REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
7.Apr.99

3. REPORT TYPE AND DATES COVERED
DISSERTATION

4. TITLE AND SUBTITLE
A FRAMEWORK FOR EFFECTIVE ALGORITHM VISUALIZATION USING ANIMATION-EMBEDDED HYPERMEDIA

5. FUNDING NUMBERS

6. AUTHOR(S)
LT COL HANSEN STEVEN R

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AUBURN UNIVERSITY MAIN CAMPUS

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
FY99-96

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

14. SUBJECT TERMS

15. NUMBER OF PAGES
257

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

18. SECURITY CLASSIFICATION OF THIS PAGE

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

DISSERTATION ABSTRACT

A FRAMEWORK FOR ANIMATION-EMBEDDED HYPERMEDIA VISUALIZATION OF ALGORITHMS

Steven Ross Hansen

Doctor of Philosophy, March 19, 1999
(M.S., Wright State University, 1986)
(B.S., Brigham Young University, 1981)

271 Typed Pages

Directed by N. Hari Narayanan

If a "picture is worth a thousand words," then why have attempts over the past decade to use pictures and animations to replace or supplement traditional instructional methods for teaching algorithms produced such disappointing results? Numerous studies and experiments have been conducted to show that pictures and animations can improve learning of challenging, abstract concepts like mathematical proofs and the algorithms used in computer science. While the pictures and animations seem to be enthusiastically received by the students, none of the studies have produced results that show consistently and conclusively that these visual tools actually *improve* learning. In fact, the accumulated empirical evidence is mixed at best, and could easily lead one to abandon the premise that animations are powerful vehicles for effectively conveying the dynamic behaviors of algorithms. However, this dissertation reports on research based on the premise that a rethinking of algorithm animation design is required in order to harness its power to enhance learning. Research reported here explores the integration of previous work in algorithm animation systems with recent developments in the cognitive and educational domains to produce a *new* model for using software visualizations to improve student comprehension. The model is based on focused learning objectives that drive a top-down design that carefully divides abstract concepts into discrete chunks for learning. The model takes a user-centered ("what do we *need* to show") view rather than a designer-centered ("what *can* we show") view, and employs hypermedia and multimodal presentation techniques to improve learning effectiveness. The key insights are that for algorithm animations to be effective, (1) they should be introduced using interactive analogies and real-world examples that serve a priming role for subsequent learning, (2) the animations should be presented within a framework that includes explanatory information in other appropriate media, and (3) the animations should be presented in varying levels of detail depending on the learner's capability. In this dissertation, we first summarize prior research on algorithm animation. Second, we discuss the theoretical foundations of our approach, architecture of the resulting hypermedia algorithm visualization system, and empirical studies that show a significant advantage for the system. We then present ablation studies that explore the features that made our framework effective, and conclude with a discussion of ways this framework can be implemented for presentation over the Internet.

A FRAMEWORK FOR ANIMATION-EMBEDDED HYPERMEDIA

VISUALIZATION OF ALGORITHMS

Steven Ross Hansen

Certificate of Approval:

James H. Cross II

James H. Cross II
Professor
Computer Science and Engineering

N. Hari Narayanan

N. Hari Narayanan, Chairman
Assistant Professor
Computer Science and Engineering

Kai-Hsiung Chang

Kai-Hsiung Chang
Professor
Computer Science and Engineering

Roland Hübscher
Assistant Professor
Computer Science and Engineering

John F. Pritchett
Dean
Graduate School

A FRAMEWORK FOR ANIMATION-EMBEDDED HYPERMEDIA VISUALIZATION OF ALGORITHMS

Steven Ross Hansen

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctorate of Philosophy

Auburn, Alabama

March 19, 1999

VITA

Steven Ross Hansen, son of Darrel and Margaret (Doxey) Hansen, was born September 23, 1956, in San Angelo, Texas. He graduated with a Bachelor of Science degree in Computer Science from Brigham Young University in June, 1981. After working as a computer systems analyst with the United States Air Force, he attended and graduated from Wright State University, Ohio with a Master's Degree in Computer Science in March,1986. He continued his career as a computer systems specialist in the Air Force and has taught computer courses for Sinclair Community College, Ohio, Texas Luthern College, Texas, and the Air Command and Staff College, Alabama. He entered Graduate School at Auburn University in January, 1996. He married Elizabeth Long, daughter of Page and Frances (Chase) Long, on May 19, 1979. They have four children: Kristina Renae Hansen, Daniel Steven Hansen, Trieste Michelle Hansen, and Kyle Steven Hansen.

DISSERTATION ABSTRACT
A FRAMEWORK FOR ANIMATION-EMBEDDED HYPERMEDIA VISUALIZATION OF ALGORITHMS

Steven Ross Hansen

Doctor of Philosophy, March 19, 1999
(M.S., Wright State University, 1986)
(B.S., Brigham Young University, 1981)

271 Typed Pages

Directed by N. Hari Narayanan

If a "picture is worth a thousand words," then why have attempts over the past decade to use pictures and animations to replace or supplement traditional instructional methods for teaching algorithms produced such disappointing results? Numerous studies and experiments have been conducted to show that pictures and animations can improve learning of challenging, abstract concepts like mathematical proofs and the algorithms used in computer science. While the pictures and animations seem to be enthusiastically received by the students, none of the studies have produced results that show consistently and conclusively that these visual tools actually *improve* learning. In fact, the accumulated empirical evidence is mixed at best, and could easily lead one to abandon the premise that animations are powerful vehicles for effectively conveying the dynamic behaviors of algorithms. However, this dissertation reports on research based on the premise that a rethinking of algorithm animation design is required in order to harness its power to enhance learning. Research reported here explores the integration of previous work in algorithm animation systems with recent developments in the cognitive and educational domains to produce a *new* model for using software visualizations to improve student comprehension. The model is based on focused learning objectives that drive a top-down design that carefully divides abstract concepts into discrete chunks for learning. The model takes a user-centered ("what do we *need* to show") view rather than a designer-centered ("what *can* we show") view, and employs hypermedia and multimodal presentation techniques to improve learning effectiveness. The key insights are that for algorithm animations to be effective, (1) they should be introduced using interactive analogies and real-world examples that serve a priming role for subsequent learning, (2) the animations should be presented within a framework that includes explanatory information in other appropriate media, and (3) the animations should be presented in varying levels of detail depending on the learner's capability. In this dissertation, we first summarize prior research on algorithm animation. Second, we discuss the theoretical foundations of our approach, architecture of the resulting hypermedia algorithm visualization system, and empirical studies that show a significant advantage for the system. We then present ablation studies that explore the features that made our framework effective, and conclude with a discussion of ways this framework can be implemented for presentation over the Internet.

ACKNOWLEDGEMENTS

The author would like to thank Dr. Mary Hegarty of the University of California, Santa Barbara for her help with the statistical analyses, Dr. N. Hari Narayanan for his patience, insight and motivating influence in the conduct of this research, Dr. Dean Hendrix and Dr. Daniela Marghitu at Auburn University for graciously supporting this research by providing access to their classes and computer laboratory resources, to the students in the Computer Science and Engineering Department for their unselfish participation in the experiments that form the backbone of this research, and to the National Science Foundation for providing generous funding to support this project. Thanks are due to family members Libby, Kristi, Danny, Trieste and Kyle for their loving support during the course of this research.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. INTRODUCTION

"A picture is worth a thousand words," so goes the old Chinese proverb. There is certainly a wealth of evidence that supports this notion of the educational richness and power of visual over textual media. Psychological studies of learning, memory, motivation, and problem solving show that coupling pictures with text forms a very effective combination to help students learn new concepts. One study, considering the tremendous advances in computer technology to produce and deliver graphical software, suggested that if a picture was worth a thousand words, then a dynamic simulation may be worth a thousand pictures (Whitney & Urquhart, 1990). Many computer science instructors and researchers have embraced this idea, and have endeavored over the past 15 years to exploit the dynamic power of animations to help students learn algorithms. Dozens of systems have been developed, enthusiastically praised by both students and instructors. However, before any new instructional technique is advocated, it should be empirically tested to show it provides an educational advantage over existing methods. Intuition and opinion alone are not sufficient.

Herein lies the perplexing paradox. Contrary to widespread intuition, despite the enthusiasm, despite the tremendous advances in multimedia technology, and despite over a decade of extensive research, attempts to use animations to facilitate learning algorithms have failed to produce compelling evidence of their instructional effectiveness and have not led to their widespread use in computer science curricula. Is it because animation is not an effective teaching medium? Should educators use these results to forgo the use of algorithm animation?

Current empirical results seem to defy our intuition, and may lead to prematurely abandoning a promising instructional approach. Therefore, we believe that further investigation is required to identify factors that lead to success, and this forms the primary goal of this research.

The purpose of this research is to study and build on previous efforts, based on the premise that previous attempts at using animation to teach algorithm behavior were unsatisfactory not because of a flaw with animation as a technique, but because of the approach used to convey the animations. Our working hypothesis is that a rethinking of algorithm animation design is required in order to harness its power to enhance learning. The goals of the research are to develop a theoretical framework of visualization design that leads to effective instruction, to conduct a series of formal empirical studies using computer science students to validate the framework, and produce a system that allows widespread use of the framework over the Internet.

This document is structured as follows. Chapter 2 contains a discussion of previous research in cognitive science, hypermedia, and algorithm animation, including a brief survey of existing algorithm animation systems and empirical studies involving those systems. Chapter 3 presents the unique combination of features that we believe are needed to provide a framework for effective learning using animation and hypermedia, and describes how those features were incorporated in the design of a system developed to test our ideas and hypotheses. Chapter 4 describes the experiments that we conducted to validate the effectiveness of our framework against popular teaching methods such as textbooks, lectures, and other animation systems. Chapter 5 dissects and analyzes the framework in a series of experiments that removed selected features and views in order to identify those components that led to the significant results reported in Chapter 4. Chapter 6 describes the efforts to make the experimental framework available to the general public as well as work performed to port the capabilities to Internet-based platforms. Chapter 7 summarizes this research and presents ideas for future research. Finally, there are several appendices that provide the materials used during the various experiments, screen captures of each of the visualizations that have been created to date, and interaction data captured during the experiments that may be of value to other research efforts.

2. PRIOR RESEARCH

The foundations of current algorithm animation research are rooted in developments and breakthroughs in other areas. This chapter highlights relevant findings in diverse topics such as graphics, multimedia, human-computer interaction design, computer-based instructional systems, and animation. A survey of algorithm animation systems follows, along with a discussion of empirical research that has been conducted to evaluate the educational effectiveness of these systems. This will set the stage for Chapter 3, which examines possible reasons for the mixed and disappointing results of current systems and ways to address the shortcomings.

2.1. RELATED WORK: 'PAVING THE WAY'

2.1.1. THE POWER OF MULTIMEDIA

The past decade has witnessed dramatic improvements in the power of computer hardware and the ease of use of graphical software programs. Coupled with extremely attractive prices and easy-to-use graphical software, superb multimedia presentations have become a widespread reality. Exploiting these features to improve the process of training and education is understandably increasing. It can be argued that the visual sense is the strongest source of human information acquisition, constantly receiving and processing millions of bits of information. Blackwell (1996) states that pictures are good at showing abstraction, that pictures are a universal form of communication, and that images can supplement other forms of communications even to the illiterate. Studies show that humans can grasp the content of a picture much faster than they can scan and understand text because of our ability to recognize spatial configurations and relationships (Kamada & Kawai, 1991), and that visual images increase learner attention (Daily, 1994). Matlin (1989) cited several studies on retention and recognition, where people had a remarkable 99.7% retention rate of images seen within 2 hours, and 63% recollection of 2560 images shown a year before. Duchastel (1978) states that pictures serve three key roles: (1) they attract the learner's attention; (2) they can help the reader understand information that is hard to describe in words; and (3) they can reduce the likelihood that information is forgotten.

Putting motion to pictures, thereby creating an animation, presents an extra dimension beyond static diagrams. A study by Hegarty (1992) showed that individuals, when presented with static diagrams of mechanical devices, generally perform *mental animations* by inferring the motion/kinematics of the objects in the diagram along the causal chains of events they perceive. Explicit, computer-presented animation, in addition to depicting aspects such as trajectory, synchronization and motion, can be used to call attention to aspects of a problem that might otherwise go unnoticed (Brown, 1988a). Animation has been shown to improve understanding of dynamic processes and machines (Reiber, 1990). Reiber, Boyce and Assad conducted a study (1990) using a computer based science lesson to teach introductory Newtonian mechanics to adults. Their results showed that students who viewed the animations were able to complete the post-tests in significantly less time.

Multimedia systems tap the synergy of combining visual images with other forms of media, as shown by several studies by Mayer (Mayer & Sims, 1994; Mayer, 1989). They observed that students were better able to recall and transfer what they had learned from a science textbook when text and illustrations were presented next to each other rather than separately. Mayer claimed that the labeled illustrations played two roles: guiding student attention and helping them build internal connections between ideas in the text. Subsequent experiments by Mayer and Anderson (1991, 1992) considered the use of animation and narration to help students understand scientific explanations. In two experiments, college students viewed animations and/or listened to narrations explaining the operation of a bicycle pump. Students who saw the animation and listened to the narration simultaneously outperformed all other groups on a creative problem solving test. Their work is an extension of the 'Dual-Coding' theory described by Paivio (1986) which suggests that humans possess two information processing systems, one that stores and represents information verbally and another that is visual. Learning is enhanced when material is presented to the student in these modes in ways that encourages the building of referential connections between the visual and verbal modes. Mayer and Moreno (1998) summarized these results as a set of principles that designers should follow to tap the potential of multimedia learning environments: designers should (1) present explanations in multiple media, (2) present the multiple media contiguously, (3) use audio to avoid splitting the student's visual attention and (4) keep media brief and focused, thereby enhancing coherence.

A study by Palmiter and Elkerton (1993) compared the use of animated demonstrations, written text and narrated animation for teaching users how to operate a particular graphical interface. They expected that the narrated animation group would perform the best with animation aiding the initial learning and narration aiding retention and transfer. Their results showed that the performances of the animation-only and narrated animation groups were very similar and better than the text group. The animation group was faster and enjoyed the lesson more, according to user surveys.

2.1.2. HYPERMEDIA IN COMPUTER-BASED EDUCATION

Hypermedia and multimedia are being used increasingly as teaching and tutoring tools. Daily (1994) cites several studies of multimedia instruction that report dramatic improvements in retention and learner attention while at the same time reducing cost and overall training time. She presents the results of several experiments comparing the effectiveness of multimedia learning to traditional classroom techniques for certain engineering applications. She concludes that multimedia greatly increased student participation and was at least as effective as traditional teaching methods. Bagui (1998) cites another eight studies that show computer-based multimedia helped people learn more quickly than classroom lecture. Crosby and Stelovsky (1995) reported similar findings, stating that the dynamic, interactive and visual capabilities of multimedia courseware led to better student performance and higher class attendance for the multimedia instruction sessions.

Narayanan and Hegarty (1998) describe research involving the efficacy of diagrams, text, and animation as teaching and training tools. They build a model of comprehension and provide guidelines for the development of interactive hypermedia manuals, particularly of machines where motion is involved. The central tenet of their model is that users follow certain steps when given a multimodal presentation: (1) they decompose the material, (2) they construct a mental model based on both referential and representational connections, (3) they determine the causal chains of events, and (4) they mentally animate the model to infer results. Properly designed hypermedia systems facilitate all these steps in ways that conventional texts cannot.

In addition to providing information in multiple modes, hypermedia technology allows multiple ways of organizing it and presenting it to the student. Most studies favor a guided but flexible navigational approach, where novice students are guided by a natural navigational path through the screens of an educational sequence, but provides knowledgeable users with the ability to bypass sections and directly navigate to desired topics. A more critical issue is how to organize educational content. Information processing theory (Miller, 1956) refers to a meaningful unit of information as a 'chunk,' and suggests that the average person can hold up to seven chunks in short term memory at a time. Blackwell (1996) suggests that the reason people learn more quickly through pictures than text is because a picture is stored in the brain as an information-rich chunk, in contrast to words that are stored and processed one at a time. Other authors suggest that chunk sizes are related to expertise (Vessey, 1985) and that chunking may help in the retention process (Wang and Arbib, 1993). Bagui (1998) claims that hypermedia systems are inherently chunked because the viewer sees information presented in a prepared series of pictures, text and sound clips connected together in a prepared sequence. Another study (Recker, Ram, Shikano, Li & Stasko, 1996) suggest that information is processed more easily and faster if it is appropriately chunked into 'cognitively relevant' pieces based on learning goals.

Simply using multimedia in computer-based education is not a guarantee for educational success, however. One criticism of multimedia training has been labeled the 'TV syndrome' or the 'hands-on, mind-off' problem, where the media is entertaining and colorful but educationally useless because it fails to engage the reasoning and inference processes that must be used to understand complex material. Addressing this concern is research in the areas of interactivity, self-diagnosis and systems based on learning objectives, discussed below.

Hypermedia is not inherently interactive simply because the reader controls navigation and pacing of the courseware. Merely watching a simulation is not sufficient to trigger learning (Reed, 1985; Pane, Corbett & John, 1996; Rappin, Guzdial, Realff & Ludovice, 1997). Yaverbaum, Kulkarni and Wood (1997) summarize several studies about the advantage of active over passive learner paradigms in hypermedia systems, stating that when students are *doing* and not just *watching*, multimedia offers serious improvements, and that the active paradigm engages the student better in the process of discovery, reflection and explanation. Theories in constructivism suggest that the learner needs to be actively involved rather than passively accepting information, and views the teaching process as a form of apprenticeship, where the expert (the teacher or carefully designed computer-based instruction) *collaborates* with the student using scaffolded lessons that nurture, coach, engage problem solving skills and encourage reflection (Narayanan Hmelo, Petrushin, Newstetter, Guzdial & Kolodner, 1995). One such system, called STABLE, (SmallTalk Apprentice-Based Learning Environment), showed that a principled approach to hypermedia design led to success even with less-than-perfect scaffolding (Guzdial & Kehoe, 1998).

In developing a multimedia learning environment to study and analyze animal behavior, researchers stated that "to expect a learner to discover [complex skills and knowledge] through free exploration in a rich multimedia environment is not enough. A pure hypermedia approach is not sufficient" (Boyle, Stevens-Wood, Feng and Tikka. 1996). To combat learner complacency, they developed learning objectives and used tests at the end of learning blocks to ensure target competencies were mastered before letting students move on to new material. They reported that over 80% of the students indicated a preference for this feature. Shikano, Recker and Ram (1996) examined how students interacted with multiple media employed by their hypermedia system called AlgoNet. They noted that hypermedia systems can falsely lead some students to the illusion that they know all the materials covered, and suggested the use of a self-diagnosis node for students to use to reflect on their understanding, and help determine which nodes to visit. Thus, scaffolding, navigational aids, and self-reflective prompts are needed to avoid the behavior noted by Pane et al. (1996), who observed that "even motivated students cannot be relied on to take full advantage of exploratory opportunities" in hypermedia systems that lack these capabilities.

Pane also observed the importance of authors using well-defined educational goals to help guide and enhance hypermedia learning. The importance of well-thought out learning objectives cannot be underestimated. Bloom (1956) showed how learning objectives are critical to success in each of the six levels in his taxonomy of learning. In the cognitive domain, which deals with the acquisition and use of information, these levels are:

- Knowledge: activities such as remembering, memorizing, recognizing, and recalling facts.
- Comprehension: activities such as interpreting, rewording, explaining, organizing, and translating information and ideas.
- Application: activities such as constructing, demonstrating, problem solving, and applying facts/rules/principles to produce a desired result.
- Analysis: activities such as decomposition, classification, identifying components, and contrasting.
- Sythesis: activities such as organizing, designing, creating/combining ideas to form a new one, and inferences about unstated behaviors.
- Evaluation: activities such as value judgements, defending, arguing, resolving controversies, and assessing correctness.

Bloom showed how instructors could find greater success in teaching when lessons were developed to meet one of these learning levels, especially when the objectives were written and stated beforehand. Reed (1985) referred to it another way, as an 'external lesson strategy,' noting its importance in order to focus student attention on pertinent features of the animated display. Yaverbaum et al. (1997) concluded that multimedia offers a good solution to certain educational ills when delivered within a framework based upon appropriate learning objectives and good design principles. These are essential for *understanding*.

2.2. ALGORITHM ANIMATION RESEARCH

2.2.1. OVERVIEW

Over ten years ago, with the unveiling of the movie *Sorting out Sorting* at SIGGRAPH-81, the idea of using graphics and animation to illustrate the dynamic behavior and functionality of computer algorithms was born. It appeared to hold great promise as an instructional aid, and since then over a hundred software visualization systems have been built (Price, Baecker & Small, 1993). Some of the best known algorithm animation systems are Balsa (Brown, 1988a, 1988c) and Tango (Stasko, 1990), which, with a host of variants and successors, seem to have been developed in the belief that algorithm animations would serve as effective supplements to help students learn about algorithms. This was a compelling goal, since computer science students generally find the subject of algorithm design particularly challenging, and any techniques that could help the learning process would save time and reduce frustration.

2.2.2. WHAT IS ALGORITHM ANIMATION?

An algorithm is a clearly specified set of instructions to be followed to efficiently and effectively solve a problem. Governed by mathematical constraints, they define a series of operations that manipulate abstract data structures over time until a terminating condition is reached. Algorithms are used extensively in computer science, and are one of the building blocks of computer software. However, unlike the tangible objects studied in other sciences, algorithms are inherently abstract entities. Not only do they lack any concrete representation in the natural world, they define dynamic processes that change over time, making them difficult to teach and to learn. Most authors and instructors make use of graphical notations and diagrams as visual aids in an effort to provide a concrete representation of the abstract components of an algorithm. Many use sequences of diagrams to help depict algorithmic behavior and changes over time. Recent breakthroughs in computer graphics technology provide new opportunities to illustrate algorithmic behavior. One such technique is animation.

Algorithm animation has been defined as the process of abstracting the data, operations and semantics of computer algorithms, and then creating moving graphical views of those abstractions (Stasko, Domingue, Brown & Price, 1998). The animation becomes a time-evolving graphical depiction of how the algorithm's operations affect the data structures associated with the algorithm. By mapping the abstract entities of the computation to the visual entities on a computer screen, algorithm animation provides the viewer with a concrete depiction of how the algorithm manipulates data items and achieves its goals over time through a series of operations. Hence, well-designed algorithm animations enable students to see the inner workings that are otherwise hidden. The typical approach employed by most animation systems is to use simple geometric shapes (bars, circles, and dots) to represent individual data elements, and to move these shapes to depict the manipulation and transformation of data by the algorithm.

Many believed that creating an animation of an algorithm is simply a process of assigning a graphical shape to display each access or update to the variables associated with an algorithm's execution. However, algorithm animation turned out to be much more difficult. In his seminal work on algorithm animation, Brown (1988a) summarizes several problems that algorithm animation designers must face, including the difficulty of capturing operations, real-time performance and limitations of information displays. Of these, deciding on *what* to depict and *how* to represent it are generally recognized as the most challenging task and the one for which the least guidance is available (Brown, 1988a; Brown & Hershberger, 1991; Stasko, Badre

& Lewis, 1993; Naps & Bressler, 1998). This is because an algorithm is more than a collection of variables. An algorithm represents a series of operations that affect the data stored in data structures. The meaning and semantics of the operations can be quite subtle, yet understanding those semantics is the crux of learning an algorithm. For example, consider the difference between an initialization operation on a variable and a swap operation. Both involve placing a value into a variable, but the swap operation involves a careful handling of the contents by using a temporary variable or key information will be lost. Because the semantics of the operations like these are not always readily apparent, deciding how best to depict and highlight those operations using graphics is the challenge the animation designer must face.

2.2.3. WHY STUDY ALGORITHM ANIMATION?

Computer science students generally find algorithm analysis and design a most challenging subject. It is easy to understand the difficulty. Algorithms manipulate abstract data structures (a tough subject in and of itself) whose contents change over time. The algorithm's behavior changes depending on the input values. Hence, algorithms are both abstract *and* dynamic entities, but the methods used to teach them (textual descriptions and diagrams) are concrete and static, not always well suited to describing dynamic processes. A survey of teaching methods (Badre, Beranek, Morris & Stasko, 1991) shows that over 80% of instructors used textbooks, diagrams and lectures to teach about algorithms. This means that teaching students how to comprehend and analyze the behavior of fundamental computer science algorithms depends mainly on an instructor's ability to impart an understanding of the dynamics involved using lectures and static media. For this reason, other methods have been sought to help students learn, beginning with the unveiling of the movie "*Sorting out Sorting*" at the 1981 ACM SIGGRAPH conference which demonstrated the use of graphics and animation to illustrate algorithm behavior. This 30-minute film took 3 years to create and used a series of color-enhanced screen captures and explanatory narrative to 'teach' nine sorting methods. The film culminated in a 'grand race,' where 2500 data items were depicted simultaneously in a separate window running each algorithm to show the performance difference in real time (see Figure 2.1).

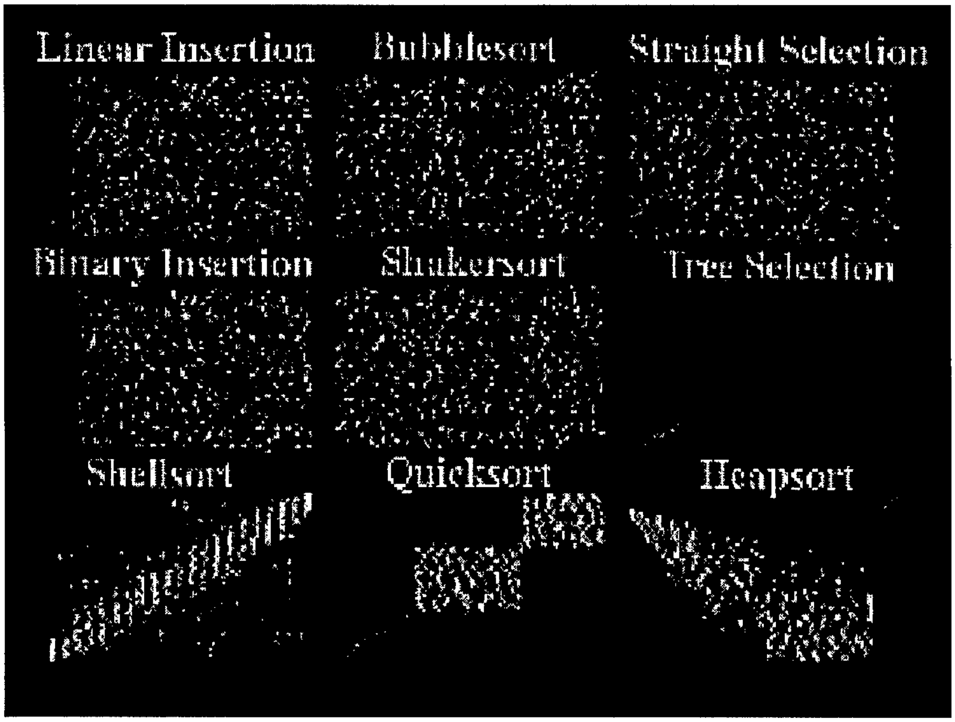


Figure 2.1. Screen Capture From the Film *Sorting Out Sorting* (Baecker, 1981)

This movie marked the beginning of research on algorithm animation and in the years since, numerous systems have been developed to facilitate learning in a wide variety of settings, such as:

- to supplement lectures on algorithms in electronic classrooms (Brown, 1988a; Stasko, 1997; Gurka & Citrin, 1996; Bazik, Tamassia, Reiss & Van Dam, 1997);
- to illustrate the basic operations of abstract data types in a computer science laboratory (Naps, 1990; Pierson & Rodger, 1998);
- to facilitate debugging logic problems (Baecker, DiGiano & Marcus, 1997);
- to help programmers understand and find performance bottlenecks in parallel programs (Heath & Etheridge, 1991; Kraemer & Stasko, 1993) ;
- to depict operating system functions (Hartley, 1994); and
- to trace the parsing of natural language processing routines (Rogers, Gaizauskas, Humphreys & Cunningham, 1997).

The general expectation with algorithm animation as an instructional aid is that providing concrete depictions should improve comprehension and facilitate learning. However, relatively little empirical evaluation of algorithm animation systems has been conducted to substantiate this belief.

2.2.4. AN OVERVIEW OF EXISTING ALGORITHM ANIMATION SYSTEMS

One of the first algorithm animation systems was Balsa (Figure 2.2) which was widely used in Computer Science classrooms at Brown University (Brown & Sedgewick, 1985). With the benefit of hindsight, a significant shortcoming with Balsa was that the animations were not informative enough—they merely illustrated 'interesting events' identified by the animator/instructor in the execution of the algorithm. The animations typically showed large numbers of data elements represented as simple geometric elements being rearranged spatially on a display. For instance, the input to a sorting algorithm was shown as a large number of dots, representing the input values to be sorted, strewn randomly on the screen. As the algorithm executed, these dots slowly rearranged themselves into a diagonal line. Such displays, without additional pictorial and verbal explanations, *impede* comprehension by making it hard for the learner to construct referential and representational connections in their mental models (Narayanan & Hegarty, 1998). Balsa was used in classrooms where, presumably, the instructor provided the needed explanations. We are not aware of any literature that presents evaluation of the system or assessment of its effect on student learning. Nevertheless, Balsa marked the beginning of the first significant research program on algorithm animation, and a variety of successor systems have followed (Price et al., 1993; Stasko et al., 1998). A selection of such systems are described briefly in the pages that follow.

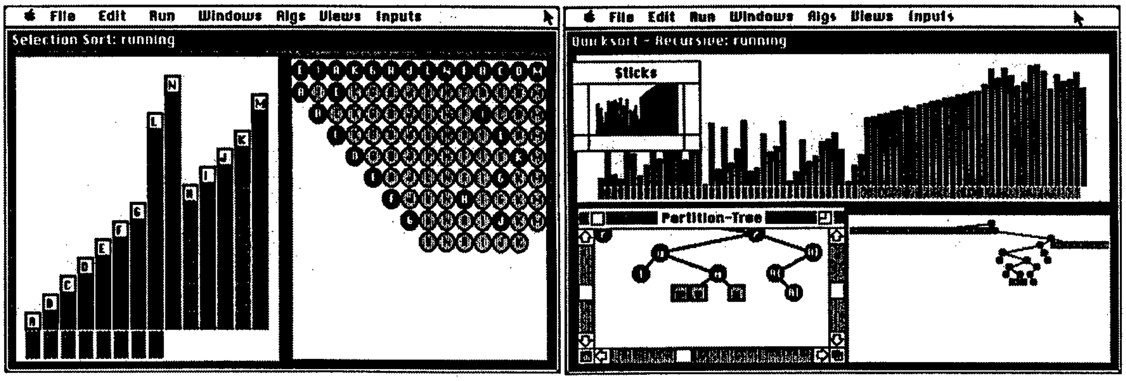

File Edit Run Windows Algs Views Inputs
Selection Sort: running
File Edit Run Windows Algs Views Inputs
Quicksort - Recursive: running
Sticks
Partition-Tree

Figure 2.2. Balsa-II Screens Views of the QuickSort and SelectionSort Algorithms (Brown, 1988a)

Zeus (Brown, 1991): Written in Modula-3, Zeus was the first animation system that supported use of color, sound, synchronized views, and introduced 3D graphics. It was designed for multi-threaded, multi-processor environments which allowed depiction of parallel programs. While it was compiled for several computing platforms, Zeus did not see widespread use outside of the laboratory. However, several weeklong programming events were conducted at the DEC Systems Research Center to demonstrate the flexibility and variety of Zeus animations. The screen capture in Figure 2.3 is copied from Brown (1994).

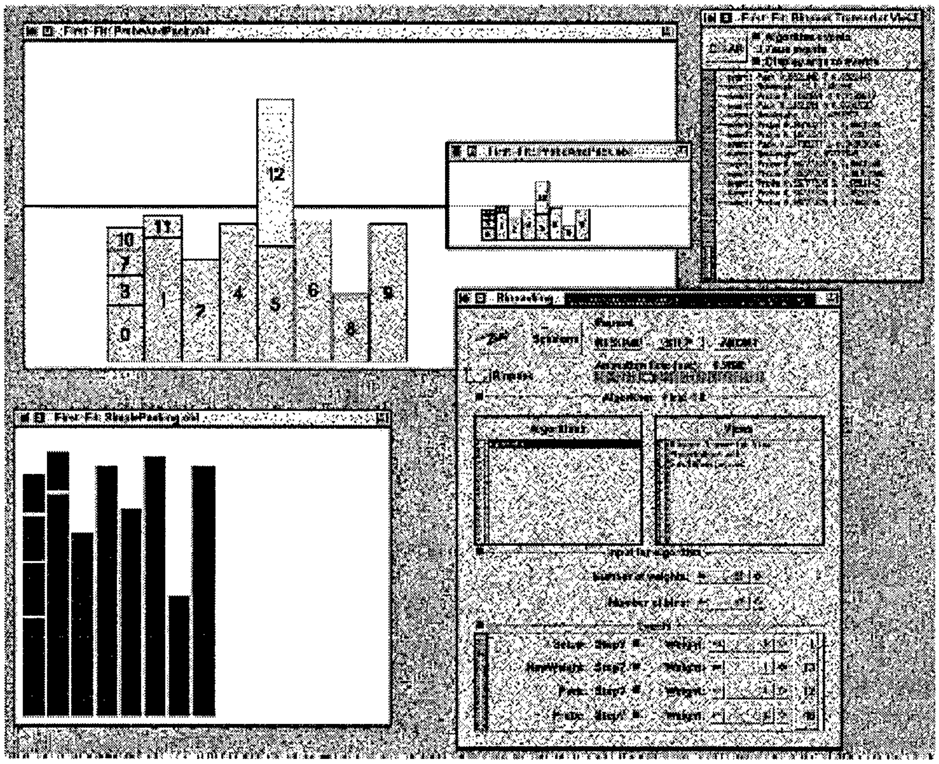
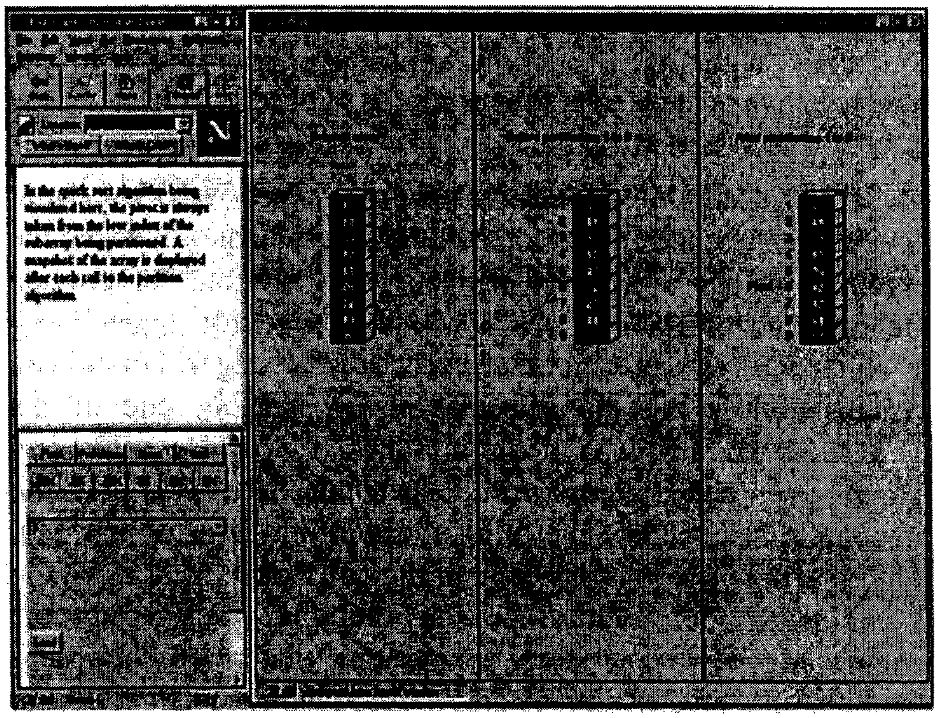


Figure 2.3. Zeus Screen View of the BinPacking Algorithm

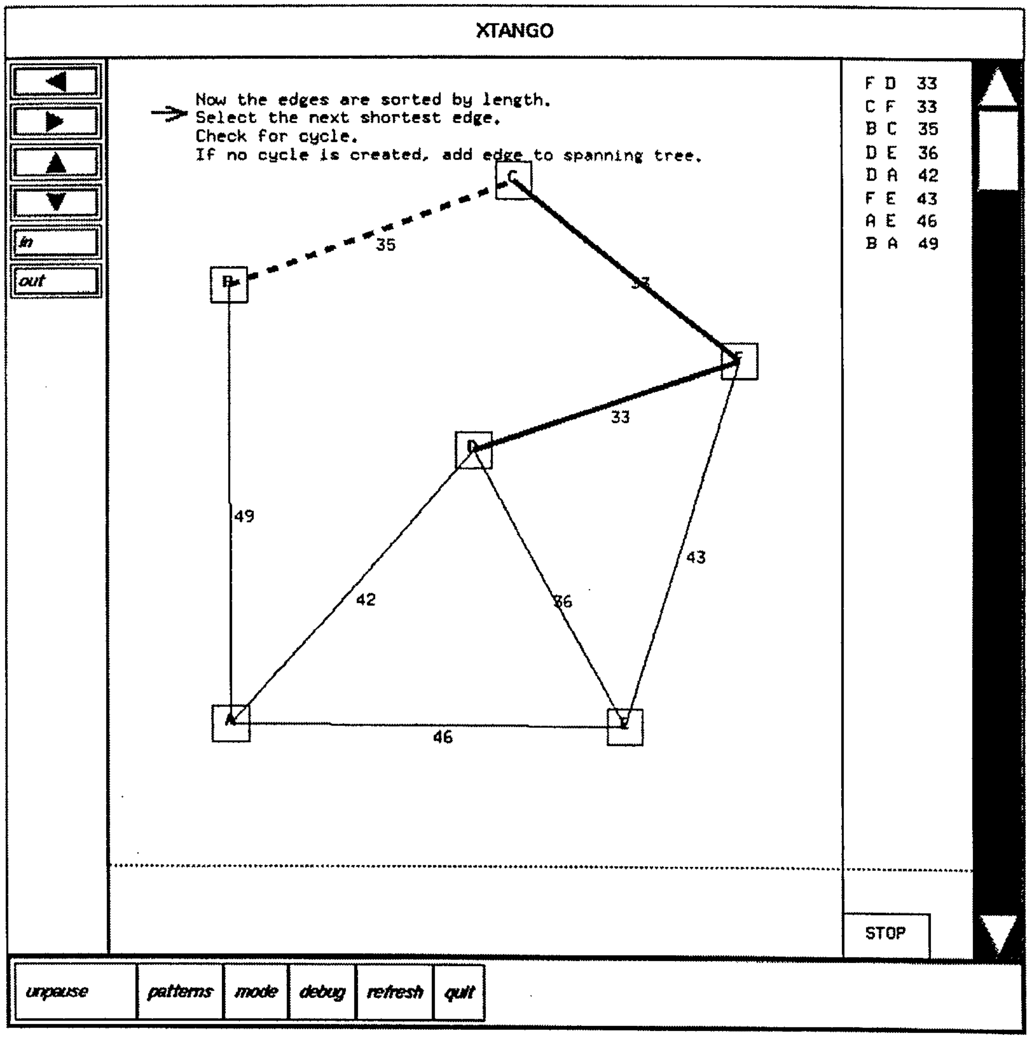
GAIGS (Naps, 1990): Developed with a focus on data structures rather than algorithm animation, GAIGS was one of the first systems to support multiple snapshot views of algorithms. In effect, this system used 'before' and 'after' pictures of algorithms in progress and relied on the user's ability to mentally animate the changes between each frame. Some research suggests this is a powerful approach because it forces mental animation (Hegarty, 1992) by not providing explicit animation for every step. WebGaigs (Naps & Bressler, 1998) extended this approach to allow delivery of snapshot animations over the Internet. A screen capture of the WebGaigs system is shown in Figure 2.4 and was adopted from Naps & Bressler (1998).



In the quick sort algorithm being visualized here, the pivot is always taken from the low index of the subarray being partitioned. A snapshot of the array is displayed after each call to the partition algorithm.

Figure 2.4. WebGaigs Screen View Showing Multiple Snapshots of the Quicksort Algorithm

Tango (Stasko, 1990): Tango enhanced Balsa's 'interesting event' approach, adding several refinements and porting the capabilities to the Unix platform. XTango (Stasko, 1992) extended those capabilities to X-Windows, and Samba and Polka (Stasko, 1997) were written to facilitate animation playback. There is currently a version for Microsoft Windows also. Tango provided a set of commands that let the author specify and manipulate the shape, size, location, and color of various graphical objects according to how the animation author wanted the operations of the algorithm to appear. The collection of Tango commands formed an 'animation script' that could be played over and over to illustrate the operation of the algorithm on a specific set of data. The script could be generated by a word processor or by carefully inserting print statements into working code representing the 'interesting events' that were selected for highlighting and illustration. Tango employed a path-transition paradigm to yield smooth movement of the graphical objects rather than the strobe-like transitions of earlier systems. The smooth movement provided a pleasing and easy to follow animation for the student. Most current experimental studies of algorithm animations have been conducted using Tango and its successors. The screen capture of XTango shown in Figure 2.5 was copied from Lawrence et al., 1994).

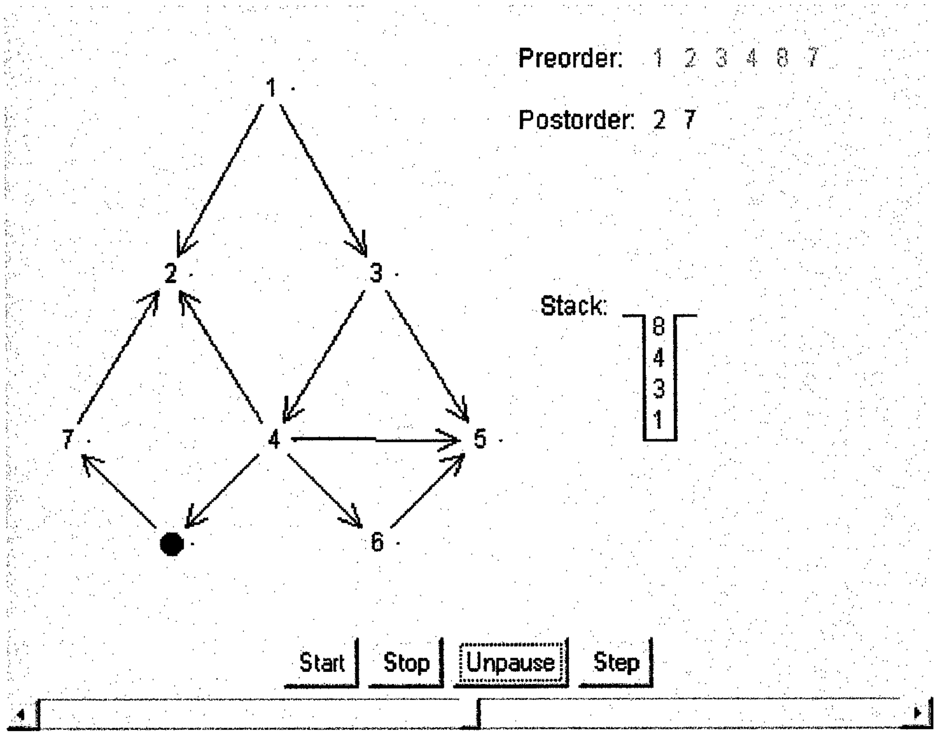


```
    Now the edges are sorted by length.
--> Select the next shortest edge.
    Check for cycle.
    If no cycle is created, add edge to spanning tree.
```

```
F D  33
C F  33
B C  35
D E  36
D A  42
F E  43
A E  46
B A  49
```

Figure 2.5. XTango Screen View of Kruskal's Algorithm

Lambada (Astrachan et al., 1996) and **JAWAA** (Pierson & Rodger, 1998): Both of these systems were built to enhance and extend the Tango framework. Like Tango, both systems make use of a graphics engine that uses a separate script file which contains commands that create and manipulate visual objects to produce the desired animation. This 'interesting event' approach allows scripts to be created from either a word processor or by inserting print statements into a working program. Unlike Tango, both Lambada and JAWAA were written in Java to allow delivery over the Internet. JAWAA provided a number of additional commands that facilitate creation and manipulation of complex data structures. Where Tango authors had to build complex data structures element by element, JAWAA authors could employ high-level commands to construct arrays, trees, graphs, stacks and other complex data structures with a single command. Figure 2.6 was taken from http://www.cs.duke.edu/~wcp/JAWAA.html.



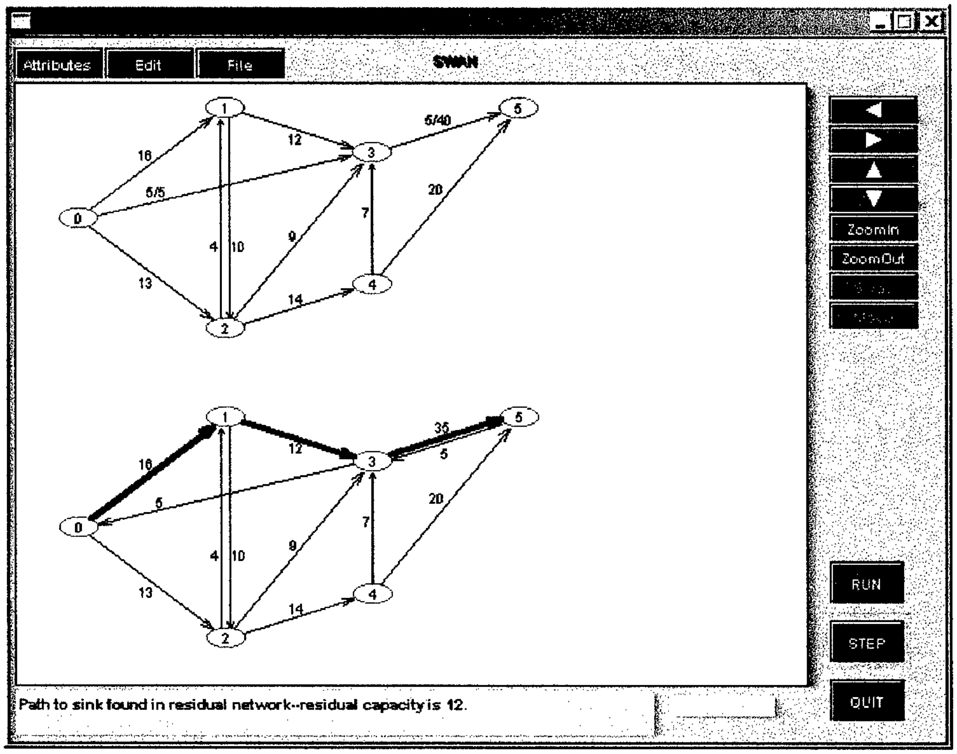
Preorder: 1 2 3 4 8 7

Postorder: 2 7

Stack: 8 4 3 1

Figure 2.6. JAWAA Screen View Showing the Depth-first Search Algorithm

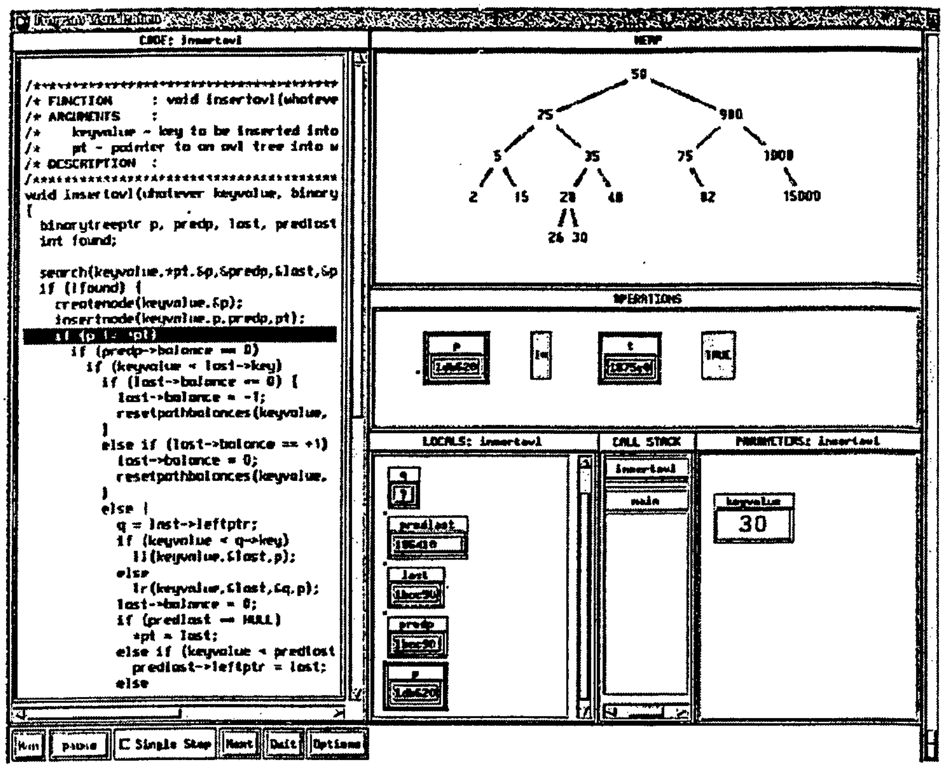
SWAN (Shaffer et al., 1996): Developed to facilitate automatic depiction of the data structures and basic execution processes of C/C++ programs and written for the MS-DOS environment. Swan views a data structure as a graph or collection of graphs including directed and undirected graphs, trees, lists and even arrays. Swan animators annotate C/C++ source code which is then compiled and produces a visual depiction of the underlying data structures of the program. The layout of the graphical components is handled automatically. The Swan interface allows the annotator to place terse explanations in a small window at the bottom of the screen to help explain the status of the execution. A sample screen view of the Swan system in action is shown in Figure 2.7 and was captured from a running version of the software that was downloaded from http://geosim.cs.vt.edu/Swan/Swan.html.



Path to sink found in residual network--residual capacity is 12.

Figure 2.7. Swan Screen View Showing a Network Flow Problem

C Animator (Sangwan & Korsh, 1998): Developed as a tool to allow visualization without annotating source code, this system provides an interesting bridge between visual program development systems and algorithm animation systems. The authors also describe their system as an excellent debugging tool. The system uses a set of specialized header files that recognize selected C/C++ variable declarations, and invokes special compiler-level calls to create visualizations of those objects automatically. Students or instructors simply make syntactic changes to data type declarations to a form that their compiler recognizes and the resulting animations are generated automatically. When executed, the program creates a set of tiled panes, containing the source code, global and local variables, function parameters, a call stack, current operations, and a visual representation of the data structure being manipulated. Figure 2.8 shows a screen view of these panes during the execution of an insert operation on an AVL tree.

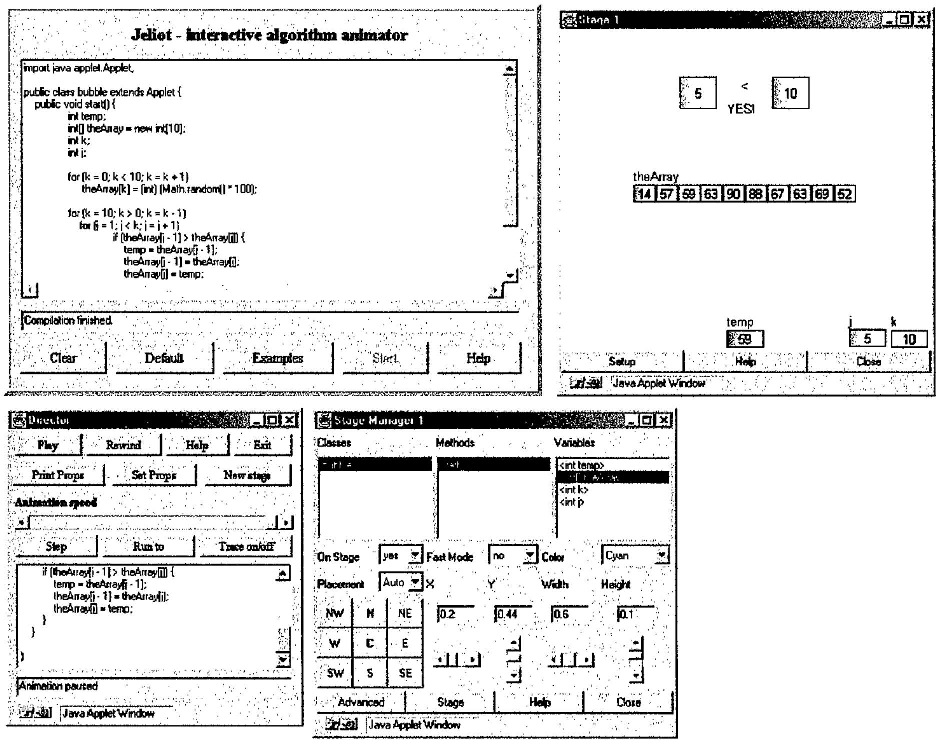


```
/*****************************************************
/* FUNCTION      : void insertavl(whateve
/* ARGUMENTS     :
/*     keyvalue - key to be inserted into
/*     pt - pointer to an avl tree into w
/* DESCRIPTION   :
/*****************************************************
void insertavl(whatever keyvalue, binary
{
  binarytreeptr p, predp, last, predlast
  int found;

  search(keyvalue,*pt,&p,&predp,&last,&p
  if (!found) {
    createnode(keyvalue,&p);
    insertnode(keyvalue,p,predp,pt);
    if (p != *pt)
      if (predp->balance == 0)
        if (keyvalue < last->key)
          if (last->balance == 0) {
            last->balance = -1;
            resetpathbalances(keyvalue,
          }
          else if (last->balance == +1)
            last->balance = 0;
            resetpathbalances(keyvalue,
          }
          else {
            q = last->leftptr;
            if (keyvalue < q->key)
              ll(keyvalue,&last,p);
            else
              lr(keyvalue,&last,&q,p);
            last->balance = 0;
            if (predlast == NULL)
              *pt = last;
            else if (keyvalue < predlast
              predlast->leftptr = last;
            else
```

Figure 2.8. The C Code Animator Application Showing an AVL Tree

Jeliot (Lahtinen et al., 1998): Jeliot was developed as a system to facilitate visualization of Java-based source code over the Internet. Animating a program with Jeliot involves a series of steps. First, the code panel is loaded with the program to be visualized. The program is a subset of the Java language, in that only a certain set of variable types are recognized by their preprocessor for animation. Once the source code is loaded, three panes appear: a controller, a stage manager, and the animation stage. The controller provides the user with an interface to characterize and manipulate the animation. The stage manager provides controls for the student to assign shapes, colors, layouts and locations to as many variables as desired. The stage is where the animated objects appear as the program executes. Figure 2.9 shows each of these panes in a screen capture of the system during execution of the selection sort algorithm, taken from http://verosaari.cs.helsinki.fi:8807/Jeliot/.



```
import java.applet.Applet;

public class bubble extends Applet {
   public void start() {
            int temp;
            int[] theArray = new int[10];
            int k;
            int j;

            for (k = 0; k < 10; k = k + 1)
                theArray[k] = (int) (Math.random() * 100);

            for (k = 10; k > 0; k = k - 1)
               for (j = 1; j < k; j = j + 1)
                        if (theArray[j - 1] > theArray[j]) {
                           temp = theArray[j - 1];
                           theArray[j - 1] = theArray[j];
                           theArray[j] = temp;
```

Compilation finished.

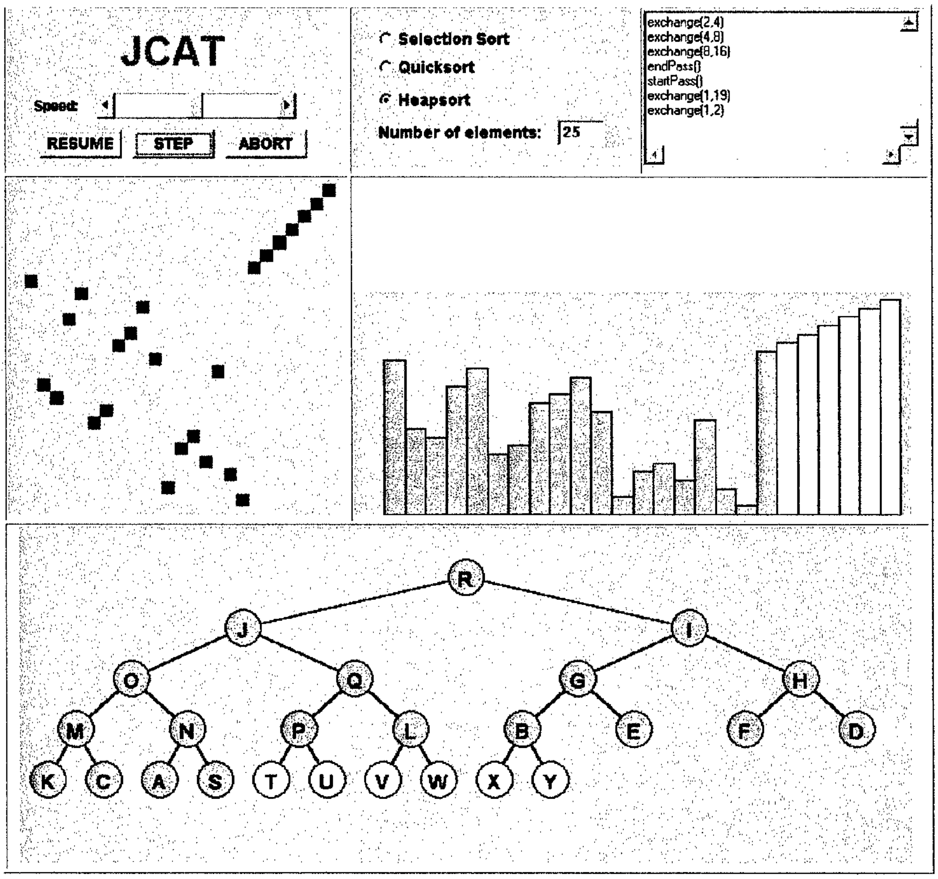
```
         if (theArray[j - 1] > theArray[j]) {
            temp = theArray[j - 1];
            theArray[j - 1] = theArray[j];
            theArray[j] = temp;
         }
   }
}
```

Animation paused

On Stage: yes
Fast Mode: no
Color: Cyan
Placement: Auto
X: 0.2
Y: 0.44
Width: 0.6
Height: 0.1

Figure 2.9. The Jeliot Framework Showing the SelectionSort Algorithm

JCAT (Brown & Najork, 1996): JCAT (Java-based Collaborative Active Textbooks) is an environment that supports Web-based animations that can be jointly viewed by multiple users. JCAT employs a form of interesting event scripting. Figure 2.10 shows a screen capture of the JCAT implementation of the quicksort algorithm, taken from http://www.research.digital.com/SRC/JCAT.

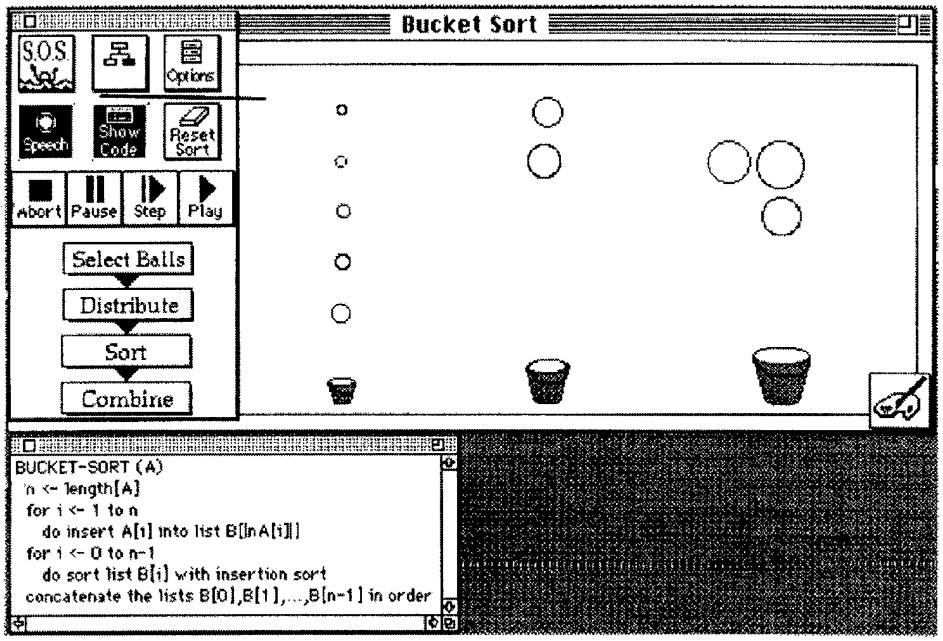


○ Selection Sort
○ Quicksort
● Heapsort
Number of elements: 25

```
exchange(2,4)
exchange(4,8)
exchange(8,16)
endPass()
startPass()
exchange(1,19)
exchange(1,2)
```

Figure 2.10. The JCAT Environment Showing the QuickSort Algorithm

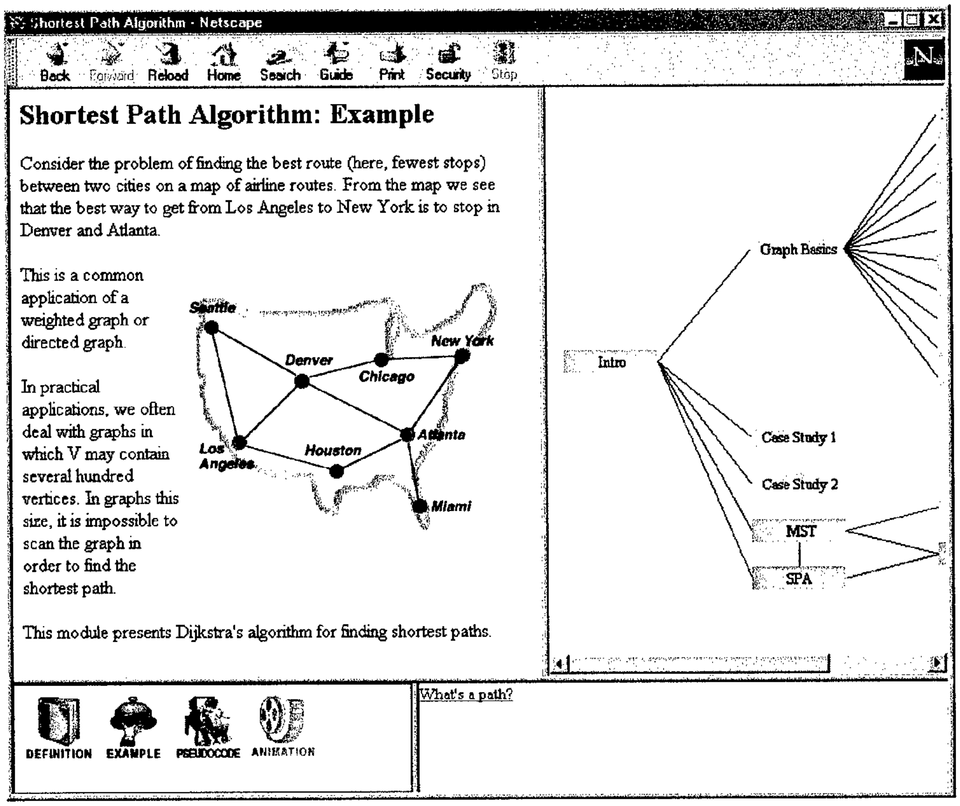
AACE (Gloor, 1992): Developed in HyperCard and currently distributed as a CDROM supplement to a textbook (from MIT Press), AACE (Animated Algorithms for Computer Science Education) was the first commercially marketed algorithm animation system. It was also one of the first systems that used real world analogies to help describe the algorithm. AACE supported interactive animations and employed multimedia presentation techniques that included multiple windows to describe the animation, such as a code window, and a window with explanatory text. Animations could not be user-developed however. Figure 2.11 shows a screen capture of the AACE system showing the bucket sort algorithm during execution, from Gloor (1992).



```
BUCKET-SORT (A)
 n <- length[A]
 for i <- 1 to n
   do insert A[i] into list B[⌊nA[i]⌋]
 for i <- 0 to n-1
   do sort list B[i] with insertion sort
 concatenate the lists B[0],B[1],...,B[n-1] in order
```

Figure 2.11. The AACE Framework Showing the Bucket Sort Algorithm

AlgoNet (Recker et al., 1995): Written in Visual Basic to explore how students interact with cognitive media types, this system never saw widespread classroom use, though some empirical usability studies with students have been conducted. AlgoNet contained lessons dealing with shortest path algorithms and included both real world case studies and examples to help introduce and reinforce subject matter. AlgoNet did not emphasize nor make heavy use of animations. The AlgoNet framework was not intended to be a general purpose algorithm animation system but rather a research framework to study use of a variety of media in cognitively relevant forms to help meet user needs and learning objectives. Figure 2.12 shows a screen capture of the AlgoNet system, taken from http://c2000.gatech.edu/brothert/research/projects/AlgoNet/start.html.



Shortest Path Algorithm: Example

Consider the problem of finding the best route (here, fewest stops) between two cities on a map of airline routes. From the map we see that the best way to get from Los Angeles to New York is to stop in Denver and Atlanta.

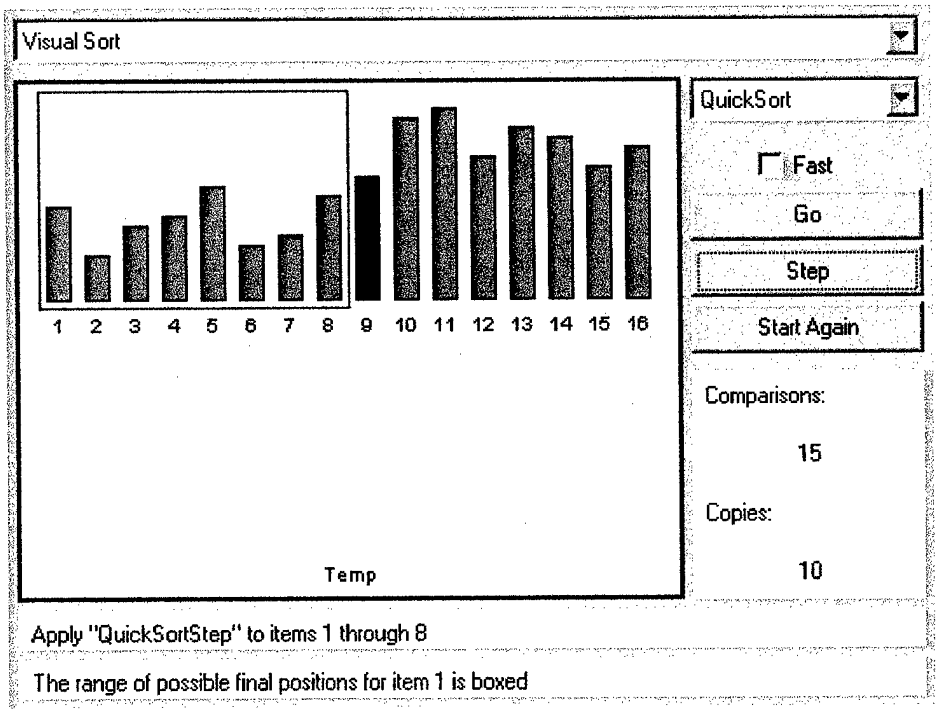
This is a common application of a weighted graph or directed graph.

In practical applications, we often deal with graphs in which V may contain several hundred vertices. In graphs this size, it is impossible to scan the graph in order to find the shortest path.

This module presents Dijkstra's algorithm for finding shortest paths.

Figure 2.12. AlgoNet Screen View

There are several other user-developed algorithm animation systems. For example, the Sun Web site includes a entertaining page showing various animated sorting algorithms (http://www.sun.com). Ecks has developed an excellent Java-based instructional tool called **XSortLab** that animates a variety of sorting algorithms, shown in Figure 2.13 and available at http://math.hws.edu/eck/cs124/labs98/lab9/xSortLab.html. This system includes two levels of explanatory statements that accompany a very intuitive graphical depiction of the algorithms. Another author, Peter Brummund, created a similar tool called the SortAnimator, shown in Figure 2.14 and available at http://www.cs.hope.edu/~alganim/animator/SortWindow.html. This tool forgoes textual explanations but includes a variety of controls for the student to tailor the presentation and playback of a number of sort algorithms. The animation is also accompanied by the algorithm code, highlighting the line being executed.


Visual Sort

QuickSort

□ Fast

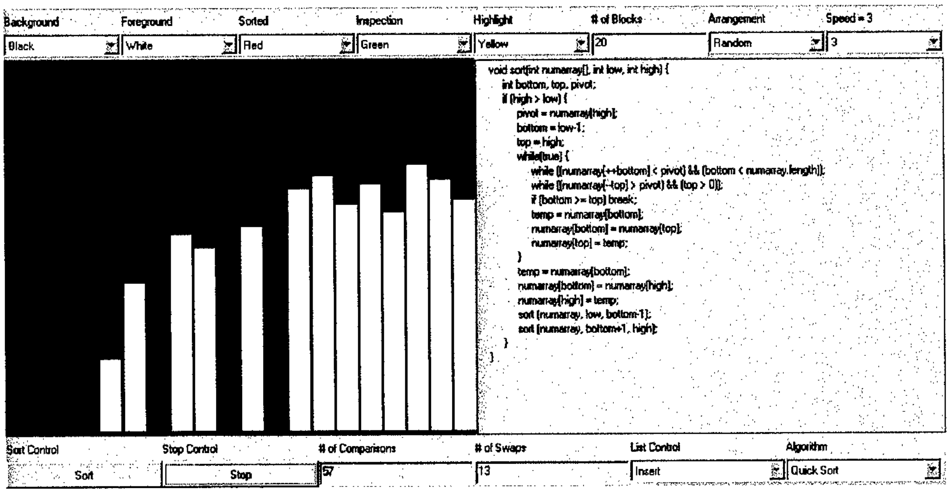
Comparisons: 15

Copies: 10

Apply "QuickSortStep" to items 1 through 8

The range of possible final positions for item 1 is boxed

Figure 2.13. The XSortLab Application Showing the QuickSort Algorithm



Background: Black
Foreground: White
Sorted: Red
Inspection: Green
Highlight: Yellow
# of Blocks: 20
Arrangement: Random
Speed = 3: 3

```
void sort(int numarray[], int low, int high) {
   int bottom, top, pivot;
   if (high > low) {
      pivot = numarray[high];
      bottom = low-1;
      top = high;
      while(true) {
         while ((numarray[++bottom] < pivot) && (bottom < numarray.length));
         while ((numarray[--top] > pivot) && (top > 0));
         if (bottom >= top) break;
         temp = numarray[bottom];
         numarray[bottom] = numarray[top];
         numarray[top] = temp;
      }
      temp = numarray[bottom];
      numarray[bottom] = numarray[high];
      numarray[high] = temp;
      sort (numarray, low, bottom-1);
      sort (numarray, bottom+1, high);
   }
}
```

Sort Control: Sort
Stop Control: Stop
# of Comparisons: 57
# of Swaps: 13
List Control: Insert
Algorithm: Quick Sort

Figure 2.14. The Sort Animator Showing the QuickSort Algorithm

There are a number of other algorithm animation systems that are discussed in the literature but are not discussed above, such as the Algorithm Explorer (McWhirter 1996), DRUIDS (Whale 1996), DynaLab (Birch et al., 1995) and others. There are several Web sites devoted to serving as clearing houses to other algorithm animation implementations. Peter Brummund maintains a Internet site with links to 43 sites that employ some form of algorithm animation (at http://www.cs.hope.edu/~alganim/ccaa/ccaa.html). Another list maintained by Hausner is at http://www.cs.princeton.edu/~ah/alganim, and yet another is at http://www.cs.duke.edu/~jeffe/compgeom/demos.html.

When chronologically viewed, successor systems made enhancements and improvements to earlier systems in several noteworthy areas:

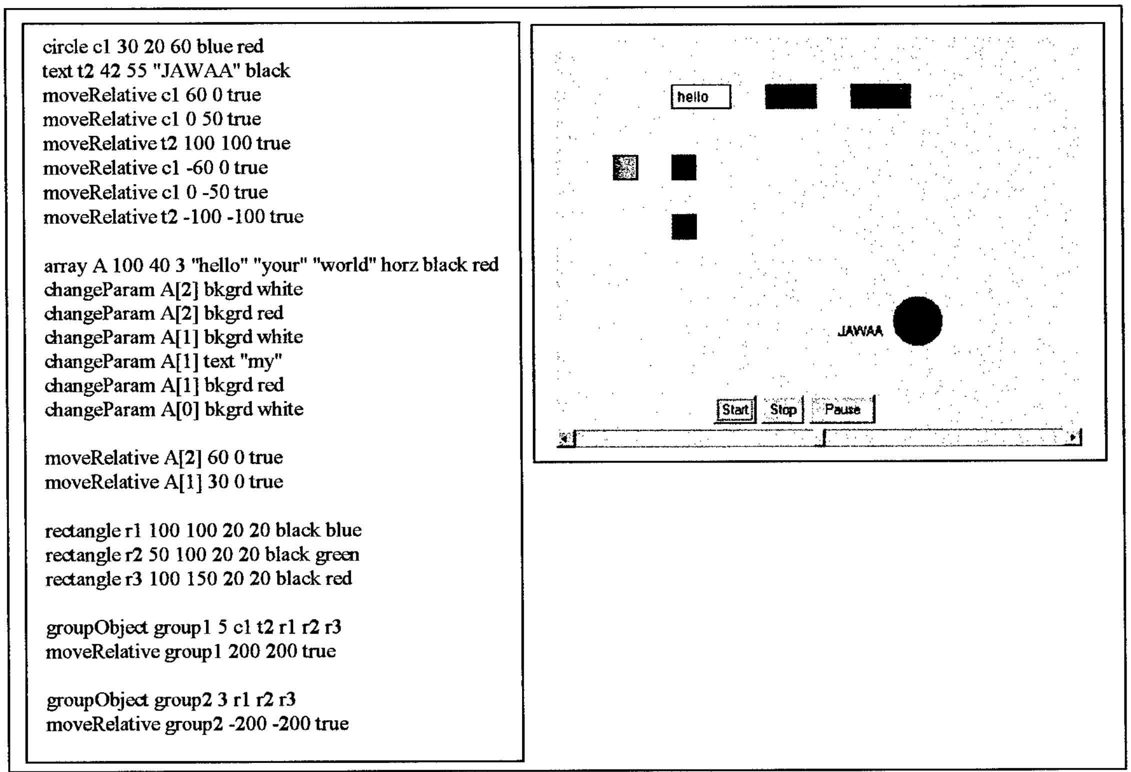
- Code independence - Creating a set of language-independent commands to control the animation, permitting instructors to create animations from a wider variety of source programs.
- Internet delivery - Allowing wider dissemination and platform independence.

- Presentation techniques (such as use of color, sound, and smooth animation) - Improving the aesthetics of the animations.
- Integration of textual explanations - Providing terse but useful descriptions of the events being depicted.
- Multiple windows - Allowing multiple views of one algorithm, or comparative views of similar algorithms.

Generally, the systems reported in the literature can be categorized into three types, listed below.

- ♦ Language Supersets (Those that provide code libraries that enhance current languages). For example, Algorithm Explorer uses specific data structure declarations to drive compiler-recognized animation sequences. These libraries modify language compilers and extend their capabilities to support animation. They generally recognize particular data structures and invoke procedures to graphically illustrate them as the program executes. These are the easiest for students to work with, in theory, because the student does not need to worry about the mechanics of illustrating the animation—the compiler automatically generates the visual components. The disadvantages are that the systems can only recognize and illustrate a fairly restrictive set of data structures, provide fairly little supporting media, are limited in the manner in which the animations can be presented, and are available for specific platforms and languages. The systems reported above that fall into this general categorization include Jeliot, Algorithm Explorer, C Code Animator, Druids, and DynaLab.
- ♦ Interesting Event Systems (Those that work with event scripts). Tango, for example, works with graphic commands generated as output from existing programs or authored

from a word processor to create and manipulate animation objects. These systems provide greater control over the design and presentation of the animation and do not restrict the animator to any specific language. The biggest disadvantage is the learning curve imposed on the animation designer to learn the manipulation commands and ensure the 'playback' environment is present for the student to view the resulting animations. The systems reported in the previous section that make use of the interesting event/scripting paradigm are JCAT, XTango, JAWAA, and Lambada. For example, the diagram in Figure 2.15 is created from the script depicted next to it:



```
circle c1 30 20 60 blue red
text t2 42 55 "JAWAA" black
moveRelative c1 60 0 true
moveRelative c1 0 50 true
moveRelative t2 100 100 true
moveRelative c1 -60 0 true
moveRelative c1 0 -50 true
moveRelative t2 -100 -100 true

array A 100 40 3 "hello" "your" "world" horz black red
changeParam A[2] bkgrd white
changeParam A[2] bkgrd red
changeParam A[1] bkgrd white
changeParam A[1] text "my"
changeParam A[1] bkgrd red
changeParam A[0] bkgrd white

moveRelative A[2] 60 0 true
moveRelative A[1] 30 0 true

rectangle r1 100 100 20 20 black blue
rectangle r2 50 100 20 20 black green
rectangle r3 100 150 20 20 black red

groupObject group1 5 c1 t2 r1 r2 r3
moveRelative group1 200 200 true

groupObject group2 3 r1 r2 r3
moveRelative group2 -200 -200 true
```

Figure 2.15. An Example of an Interesting Event Script and the Animation it Creates

- Dedicated Systems (Those animations created without provisions for instructors to author new sequences). For example, AlgoNet, AACE, and XSortLab are packaged learning environments/applications that cannot be extended by end-users. They require extensive coding to implement animation sequences, but allow much richer and interactive tutorials. Other systems include SWAN, DRUIDS, and GAIGS. These provide the greatest control of media presentation to the learner and are relatively easy for the student to use, but these also require a tremendous investment by the author/animator in time and effort.

The animation systems that have enjoyed the widest use and longest 'shelf life' have been the ones that are most flexible and accommodate the widest variety of animations. Tango and its clones and successors seem to be the most popular animation environments currently available. Built on the 'interesting event' paradigm, the Tango graphic command set allows a wide variety of animation styles. Scripts can be generated by inserting output statements into a fully functioning program, or can be produced from a word processor. This powerful arrangement facilitates a number of possible animation styles, and has even been used by students where they assume the role of the instructor and create an animation of their own to depict their visualization of various algorithms (Stasko, 1997).

Most of the empirical research so far has been conducted using the interesting event systems, which seem to provide greater user control over creation and modification of animation scripts, as will be seen in the following section.

2.2.5. EMPIRICAL STUDIES OF ALGORITHM ANIMATION EFFECTIVENESS

Guided mainly by intuition, most of the systems discussed in the previous section were developed in the belief that algorithm animations would serve as successful tools for students to learn about algorithms (Pierson & Roger, 1998; Sangwan & Korsh, 1998; Shaffer et al., 1996; Whale, 1996; Stasko & Patterson, 1992; Stasko, 1997). This belief has a strong intuitive basis. Students have always had a relatively difficult time understanding abstract mathematical notions, especially when these included dynamics of how algorithms manipulate data, so concretizing these notions graphically and animating these to illustrate the dynamic behavior ought to improve learning. So while it is certainly possible to learn about an algorithm without using animation, it *seems* almost obvious that a student could learn faster and more thoroughly with one.

Subjective studies are almost unanimous in their praise for animated algorithms. Students, exposed to animations, have reported that they felt the animations assisted them in understanding the algorithm (Stasko et al., 1993; Naps, 1990; Naps & Bressler, 1998; Gloor, 1992; Stasko, 1997; Gurka & Citrin, 1996). Instructors echo similar positive comments and many embrace their use as an integral part of their teaching method (Naps & Bressler, 1998; Gurka & Citrin, 1996; Stasko, 1997; Brown, 1988c; Eck, 1998). Unfortunately, the elation has been dampened for students and instructors when formal empirical research about the benefits of animation in computer science and elsewhere began to show disappointing results (Reiber et al., 1990; Badre et al., 1991; Palmiter & Elkerton, 1991; Stasko et al., 1993; Byrne et al., 1996).

Badre et al. (1991) conducted a faculty survey to identify algorithm teaching practices and conducted an observational study of two groups of students working with a shell sort algorithm. One group saw the animation while the other did not. They found that student performance in the animation condition varied with individual background and that good students did well regardless of whether they used the animation or not.

Lawrence (1993) conducted a series of experiments to evaluate student preferences and performance using various animation presentation techniques. Groups viewed animations with slightly varied components, and both subjective and empirical results were gathered. Some of the subjective statements indicated, for example, user preferences for vertical over horizontal bars, for bars over dots, for solid images rather than hollow ones, and so on. She conducted several experiments comparing specific animation conditions, finding that groups viewing animations with color cues performed *worse* than groups viewing monochrome animations, and that groups viewing animations that included ‘conceptual labels’ (a terse one line explanation of what the animation was doing) performed significantly better than groups viewing animations without such explanatory statements. Finally, she reported a statistically significant benefit to students who were allowed to interact with the animation system by entering their own data sets as input to algorithms over the group that passively watched already prepared animations. The results supported her hypotheses for the positive effects of conceptual labeling and for student interaction, but found the negative effects of using color in animation somewhat surprising, leading her to recommend sparse and judicious use of color to highlight well-labeled events and to avoid arbitrary coloring schemes. Her results are insightful and provide guidelines for algorithm designers to consider, but do not compare the effectiveness of learning algorithms by animation compared to other traditional teaching methods.

Another study was conducted by Stasko, Badre and Lewis (1993) which used an interactive animation to teach a priority queue algorithm to computer science graduate students under text-only and text-and-animation conditions. The study hypothesized that animation would aid procedural understanding but found that the animation group did not perform any better than the control group on questions testing procedural knowledge. They found the effects of the animation was not as strong as expected (“non-significant trend favoring the animation group”). They attribute the lack of performance of the animation group to a property of most visualizations, which is that an animation typically represents an expert’s understanding of the algorithm, not a novice’s. They state,

> “For a student to benefit from the animation, the student must understand both the mapping [from the algorithm to the graphics] and the underlying algorithm on which it is

> based...Students just learning about an algorithm do not have a foundation of understanding upon which to construct the visualization mapping." (Stasko et al., 1993, p65)

Lawrence, Badre & Stasko (1994) conducted a detailed 2 X 2 study of students learning about an algorithm under lecture-only, lecture-and-laboratory, lecture-and-animation, and lecture-animation-laboratory conditions. In the laboratory condition, students either participated passively by watching, or actively by specifying different inputs to the algorithm. They found no significant difference between lecture-only and lecture-and-animation conditions, but they did find a significant ($p<.05$) effect in the active lab conditions on tests measuring conceptual knowledge.

Byrne, Catrambone & Stasko (1996) also found limited learning effects in undergraduates using interactive animation. Their study examined the effects of animation and making predictions. In learning new algorithms, some students viewed animations and some were prompted to make predictions about an algorithm's operation on novel data sets. One experiment used novice students studying the relatively simple depth-first search algorithm, and the other used expert students with a more challenging binomial heap algorithm. In both cases they used a 2 X 2 design with animation/no-animation and prediction/no-prediction conditions. They found that the ability to make, test and receive feedback on predictions contributed to learning. But animations did not seem to confer any significant advantage over paper-and-pencil predictions. Thus, their conclusion is that animations, as traditionally conceived, may have less effect on learning than generally assumed. They conjecture that benefits from animations may accrue only if the learner has the right amount of prior knowledge, not so much that the information provided by the animation is redundant, but not so little that the information is difficult to understand and integrate.

Hundhausen (1996) provides an excellent summary of the results of 29 empirical studies pertaining to the comprehension efficacy of algorithm animations and more generally, software visualization. Of the 29 experiments, only the three reported by Lawrence (1993) generated statistically significant results, and these dealt with design issues rather than learning effectiveness over traditional teaching methods. He observes, succinctly summarizing the paradox that our research will address:

> "The results have been mixed, with a majority of the studies failing to obtain the statistically significant result for which their authors had hoped. In stark contrast, questionnaire and survey data have painted an overwhelmingly positive picture of software visualization effectiveness; most people who use software visualizations seem to think that it helps them." (Hundhausen, 1996, p22)

This intuitive belief in the benefit(s) of visualization is not borne out by empirical studies. Therefore, our research has two thrusts: (1) to design better algorithm visualizations and (2) to empirically and statistically demonstrate their effectiveness.

2.3. THE PROBLEM: MOVING FROM ANIMATION TO VISUALIZATION

Though research on designing and deploying algorithm animation spans over 15 years, including dozens of experiments conducted to test whether animations lead to improved understanding, all one can say about this accumulated evidence is that the results, at best, are mixed. On one hand, subjective comments by students and instructors applaud the use of animation as a teaching tool, citing it as enjoyable, entertaining, interesting, and even revealing. On the other hand, the empirical evidence has been terribly disappointing in confirming the effectiveness of algorithm animation as a teaching tool. The general conclusion that emerges is that algorithm animation, as traditionally constructed and used, may not be as effective for learning as had been expected. The mixed, disappointing results can lead one to abandon the premise that animations are powerful vehicles for effectively conveying the dynamic operations of algorithms on data structures. The following quote succinctly expresses the frustration felt by researchers working in this area:

> "Unfortunately, the viability of algorithm animations as instructional aids remains rooted in intuition. No substantive empirical evidence has ever been presented to support these claims." (Stasko, Badre & Lewis, 1993, p61).

Furthermore, the vast majority of current algorithm animation systems, with a few exceptions, remain on the shelves as dust-gathering research prototypes, or made available over the web for the curious to experiment with. Even in those few cases where algorithm animations and animation building systems have been deployed in undergraduate classrooms, systematic studies of their effectiveness to test whether these do indeed lead to improved understanding of algorithms are yet to be conducted. There is also little research that can shed light on issues such as whether combining animations with other kinds of explanations can improve learning, and if so, what might the principles be for designing such explanations.

In the studies that failed to find significant benefits to using animations, the following explanations seem plausible:

- *There are no, or only limited, benefits using algorithm animation.*

  This is contrary to intuition and other positive results in non-algorithmic areas. Hence, if we accept the premise that animations are indeed useful, then we must rethink the process of algorithm animation design in order to harness its power to enhance learning. Perhaps the results of earlier work were unsatisfactory not because of a flaw with animation as a technique, but because of the approach used to convey the animations. As stated earlier, perhaps the flaw in previous algorithm animation systems was not in the *use* of animation, but in the *design* of the system employed to deliver the algorithm animation to the student.

- *There are benefits to using algorithm animation but the measurements used in the studies are not sensitive to them.*

  Many of the studies focused exclusively on student performance, but did not measure other factors that animations might affect, such as speed of learning, student motivation, student satisfaction, and long-term retention. Evidence of effectiveness in any of these measures could present a valid argument for acceptance of algorithm animation systems. However, our chief focus will be to measure student performance, and we will leave the other metrics for future research. The experiments in Chapter 4 concentrate on measuring student performance by comparing groups that interact with animated algorithms and groups that learn algorithms using traditional teaching methods.

- *Something in the design of the experiment or animation prevented participants from receiving the benefits.*

  It is quite possible that the experiments encountered factors, such as buggy software, awkward user interface, inadequate or faulty hardware, and insufficiently developed animations that led to less than satisfactory results and skewed the findings. For example, some experimenters mentioned not properly isolating the independent variables between the groups (Hundhausen, 1996), improper phrasing of questions (Byrne, Catrambone & Stasko, 1996; Crosby & Stelovsky, 1995), and having questions that did not match the learning material (Reed, 1985). One author stated his system was simply too unstable to contemplate formal testing (Whale, 1996). These problems shaped our efforts. However, one study mentioned a reason that falls within the scope of our research: Byrne et al., (1996) suggested the animations used might not have been 'the right ones,' that they may not have been properly designed. Since a 'bad' animation could negatively affect results, we dedicated a portion our research on studying factors and features that make a good animation. The experiments in Chapter 5 explore the effectiveness of a variety of features and design approaches that give insight to future researchers and developers concerning the design of better algorithm animations.

Before presenting the experiments, in the next chapter we will present and describe the framework we developed to address these issues and concerns.

3. TOWARDS A FRAMEWORK FOR EFFECTIVE VISUALIZATION (OR 'WHAT WAS MISSING BEFORE')

This research is based on the hypothesis that animations are indeed powerful vehicles for effectively conveying the dynamic behavior of algorithms, but that a rethinking of algorithm animation design is required in order to harness its power to enhance learning. Research has shown clearly that animation, by itself, is not enough. What is needed is a new framework that leads to effective visualization. The Oxford English Dictonary (Simpson & Weiner, 1989) defines visualization as "the power or process of forming a mental picture or vision of something not actually present to the sight." Where animation is concerned with moving graphic objects to create the illusion of motion, visualization involves use of animation and a host of other techniques to create a clear mental image in the mind of the observer. Visualization then, is a much richer and more involved process than merely watching an animation.

3.1. COMPONENTS OF THE FRAMEWORK

Our research explores the integration of previous work in algorithm animation systems (e.g., Kehoe & Stasko, 1996; Recker, et al., 1995) with recent developments in the cognitive and educational aspects of multimedia (e.g., Crosby & Stelovsky, 1995; Narayanan & Hegarty, 1998) to produce a novel approach to designing algorithm visualizations to improve student knowledge and comprehension. This approach is based on explicating learning objectives that drive a top-down design process which carefully divides abstract concepts into discrete chunks for learning.

Unlike previous work, our model takes a user-centered ("what do we *need* to show") view rather than a technology-centered ("what *can* we show") view, and employs hypermedia and multimodal presentation techniques to improve learning effectiveness. We call the resulting framework, containing text, diagrams, audio as well as animations, a *hypermedia visualization.* Under this new framework, an algorithm visualization is more than a mere animation. It describes a scaffolded, apprenticeship-oriented environment (Soloway et al, 1996; Guzdial & Kehoe, 1998) that elicits active student participation using a carefully orchestrated presentation of information in various media (such as animations, text, static diagrams, aural narratives, etc.) with appropriate temporal, spatial and hyperlink connections to semantically related components. The foundation of this new framework integrates ideas encountered in related work with original concepts developed in our research.

3.1.1. OBJECTIVE-BASED DESIGN

Our framework employs top down design techniques that are based on learning objectives. As noted by Stasko, Badre and Lewis (1993) and Reed (1985), learning objectives are critical to educational success. The objectives determine what the student needs to know and to what depth (Bloom, 1956), and ultimately drive the content of the views, the animations, the interactions, and the questions to ask to measure understanding. Hundhausen (1996) stated:

> "the lack of a well-defined task objective places learners in an unrealistic (and potentially disconcerting) situation in which they are to explore an animation without a clear idea of what they are supposed to get out of it. As learners engage in an aimless process of discovery learning, there is no guarantee that they will stumble upon the insights into algorithmic behavior that could help them on the upcoming test. Thus the lack of a concrete objective may serve to rob an animation learning session of its putative benefits." (Hundhausen, 1996, p18)

Using Bloom's taxonomy of learning (see Chapter 2.1.2), an example of application-level learning objectives for the insertion sort algorithm might include:

- Main Learning Objective
    - Demonstrate understanding of the insertion sort algorithm to include its purpose, key steps, data structure manipulations and terminating condition(s) as applied to various sorting problems
- Sub-objectives
    - Express the insertion sort algorithm in high level terms, such as pseudocode or english
    - Describe the overall strategy insertion sort uses to sort data
    - Explain the purpose of the outer loop of the insertion sort algorithm
    - Identify the variables that control the outer and inner loops of the insertion sort algorithm
    - Explain the purpose of the inner loop of the insertion sort algorithm
    - Describe how a single element is placed into proper sequence
    - Explain the conditions that result in insertion sort terminating successfully
    - Given any set of input, trace the execution of the steps the insertion sort algorithm would follow

Objectives stated in this way give direction to the learner *and* the designer. For the designer, they are the foundation of the various visualization development tasks or steps (Hundhausen, 1996), including identifying the interesting events that the animation should illustrate (Brown, 1988a; Stasko, 1990) and providing an outline of the supporting materials and techniques (Lawrence, 1993; Pane, et al., 1996) that should be used. For the student, clearly stated objectives provide focus, purpose and direction to learning.

3.1.2. MULTIMODAL PRESENTATIONS

A unique aspect of our framework is that the focus is shifted—it is not on the animation itself, but on providing relevant and sufficient information in *appropriate* media (Rappin et al, 1997) to support achieving the learning objectives. Other research discussed the importance of collocating graphics with textual descriptions (Stasko, Badre & Lewis, 1993; Mayer, 1989; Lawrence, 1993), and of appropriate use of audio cues and narratives (Brown & Hershberger, 1991; Mayer & Anderson, 1991; Bagui, 1998; Hundhausen 1996; Mayer & Moreno, 1998). Our framework involves *embedding* animations within a hypermedia visualization, along with textual descriptions, audio and diagrams.

An original feature is the introduction of different forms of text, such as:

- **static descriptions**- passages that are presented in panels that remain visible to the user and do not change, such as introductions, definitions and pseudocode;
- **animated text**- passages in which lines are highlighted in synchronization with other screen objects;
- **contextual explanations**- text that is provided in response to specific conditions determined by the state of the system.

Another capability that is not unique to our framework but bears brief explanation is the use of multiple forms of audio to enhance the hypermedia experience, including:

- **sound cues**- audio 'bells' or 'buzzers' that signal selected conditions, such as the completion of a loop or invoking a swap operation;
- **redundant narrative**- an audio segment that mirrors text already visible to the user. Audio used in this way employs the dual-coding theory (Paivio, 1986) and is thought by some to help reduce cognitive load. It also allows the user to visually roam to view other parts of the screen since the ears are receiving the words in place of the eyes;

- **contextual cues**- audio segments that direct user attention to specific features or interactions that should be explored further. This provides a form of scaffolding by providing the user with prompts and hints that are not directly related to the content of the animation, but can significantly deepen the learning experience by highlighting features that could be very useful and educational to the student.

3.1.3. BRIDGING ANALOGIES

Another unique approach embraced by our framework is the use of real world analogies, cases and examples to introduce and illustrate key concepts. An analogy is a learning device in which knowledge about entities in a well-known domain (the source) is mapped into another lesser-known domain (the target). At the highest level, students are introduced to the algorithm's basic operations using animated real world analogies, and are provided with a bridge between the analogy and the abstract components of the algorithm as well as the concrete graphical representations used to depict the algorithm in later animations. This approach draws from the observations that students tend to employ analogies in describing how algorithms operate (Douglas, Hundhausen & McKeown, 1995; Stasko, 1997), and that analogies can serve to provide a form of scaffolding (Hmelo & Guzdial, 1996) for subsequent learning. Related research (Kolodner, 1993) suggests that using analogies, cases and examples to introduce and teach complex physical systems is crucial to building mental models because learners can draw from similar principles extracted from overlapping experiences. Analogy allows the application of preexisting conceptual structure to new problems and domains, and hence supports the rapid learning of new systems. Of all the learning processes, analogy is the only one that offers a mechanism for the acquisition of substantial knowledge structures in a brief span of learning (Gentner, 1989; Brophy & Schwartz, 1998). In our framework, the conceptual gaps between the analogy, the abstract components of the algorithm and the concrete graphical representations used in the animation are bridged using explanatory text to help build the necessary referential connections (Mayer & Sims, 1994; Narayanan & Hegarty, 1998) for better comprehension. These analogies set the stage for subsequent detailed learning, and may improve long term retention (Gentner, 1989).

We have identified three aspects of real world examples that characterize the analogies we used to illustrate or introduce algorithms at the conceptual level:

- **Interactivity**- This refers to the level of activity required by the user to receive the analogy such as passively-viewed static text and diagrams, a paced animation (where the user controls the playback tempo of an animation sequence), a simulation (where the user inputs values into a simulation program that processes and reports results), or a game (where the user competitively interacts with the program). We implemented examples that involved each of these various levels of user interactivity. At the simplest level, the bubble sort analogy simply provided a real world example of bubbles that rise in a flask of water. The simplest analogy provides no more than an example for the user to associate the name of the algorithm to its most basic behavior. Another level allowed the user to view an animated, user-paced example. The most complex were the analogies that were interactive, almost approaching a simulation for the user to explore. For example, the Merge example showed the student, using two stacks of cards, how the merge operation worked, then allowed the student to try merging the cards in a race against time. In the Shortest Path analogy, students were shown how the algorithm worked by testing various paths until the lowest fare was determined, then randomly changed the fares and let the student 'try his hand' against the analogy by clicking on cities that would be on the shortest path.

- **Fidelity**- This refers to the similarity of the mapping of attributes and relations between the source and target domains. A good analogy should closely parallel the central properties of the algorithm. If the analogy is too 'different', then it stands a very good chance of confusing the student, or imposes an excessive cognitive load in trying to bridge the components of the analogy with the algorithm, defeating the purpose of using an analogy. Gentner (1989) identifies several levels of similarity: literal similarity (where the attribute and relational mappings are one-to-one), mere appearance (where the attributes match but the relations do not); abstraction (where attributes don't match but the relations do); and anomaly (where nothing maps). Using these terms, our work deals with analogies with literal similarity and mere appearance to the algorithms in our system.
- **Realism**- This refers to how common the chosen example might be. Analogies that bring to mind common, every-day activities or processes are more likely to lead to better comprehension and long-term retention. Ideally, the example should represent an event or situation that everybody has witnessed or experienced. The more a student can relate to the analogy, the richer the content the analogy brings to the learning process.

While we look optimistically to the use of analogies and examples to introduce and illustrate the algorithms to students, we recognize that their use can also present problems. Good analogies take effort and thought by the designer to create. Analogies take thought for the student to understand and mental effort to build the referential connections between domains. Analogies draw from human experience which is different for each individual, so what constitutes a good algorithm for some might be quite confusing to another. Furthermore, there will certainly be algorithms for which no commonly available example exists.

3.1.4. VIEWS AT MULTIPLE LEVELS

Our framework incorporates the ability to view algorithm behavior in various levels of detail. Studies indicate improved results when multiple views are employed in large problems (Muthukumarasamy & Stasko, 1995; Robertson, 1991), and scaffolding theorists support the idea of providing students with levels of detail (Guzdial & Kehoe, 1998). At the highest level of our framework, a 'Conceptual View' introduces the algorithm in general terms accompanied by a real world analogy. Next, a "Detailed View" focuses on and animates specific algorithmic operations in tandem with pseudocode highlighting and textual explanations. Finally, animations in a "Populated View" show the algorithm's aggregate performance and behavior on large data sets. This multi-level design approach addresses the debate by Byrne et al (1996) over whether animations are better suited for presenting the 'big picture' or for illustrating details. They stated: "There is the paradoxical problem that an animation that shows the big picture or emerging qualities might be appreciated only by those who already understand the algorithm at the mechanical level."

3.1.5. SEMANTIC CHUNKING

Most animations provide playback capabilities in the form of VCR-like controls, allowing the user controls such as Forward, Pause, and Cancel. A novel feature of our framework is what we call 'semantic chunking' which entails subdividing the animation into logical operations at different levels of detail, and allowing the user to control the granularity of the playback. The concept of chunking deals with the practical limits of a person's short term memory. Miller (1956) states that the average person recognizes about seven units or chunks of information at a time for retention in short term memory. The size and complexity of each chunk is small with novices, but grows with experience as smaller chunks are clustered into larger concepts (Vessey, 1985). Our framework presents algorithm animations in *discrete chunks* accompanied by explanations of the specific actions being accomplished. By chunking animations into meaningful "bite-sized" pieces and providing logical pauses between chunks, the student is able to better digest the abstract, dynamic information being presented. Allowing the student to adjust the size of chunking tailors the flow of information to meet individual needs. This is in stark contrast to most current algorithm animation systems which present the detailed dynamics as a one-shot, stand-alone show that is entertaining to watch but tends to obscure the very details a student needs to learn.

We determined that the following levels of chunking needed to be implemented for our prototype system:

- **Statement level**- pause the execution at each logical step of the algorithm.
- **Pass**- pause the execution upon completion of a logical unit of statements, such as a logic operation or a single pass through an iterative sequence.
- **Completion**- allow the execution to proceed to completion.

Most animation systems we reviewed implemented statement level pausing, and virtually all allowed the animation to proceed to completion. Some added a feature to alter the speed of the animation, similar to a fast-forward button on a VCR, but to our knowledge, no system explored chunking levels in the way proposed above. Our work proposes three levels, but future research may discover that more levels are needed to adequately 'decompose' the semantic structure of more complex algorithms.

3.1.6. PURPOSEFUL INTERACTION

Our framework emphasizes encouraging student participation by allowing *rich interactions* with the animations and using *probes* or questions that stimulate thinking and foster self-explanations. The central notion of constructivism is that understanding and learning are active, generative processes (Soloway et al, 1996) involving self-explanation (Chi, Bassok, Lewis, Reimann & Glaser, 1989), problem solving and reflective articulation (Narayanan et al., 1995). Experiments conducted by Lawrence et al. (1994) indicate a learning advantage for students who had active involvement in the creation of input values to algorithm animations. Other researchers espouse incorporating animations in an active learning environment rather than a passive viewing one, that allows students to interact with the data (Stasko et al, 1993; McWhirter, 1996; Pierson & Rodger, 1998; Sangwan & Korsh,1998), or go so far as to have students build their own animations in the role of an instructor (Stasko, 1997; Naps & Bressler, 1998; Dershem & Brummond, 1998; Ford, 1993). In contrast, research in different domains indicates that passively watching an animation does not facilitate learning (Hundhausen, 1996). Therefore, the visualization framework we propose incorporates a variety of features to stimulate purposeful interactions.

Our framework supports interaction at the data input level by prompting students to input data sets of their choosing in order to explore algorithm behavior more thoroughly. Additionally, our framework supports reflective thought and self-explanation through use of questions that are posed periodically to the student. The simplest form is called a "tickler", which is a question that pops up in random order but always in an appropriate context. Tickler questions are open-ended, focus student attention on specific issues, challenge their understanding and promote self-explanations to improve comprehension. Their answers are not entered into the computer nor is feedback provided. Below are several examples of tickler questions used in conjunction with a specific step of the SelectionSort algorithm visualization:

- On this pass, which element will move into the leftmost unsorted position?
- How many swaps will occur in this pass?
- How many passes will it take until the remaining values are in place?
- What will the inner loops beginning and ending values be?
- As the sort progresses, the boxes to the left side get grayed-out...Why?
- Could you write the pseudocode for this algorithm?

Another way to encourage self-reflection is to prompt for predictions about expected actions or outcomes. As with tickler questions, verifying correctness is left to the student and not checked by computer. Unlike ticklers, the computed result is presented to the user for comparison. An example of a predictive question for the SelectionSort algorithm would be to ask the student to predict how many comparisons (or swaps) the algorithm will perform for the entire execution of a selected data set, then present the actual tally upon completion.

The last type of question we employ are questions for which computer validation is performed and feedback provided. These questions can be true/false, multiple choice, and involve orderingWe also place multiple choice questions, requiring students to enter answers in order to proceed further, at "articulation points" between modules of the visualization. In this case, immediate feedback is provided by the system.

Posing questions will help combat a shortcoming reported in other multimedia systems called the 'TV syndrome' or the 'hands-on, mind-off' situation by making student interaction active and purposeful. These questions are tied to learning objectives set out by the designer. These questions and feedback help students self-diagnose their learning and allow reflection on what they know, what they don't, and where to find relevant information.

3.2. PROTOTYPE DEVELOPMENT

To test the theoretical framework for the hypermedia visualization system described in the previous sections, an algorithm visualization prototype needed to be constructed and empirically tested. The prototype system that implements the framework is called the Hypermedia Algorithm Visualization System (HalVis). HalVis is written using Asymmetrix Toolbook, a prototyping and hypermedia authoring environment, to test the components of the framework. As of this writing, it contains visualizations of four sorting algorithms (BubbleSort, SelectionSort, MergeSort and QuickSort), the Merge algorithm, and one graphing algorithm (Shortest Path). A number of empirical studies using these algorithm visualizations have been conducted. The sections below describe the architecture of the HalVis prototype in its current form. Subsequent chapters present results from the empirical studies conducted to validate and analyze the framework.

3.2.1. HALVIS ARCHITECTURE

HalVis has been constructed using five main modules, forming an architectural template for algorithm visualizations (Figure 3.1). The modules are the Conceptual View, the Detailed View, the Populated View, the Questions module and the Fundamentals module. Each algorithm is presented using the template described below.

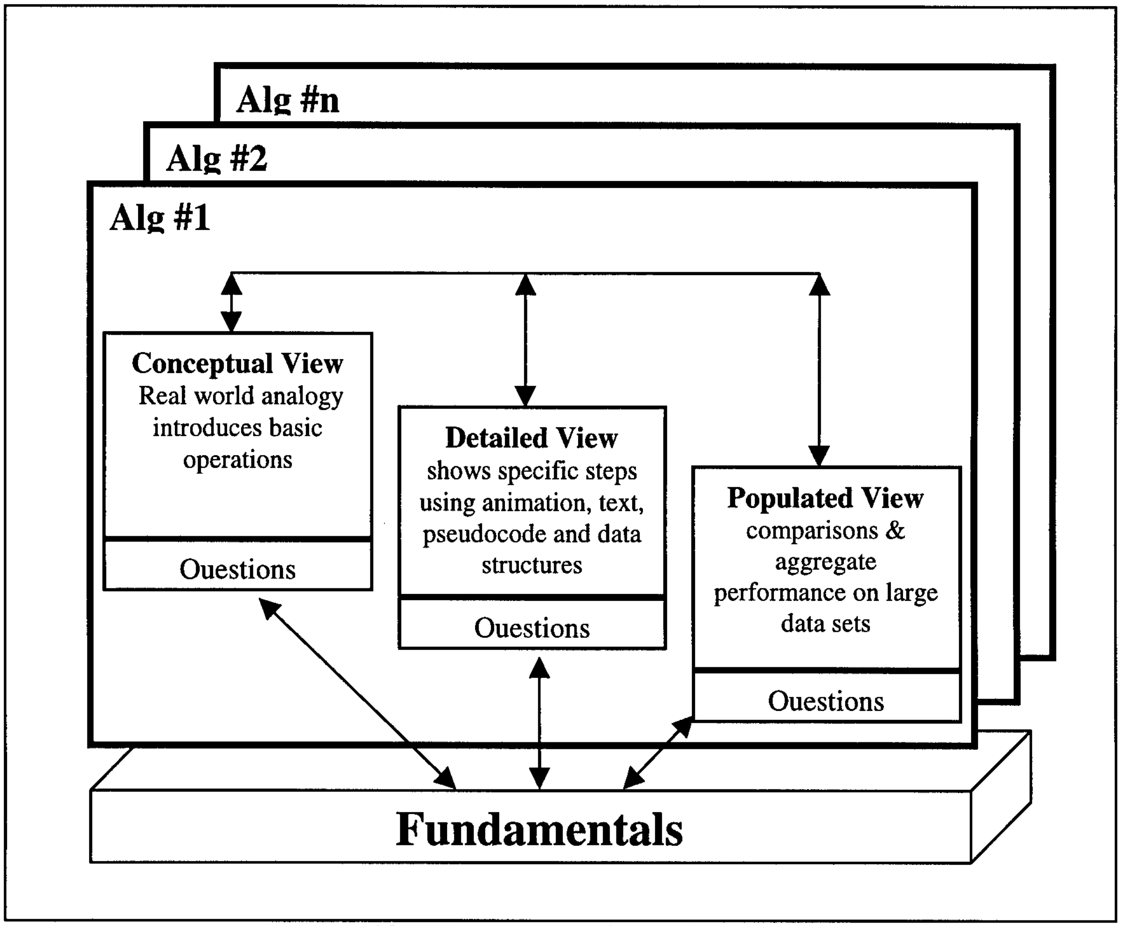

Alg #n
Alg #2
Alg #1
Conceptual View
Real world analogy introduces basic operations
Ouestions
Detailed View
shows specific steps using animation, text, pseudocode and data structures
Ouestions
Populated View
comparisons & aggregate performance on large data sets
Ouestions
Fundamentals

Figure 3.1. The HalVis Architecture

3.2.2. FUNDAMENTALS MODULE

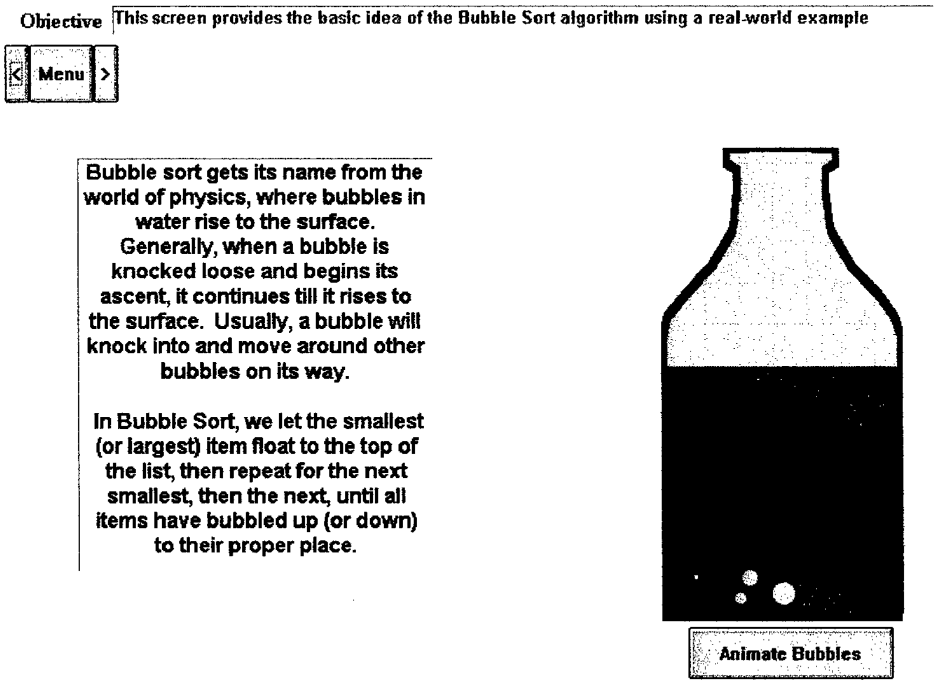
This module contains information about basic building blocks common to virtually all algorithms. Examples include Comparing & Swapping Data, Looping Operation, Recursion, and so on. Topics can be added to the Fundamentals module as needed to address student's prior knowledge or the prerequisites of a particular algorithm. Generally, this module is accessible only through hyperlinks from other modules, so that the basic information is presented *on demand* (in response to a learner request in the form of clicking on a hyperlink) and *in context* (of algorithm-specific information within which the hyperlink is embedded).

3.2.3. THE CONCEPTUAL VIEW

This module introduces a specific algorithm in very general terms using a real world analogy. We wanted examples that mapped closely to the algorithms being studied, and gave high priority to having the properties discussed in Section 3.1.3 (fidelity, interactivity and realism). In most cases, we created the analogies and examples based on personal experiences or analogies we had seen used effectively in other classes or textbooks. An example of each of the conceptual view screens and the associated analogy implemented in the prototype is listed in Table 3.1 and shown in Figures 3.2-3.14. For instance, BubbleSort (Figure 3.2) is introduced using a flask of water with bubbles that rise to the surface according to their size (Figure 3.3), starting with smallest bubbles and gradually working up to the largest ones. Figures 3.4 and 3.5 show screens from the conceptual view of the MergeSort algorithm, where cards are animated to illustrate dividing and merging to create a sorted sequence. Some analogies are more elaborate and interactive than others. For example, two of the analogies (Merge and Shortest Path) are presented as animated examples followed by a game/simulation that lets the student interact with the example while following the operations of the algorithm. This module uses animations, text and audio to provide the student with a "big picture" description, a visual example to aid long-term retention, and sufficient bridging information to proceed from the visual elements in the analogy to the data structures and algorithm operations in later modules.

Algorithm	Analogy employed	Interactivity	Fidelity	Realism
Bubble Sort	Bubbles in a flask that rise to the surface	• None	Mere appearance	High
Selection Sort	A line of people of various height that need to be placed in order. The tallest person of each pass would raise his hand	• Paced animation	Literal similarity	High
MergeSort	A deck of cards	• Paced animation	Literal similarity	Medium
Merge	A deck of cards	• Timed simulation	Literal similarity	High
QuickSort	A line of people of various height that need to be placed in order. Groups split around a 'pivot' person	• Paced animation	Literal similarity	Medium
Shortest Path	Airfares between various connecting cities and a common destination	• Paced animation • Randomized simulation	Literal similarity	High

Table 3.1. List of Analogies Used in HalVis



Objective This screen provides the basic idea of the Bubble Sort algorithm using a real-world example

Bubble sort gets its name from the world of physics, where bubbles in water rise to the surface. Generally, when a bubble is knocked loose and begins its ascent, it continues till it rises to the surface. Usually, a bubble will knock into and move around other bubbles on its way.

In Bubble Sort, we let the smallest (or largest) item float to the top of the list, then repeat for the next smallest, then the next, until all items have bubbled up (or down) to their proper place.

Figure 3.2. The Conceptual View Screen of the Bubble Sort Algorithm

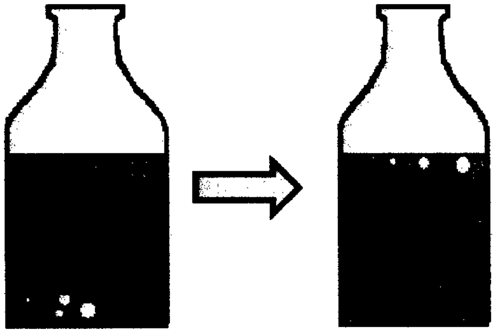
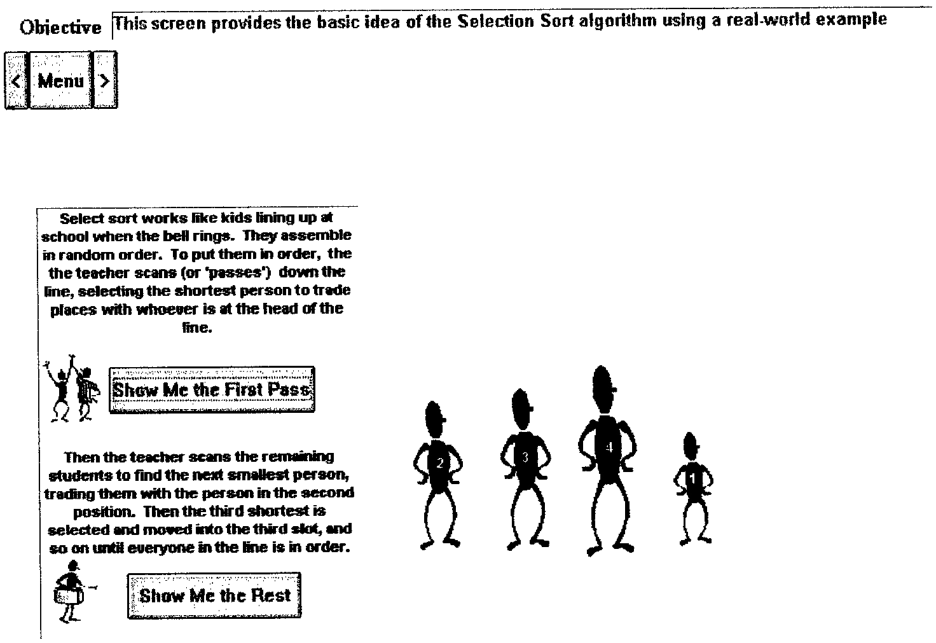


Figure 3.3. The Bubble Sort Animated Analogy



Objective This screen provides the basic idea of the Selection Sort algorithm using a real-world example

Select sort works like kids lining up at school when the bell rings. They assemble in random order. To put them in order, the the teacher scans (or 'passes') down the line, selecting the shortest person to trade places with whoever is at the head of the line.

Then the teacher scans the remaining students to find the next smallest person, trading them with the person in the second position. Then the third shortest is selected and moved into the third slot, and so on until everyone in the line is in order.

Figure 3.4. The Conceptual View Screen of the SelectionSort Algorithm

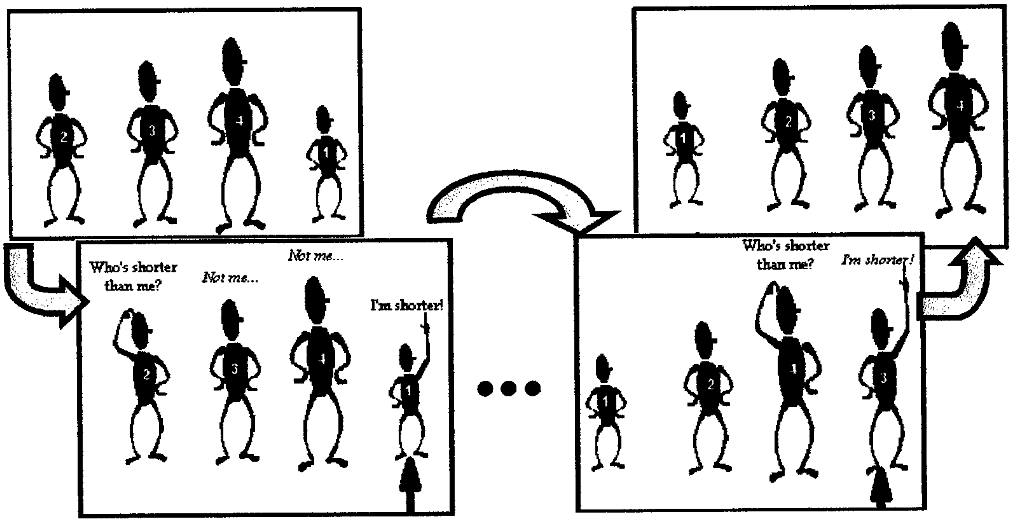
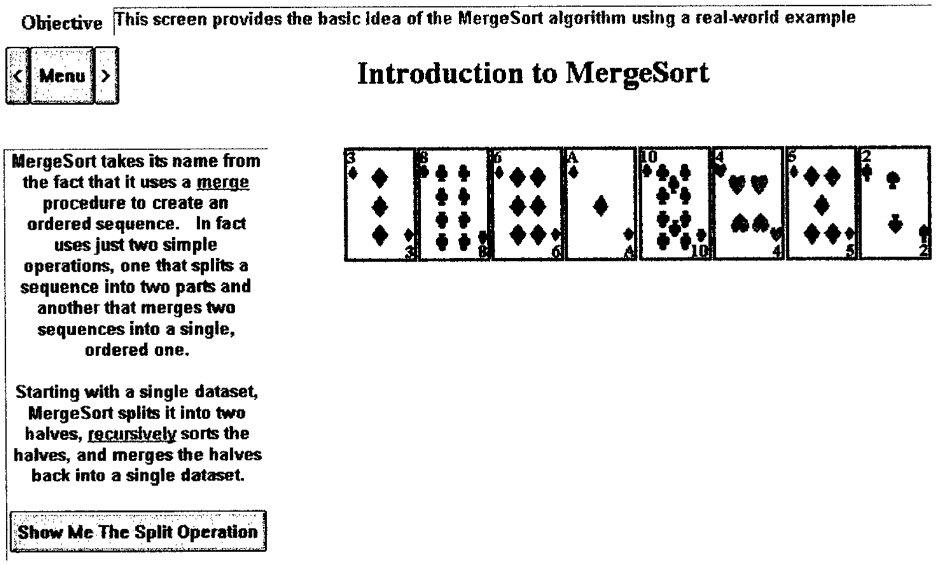


Figure 3.5. The SelectionSort Algorithm Animated Analogy



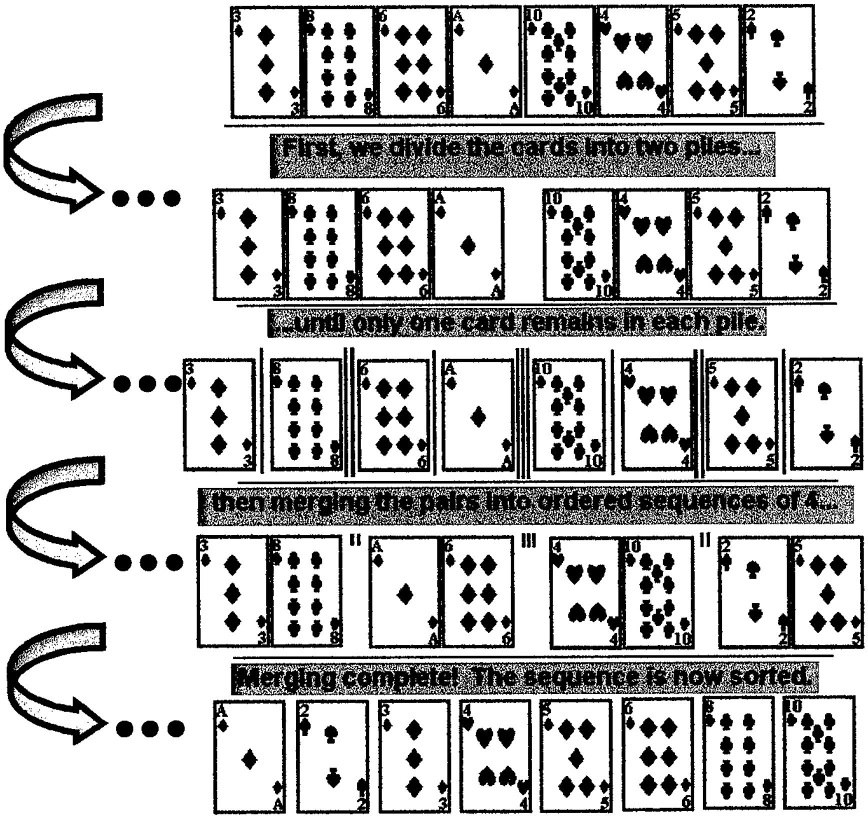
Objective This screen provides the basic idea of the MergeSort algorithm using a real-world example

Introduction to MergeSort

MergeSort takes its name from the fact that it uses a merge procedure to create an ordered sequence. In fact uses just two simple operations, one that splits a sequence into two parts and another that merges two sequences into a single, ordered one.

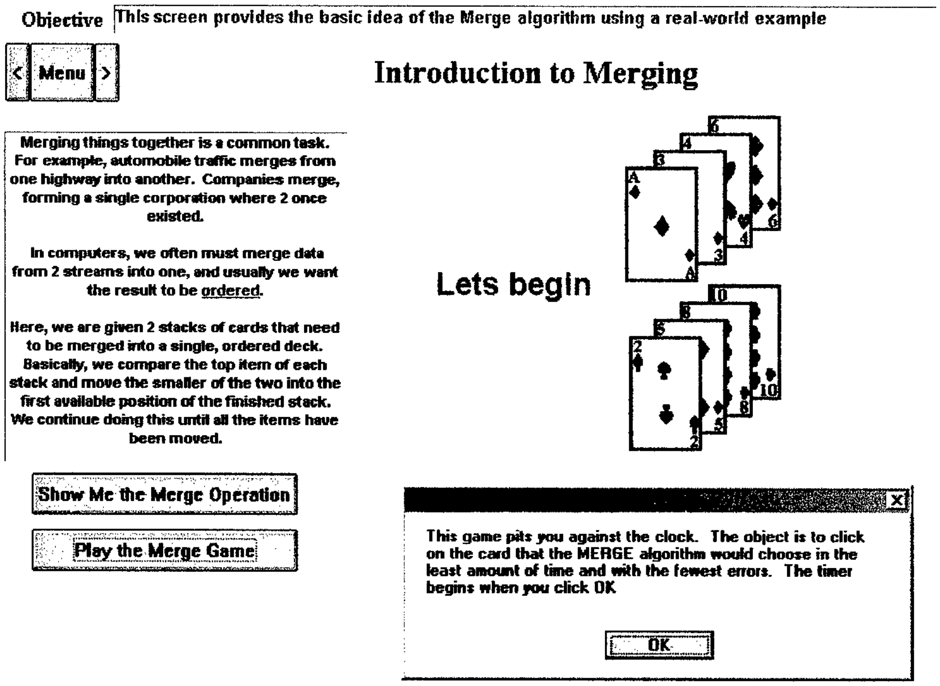
Starting with a single dataset, MergeSort splits it into two halves, recursively sorts the halves, and merges the halves back into a single dataset.

Figure 3.6. The Conceptual View Screen of the MergeSort Algorithm



First, we divide the cards into two piles...
...until only one card remains in each pile.
then merging the pairs into ordered sequences of 4...
Merging complete! The sequence is now sorted.

Figure 3.7. The MergeSort Algorithm Animated Analogy



Objective This screen provides the basic idea of the Merge algorithm using a real-world example

Introduction to Merging

Merging things together is a common task. For example, automobile traffic merges from one highway into another. Companies merge, forming a single corporation where 2 once existed.

In computers, we often must merge data from 2 streams into one, and usually we want the result to be ordered.

Here, we are given 2 stacks of cards that need to be merged into a single, ordered deck. Basically, we compare the top item of each stack and move the smaller of the two into the first available position of the finished stack. We continue doing this until all the items have been moved.

Lets begin

This game pits you against the clock. The object is to click on the card that the MERGE algorithm would choose in the least amount of time and with the fewest errors. The timer begins when you click OK

Figure 3.8. The Conceptual View Screen of the Merge Algorithm

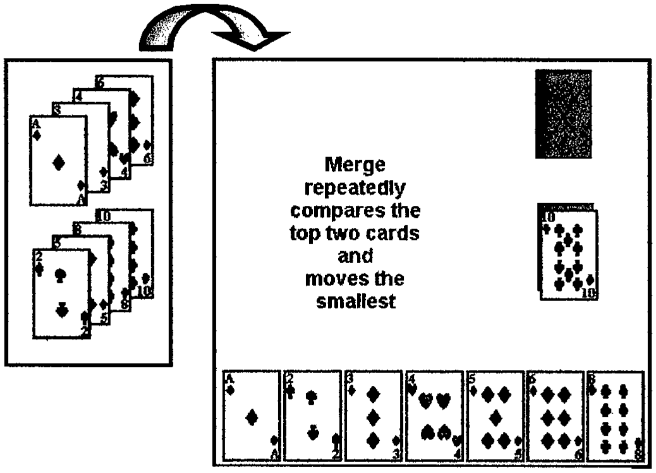
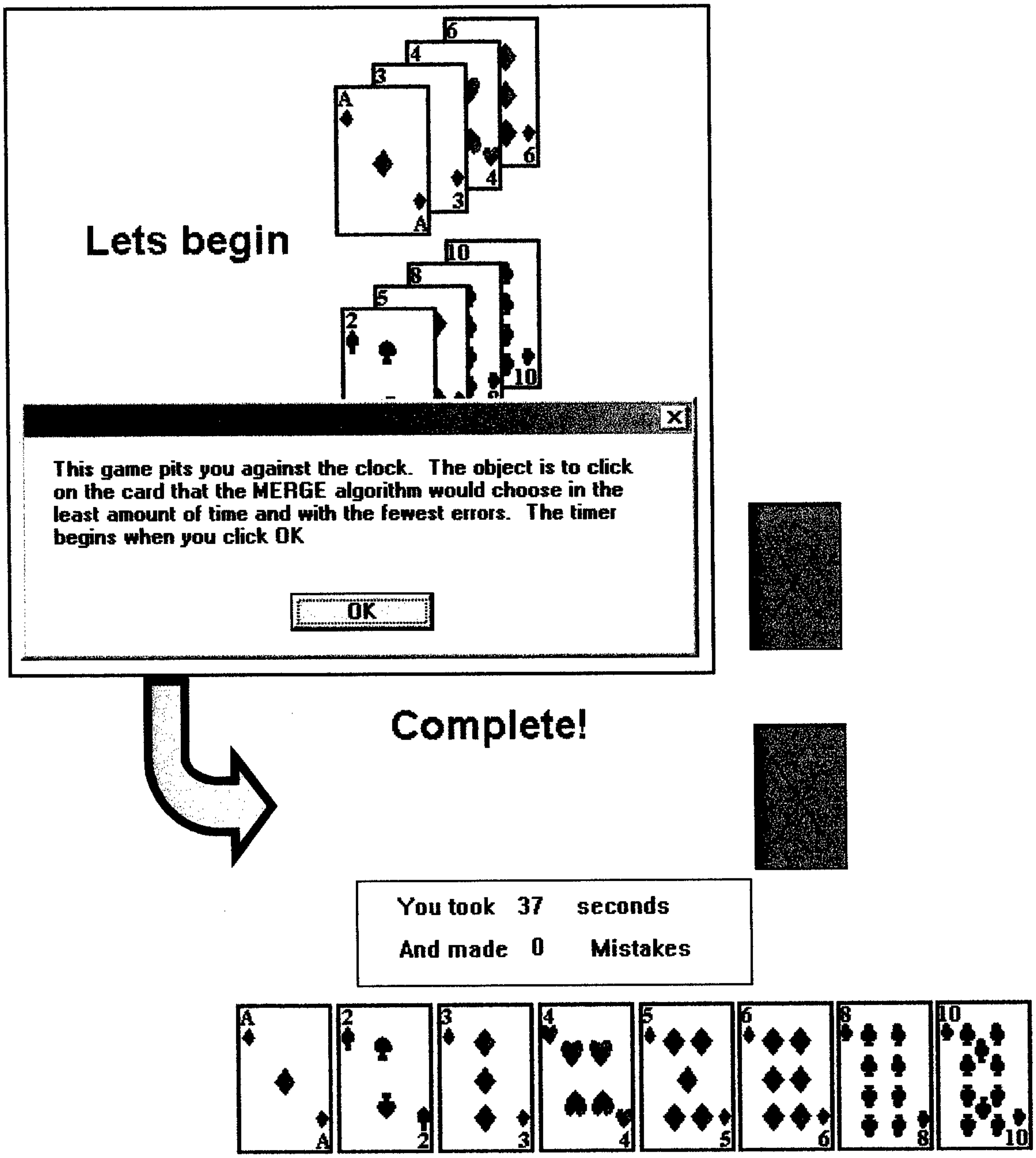


Figure 3.9. The Merge Algorithm Animated Analogy

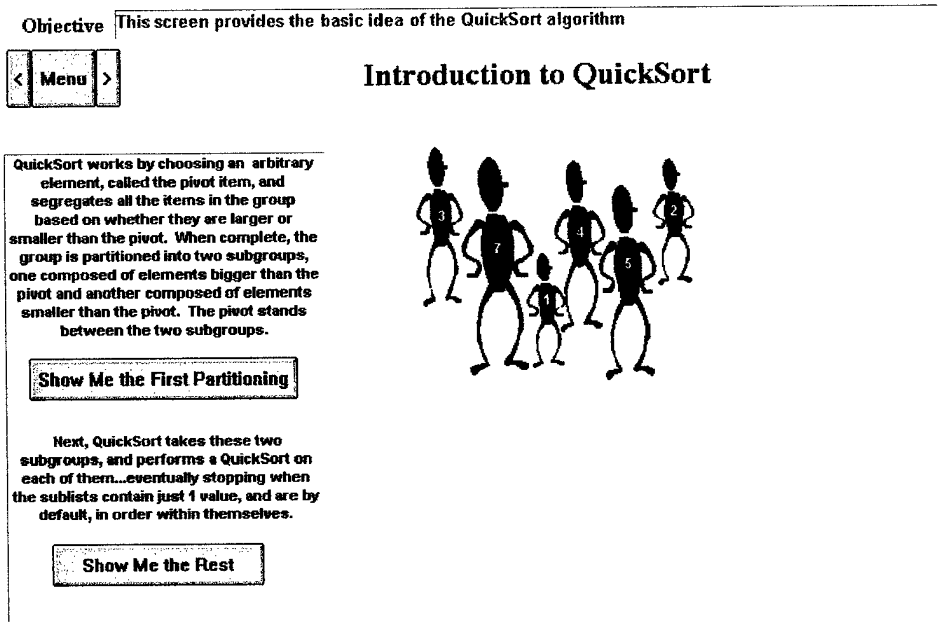


This game pits you against the clock. The object is to click on the card that the MERGE algorithm would choose in the least amount of time and with the fewest errors. The timer begins when you click OK

You took 37 seconds

And made 0 Mistakes

Figure 3.10. The Merge Algorithm Interactive Simulation



Objective This screen provides the basic idea of the QuickSort algorithm

Introduction to QuickSort

QuickSort works by choosing an arbitrary element, called the pivot item, and segregates all the items in the group based on whether they are larger or smaller than the pivot. When complete, the group is partitioned into two subgroups, one composed of elements bigger than the pivot and another composed of elements smaller than the pivot. The pivot stands between the two subgroups.

Next, QuickSort takes these two subgroups, and performs a QuickSort on each of them...eventually stopping when the sublists contain just 1 value, and are by default, in order within themselves.

Figure 3.11. The Conceptual View Screen of the QuickSort Algorithm

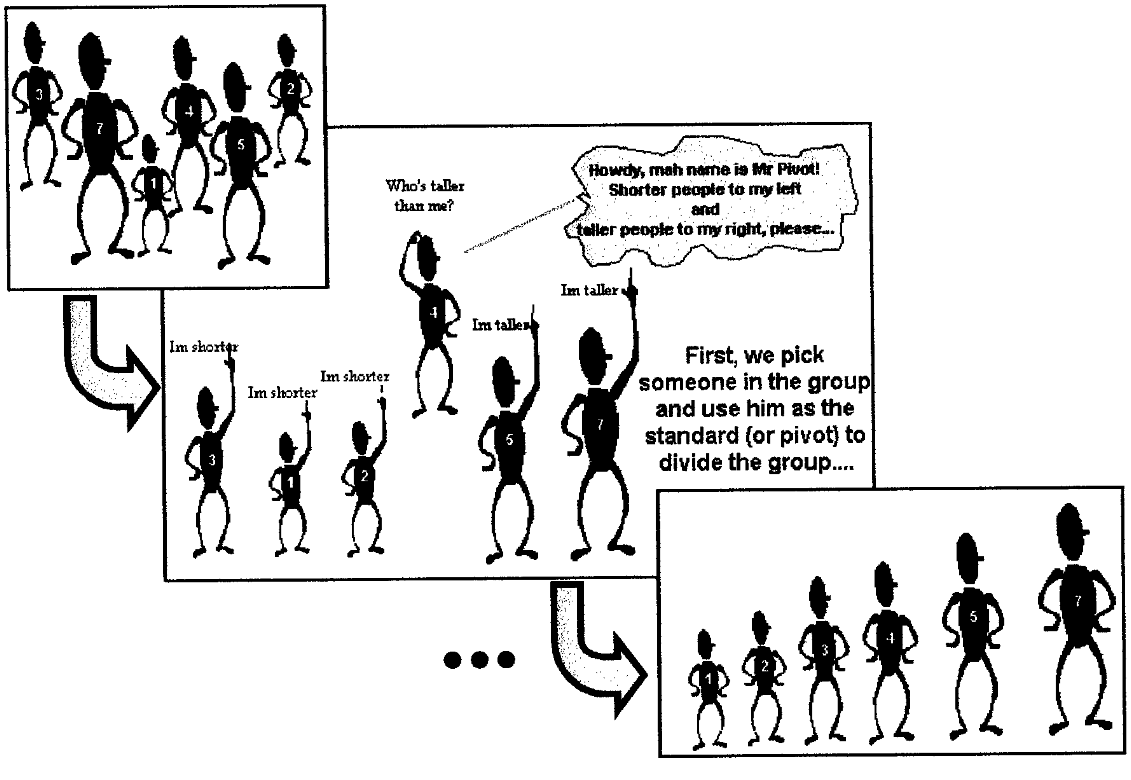
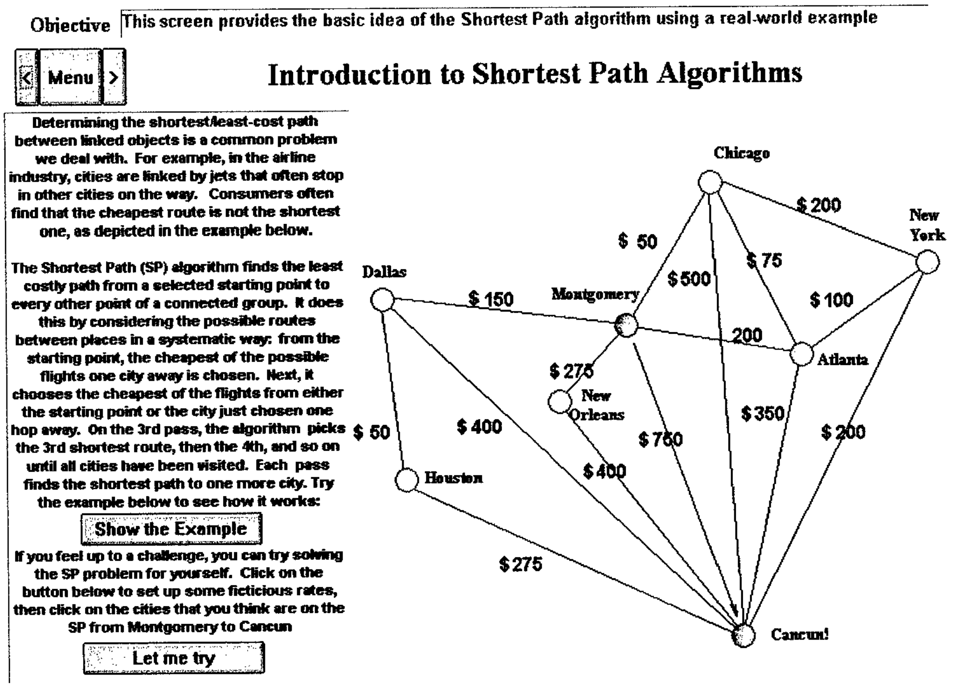

Who's taller than me?
Howdy, mah name is Mr Pivot! Shorter people to my left and taller people to my right, please...
Im taller
Im taller
Im shorter
Im shorter
Im shorter
First, we pick someone in the group and use him as the standard (or pivot) to divide the group....

Figure 3.12. The QuickSort Animated Analogy



Objective This screen provides the basic idea of the Shortest Path algorithm using a real-world example

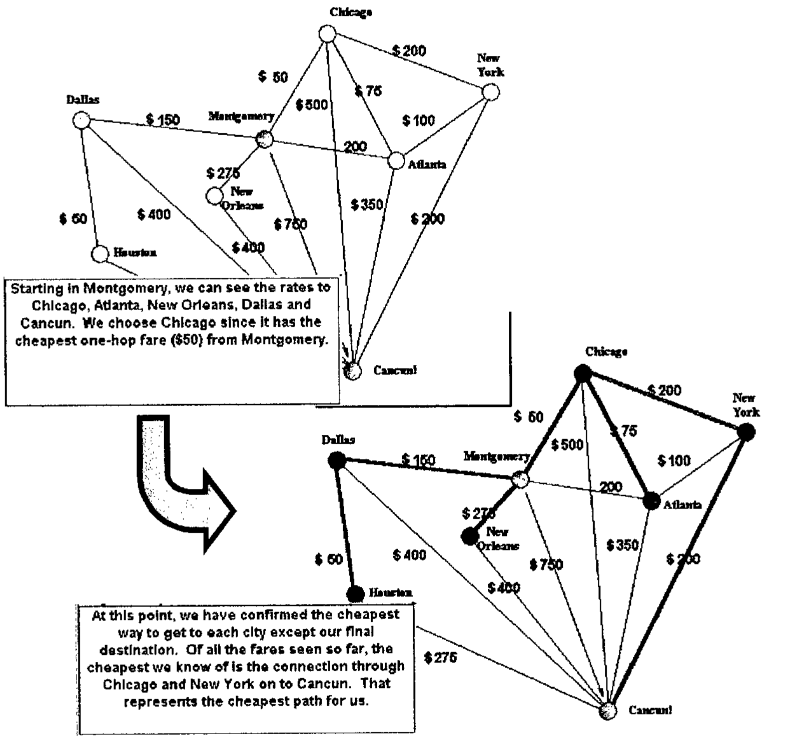
Introduction to Shortest Path Algorithms

Determining the shortest/least-cost path between linked objects is a common problem we deal with. For example, in the airline industry, cities are linked by jets that often stop in other cities on the way. Consumers often find that the cheapest route is not the shortest one, as depicted in the example below.

The Shortest Path (SP) algorithm finds the least costly path from a selected starting point to every other point of a connected group. It does this by considering the possible routes between places in a systematic way: from the starting point, the cheapest of the possible flights one city away is chosen. Next, it chooses the cheapest of the flights from either the starting point or the city just chosen one hop away. On the 3rd pass, the algorithm picks the 3rd shortest route, then the 4th, and so on until all cities have been visited. Each pass finds the shortest path to one more city. Try the example below to see how it works:

If you feel up to a challenge, you can try solving the SP problem for yourself. Click on the button below to set up some ficticious rates, then click on the cities that you think are on the SP from Montgomery to Cancun

Figure 3.13. The Conceptual View of the Shortest Path Algorithm



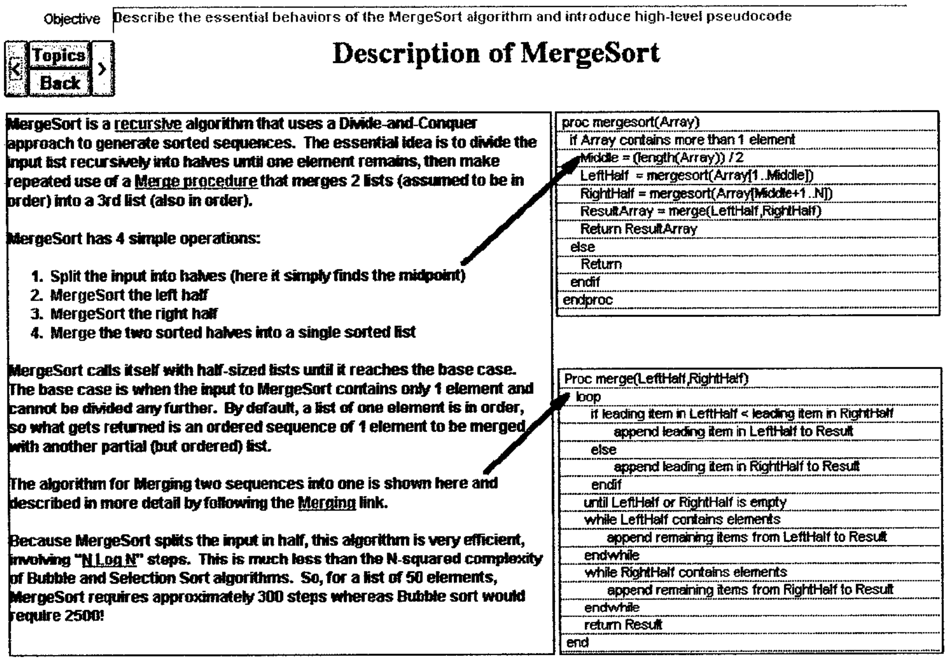
Starting in Montgomery, we can see the rates to Chicago, Atlanta, New Orleans, Dallas and Cancun. We choose Chicago since it has the cheapest one-hop fare ($50) from Montgomery.

At this point, we have confirmed the cheapest way to get to each city except our final destination. Of all the fares seen so far, the cheapest we know of is the connection through Chicago and New York on to Cancun. That represents the cheapest path for us.

Figure 3.14. The Shortest Path Algorithm Animated Analogy/Simulation

3.2.4. THE DETAILED VIEW

This module describes the algorithm at a very detailed level using two presentations. One consists of a detailed textual description of the algorithm alongside a pseudocode representation of it (Figure 3.15). Embedded in the text are hyperlinks to related information in the Fundamentals module. The second presentation (Figure 3.16) contains four windows that depict various aspects of the algorithm's behavior. The Execution Animation window shows how steps of the algorithm modify data structures using smooth animation. The animation is chunked at multiple levels of granularity corresponding to semantically meaningful units of the algorithm's behavior, with the level of chunking selectable by the learner. At the lowest level, the animation displays the execution of an individual statement, pausing for the learner's signal to proceed. The next level corresponds to a logical operation, like completion of a single pass in a loop. At the highest level, the animation proceeds to completion without pausing. The Execution Status Message window provides comments and textual feedback to the student about key events and actions during execution. This is also available as an audio commentary. The Pseudocode window shows the steps involved in the algorithm, which are highlighted synchronously with the animation. Finally, the Execution Variables window displays a scoreboard-like panorama of the variables involved in the algorithm and their changing values. Before launching the animation, students can change the data input to the algorithm as well as the speed and granularity of animation and feedback using a control panel (Figure 3.17). Execution of each step of the algorithm affects the display in the four windows simultaneously. Figure 3.16 shows seven data elements to be sorted using the MergeSort algorithm. When the user presses the ShowMe button, the four windows spring to life, moving the seven data items as needed and pausing between chunks until the algorithm is finished. HalVis intentionally limits the number of data items in the Execution Animation window to focus attention on the micro-behavior of the algorithm.



Objective: Describe the essential behaviors of the MergeSort algorithm and introduce high-level pseudocode

Description of MergeSort

MergeSort is a recursive algorithm that uses a Divide-and-Conquer approach to generate sorted sequences. The essential idea is to divide the input list recursively into halves until one element remains, then make repeated use of a Merge procedure that merges 2 lists (assumed to be in order) into a 3rd list (also in order).

MergeSort has 4 simple operations:

1. **Split the input into halves (here it simply finds the midpoint)**
2. **MergeSort the left half**
3. **MergeSort the right half**
4. **Merge the two sorted halves into a single sorted list**

MergeSort calls itself with half-sized lists until it reaches the base case. The base case is when the input to MergeSort contains only 1 element and cannot be divided any further. By default, a list of one element is in order, so what gets returned is an ordered sequence of 1 element to be merged with another partial (but ordered) list.

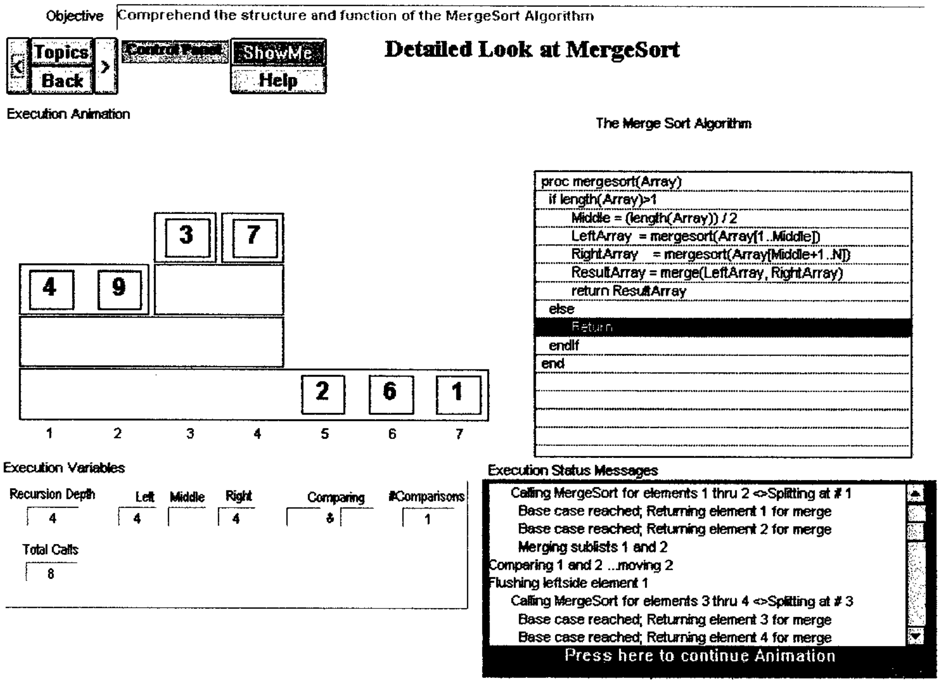
The algorithm for Merging two sequences into one is shown here and described in more detail by following the Merging link.

Because MergeSort splits the input in half, this algorithm is very efficient, involving "N Log N" steps. This is much less than the N-squared complexity of Bubble and Selection Sort algorithms. So, for a list of 50 elements, MergeSort requires approximately 300 steps whereas Bubble sort would require 2500!

```
proc mergesort(Array)
 if Array contains more than 1 element
   Middle = (length(Array)) / 2
   LeftHalf  = mergesort(Array[1..Middle])
   RightHalf = mergesort(Array[Middle+1..N])
   ResultArray = merge(LeftHalf,RightHalf)
   Return ResultArray
 else
   Return
 endif
endproc
```

```
Proc merge(LeftHalf,RightHalf)
 loop
    if leading item in LeftHalf < leading item in RightHalf
        append leading item in LeftHalf to Result
    else
        append leading item in RightHalf to Result
    endif
   until LeftHalf or RightHalf is empty
   while LeftHalf contains elements
       append remaining items from LeftHalf to Result
   endwhile
   while RightHalf contains elements
       append remaining items from RightHalf to Result
   endwhile
   return Result
end
```

Figure 3.15. The Description Screen for the MergeSort Algorithm



Objective: Comprehend the structure and function of the MergeSort Algorithm

Detailed Look at MergeSort

The Merge Sort Algorithm

```
proc mergesort(Array)
  if length(Array)>1
      Middle = (length(Array)) / 2
      LeftArray  = mergesort(Array[1..Middle])
      RightArray    = mergesort(Array[Middle+1..N])
      ResultArray = merge(LeftArray, RightArray)
      return ResultArray
  else
      Return
  endIf
end
```

Execution Variables

Recursion Depth	Left	Middle	Right	Comparing	#Comparisons
4	4		4	&	1

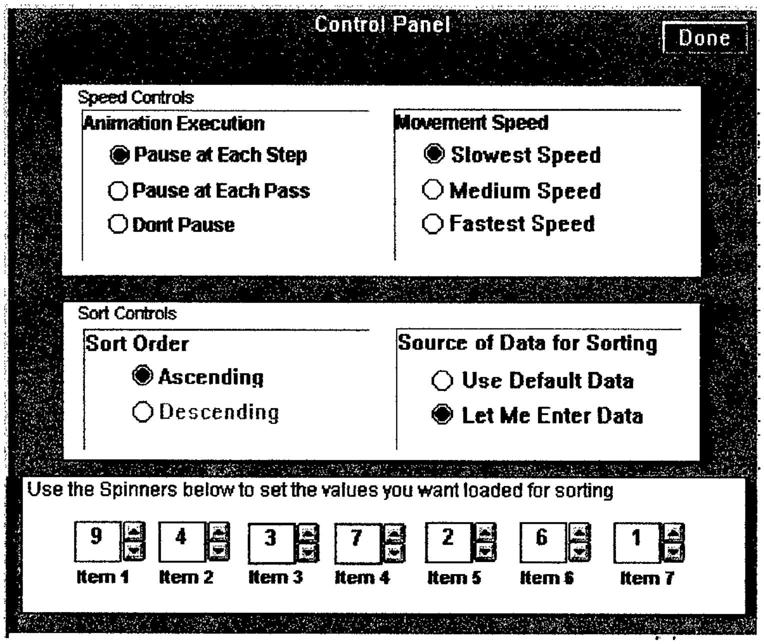
Total Calls: 8

Execution Status Messages

```
    Calling MergeSort for elements 1 thru 2 <>Splitting at # 1
     Base case reached; Returning element 1 for merge
     Base case reached; Returning element 2 for merge
     Merging sublists 1 and 2
Comparing 1 and 2 ...moving 2
Flushing leftside element 1
    Calling MergeSort for elements 3 thru 4 <>Splitting at # 3
     Base case reached; Returning element 3 for merge
     Base case reached; Returning element 4 for merge
```

Press here to continue Animation

Figure 3.16. The Detailed View Screen of the MergeSort Algorithm



Speed Controls

Animation Execution

● Pause at Each Step
○ Pause at Each Pass
○ Dont Pause

Movement Speed

● Slowest Speed
○ Medium Speed
○ Fastest Speed

Sort Controls

Sort Order

● Ascending
○ Descending

Source of Data for Sorting

○ Use Default Data
● Let Me Enter Data

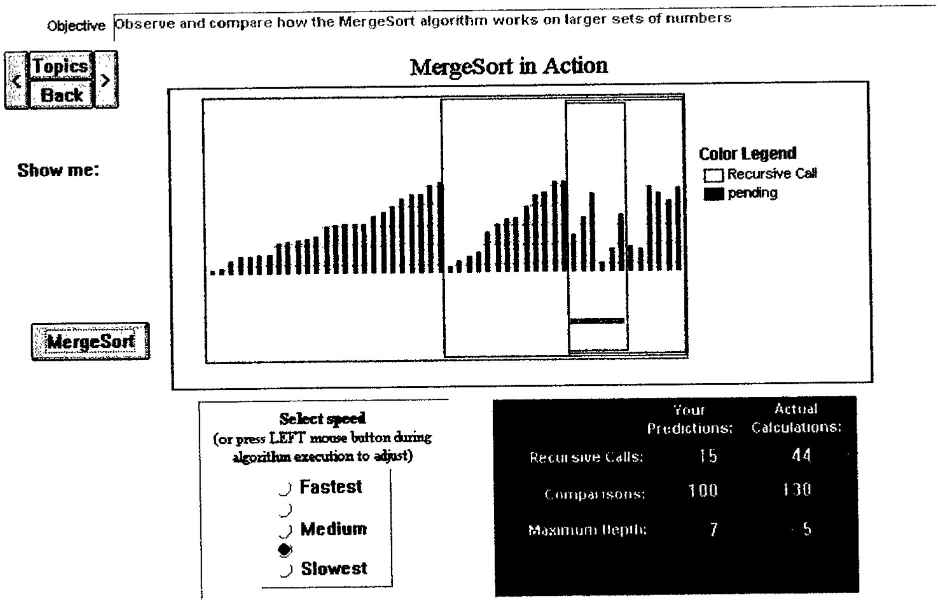
Use the Spinners below to set the values you want loaded for sorting

Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7
9	4	3	7	2	6	1

Figure 3.17. A View of the Control Panel of the Detailed View Screen

3.2.5. THE POPULATED VIEW

This module is intended to provide students with an animated view of the algorithm on large data sets to make its macro-behavior explicit. In this birds-eye view (see Figure 3.18), many of the details presented in the detailed view are obscured to enhance the student's focus on the algorithm's performance. Notice, for example, that the pseudocode is not shown. The variables are not shown as numbers but rather as colored bars. Animations and animation controls in this module are similar to those in previous systems, but there are two novel features. One is a panel of counters that show pertinent performance-oriented information such as number of comparisons, swaps, recursive calls, and so on. Another is a facility for the student to make a prediction about different parameters of algorithm performance and then compare those against the actual performance when the animation is running. When the learner presses the ShowMe button, the algorithm prompts for predictions, initializes the bars (data) into random, ascending or descending order, and proceeds to execute the algorithm. During execution, the bars change color and move about, accompanied by audio explanations and cues. Color coding is used to convey information such as already processed data and data elements currently being processed.



Objective Observe and compare how the MergeSort algorithm works on larger sets of numbers

Select speed
(or press LEFT mouse button during algorithm execution to adjust)

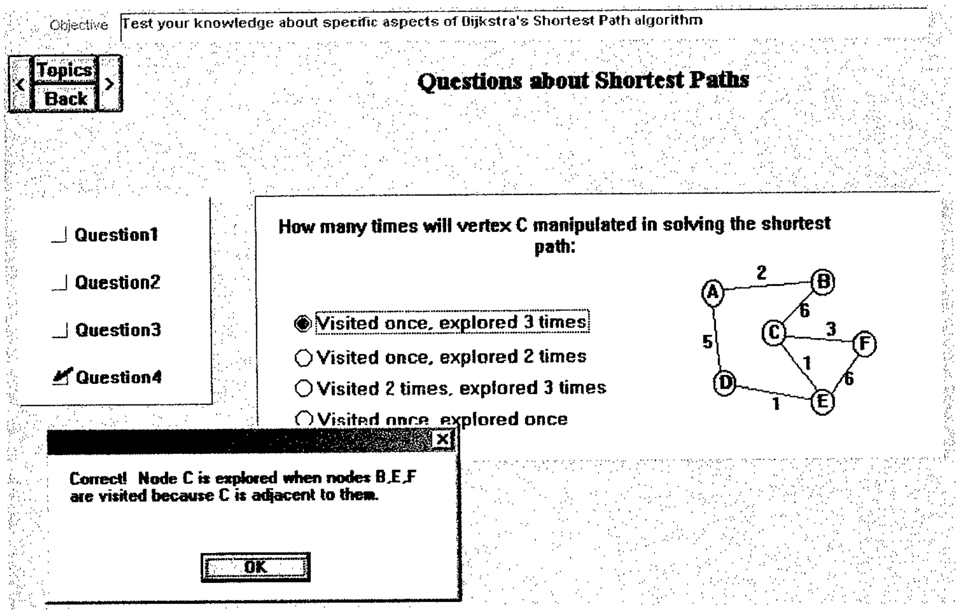
○ Fastest
○
○ Medium
●
○ Slowest

	Your Predictions:	Actual Calculations:
Recursive Calls:	15	44
Comparisons:	100	130
Maximum Depth:	7	5

Figure 3.18. The Populated View Screen

3.2.6. QUESTIONS MODULE

This module presents the student with several questions at articulation points between the other modules to facilitate and help them self-assess comprehension. A combination of multiple choice, true-false, and algorithm debugging questions are provided. Students get immediate feedback on their answers (Figure 3.19). HalVis also uses tickler questions as shown in Figure 3.20. The context-sensitive ticklers help focus the student's attention on key aspects of the algorithm being studied. Collectively, the use of questions promotes self-reflection, self-explanation, and sparks student interest.



Objective: Test your knowledge about specific aspects of Dijkstra's Shortest Path algorithm

Questions about Shortest Paths

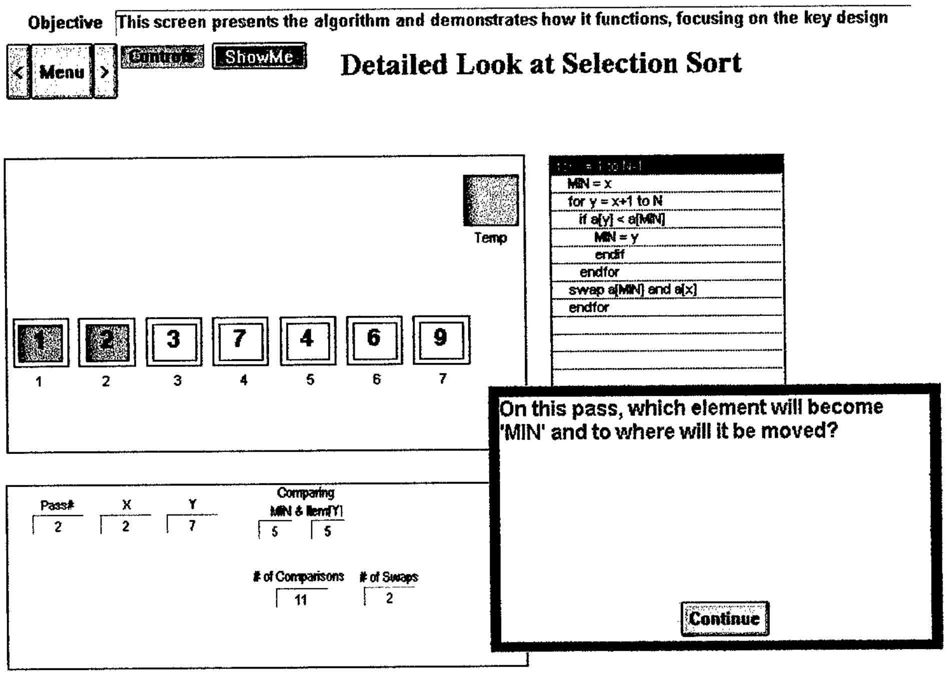
- Question1
- Question2
- Question3
- ✓ Question4

How many times will vertex C manipulated in solving the shortest path:

- ● Visited once, explored 3 times
- ○ Visited once, explored 2 times
- ○ Visited 2 times, explored 3 times
- ○ Visited once, explored once

Correct! Node C is explored when nodes B,E,F are visited because C is adjacent to them.

Figure 3.19. Feedback-style Question



Objective: This screen presents the algorithm and demonstrates how it functions, focusing on the key design

Detailed Look at Selection Sort

```
for x = 1 to N-1
   MIN = x
   for y = x+1 to N
     if a[y] < a[MIN]
        MIN = y
        endif
     endfor
   swap a[MIN] and a[x]
   endfor
```

Pass#	X	Y	Comparing MIN & Item[Y]	
2	2	7	5	5

# of Comparisons	# of Swaps
11	2

On this pass, which element will become 'MIN' and to where will it be moved?

Figure 3.20. Tickler-style Question

4. VALIDATING THE FRAMEWORK

We conducted empirical evaluations to validate the effectiveness of the framework outlined in the previous chapter. This chapter describes how the experiments were conducted and the results that were obtained.

4.1. EFFECTIVENESS: WHAT ARE WE LOOKING FOR?

Effectiveness is a broad concept with several aspects to consider before diving into formal research. The factors that we believe are significant to consider are:

- **Comprehension**: This refers to how closely the student came to meeting the learning objectives set forth by the instructor. Generally this is measured by testing the student. A system that improves comprehension over alternative systems would be regarded as more effective. Alternatively, this could be referred to as short-term retention.
- **Retention**: This is a measure of comprehension over time. Research indicates humans have short term memory, capable of retaining small bits of information for a matter of minutes, and long term memory, capable of recalling information for years. A system that improves long term retention is clearly more effective.
- **Speed**: How long did it take the student to meet the learning objectives? A system that helps students learn material more quickly without sacrificing the other factors would be considered more effective.

- **Satisfaction**: This refers to how engaging the system is to the student. Before and during the session this could be called motivation. After the fact, this is often referred to as satisfaction. Either is a difficult thing to measure quantitatively, and are generally reported as subjective figures. Generally, a system that fosters greater learner satisfaction without negatively affecting the other measures should be considered a more effective system. This is not always the case though.

Notice that some of the measures can lead to a conflict. Consider speed and satisfaction, for example. Because a motivated student is more likely to spend greater amounts of time with software that is engaging and rewarding, it does not necessarily follow that such a system is less effective. Nor does it follow that a system that takes less time but has extremely low satisfaction is better or worse either. Researchers simply have to determine which measures are most important for their domain. In our experiments, we focus on comprehension and retention as the primary measures of the effectiveness of our theoretical framework. We believe (and hope) that student satisfaction with our system will be very positive. We report speed measures anecdotally.

We justify this position by observing that previous empirical research resulted in virtually unanimous student satisfaction with algorithm animations as a learning method, but none showed that the use of animation leading to increased student performance. Hence, for our purposes, we recognize the importance of speed in learning, but place higher priority on achieving higher comprehension, retention and satisfaction. In the sections that follow, we use the terms comprehension and 'learning' interchangeably, referring to the improvement in student performance as measured by the difference between individual pre-test and post-test scores.

4.2. EXPERIMENTS WITH HYPERMEDIA ALGORITHM VISUALIZATIONS

We conducted experiments to evaluate and substantiate the effectiveness of hypermedia visualization of algorithms, specifically to validate our belief that a hypermedia system incorporating text, images, and animations would be an effective tool for teaching students about algorithms. Specifically, our experiments were designed to compare using the HalVis framework to learn about algorithms with the methods currently used by instructors to teach their students. Common teaching instruments include textbooks, lectures, and laboratory experiences. More recently, instructors have also started using algorithm animations (Stasko, 1997). A survey of computer science instructors (Badre et al., 1991) showed that over 80% of instructors use at least one of these methods to teach algorithms to students. Previous experiments on the effectiveness of algorithm animations as teaching tools compared experimental groups employing a combination of instructional media, as shown in Table 4.1.

Research Reference	**Subj. Level**	**Comparison Groups**	
Lawrence et al. 1994	K13	Animation + Lecture + Laboratory	Slides + Lecture + Lab
Crosby & Stelovsky 1995	K13	Animation + Lecture + Homework	Slides + Lecture + Homework
Stasko et al. 1993	K17	Animation + Text	Text
Badre et al. 1991	K14	Animation + Handout	Lecture + Handout
Byrne et al. 1996	K15	Animation + Video Lecture + Text	Diagrams + Video Lecture + Text

Table 4.1. Prior Experiments Involving Algorithm Animation

A criticism of previous empirical research was a failure to sufficiently isolate the visualization from other learning techniques (Hundhausen, 1996). To avoid confounding the various factors, we designed experiments to specifically compare our visualization framework with learning from a textbook (Experiment I), learning from a textbook that includes completing exercises (Experiment II), and learning from a lecture (Experiment III). While we acknowledge that most instructors use a combination of teaching methods, our approach is to isolate each method and compare it with our AV framework for statistical analysis. Lastly, we designed an additional experiment (Experiment IV) to replicate the environment used by several previous researchers who combined text with animation and compared it to the hypermedia-only approach.

Our hypothesis was that students would learn more effectively using HalVis than from other teaching methods, as indicated by their performance in pre-tests and post-tests. The five individual experiments are summarized in Table 4.2, showing the student level, the learning media used by the comparison groups, and the algorithm(s) studied. The first experiment compared learning with HalVis to learning from textbooks alone. To gain insight into the effects of algorithm complexity and student ability, the experiment had two components, one that exposed novice students to a relatively simple algorithm, and the other that exposed more advanced students to two algorithms of moderate complexity. Extending this comparison further, Experiment II compared learning with HalVis to learning from a compilation of the best descriptions and depictions extracted from a survey of 19 textbooks followed by solving a set of exercises. The third experiment compared learning from HalVis to learning from lectures. Finally, the fourth experiment compared learning from HalVis to learning from the combination of a typical algorithm animation and text. These experiments, we felt, would help us determine the comparative effectiveness of the HalVis framework. In these experiments we used pre- and post-tests to measure students' ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. We did not differentiate between visual and verbal learners since HalVis contains rich textual and visual presentations to support both kinds of learner dispositions.

	Level	**Comparison Groups**		**Algorithm(s)**
Experiment I a	K14	HalVis	Text + Diagrams	MergeSort (MS)
Experiment I b	K15	HalVis	Text + Diagrams	MergeSort QuickSort (QS)
Experiment II	K14	HalVis	Text + Diagrams + Exercises	BubbleSort (BS) SelectionSort (SS)
Experiment III	K14	HalVis	Lecture	SelectionSort MergeSort
Experiment IV	K15	HalVis	Text + Animation	Shortest Path (SP)

Table 4.2. Experiment Summary

The general format of each of the experiments involved:

- Gathering participants from appropriate computer science courses.
- Dividing participants into balanced, matched groups. Balanced groups were necessary to ensure that results were not skewed or biased by differing group capabilities. The matched groups were created by random assignment from a rank-ordered list of subjects based on results of a demographic survey that gathered information regarding academic performance and potential (GPA, ACT/SAT scores, class standing, etc). In experiments involving two groups, successive pairs of subjects were drawn from the rank-ordered list and randomly assigned to one group or the other. In experiments involving four groups like the ablation studies in the next chapter, four subjects were drawn from the list at a time, and randomly assigned to one of four groups. Finally, after the matched groups were created, the treatment condition that each group was to receive for the experiment was randomly designated to eliminate any possibility of bias.

- Administering a Prior Knowledge Survey ("pre-test") to assess pre-existing individual knowledge about the algorithm to be studied, thereby establishing a baseline from which to measure improvement more precisely. The pre-test was different from the demographic survey used to create the groupings, in that the pre-test measured algorithmic knowledge about the specific algorithm the group was about to study in the experiment whereas the survey merely gathered information about general academic potential. Without a pre-test, measuring individual improvement would be difficult and could invalidate our results.
- Conducting the experiment.
- Administering a Knowledge Improvement Survey ("post-test") to evaluate the participant's knowledge about the algorithm following the experiment.
- Gathering student comments and impressions about the software they used in a survey, if appropriate, and examining the logs that recorded student activity while using the hypermedia software.

4.2.1. EXPERIMENTS IA AND IB: COMPARISON OF HALVIS WITH TEXTUAL LEARNING

This experiment consisted of a pair of studies intended to explore our hypothesis that students would learn more effectively using the HalVis animation-embedded hypermedia framework than by printed materials from a textbook. We conducted the experiment in two phases using similar procedures but students at different levels in their computer science curriculum. The first experiment involved 28 novice students enrolled in an introductory data structures and algorithms class. The second experiment employed 22 more experienced students learning two algorithms of greater sophistication.

4.2.1.1. Experiment Ia

Subjects:

The experiment involved 33 undergraduates enrolled in an introductory data structures and algorithms course at Auburn University. Subjects received course credit for their participation. In the first week of the quarter, subjects completed a demographic survey providing information such as GPA, ACT and SAT scores. We used this information to rank students and create a matched pair of groups: one group (called the "Text" group) would learn about the MergeSort algorithm using textbook descriptions, and another group (called the "Algorithm Visualization" (AV) group) would learn the MergeSort algorithm using the HalVis algorithm visualization tool. The groups were initially balanced but five students did not complete all facets of the experiment and their data is not included. Of the 28 students completing all aspects of the experiment, twelve students were in the Text group and sixteen students were in the AV group. The loss of the five students did not imbalance the groups and skew the results. The average GPA and ACT score of the Text group was 2.9 and 28, respectively. The average GPA and ACT score of the AV group was 3.0 and 27, respectively.

Materials

The Text group received a photocopied six-page extract from a textbook (Dale, Lilly & McCormick, 1996) that discussed the MergeSort algorithm. The handout included a description and analysis of the algorithm, various diagrams, and program code.

The AV group learned about the MergeSort algorithm using the HalVis system with no supplementary materials provided.

A pre-test/post-test combination (see Appendix B.4.1) measured individual learning performance with nine questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithm and the post-test measured changes resulting from experimental conditions.

Procedure

We structured the experiment to follow class lectures covering basic program design and fundamental data structures, but precede lectures that covered sorting algorithms. Towards the middle of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the MergeSort algorithm. The pre-test results helped us verify that the two groups were evenly balanced, and provided a baseline against which to compare subsequent changes. The pre-test scores indicated that the subjects did not know this algorithm and that the groups were evenly distributed (Average = 27% for the Text group and 28% for the AV group).

During the following week, the AV group met in a public computer laboratory. They were given a five minute introduction to HalVis, which oriented them to the various screens they would encounter and provided them with basic navigational tips. The students were then assigned to a computer and instructed to interact with the software until they felt they understood the MergeSort algorithm. The computers were Pentium-class systems with 15 inch color monitors. Subjects were not given any text material to study, nor had they been exposed to the MergeSort algorithm in the class prior to the experiment. There was no time limit, so when each subject indicated he/she was done, he/she was given a post-test that helped measure knowledge improvement. No student in the AV group took more than 60 minutes for the entire experiment.

On the same day the Text group met in a classroom, and was provided with photocopied handout describing the MergeSort algorithm. They were not provided any other information, nor had they been exposed to the MergeSort algorithm during class lectures. They were asked to learn the MergeSort algorithm from the materials provided, with no time limit imposed. When they finished studying the explanatory materials provided, they were given a post-test and allowed to leave. No student in the text group took more than 45 minutes for the entire experiment.

Results

The overall results are summarized as box plots in Figure 4.1. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group.

The pre-test results indicate that both groups were equally unfamiliar with the MergeSort algorithm. The post-test averages show a significant improvement for the AV group over the Text group. The AV post-test average was 74% compared to the Text group's 43%, and the results are significant for both the overall performance (F(1,27)=10.9, p<0.003) and for improvement (F(1,27)=6.7, p<0.015). The statistical summary is shown in Table 4.3.

Statistical Summary			
	Pre-Test	Post-Test	Improvement (raw)
Text Group	27%	43%	16%
AV Group	28%	74%	46%
F(1,27)	0.01	10.9	6.7
Significance level	p<0.93	p<0.003	p<0.015

Table 4.3. Experiment Ia Statistical Summary

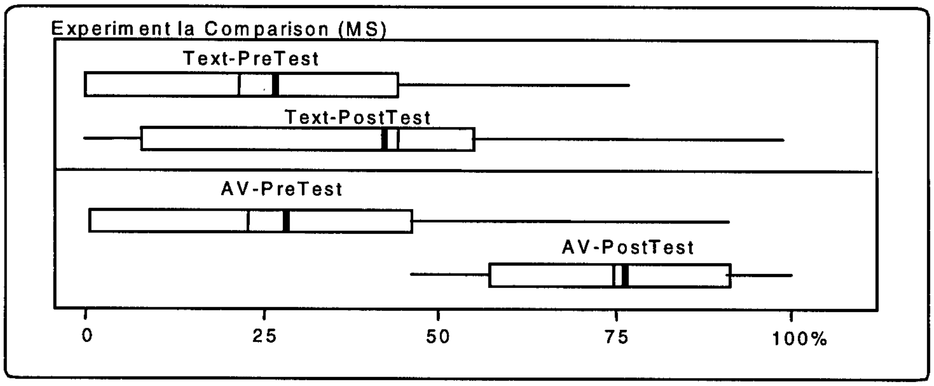
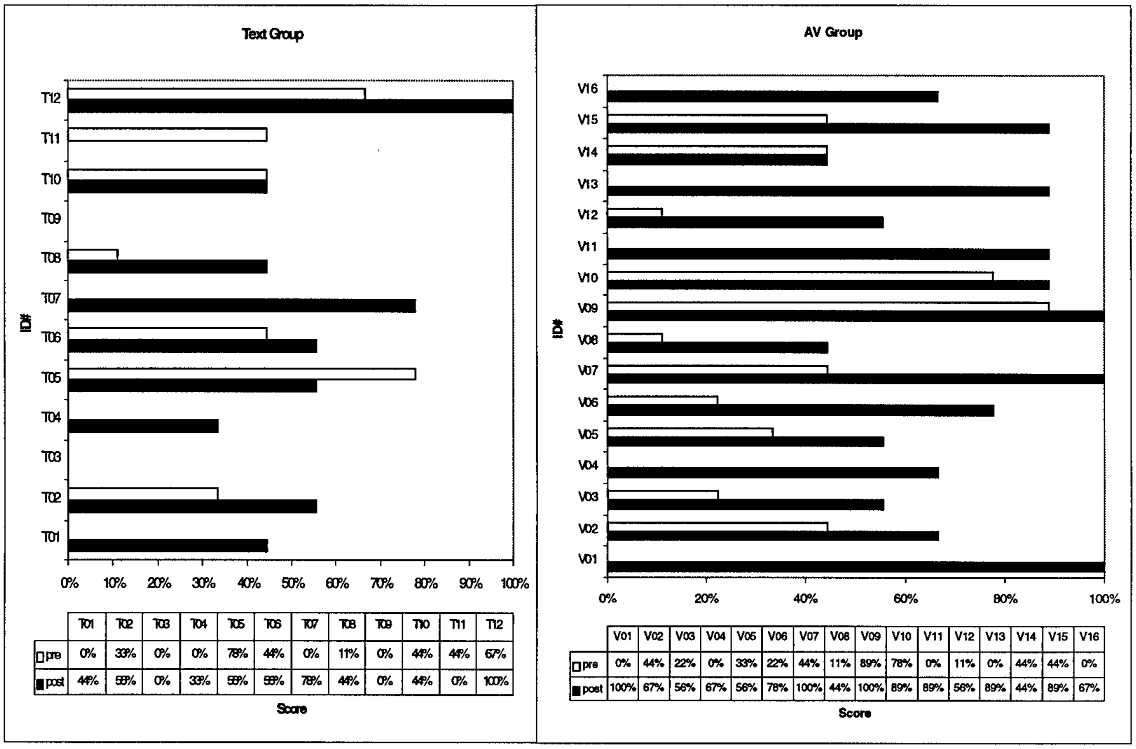


Figure 4.1. Experiment Ia Box Plots

Figure 4.2 shows the individual pre-test and post-test scores of each subject by group. Each participant is indicated by his/her randomly assigned ID number on the vertical axis. Pairs of horizontal bars indicate each participant's test performance. The light bars represent pre-test performance and the dark bars show post-test performance. No bar is shown when the corresponding score is zero. The tables below provide the percentage grade obtained by each participant in pre- and post-tests. It is interesting that every subject in the AV group improved his/her knowledge, but two subjects in the Text group (T11 and T05) actually did worse.



	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10	T11	T12
□ pre	0%	33%	0%	0%	78%	44%	0%	11%	0%	44%	44%	67%
■ post	44%	56%	0%	33%	56%	56%	78%	44%	0%	44%	0%	100%

	V01	V02	V03	V04	V05	V06	V07	V08	V09	V10	V11	V12	V13	V14	V15	V16
□ pre	0%	44%	22%	0%	33%	22%	44%	11%	89%	78%	0%	11%	0%	44%	44%	0%
■ post	100%	67%	56%	67%	56%	78%	100%	44%	100%	89%	89%	56%	89%	44%	89%	67%

Figure 4.2. Experiment I Individual Results, by Group

Discussion

These results suggest that novice students perform better in answering conceptual and procedural questions about the MergeSort algorithm after learning from a hypermedia algorithm visualization than after studying a typical textbook. However, there are several factors that must be mentioned to keep these results in perspective. First, one could argue that a different textbook could have led to different results. We believe we reduced the possibility of this form of bias by using a text that was competitively selected from books reviewed by a group of professors and graduate students in the Computer Science Department at Auburn University. While we consider the material we used was from a well-written book, Experiment III was designed to probe this issue further. Second, only novice students participated in this experiment. It is possible that more advanced students may benefit more from a textbook explanation of an algorithm. Experiment Ib investigated this possibility.

Third, difference in student motivation between the groups could have influenced the results. The level of enthusiasm observed in the HalVis group was much higher than in the Text group. The novelty of the visualization and the interactive features of HalVis seemed to engage the students' interest. In contrast, there was nothing new or uniquely motivating for the Text group. Fourth, familiarity with the learning materials provided could have had an influence. The Text group did not have to acquaint themselves with a new user interface, software system or learning from interactive visualizations. They were all familiar with reading and learning from a textbook. The students in the AV group had to contend with a new interface and a different way of learning. If this factor indeed played a role, the AV group exhibited a higher level of comprehension despite any additional cognitive effort involved in learning the interaction and navigation facilities of HalVis. The opposite could also be the case, however, in that the computer science students in the Text group might have performed poorly because of a pronounced discomfort with printed materials of any kind and would have demonstrated a preference for any kind of computerized alternative.

4.2.1.2. Experiment Ib

This experiment was similar to Experiment Ia in that the goal was to compare the effectiveness of learning using HalVis to learning from printed materials from a textbook. Our aim was to test whether results of Experiment Ia could be replicated with more sophisticated algorithms and higher level students. We asked students to learn the MergeSort and QuickSort algorithms. Unlike the previous experiment, these participants completed all components of the experiment in one day: a pre-test, learning two algorithms, and a post-test.

Subjects

This experiment involved 22 undergraduate computer science students enrolled in a third year algorithm analysis course at Auburn University. Like Experiment Ia, participants were ranked based on academic ability (using on course performance up through mid-term grades in the course) and assigned to a matched pair of groups: a "Text" group and an "Algorithm Visualization" (AV) group. Students were given extra credit for participating. Eleven students in the Text group and eleven students in the AV group completed the experiment. The average class mid-term score for subjects in the Text group was 21.5 and 21.6 for subjects in the AV group.

Materials

The Text group received a ten page photocopied extract from their course textbook (Weiss, 1993) that discussed the MergeSort and QuickSort algorithms.

The AV group learned about the MergeSort and QuickSort algorithms using the HalVis system with no supplementary materials provided.

A pre-test/post-test combination measured individual learning performance with 18 questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithms and the post-test measured knowledge improvement resulting from the experimental conditions.

Procedure

We structured the experiment to precede the class lectures that dealt with sorting algorithms. Towards the middle of the quarter, on the day of the experiment, all participants met in a classroom and completed the pre-test. Afterwards, members of the AV group were taken to a public computer laboratory, while the Text group remained in the classroom.

In the computer laboratory, the AV group was given a brief navigation-only orientation to the HalVis system, then assigned to individual computers to interact with the software and learn the algorithms. Students were allowed to take as much time as needed. They did not have access to any supplementary materials. As each subject finished interacting with the visualizations, a post-test was given. All subjects completed the experiment in less than two hours.

The Text group was given the extract from their course textbook. This contained a typical combination of textual descriptions and explanations, diagrams, pseudocode and program examples. Like the AV group, there was no time constraint. When a subject signaled completion of studying the materials, he/she was given the post-test and allowed to leave. All students in the Text group completed the pre-test, studied the textual materials and completed the post-test in less than 90 minutes.

Results

The overall results are summarized as box plots in Figure 4.3. The pre-test results indicate that both groups were equally unfamiliar with both algorithms. The post-test averages show a significant improvement for the AV group over the Text group. The AV post-test average was 63% compared to the Text group's 44%, and the results are significant for both the overall results ($F(1,21)=4.96, p<0.038$) and for improvement ($F(1,21)=9.29, p<0.006$). The statistical summary is given in Table 4.4, showing both aggregate and algorithm-specific results for each group.

Statistical Summary			
	Pre-test	Post-test	Improvement (raw)
Text (MS)	44%	53%	9%
AV (MS)	48%	71%	23%
Text (QS)	10%	35%	25%
AV (QS)	4%	55%	51%
Text (MS+QS)	27%	44%	17%
AV (MS+QS)	26%	63%	37%
F(1,21)	0.02	4.96	9.29
P	p<0.89	p<0.038	p<0.006

Table 4.4. Experiment Ib Statistical Summary

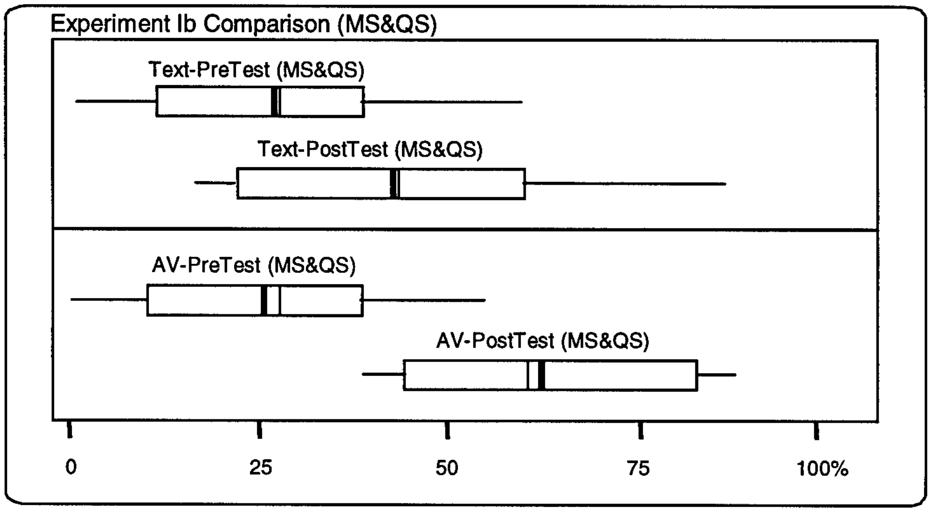


Figure 4.3. Experiment Ib Box Plots

Discussion

These results parallel those of Experiment Ia in suggesting that students perform better in answering conceptual and procedural questions about the MergeSort and QuickSort algorithms after using a hypermedia visualization system to learn than after studying a typical textbook. As shown in Table 4.5, the more advanced status of the students led to higher prior knowledge scores for the MergeSort algorithm (44% and 48%), which was much higher than the pre-test levels observed in the novice students of Experiment I (27% and 28%). This is not a surprising result, since it is reasonable to expect that the more advanced students could have been exposed to the MergeSort algorithm before, and even if not, to grasp the essential concepts more readily than novice students. Together, these two experiments suggest that learning by visualization leads to better comprehension than by printed text for novice and more advanced students. As Table 4.5 indicates, AV groups improved their performance by approximately two to three times compared to the performance improvement of the Text groups in the two experiments. Interestingly, the AV groups in both experiments reached the same level of performance after interacting with HalVis though they started off with different levels of prior knowledge.

	Experiment Ia		Experiment Ib	
	Pre-test	Post-test	Pre-test	Post-test
Text Group	27%	43%	44%	53%
AV Group	28%	74%	48%	71%

Table 4.5. Comparison of Results of Experiments Ia and Ib

As with Experiment Ia, there are several factors to consider to keep these results in perspective, such as textbook quality, student experience and ability, motivation, and familiarity with learning approaches used in the experiment. This experiment added a significant load to the learning task, requiring students to learn two fairly challenging algorithms in one session, as well as complete a pre-test and a post-test in the same session. This cognitive load could have affected the test performance of both groups.

4.2.2. EXPERIMENT II: COMPARISON OF HALVIS TO LEARNING FROM TEXT WITH EXERCISES

This experiment was conducted in a similar fashion to the ones previously described, involving students relatively new to algorithmic study, and compared the effectiveness of learning from HalVis to learning from text *and* problem solving. The goal was to provide one group with the *best* possible descriptive and depictive printed materials and a set of exercises, in order to investigate the limits of learning from HalVis by comparing it with learning from carefully designed, detailed textual and diagrammatic explanatory materials coupled with problem solving. We chose to use the BubbleSort and SelectionSort algorithms for this experiment. While these are relatively simple algorithms, we felt that asking participants in this experiment who were novice students, not yet exposed in depth to the subject of sorting, to learn both algorithms in one session would represent a reasonable cognitive load.

Subjects

The experiment involved 25 undergraduate computer science students enrolled in an introductory data structures and algorithms course at Auburn University. Subjects received course credit for their participation. In the first week of the quarter, they completed a demographic survey providing information such as GPA, ACT and SAT scores. We used this information to rank students and randomly assign them to form two matched groups: one group (called the "Text" group) that would learn about the BubbleSort and SelectionSort algorithms using a handout that we created and then completing a series of problem solving exercises, and another group (called the "Algorithm Visualization" (AV) group) that would learn the same algorithms using the HalVis algorithm visualization tool alone. Twelve students in the Text group and thirteen students in the AV group completed the experiment. The average GPA and ACT scores for the Text group were 2.92 and 28 respectively. The average GPA and ACT scores for the AV group were 2.96 and 27, respectively.

Materials

The Text group received an eight-page explanation that contained both textual descriptions and graphic depictions of the BubbleSort and SelectionSort algorithms, along with several exercises. The handout was subjected to a stringent review by a group of faculty and graduate students to address the issue of the quality of text used in the previous experiments. After reviewing the descriptions, depictions and examples of the BubbleSort and SelectionSort algorithms contained in nineteen textbooks published between 1974 and 1997 (Aho, Hopcroft & Ullman, 1974; Aho, Hopcroft & Ullman, 1983; Baase, 1988; Dale, Lilly & McCormick, 1996; De Lillo, 1993; Harel, 1992; Horowitz & Sahni, 1978; Kingston, 1990; Knuth, 1973; Kozen, 1992; Manber, 1989; Nance & Naps, 1995; Reingold & Hansen, 1983; Rowe, 1997; Sedgewick, 1988; Shaffer, 1997; Singh & Naps, 1985; Weiss, 1993; Wirth, 1986), we selected the best explanations we could find. These explanations were then edited to increase clarity and merged to create a handout containing textual and pictorial explanations of the two algorithms. We also developed and included a set of “end of chapter” style exercises in this handout for students to solve after perusing the explanations. This handout is provided in Appendix B.5.2.

The AV group learned about the BubbleSort and SelectionSort algorithms using the HalVis system with no supplementary materials provided.

A pre-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. A copy of the post-test is included in Appendix B.5.1.

Procedure

As with the previous experiments, we timed the experiment to follow class lectures covering basic program design and fundamental data structures, but precede those that covered sorting algorithms. Towards the middle of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the BubbleSort and SelectionSort algorithms. In addition to providing a baseline against which to compare subsequent changes, the pre-test results also helped us verify that the two groups were evenly balanced.

The following week, the AV group met in a computer laboratory on campus. They were given a five minute introduction to HalVis, which oriented them to the various screens they would encounter and provided them with navigational tips. The students were then assigned to a computer and instructed to interact with the software until they felt they understood the two algorithms. The computers were Pentium-class systems with 15 inch color monitors. Subjects were not given any text material to study, nor had they been exposed to the algorithms in class lectures. There was no time limit, so when each subject indicated he/she was done, he/she was given a post-test that helped measure knowledge improvement. No student in the AV group took more than 90 minutes for the entire experiment.

On the same day, the Text group met in a classroom, and was provided with the handout described above. They were asked to read and understand the materials and then to solve the set of exercises at the end. They were not provided with any additional information, nor had they been exposed to these algorithms in class lectures. When they finished studying the descriptive materials and attempting the exercises, they were given a post-test and allowed to leave. No student in the text group took more than 60 minutes for the entire experiment.

Results

The overall results are shown in Figure 4.4. The pre-test scores indicated that the subjects did not know these algorithms and that the groups were evenly matched (Average = 35% for the Text group and 31% for the AV group). The post-test averages show an improvement of 30% for the AV group to only 22% for the Text group. These results, while indicating better learning for the AV group, are *not* statistically significant as can be seen in Table 4.6.

Statistical Summary			
	Pre-test	Post-test	Improvement (raw)
Text	35%	57%	22%
AV	31%	61%	30%
F(1,24)	0.36	0.32	0.82
P	p<0.55	p<0.57	p<0.37

Table 4.6. Experiment II Statistical Summary

The results are summarized as box plots in Figure 4.4. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group. The box plots reveal a wider dispersion of scores in the post-test results of the Text group, but a much tighter clustering in the post-test results of the AV group.

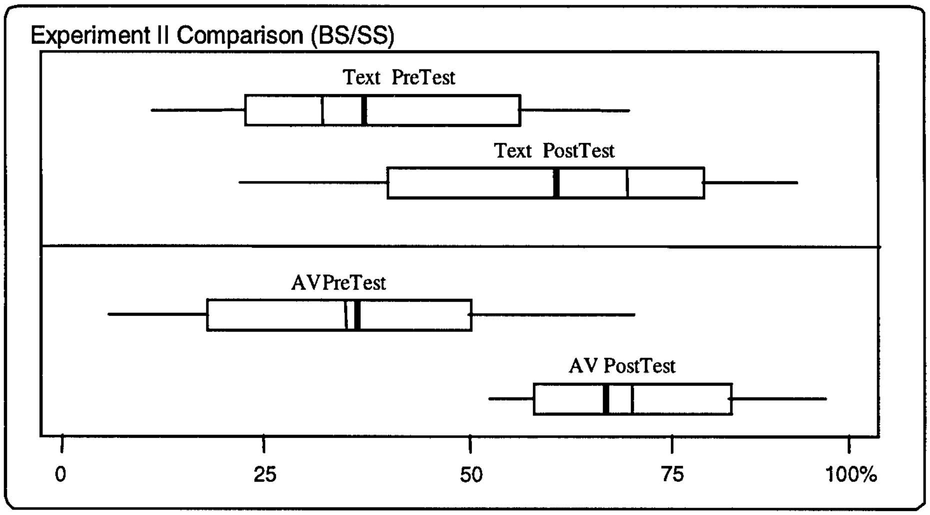


Figure 4.4. Experiment II Box Plots

Figure 4.5 shows the individual pre-test and post-test scores of each subject by group. Each participant is indicated by his/her randomly assigned ID number on the vertical axis. Pairs of horizontal bars indicate each participant's pre- and post-test scores. The light bars represent pre-test performance and the dark bars show post-test performance. The tables below provide the percentage grade obtained by each participant in pre- and post-tests. Two subjects in the Text group (T06 and T07) and one subject in the AV group (V10) did worse after the experiment than they did on the pre-test. This is the first time, in the three experiments conducted thus far, where the performance of a member of the AV group decreased.

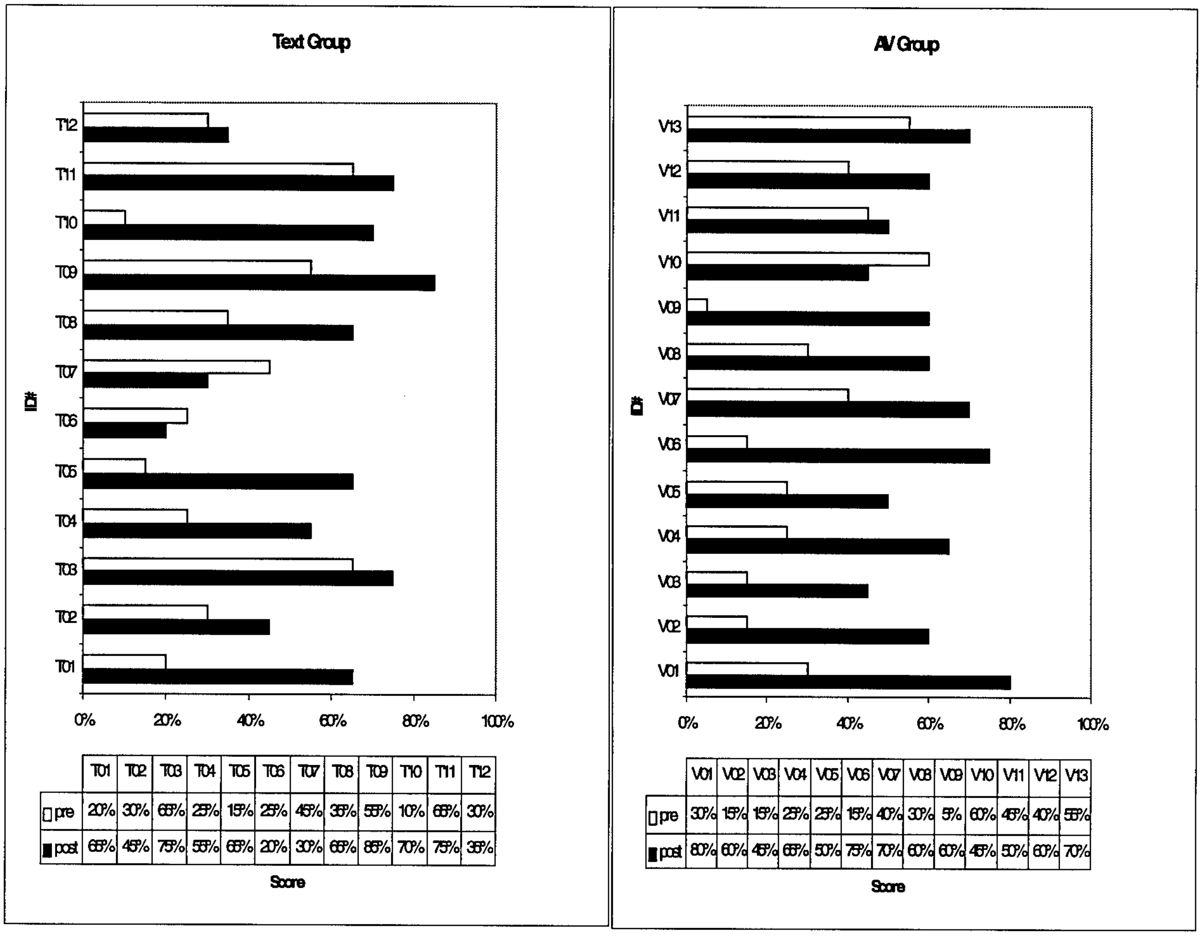


Figure 4.5. Experiment II Individual Results, by Group

Figure 4.6 shows performance by question (see Appendix B.5.1. for the questions) across the two groups for the post-test. For each question on the vertical axis, the horizontal axis provides the number of subjects who answered it correctly. It can be seen that only in three questions did the text group outperform the AV group: questions 7 and 11 dealing with worst case orderings, and question 10 that examined the number of swap operations the SelectionSort algorithm would require.

Figure 4.7 shows performance across pre- and post-tests by the AV group. For each question on the vertical axis, the horizontal axis provides the number of subjects who answered it correctly. If none answered a question correctly (Q5b, pre-test), the corresponding bar is not shown. This figure illustrates the substantial increase in the number of students answering correctly the questions dealing with algorithm recognition (Q1, Q3), behavior (Q6, Q8, Q10, Q12, Q14) and data ordering (Q5, Q9, Q13) after interacting with the visualizations of the algorithms.

Discussion

The combination of using simpler algorithms, significantly improving the text and asking students to engage in problem solving had a marked impact on the Text group's performance. It improved to a level on par with that of the AV group. Our conclusion from this experiment is that AV appears to be as effective for novice students to learn about algorithms as learning from carefully crafted textual materials coupled with problem solving exercises. Factors we did not control for, such as motivation and familiarity with textual descriptions and exercises may also have influenced the results.

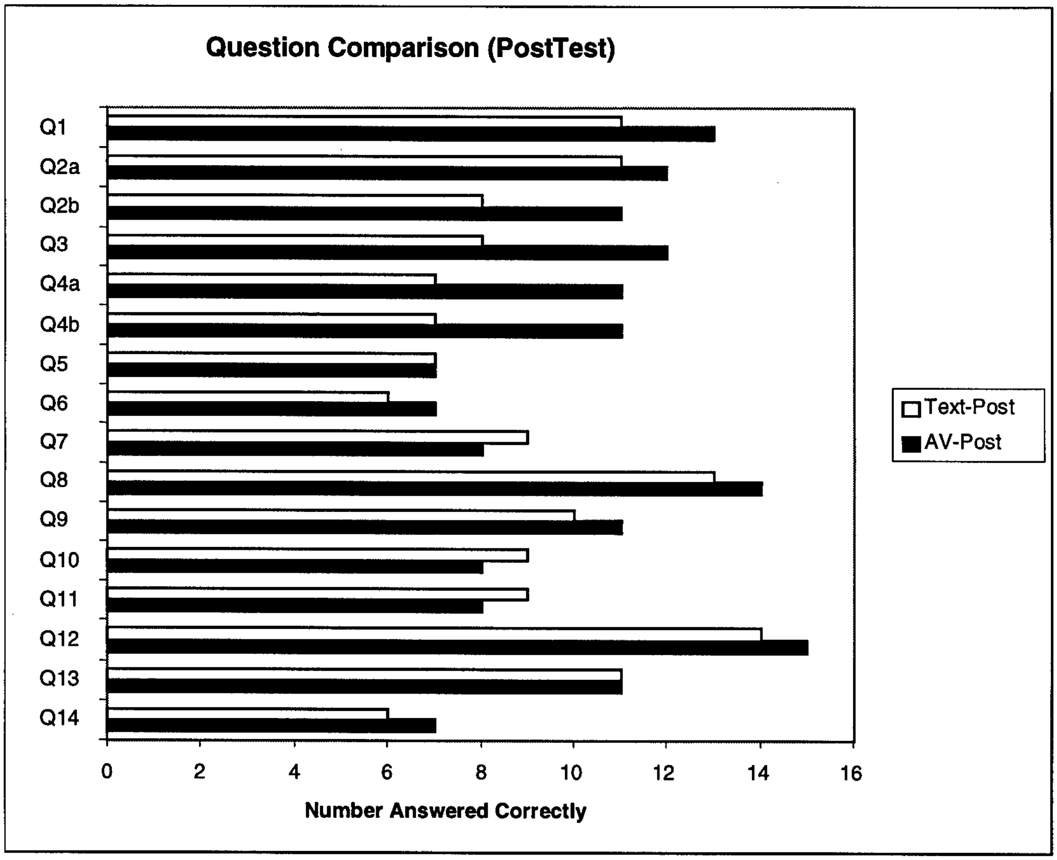


Figure 4.6. Comparison of Post-test Responses, by Group

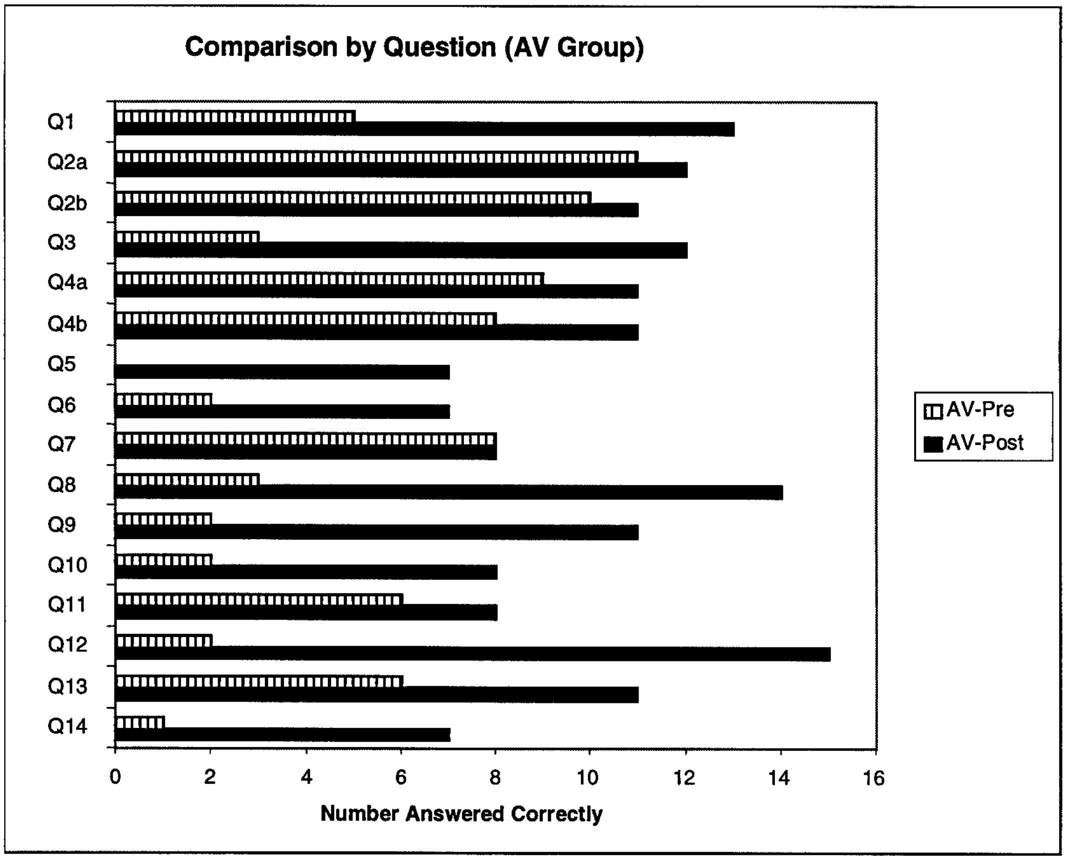


Figure 4.7. Comparison of AV Group Pre-test and Post-test Responses

4.2.3. EXPERIMENT III: COMPARISON OF HALVIS AND LECTURE-BASED LEARNING

This experiment was designed to compare HalVis with classroom lectures, and also to investigate how HalVis and lectures can together contribute to learning. We wanted to (1) verify our hypothesis that students learning from HalVis would outperform students learning by lecture alone, (2) measure additional learning obtained by combining lecture *and* HalVis, and (3) investigate whether the order (HalVis before lecture, or vice versa) would make a difference in performance. Participants in this experiment were novice computer science students, and the algorithms used were SelectionSort and MergeSort.

Subjects

The experiment involved 27 undergraduates enrolled in an introductory data structures and programming course at Auburn University. Subjects received extra credit for their participation. In the first week of the quarter, the subjects completed a demographic survey providing information such as GPA, ACT and SAT scores. We used this information and current class standing to rank and assign students to a matched pair of groups, a Lecture-Visualization (LV) group and a Visualization-Lecture (VL) group. The LV group received a class lecture discussing the algorithms, then interacted with the visualizations of the two algorithms in a computer laboratory. The VL group interacted first with HalVis, then attended the class lecture covering the algorithms. There were nine students in the LV group and eleven in the VL group that completed the three components of the experiment; seven students (six in the LV group and one in the VL group) did not complete all three components and their data is not included in the analysis below. The average mid-course grade for students in the LV group was 86% and the average for the VL group was 82% of the points possible at that time in the course.

Materials

All participants attended a lecture on the two sorting algorithms provided in two consecutive 50-minute class sessions conducted by Dr. Dean Hendrix, a member of the Auburn University Computer Science and Engineering Department faculty. His lecture consisted of verbal instruction accompanied by blackboard diagrams, overhead transparencies, and a lecture summary handout. He responded to several questions from students in the class during the lecture.

Both groups interacted with the same visualizations of both the SelectionSort and MergeSort algorithms, with no supplementary materials provided.

A pre-test/mid-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes.

Procedure

The phases of this experiment were carefully synchronized with the class syllabus. The week before the scheduled lecture about sorting algorithms, students were given a pre-test to measure prior knowledge about the two algorithms and to obtain a baseline to measure subsequent changes.

The day before the lecture, the VL group met in a computer laboratory on campus, interacted with HalVis to learn the two algorithms, and completed a mid-test. This test measured changes in knowledge resulting from HalVis interaction. The same lecture was attended by both of the groups. We chose to use a regular classroom lecture over a videotaped one to allow student interaction with the professor and to simulate a realistic learning environment. Having both groups attend the same lecture eliminated variations between separate lectures.

The day after the lecture, the LV group met in a computer laboratory on campus and first completed the same mid-test taken by the VL group. This test measured changes in knowledge resulting from the lecture for the LV group. Then the group was assigned to computer terminals and asked to interact with HalVis to learn the two algorithms. When they felt they understood the algorithms, they were asked to complete a post-test and allowed to leave. On this same day, the VL group met in a classroom and completed the same post-test. The post-test measured the final knowledge level of the two groups after both the lecture and the interactive sessions.

We designed the experiment to minimize outside interactions that might affect the results. First, while we did not explicitly instruct students not to read the course textbook or try to learn more about the algorithms from other sources, only one of the algorithms was covered in the course textbook. A question in the mid- and post-tests asked the students whether they had read about the algorithms elsewhere. None of the students indicated on the mid-test that they had read about the algorithms, but four indicated on the post-test that they had. We did however ask students to refrain from discussing any aspect of the experiment during its course.

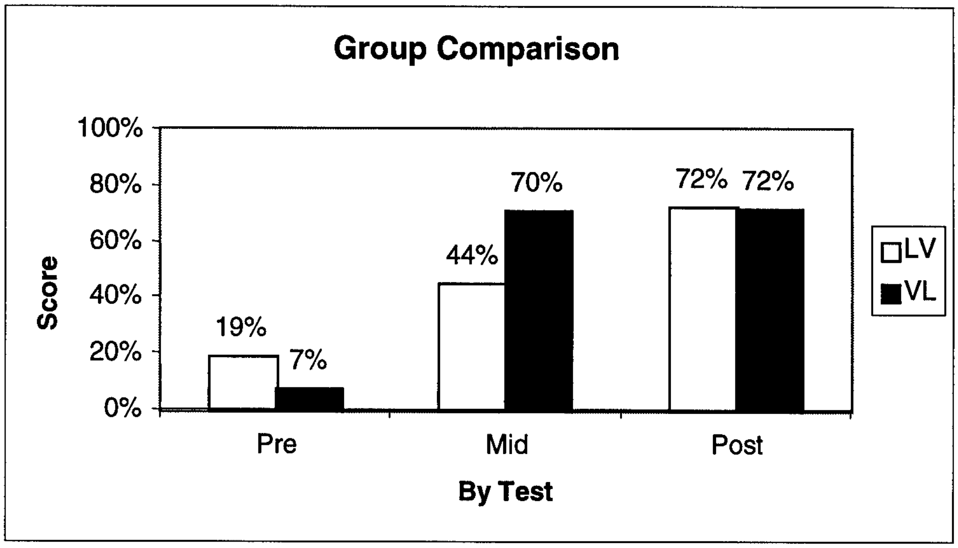


Figure 4.8. Comparison of Group Pre-Test, Mid-Test and Post-Test Performance

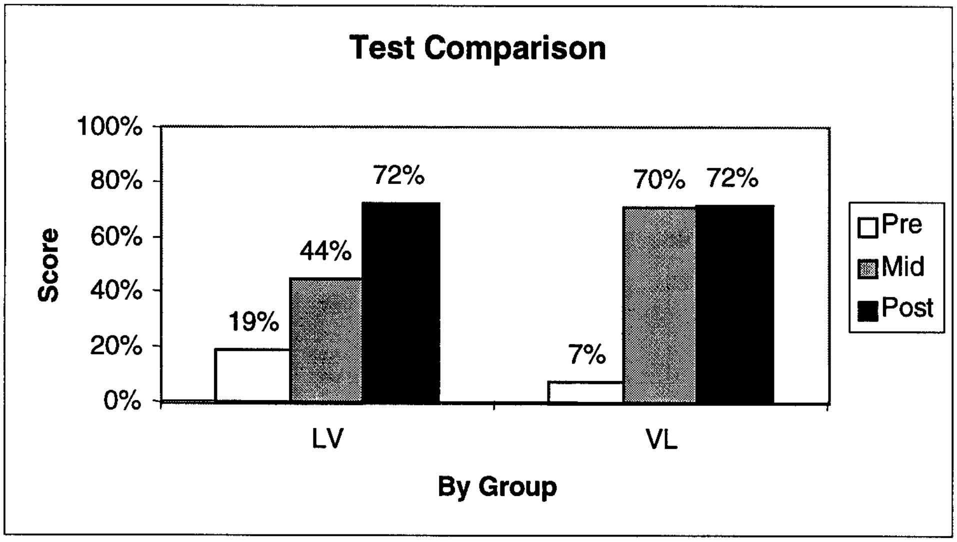


Figure 4.9. Comparison of Group Performance Change in Tests

Results

Examining the results depicted in Figure 4.8, we see that both groups were relatively unfamiliar with the algorithms based on their pre-test averages (7% for the VL group and 19% for the LV group). The mid-test results indicate the VL group learned more than the LV group:

following the session with HalVis, the VL group average score was 70% compared to 44% for the LV group after the lecture, representing an improvement of 63% for the VL group and just 25% for the LV group. Hence, despite having less prior knowledge about the algorithms, in the mid-test, the VL group, after interacting with the algorithm visualizations, significantly outperformed the LV group that received a classroom lecture. The improvement in *additional* knowledge gained by the VL group from the subsequent lecture session was marginal (scores rose from 70% on the mid-test to 72% on the post-test), whereas the visualization helped the LV group catch up with the VL group by the time of the post-test, with their scores rising from 44% on the mid-test to 72% on the post-test.

Another view of the results is depicted in Figure 4.9, showing the improvement in each group's performance as measured by test following the visualization and lecture treatments. Again, large increases in knowledge as measured by test performance occurred in both groups as a result of interacting with algorithm visualizations. The LV group experienced a 25% improvement in average score after receiving the lecture, then improved another 28% following the AV interaction. The VL group experienced a 63% improvement after the visualization, while the following lecture provided an improvement of only 2%. Another interesting observation is that both groups eventually reached similar levels (72%) of performance. The LV group showed steady increases following the lecture and then the visualization. The VL group showed a substantial increase resulting from visualization alone, to which the lecture session did not add considerably.

These results appear to indicate that interactive hypermedia visualizations produce greater gains when prior knowledge is limited, and that a conventional lecture does not appear to provide significant additional learning benefits. On the other hand, students with prior knowledge gained by conventional instruction can *also* benefit from hypermedia visualizations.

Statistical support for these conclusions is provided in Tables 4.7, 4.8 and 4.9. Table 4.7 shows the between-group statistical results. The post-test results show that the order in which lectures and visualizations are presented does not appear to make a difference, as both groups scored approximately the same (72% for the VL group and 72% for the LV group, (F(1,19)=0.001, p<0.97)). While these post-test results are not significantly in favor of either group, Table 4.8 shows that the mid-test performance results are significantly in favor of the group that interacted with the visualization first (VL group) compared to the group (LV group) receiving the lecture (F(1,19)=11.87, p<0.033).

	Pre-test	Mid-test	Post-test
LV	19%	44%	72%
VL	7%	70%	72%
F(1,19)	1.78	5.3	0.001
p	p < 0.198	p < 0.033	p < 0.97

Table 4.7. Experiment III Statistical Summary: Between Groups (Overall Performance)

Table 4.8 shows that the score improvement from the pre-test to the mid-test favored the group receiving the visualization, which for this first phase of the experiment was the VL group (F(1,19)=26.89, p<0.0001). Additionally, the score improvement from the mid-test to the post-test also favored the group receiving the visualization, which for this second phase of the experiment was the LV group (F(1,19)=11.87, p<0.003). These results suggest that the HalVis visualization leads to a significant improvement in learning.

	Pre-to-Mid-Test Improvement	Mid-to-Post-Test Improvement	Overall Improvement (Pre-to-Post)
LV	25%	28%	53%
VL	63%	2%	65%
F(1,19)	26.89	11.87	1.16
p	p < 0.00001	p < 0.003	p < 0.29

Table 4.8. Experiment III Statistical Summary: Between Groups (Improvement)

Specific test-to-test improvements for the two groups are shown in Table 4.9. Here, the only result that did not yield statistical significance was the improvement in performance for the VL group, when they went from having interacted with the visualization to receiving the lecture. The lecture added only 2% to the overall performance. But this effect is not statistically significant, indicating that the visualization prior to the lecture must have been the primary cause of overall improvement. In all other cases, both the lecture and the visualization resulted in significant improvements in knowledge.

		pre-to-mid	mid-to-post	pre-to-post
LV	F(1,19)	6.35	5.037	21.62
	P	p < 0.023	p < 0.039	p < 0.0002
VL	F(1,19)	43.97	0.016	73.44
	p	p < 0.00002	p < 0.899	p < 0.0000004

Table 4.9. Experiment III Statistical Summary: Within Group (Improvement)

These results are summarized as box plots in Figure 4.10. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group. The distribution of scores is interesting. The distribution in the LV group appears to be similar in each of the three tests, with quartiles that are approximately equidistant from a well-centered mean. The distributions are not as uniform across the tests in the VL group. The pre-test is tightly clustered and positively skewed. The mid-test following the visualization shows a wide distribution of scores, which then tightens up following the lecture to a more normal-appearing distribution. This seems to indicate that individuals may not have uniformly benefited from the visualization to the same extent, and that this dispersion was somewhat remedied by the lecture which presumably benefited those who did not significantly gain from the visualization. These individual differences are masked by the 2% overall improvement of the VL group from the lecture. This argues for hypermedia algorithm visualizations supplementing, rather than replacing, traditional instruction.

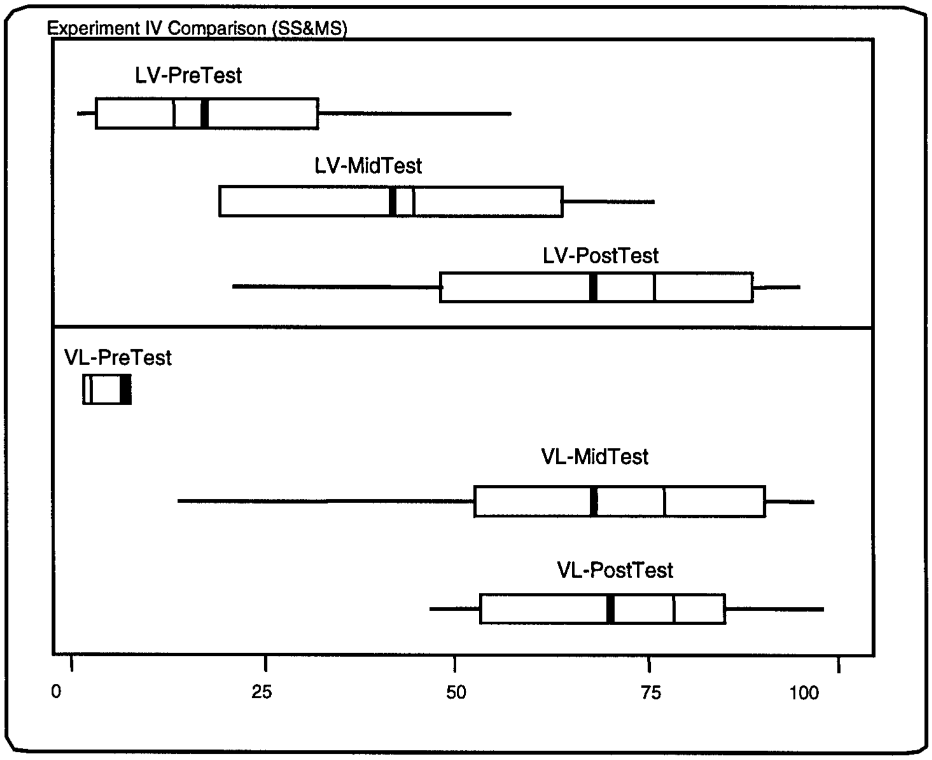


Figure 4.10. Experiment III Box Plots

Discussion

This experiment shed interesting insights on the three hypotheses that we set out to investigate. First, the results support the hypothesis that learning by visualization is more effective than learning by lecture alone. The mid-test results captured the improvement caused by visualization in the VL group and lecture in the LV group. The results of this first phase were statistically significant (from table 4.7: F(1,19)=5.3, p<0.03).

Second, these results suggest that the *combination* of learning by visualization and lecture leads to improvements over learning by visualization or lecture alone, as both groups' scores improved with each added phase/treatment. However, the improvement gains after groups received the second treatment were not uniform. The improvement observed by adding the visualization to the LV group was large (28%) and statistically significant (table 4.9: F(1,19)=5.037, p<0.039). However, the improvement observed in adding the lecture to the VL group was only 2% and not significant (from Table 4.9: F(1,19)=0.016, p<0.89).

Third, as to the impact of the presentation order in the *final* outcome, these results suggest that order does not matter in terms of overall performance. After completing *both* phases, the two groups performed at about the same level (~72%, F(1,19)=0.001, p<0.97). However, as noted above, the greatest jump in performance of all the pairings was the 63% improvement observed in the VL group following the visualization. It should be noted that in this experiment, the lecture and the visualization were fairly redundant in their content, with both approaches giving similar basic information about the algorithm including general and detailed descriptions and examples of the algorithm in action. This insight would suggest that instructors that couple classroom lecture with hypermedia visualization can and should alter the depth and content of the lecture to be less redundant with the basic notions covered effectively by visualization software. Preceding the lecture with a visualization lab should free the instructor from spending valuable lecture time covering the algorithm basics to allow delving into deeper material about the algorithm such as performance issues, design trade-offs, boundary cases and so on.

Naturally, there are factors that could have influenced these results. The most obvious one is the quality of the lecturer. To address this issue, we requested the services of a highly rated (by students) professor who had taught introductory computer science courses several times. Dr. Dean Hendrix is known to be an excellent lecturer. We developed the experiment so that all the participants would attend a single lecture, to avoid the possibility that one lecture session could have covered the material in a different way than another. We also chose to have a live lecture instead of a videotaped one, to allow teacher-student interaction more typical of a classroom environment.

One factor we did not control for was that some students might have read additional textual materials between the phases of the experiment. To reduce this possibility, we intentionally used the SelectionSort algorithm which was not mentioned in the course textbook. MergeSort was however covered in the textbook. We asked the students how much time they spent reading the text, if at all. Only four indicated that they had read the text, and the average time indicated was 10 minutes. We did not detect any significant differences in performance between the algorithm described in the textbook and the algorithm not covered in the textbook.

4.2.4. EXPERIMENT IV: COMPARISON OF HALVIS AND LEARNING FROM TEXT & CONVENTIONAL ANIMATION

This experiment was designed to compare our algorithm visualization framework to the environment typified by previous empirical studies which measured learning after participants viewed a textual description of an algorithm and then viewed an animation. The algorithm we selected was Dijkstra's Shortest Path algorithm. It is conceptually difficult, and is different in style (a graph algorithm) from all algorithms used in the previous four experiments (sorting algorithms). Our hypothesis was that a properly designed hypermedia visualization would be a more effective tool for learning algorithms than a conventional algorithm animation combined with text.

Subjects

This experiment involved 38 undergraduate computer science students enrolled in a third year algorithm analysis course at Auburn University. Like previous experiments, participants were ranked based on their course performance up through mid-term grades, GPA, and ACT/SAT scores, and randomly assigned to create two matched groups: a "Tango" group and a "HalVis" group. Students were given extra credit for participating. Twenty students in the Tango group and eighteen students in the HalVis group completed the experiment. There were two students assigned to the HalVis group that did not complete the experiment and while their results are not included, their absence did not appear to bias the groups as will be seen in the section discussing the results. The average GPA and ACT scores for the HalVis group were 3.20 and 27.4, respectively. The average GPA and ACT scores for the Tango group were 3.13 and 28.1, respectively.

Materials

Tango Algorithm Animation: One of the most mature and widely available algorithm animation platforms is the Tango software suite developed by Dr. John Stasko (Stasko, 1990; Stasko, 1997), publicly available from Georgia Institute of Technology at ftp.cc.gatech.edu:/pub/people/stasko. The Tango software distribution executes on Windows95 systems and includes a library of animated algorithms. Three researchers in our group carefully examined eight animations of the Shortest Path algorithm available in this distribution, and selected one that appeared to be the most complete, easiest to understand, and which most closely matched the features of the HalVis system (i.e., use of multiple representations, contextual descriptions and animated pseudocode).

Hypermedia Algorithm Visualization: A visualization for the Shortest Path algorithm was built and provided in HalVis.

Handout: The Tango group received a supplement to help them learn the Shortest Path algorithm, which consisted of a five page extract from their textbook (Weiss, 1992). This was done to simulate the conditions under which Tango-style animations were previously experimentally evaluated (Lawrence et al., 1994), when the visualization groups received textual supplements in addition to the visualization.

Test Questions: A pre-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithm. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithms and the post-test results measured changes resulting from the experimental conditions.

Procedure

As with the previous experiments, we timed the experiment to precede the course lectures that covered the subject of graph algorithms. Towards the end of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the Shortest Path algorithm. In addition to providing a baseline against which to compare subsequent changes, the pre-test results also helped us verify that the two groups were evenly balanced.

In the following week, both groups met in the same public computer laboratory on campus, but at different times. Both groups received a brief, navigation-only orientation to the software they were to use, then were assigned to a computer and instructed to interact with the visualization until they felt they understood the algorithm. The computers were Pentium-class systems with 15 inch color monitors.

Members of the HalVis group were not given any text material to study, nor had they been exposed to the algorithm earlier in class. There was no time limit for either group, so when each subject indicated he/she was done, he/she was given a post-test to measure knowledge improvement. No student in the HalVis group took more than 90 minutes for the entire experiment.

Members of the Tango group received an extract from their textbook describing the Shortest Path algorithm and were assigned to a computer to interact with the animation. They were not provided with any other information, nor had they been exposed to this algorithm during class lectures. When they indicated they understood the material, they were given a post-test and allowed to leave. No student in the text group took more than 60 minutes for the entire experiment.

Results

Examining the results shown in Table 4.10 below, we see that both groups were relatively unfamiliar with the algorithm based on the pre-test averages (23% for the Tango group and 22% for the HalVis group). The post-test results show that the HalVis group's scores improved to 89% while the Tango group improved to 71% (F(1,37)=12.75, p<0.001).

The results are also summarized as box plots in Figure 4.11. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group. The distribution of scores is interesting. Generally, the HalVis pre-test score distribution is tight and normal looking, but there are two outliers (shown as black dots) that scored very well, indicating prior knowledge of the Shortest Path algorithm. The distribution of the HalVis group's post-test scores indicates a tighter clustering, with one outlier at 69% (shown as a black dot), compared to the post-test score distribution of the Tango group. The post-test score distribution of the Tango group also shows (as a black dot) the presence of one outlier who scored extremely poorly.

Statistical Summary			
	Pre-test	Post-test	Improvement (raw)
Tango	23%	71%	48%
HalVis	22%	89%	68%
F(1,37)	0.01	12.75	4.79
P	p<0.91	p<0.001	p<0.035

Table 4.10. Experiment IV Statistical Summary

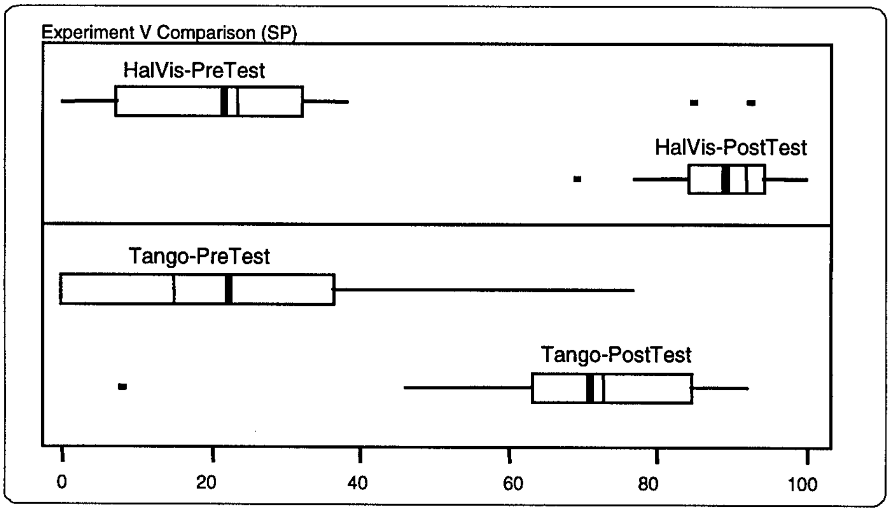


Figure 4.11. Experiment IV Box Plots

Discussion

This experiment compared our HalVis algorithm visualization framework with an animation generated from a popular algorithm animation package. We chose Tango since Tango and its predecessors form a set of algorithm animations that have not only been extensively described in the literature (Stasko, 1990; Stasko, 1997) but also have been the subjects of significant experimental analyses reported in the literature (Byrne et al., 1996; Kehoe & Stasko, 1996; Lawrence et al., 1994; Stasko, 1997; Stasko et al., 1993). The Tango animation was well-paced, showed good use of color to highlight algorithm actions, included a brief textual introduction, contained contextual explanations and provided the student with the algorithm's pseudocode, whose lines were highlighted synchronously as the animation proceeded. We supplemented this animation with pages describing the algorithm from the course textbook (Weiss, 1993) in order to provide the student with as much information about the algorithm as possible in a standalone setting, and to replicate as closely as possible the conditions of algorithm animation experiments reported by other researchers. The results indicate that our framework for hypermedia algorithm visualization design is more effective than an algorithm animation representative of current approaches.

How did our Tango group compare to previous experiments reported in the literature that used Tango under similar circumstances? One experiment reported in (Lawrence et al., 1994) compared groups using the Tango animation system in conjunction with a lecture and active or passive laboratory assignments to learn Kruskal's Spanning Tree algorithm. In their study, one group's conditions closely matched that of our experiment: the group that received a prepared lecture (roughly corresponding with our group that received text) and a Tango/Polka animation that contained contextual descriptions but did not permit data modification (passive laboratory).

The comparison is shown in Table 4.11. While there are many factors that render an exact comparison impossible, such as the different algorithms used, the general results appear to suggest that our experimental group using Tango performed comparably with their corresponding experimental group.

Comparison Summary		
	Pre-test	Post-test
Lawrence et al, 1994	N/A	75%
HalVis	22%	89%
Tango	23%	71%

Table 4.11. Comparison of Results with Lawrence et al. (1994)

As with the other experiments, there are several factors potentially influencing the results. The high scores of the HalVis group in the post-test indicates a possible ceiling effect that might have suppressed a greater separation between the groups. Another factor could be the quality of the Tango animation. We attempted to address this by selecting the best and most comparable animation from the eight supplied with the Tango distribution files. We believe that we chose a representative animation. Nevertheless, it is possible that a different Tango animation might have led to different results for the Tango group. A similar argument could be made for the text (photocopied pages from the course textbook) that was provided to the Tango group. The textbook has been used for several years at Auburn, and it is considered to be a good choice for an algorithm analysis course.

4.3. GENERAL DISCUSSION

4.3.1. COMPREHENSION

Our experiments were designed to test the effectiveness of a novel framework for hypermedia algorithm visualizations, a framework that embeds animations in a context and knowledge providing hypermedia environment, against traditional methods of instruction. Comparisons with learning from a textbook, learning by reading carefully constructed textual explanations and solving problems, learning from lectures, and learning by interacting with an algorithm animation representative of extant research on the topic, all indicated the significant advantages of the HalVis framework from a self-directed learning perspective. Results from four of the five experiments were statistically significant for different levels of students and different kinds of algorithms. The performance of students in Experiment II showed a positive but non-significant trend favoring visualizations, suggesting that a hypermedia visualization can be as effective as learning from a well-crafted mixed mode explanation (text + diagrams) combined with problem solving.

4.3.2. SATISFACTION

In addition to measuring quantitative performance, we captured a log of user activity and gathered subjective evaluations from participants to gain insight from their perception of the system and the visualization experience. We found some of the comments and suggestions useful in helping to shape and modify the appearance and navigational structure of the system. Preliminary analysis of the user logs showed that students executed the detailed-level animations an average of three times and viewed the populated-level animation an average of four times. This was an interesting finding, since we anticipated the students would run the detailed-level animations more than this, and the populated animation less. We found that over half of the students experimented with entering their own data for at least one execution of the algorithm at the detailed level, and every student elected to make performance predictions during at least one execution at the populated level.

Students overwhelmingly (87%) enjoyed interacting with the visualization and 92% indicated they would prefer learning by visualization if given the choice in the future. Extracts of some of the positive comments include: “Animation made it easier to understand ... more interesting ... much better than reading a book ...”. Students liked “...being able to rerun the animation until you understand ... step by step...,” as well as the questions and ability to make predictions. One student said, “ I liked being able to see [efficiency]--I didn't realize there was that much difference.” And another noted, “I liked the fact that the voice explains what is going on while the animation is going on." There were some negative comments too. Students mentioned constructive things like: “It could use more sound,” and “The reoccurring text was annoying.” Interestingly, there were conflicting comments like one who said, “It was too detailed,” contrasted to another that observed, “It was too simplistic.”

These results parallel virtually every study we have read. Detailed figures are available in Appendix C for those interested in gleaning more from these subjective comments.

4.3.3. RETENTION

Another measure of success for educational systems is increased information retention over time. Many researchers speculate that hypermedia systems and animations are information-rich, engaging, and contain a wealth of content that may enhance long term retention. Research has shown that people remember minute details contained in pictures much more accurately than information described in text (Mayer, 1989), but very little research has been reported about the long term effects on retention of viewing an animation. Our belief is that several facets of the

HalVis framework can enhance long term retention, particularly the use of animation in the Detailed View and the use of a real-world analogy in the Conceptual View to introduce the algorithm. Yet we recognize the many mitigating factors that make quantitative empirical research difficult. There are many confounding variables that intervene in the time between subjects viewing an animation and being tested weeks, months or years later to measure long-term retention. Nonetheless, we desired to gather some retention data to explore this area.

One year after Experiments I through IV, we found twelve subjects that had been involved in learning the MergeSort algorithm who volunteered to participate in a retention study. Six had learned the MergeSort algorithm as a member of a Text-only group, and six had learned the algorithm using HalVis. They took a brief retention test involving questions very similar to ones they had seen on pre- and post-tests a year earlier to measure their recall of this algorithm. The subjects from the visualization group did 66% and the subjects from the Text group averaged 60% on the retention post-test. The results were not statistically significant, but the experiment was not intended to produce such formal results because of the many intervening and confounding factors that made what were once matched groups now unpredictably different. We could not control nor account for experiences they might have had over the year between the post-test for the initial experiment and the retention post-test that could have reinforced or obfuscated the concepts they had learned a year earlier. However, the results do reflect a slightly positive trend favoring the animation condition, and constitute one of the first quantifiable, albeit informal, retention studies in algorithm visualization research. Perhaps a more significant indicator of retention is the fact that five of the six visualization students, when asked what they remembered most about the system, specifically recalled the analogy.

4.3.4. SPEED

Another measure of success that researchers attribute to hypermedia systems is that they have the potential to help students learn the same information (or more) in less time. Our experience has been just the opposite. We did not place restrictions on the amount of time students could spend in any facet of the experiments described in this section, unlike previous researchers who limited the amount of time subjects could spend interacting with the learning media involved in their research. We did not keep strict measures on the amount of time by individual, but we did keep anecdotal figures as groups completed the experiments. We observed that the Text groups of Experiment I took less time (~20 minutes) than the AV groups (~35 minutes); in Experiment II the Text group took a bit longer (~30 minutes) because of the exercises. The lecture in Experiment III took about the same amount of time as the AV session (~60 minutes). While this data could be taken as a strike against hypermedia, we suggest a different interpretation: subjects found the hypermedia presentation more engaging and motivating, and either didn't notice the time passing or found the added time rewarding enough to continue anyway. We informally asked several students how much time they used the HalVis software, and in every case, their recall was at least 15 minutes less than the time they actually spent using the system. Hence, students spent more time interacting with the visualizations, but didn't notice. Most instructors would declare any teaching approach a success that motivates students to spend additional time with a subject because they *want* to, not because they *need* to.

4.4. CRITICISMS

The results reported in this chapter are among the first to provide statistical support of the use of animations to enhance learning algorithms. This is encouraging and exciting, but should be kept in perspective. In this section, we cast a critical eye to our work and present some of the issues that must be considered to keep our results in proper context, and to help guide future research.

- Design bias: The same group designed the HalVis system, the learning objectives, the pre- and post-tests and performed the grading. While we attempted to be fair and unbiased at all stages of research, we recognize that using independent proctors, graders and designers could have helped address this criticism.
- Inauthentic setting: The setting for conducting the experiments was not 'authentic,' which means our results might not be replicated if HalVis was used to supplement or replace typical courseware. Even though these experiments were conducted in classrooms and labs in the Computer Science department, all the subjects involved in our experiments understood they were participating in research. The sequence of grouping the students, issuing demographic surveys, pre-tests, post-tests and satisfaction surveys is not consistent with typical courseware. It is likely that many subjects allowed extra tolerance, patience and effort that might not be the case in day-to-day use. Of course, the opposite is also possible, which is that the students understood that only their participation and not their results was being considered for grading and credit, and performed only the minimum necessary to get by. At any rate, the research setting did not represent the envisioned used of algorithm visualization as a teaching tool, and the results reported here might not be replicated in authentic use.

- Group Size: Generally, as group sizes increase in statistical research, the effects of variation are decreased and a more representative population average can be reported. Our work involved groups with as few as nine and as many as twenty subjects, which are smaller than ideal for making statistical comparisons. Having more subjects would have added to the confidence and reliability of the results.
- Limited Algorithmic Domain: Our experiments involved a limited number of algorithms that covered selectected sorting, merging and graphing problems. In these limited and relatively simple domains, the HalVis framework demonstrated encouraging results. However, it is fair to ask if our results would be replicated for other, possibly more complicated algorithms.

4.5. SUMMARY OF EMPIRICAL COMPARISONS

The following summarizes conclusions from the five experiments discussed in this chapter:

- Advanced as well as novice students perform better in answering conceptual and procedural questions about certain fundamental algorithms after interacting with the HalVis hypermedia algorithm visualization framework than after studying explanations found in typical textbooks on algorithms.
- The HalVis hypermedia algorithm visualizations appear to be as effective a learning aid for novice students to learn about selected algorithms as learning from carefully crafted textual and diagrammatic explanations combined with solving a set of problems.
- Novice students gain more knowledge after interacting with the HalVis hypermedia algorithm visualization framework than after hearing a typical classroom lecture. Furthermore, lecture and visualizations supplementing each other provide the best learning scenario, and the order of presentation does not seem to influence the overall extent of learning.

- It appears that interactive hypermedia algorithm visualizations are more effective when prior knowledge is limited. However, students with prior knowledge from conventional instruction *also* derive a significant learning benefit from algorithm visualizations.
- Individual differences in learning from algorithm visualizations exist. These differences may be compensated by the use of multiple modes of instruction. This argues for hypermedia algorithm visualizations supplementing, rather than replacing, traditional instructional methods.
- Finally, the framework for algorithm visualization design that HalVis exemplifies appears to be a more effective teaching tool than previous algorithm animation designs.

The general conclusion is that interactive hypermedia algorithm visualizations modeled after the HalVis framework (a system in which animations are embedded within a knowledge and context providing hypermedia environment) can provide significant benefits to learners as an educational medium for self-directed and self-paced learning, either by itself or even more so in combination with other instructional media.

There are a number of possible reasons for this. First, in comparison with previous animation systems that only presented animations with some textual feedback, HalVis allows the student to learn incrementally by starting from a real world analogy and transitioning to the algorithm itself. Second, the hypermedia structure allows a student access to fundamental building blocks of algorithmic knowledge in-context and on-demand. Third, a learning objective-based design approach and the hypermedia structure surrounding animations have allowed us to divide dynamic information into manageable and meaningful pieces, and present each piece using animation chunks. This makes it easier for students to pause and reflect, repeat, or access other relevant information through hyperlinks while watching animations. Furthermore, animation chunks are presented in synchrony with other representations in other media. These novel features, we believe, result in the dynamic information being conveyed better in context, and therefore in a more comprehensible fashion. Fourth, rather than providing just one view of an animation as has been the typical approach, HalVis presents three kinds of animations (analogical, micro-level and macro-level), so that the macro behavior is seen after the micro behavior is seen and understood, both following an analogical introduction to the algorithm. Fifth, our framework allows students to actively engage themselves in the visualization by changing data inputs, making performance predictions, and reflecting on questions that pop up in context, all contributing to better learning.

The following chapters discuss a series of ablation experiments designed to measure the differential contributions to learning of these various features and subsequent refinements to the HalVis framework.

5. ABLATION STUDIES (WHAT MADE A DIFFERENCE?)

The previous chapter focused on comparing a richly endowed hypermedia system with other learning media to validate the hypothesis that animation-embedded software could lead to the effective learning results that eluded former researchers. This chapter describes a series of experiments that were conducted to focus on the specific components that lead to the successful results we observed in Experiments I through IV. Three experiments were devised to probe specific aspects of the HalVis framework by comparing a control group using a fully-enabled version with test groups that had one or more features or views disabled. In Experiment V, we wanted insight on the impact of removing features such as chunking, questioning, and the animated pseudocode. Experiment VI tested the removal of a single view and Experiment VII explored the impact of removing two views. Table 5.1 indicates the features and views that were removed from each group.

	Ablation of Features				Ablation of Views							
	Experiment V Groups				Experiment VI Groups				Experiment VII Groups			
	1	2	3	4	1	2	3	4	1	2	3	4
Conceptual View w/analogy						▨					▨	▨
Detailed View							▨			▨		▨
• Chunking		▨					▨			▨		▨
• Tickler Questions				▨			▨			▨		▨
• Animated pseudocode			▨				▨			▨		▨
• Animation							▨			▨		▨
• User selectable input							▨			▨		▨
• Contextual messages							▨			▨		▨
• Execution variables							▨			▨		▨
Populated View								▨		▨	▨	
• Predictions				▨				▨		▨	▨	
• Animation								▨		▨	▨	
• User selectable input								▨		▨	▨	
Fundamentals Screens												
Description Screen w/pseudocode												
Questions Screen				▨								

Table 5.1. Ablation Experiment Summary

5.1. EXPERIMENT V: ABLATION OF FEATURES

This experiment was designed to isolate selected features from our algorithm visualization framework to measure their respective impact and effectiveness. Of the unique features incorporated in our framework, the three we identified as the most likely to have an impact were chunking, animated pseudocode in the Detailed View, and use of questions. We believed that the contextual explanations and the user's ability to interact with the input data for the animation were also important, but these features have already been empirically identified as significant (by Lawrence, 1993) so we did not pursue testing them. Our hypothesis was that each of these features, when removed from the overall framework, would lead to less effective learning performance by students compared to a control group receiving the full HalVis framework.

Subjects

This experiment involved 40 undergraduate computer science students enrolled in a third year algorithm analysis course at Auburn University. Like the experiments described in the previous section, participants were ranked based on their course performance up through mid-term grades, GPA, and ACT/SAT scores, and creating matched groups by taking the top four subjects from the ranked list and randomly assigning them to one of the four following groups:

- Control Group: This group would interact with a fully enabled version of HalVis.
- No-Chunking Group: This group would interact with a version of HalVis for which the chunking capability was removed.
- No-Pseudocode Group: This group would interact with a version of HalVis for which the pseudocode pane on the Detailed View was removed.
- No-Questions Group: This group would interact with a version of HalVis for which all questions were removed.

Students were given extra credit for participating. The group demographics are shown in Table 5.2.

Group Name	Number of Subjects	ACT Average	GPA Average
Full (control) Group	10	26	3.3
No-Chunking	10	28	3.1
No-Pseudocode	10	28	3.2
No-Questions	10	28	3.1

Table 5.2. Experiment V Demographic Summary

Materials

Each of the groups interacted with the HalVis framework to learn the QuickSort algorithm. Selected features of the HalVis system were disabled or removed to facilitate each of the experimental groups. The Control group received a version of HalVis with all features enabled. The No-Chunking group interacted with a version of HalVis for which the chunking feature was disabled in the Detailed View. This meant the animation would proceed to completion without any pauses. The user could control the speed of the animation, however. The No-Animated Pseudocode Group interacted with a version of HalVis in which the pseudocode window was removed in the Detailed View. The No-Questions Group interacted with a version of HalVis in which all forms of tickler and feedback questions were disabled. None of the groups received any additional handout information, just the version of HalVis associated with their experimental group.

A pre-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithm. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithms and the post-test results measured changes resulting from the experimental conditions.

Procedure

As with the previous experiments, we timed the experiment to precede the course lectures that covered the subject of sorting algorithms. Towards the end of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the QuickSort algorithm. In addition to providing a baseline against which to compare subsequent changes, the pre-test results also helped us verify that the four groups were evenly balanced.

In the following week, the groups met in the same public computer laboratory on campus, but at different times. Each group received a brief, navigation-only orientation to the version of HalVis they were to use, then were assigned to a computer and instructed to interact with the visualization until they felt they understood the algorithm. The computers were Pentium-class systems with 15 inch color monitors. There was no time limit for any group, so when each subject indicated he/she was done, he/she was given a post-test to measure knowledge improvement. No student took more than 60 minutes for the entire experiment.

Results

The average group improvement is shown in Figure 5.1, and summarized in more detailed box plots in Figure 5.2. As we anticipated, the group that showed the greatest improvement was the group that interacted with the full version of HalVis, improving 55%. The group that improved the least was the group that interacted with the version of HalVis for which the Chunking capability was removed; their scores did improve, but only 35% which was the poorest of the four groups. The group that did not have any questions, predictions or ticklers improved 44%, and the group that lacked the window showing the highlighted pseudocode in the

Detailed View improved a surprising 51%. None of these results were statistically significant as shown in Table 5.3. However, there is a positive trend indicating the value of each of the ablated features.

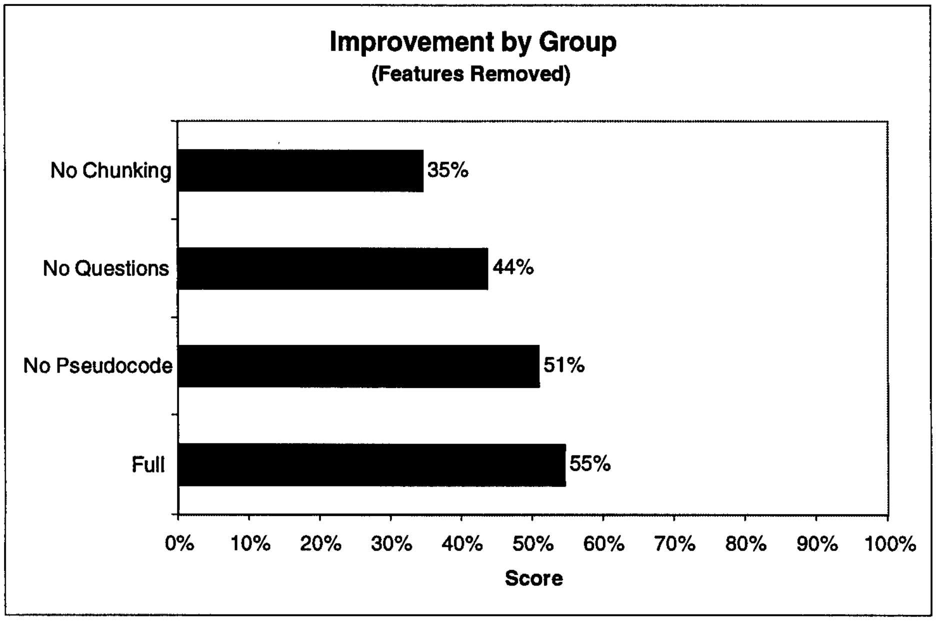


Figure 5.1. Improvement by Group for Experiment V (Features Removed)

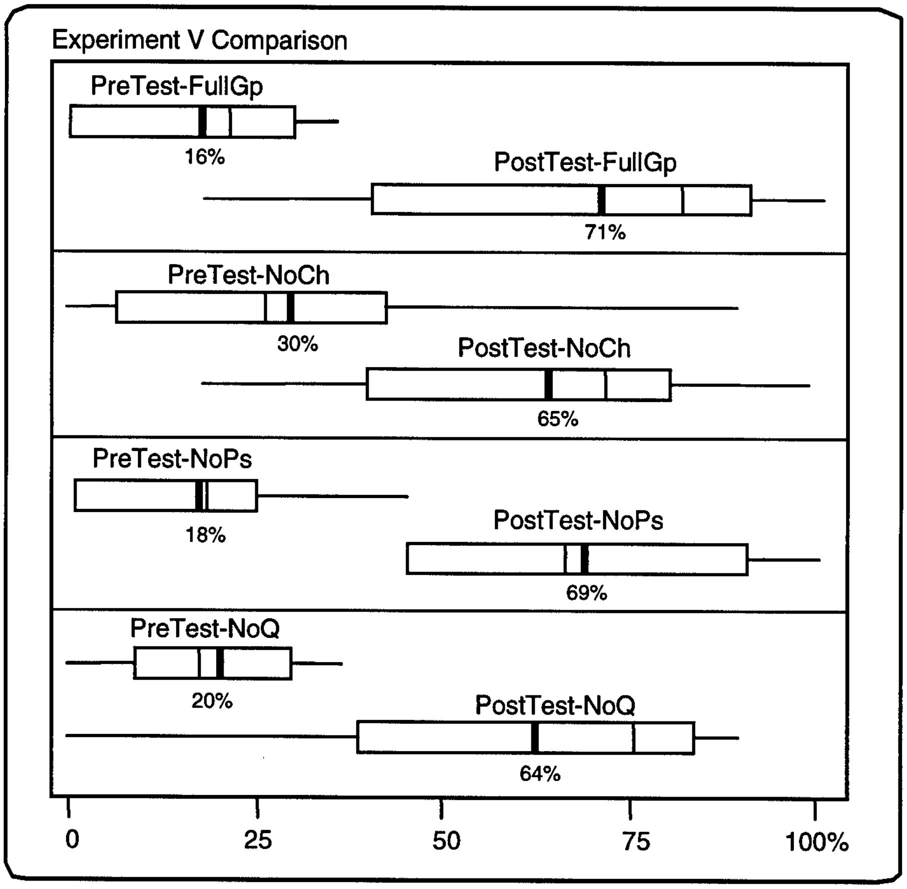


Figure 5.2. Experiment V Box Plots

Statistical Summary		
Group	Average Improvement	Variance
Full	55%	13.8%
NoChunking	35%	5.7%
NoPseudocode	51%	5.0%
NoQuestions	44%	11.0%
F(3,37)	0.88	
p	p < 0.46	

Table 5.3. Statistical Summary for Experiment V (Features Removed)

Discussion

An interesting trend emerges from the navigational and usage data analysis. From the navigational logs captured by the HalVis software, we noted that the No-Pseudocode group not only spent the most time with the system (Figure 5.3), they also navigated back and forth between the Description screen that contained the pseudocode and the Detailed View that presented the visualization with a frequency three times that of the other groups. Several members in the group commented on how inconvenient such a design 'oversight' made using the software. It is also interesting to note that the No-Questions group spent an average of four minutes less interacting with the system, suggesting that the presence of questions throughout the system increases the interaction which is a positive outcome that might account for a portion of the improved performance observed in the control group. Finally, the No-Chunking group spent the least amount of time using the system (26 minutes) and produced the lowest improvement scores on the performance tests (35%). Yet, this group ran the animations about 40% *more* than their counterparts in the other groups. Even though they used the system more, without chunking, they got less out of the system for their efforts. Furthermore, we did not register any complaints or comments from this group that would indicate a sense of giving up out of frustration—these subjects felt they knew the material as well as their counterparts when in fact they did not.

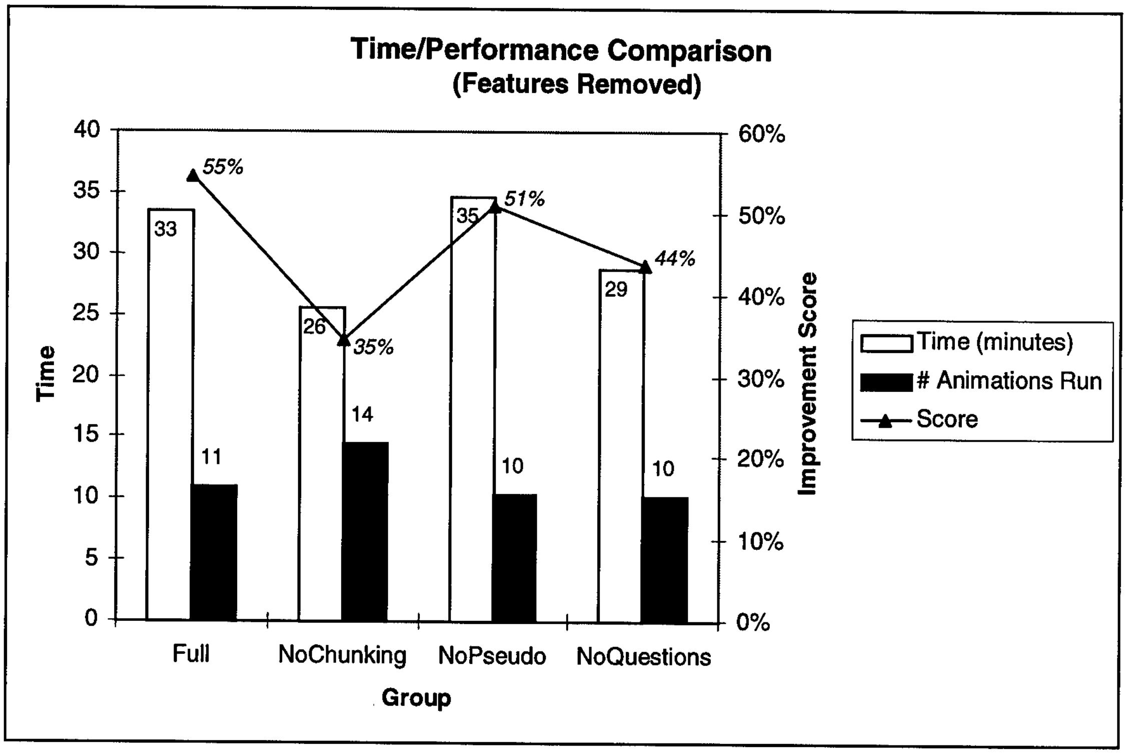


Figure 5.3. Time/Performance Comparison for Experiment V (Features Removed)

Figure 5.4 shows the number of times subjects in each group invoked the animation sequence in the Detailed View and the Populated View. The most notable trend is that the No-Chunking group ran the animations more than all the other groups. Without chunking, the Detailed View animations would generally run to completion in less time because the semantic pauses would not slow the execution. In the groups with Chunking enabled, we observed subjects taking advantage of the pauses to examine, probe and correlate the information in the various panes on the screen. Without the system-inserted pauses, the subjects apparently did not feel they were learning the subtle nuances of the algorithm and attempted to make up for this deficit by rerunning the animation multiple times.

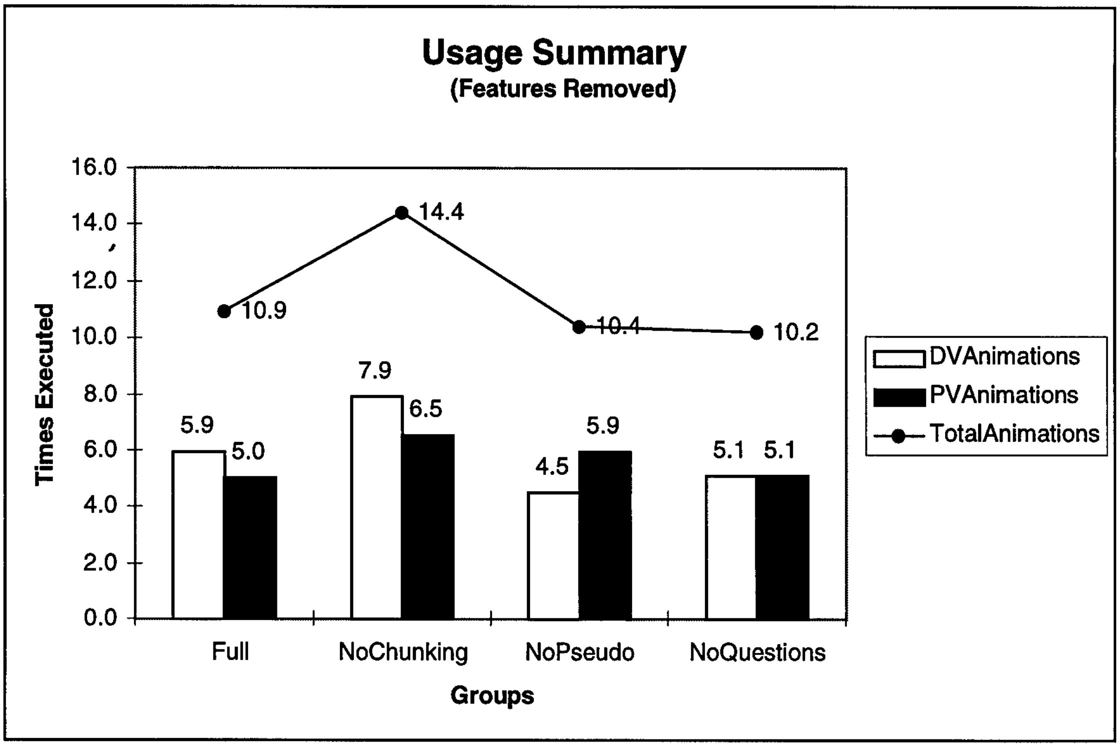


Figure 5.4. Animation Execution Summary for Experiment V

The lack of statistical significance between the improvement levels of the groups was a minor disappointment. We feel that the small group sizes contributed to this outcome. This outcome could also be an indicator that each of the features that were ablated, in a small and measurable way, *contributes* to better performance but that good performance is not exclusively dependant on any *single* feature by itself. In other words, a well designed hypermedia framework allows users to circumvent minor shortcomings by drawing from other related information in the system. We noted this in the No-Chunking group where they ran the animations more times to make up for the lack of pauses, and in the No-Pseudocode group as they navigated back and forth to the screen that possessed the information they felt they needed to understand the algorithm and meet the learning objectives.

5.2. EXPERIMENT VI: ONE VIEW REMOVED

This experiment was designed to elide one of the three views (the Conceptual View, the Detailed View, and the Populated View), leaving the subject with the other two as a means of measuring the contribution each view makes to learning effectiveness. Our hypothesis was that the most important view was the Detailed View and that it would prove to be the most valuable because of the amount of information and interaction it provided. Therefore, the groups that interacted with this view would outperform the group that was denied this view. We believed that the Populated View would follow in significance and felt that the Conceptual View would lag well behind the other two.

Subjects

This experiment involved 32 undergraduate computer science students enrolled in a third year algorithm analysis course at Auburn University. Students were given extra credit for participating. Like the experiments described in the previous section, participants were ranked based on GPA and ACT/SAT scores, and randomly assigned to the following matched groups as indicated below and summarized in Table 5.4:

- Full (control) Group: This group was referred to as the CDP group, which stands for the views they would have access to in the HalVis framework (C=Conceptual View; D=Detailed View; P=Populated View). This group would interact with a fully-enabled version of HalVis.
- No-Conceptual-View Group: This group was referred to as the DP group. This group would receive a version of HalVis in which the Conceptual View was disabled. The key views this group used were the Detailed and Populated Views.

- No-Detailed-View Group: This group was referred to as the CP group. This group would interact with a version of HalVis in which the Detailed View was disabled. The key views this group used were the Conceptual and Populated Views.
- No-Populated-View Group: This group was referred to as the CD group. This group would receive a version of HalVis in which the Populated View was disabled. The key views this group used were the Conceptual and Detailed Views.

Group Name	Number of Subjects	ACT Average	GPA Average
Full (control) Group: "CDP"	7	28.4	2.8
No-Conceptual-View: "DP"	8	28.6	2.9
No-Detailed-View: "CP"	9	28.9	2.8
No-Populated-View: "CD"	8	26.3	3.1

Table 5.4. Experiment VI Demographic Summary

Materials

Each of the groups interacted with the HalVis system to learn the QuickSort algorithm. We constructed a different version of the HalVis system, with the selected view removed, for each of the experimental groups. The CDP group received a version of HalVis with all features enabled. The DP group interacted with a version of HalVis for which the Conceptual View was removed. The CP Group interacted with a version of HalVis in which the Detailed View was removed. The DP Group interacted with a version of HalVis in which the Populated View was removed. None of the groups received any additional handout information, just the version of HalVis associated with their experimental group.

A pre-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithm. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithms and the post-test results measured changes resulting from the experimental conditions.

Groups were exposed to versions of HalVis as indicated in Table 5.5.

	CDP Group (Control)	CD Group	DP Group	CP Group
Conceptual View	Y	Y	-	Y
Detailed View	Y	Y	Y	-
Populated View	Y	-	Y	Y
Description Screen	Y	Y	Y	Y
Questions Module	Y	Y	Y	Y

Table 5.5. Views Available to Groups for Experiment VI (1-View Removed)

Procedures

As with the previous experiments, we scheduled the experiment to precede the course lectures that covered the subject of sort algorithms. Towards the end of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the QuickSort algorithm. In addition to providing a baseline against which to compare subsequent changes, the pre-test results also helped us verify that the groups were evenly balanced.

In the following week, the groups met in the same public computer laboratory on campus, but at different times. Each group received a brief, navigation-only orientation to the version of HalVis they were to use, then were assigned to a computer and instructed to interact with the visualization until they felt they understood the algorithm. The computers were Pentium-class systems with 15 inch color monitors. We imposed no time limits on any group, so when each subject indicated he/she was done, he/she was given a post-test to measure knowledge improvement. No student took more than 60 minutes for the entire experiment.

Results

Figure 5.5 shows the average group improvement, and Figure 5.6 summarized the results as box plots showing the average (the dark vertical bar), the 25-75% quartile range (the box), and the range of values (the lines extending from the quartile box). As expected, the group that received all three HalVis views performed the best. However, examining the results reveals an interesting observation—the impact of the Conceptual View. We expected that the final order of the groups would be CDP, DP, CD, then CP, simply because the Detailed View portrays so much of the inner workings of the algorithm, and the Populated View shows the macro behavior. Our results did not confirm this hypothesis. Instead, contrary to our expectations, the groups that performed best were not the ones exposed to the Detailed View but rather the groups that interacted with the Conceptual View. The improvement for the groups that received the Conceptual View with *any* other view combination more than tripled the improvement of the DP group.

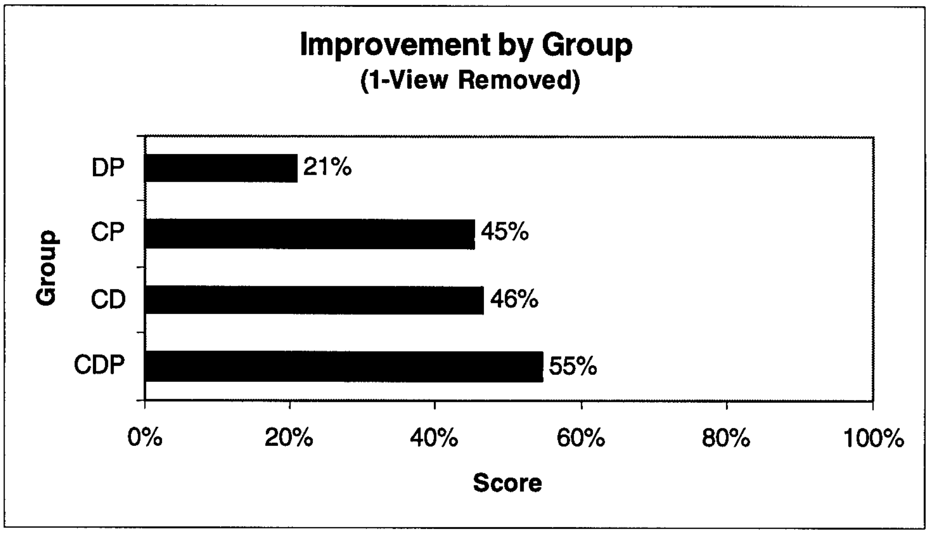


Figure 5.5. Improvement by Group for Experiment VI (1-View Removed)

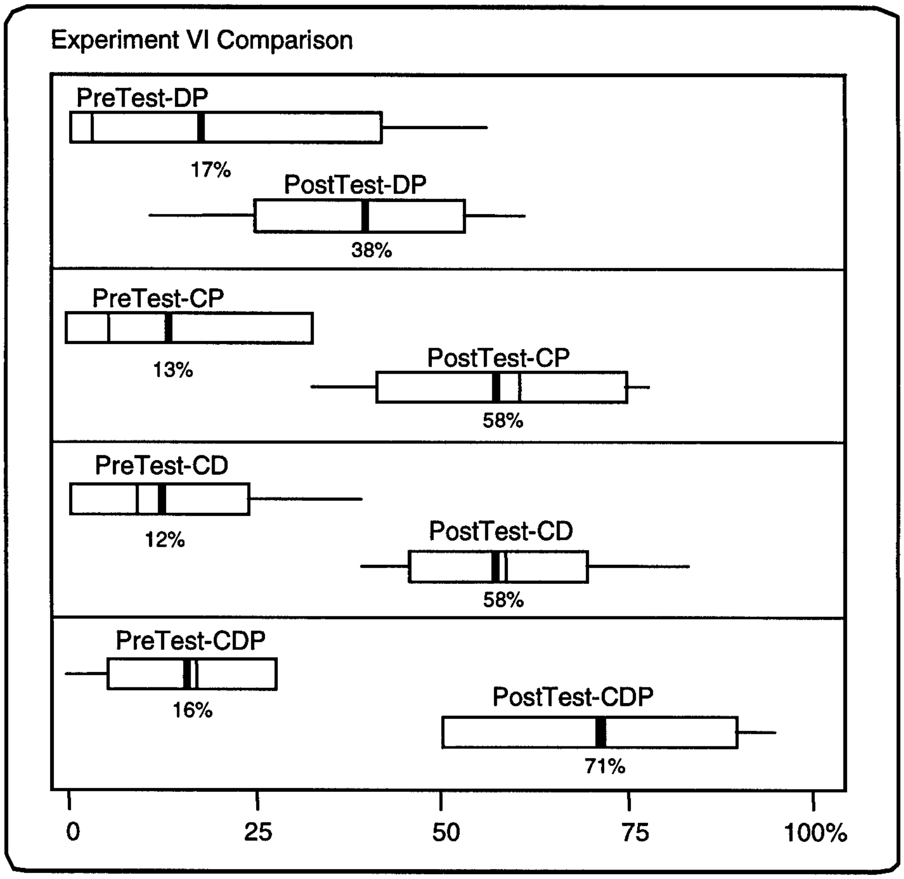


Figure 5.6. Experiment VI Box Plots

Figure 5.7 shows the amount of time subjects interacted with the HalVis software and confirms our expectation that subjects provided with the Detailed View would spend more time with the system, but the differences are not large.

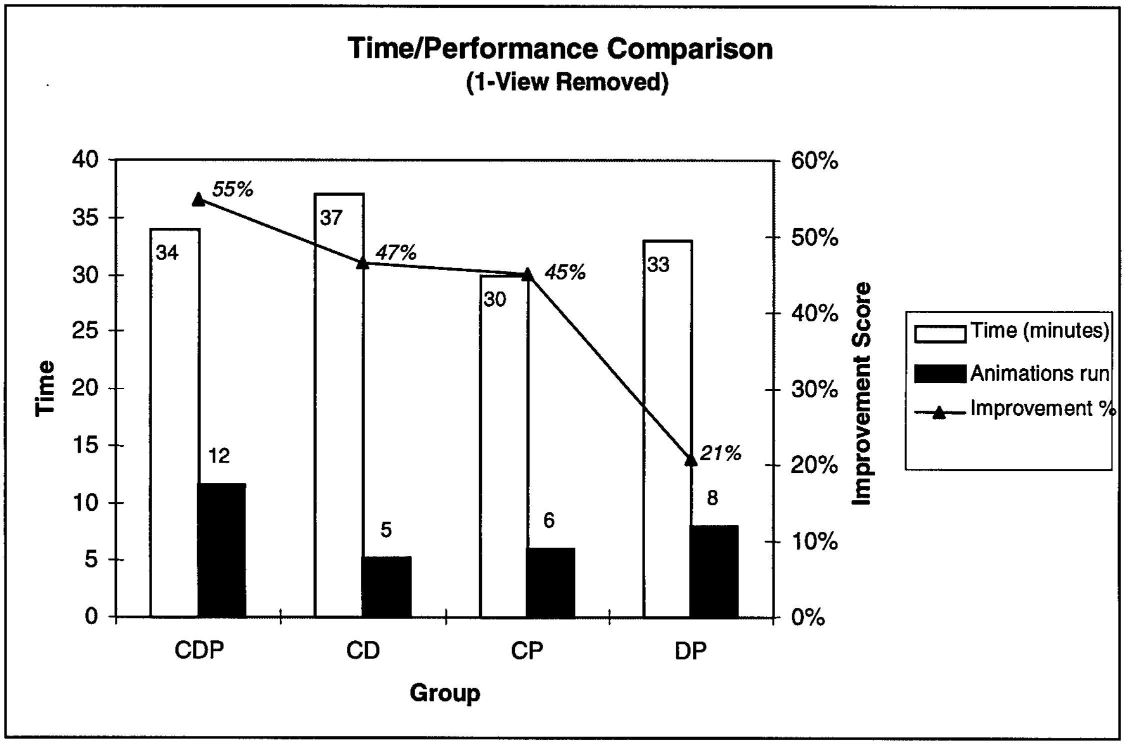
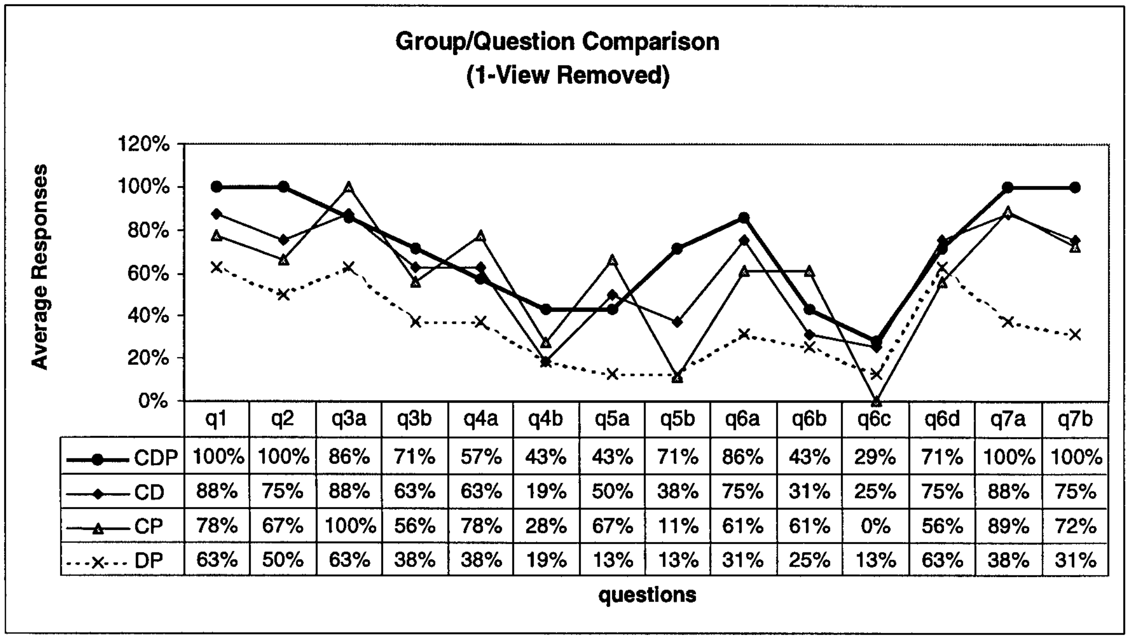


Figure 5.7. Time/Performance Comparison for Experiment VI (1-View Removed)

Figure 5.8 shows the average responses for each of the questions by group. Interestingly, there are four questions in which the CD group outperformed the CDP group. These questions dealt with aggregate performance concepts such as impact of the pivot selection and order of the data on overall results. In none of the questions did the DP group excel. Hence, the differences between the CD and CP group shed some insight into the different ways in which the Detailed View and the Populated View assist learners.



	q1	q2	q3a	q3b	q4a	q4b	q5a	q5b	q6a	q6b	q6c	q6d	q7a	q7b
—●— CDP	100%	100%	86%	71%	57%	43%	43%	71%	86%	43%	29%	71%	100%	100%
—◆— CD	88%	75%	88%	63%	63%	19%	50%	38%	75%	31%	25%	75%	88%	75%
—▲— CP	78%	67%	100%	56%	78%	28%	67%	11%	61%	61%	0%	56%	89%	72%
---×--- DP	63%	50%	63%	38%	38%	19%	13%	13%	31%	25%	13%	63%	38%	31%

Figure 5.8. Experiment VI Comparison of Question Responses

Table 5.6 shows a partial statistical summary of the experiment, presenting data for comparisons between groups that were significant or noteworthy. The data was statistically significant for groups that received the Conceptual View in any combination, compared against the DP group that did not. On the other hand, statistical significance was not detected in the comparison between the control group and the CP group, although there is a slightly positive trend favoring the control group.

Statistical Summary							
Statistically Significant Pairings						NonSignificant	
CDP	55%	**CP**	45%	**CD**	47%	CDP	55%
DP	21%	**DP**	21%	**DP**	21%	CP	45%
F(1,14)	16.71	F(1,16)	8.99	F(1,15)	7.16	F(1,15)	1.45
p	p < 0.001	p	p < 0.01	p	p < 0.018	p	p < 0.25

Table 5.6. Statistical Summary for Experiment VI (1 View Removed)

Discussion

Perhaps the most noteworthy observation from the results of this 2-view ablation study was the effect of the Conceptual View in priming the learning of information presented in subsequent views. The groups that interacted with the Conceptual View in any combination with other views performed better than the group that lacked the Conceptual View. Comparing the difference between the number of times the animations were executed (Figure 5.9), it appears that having the Conceptual View motivated the CDP group to invoke the Detailed View animations nearly twice as many times, and the Populated View animation nearly three times more than the DP group. The impact of the Conceptual View was examined further in the next study that singled each view out.

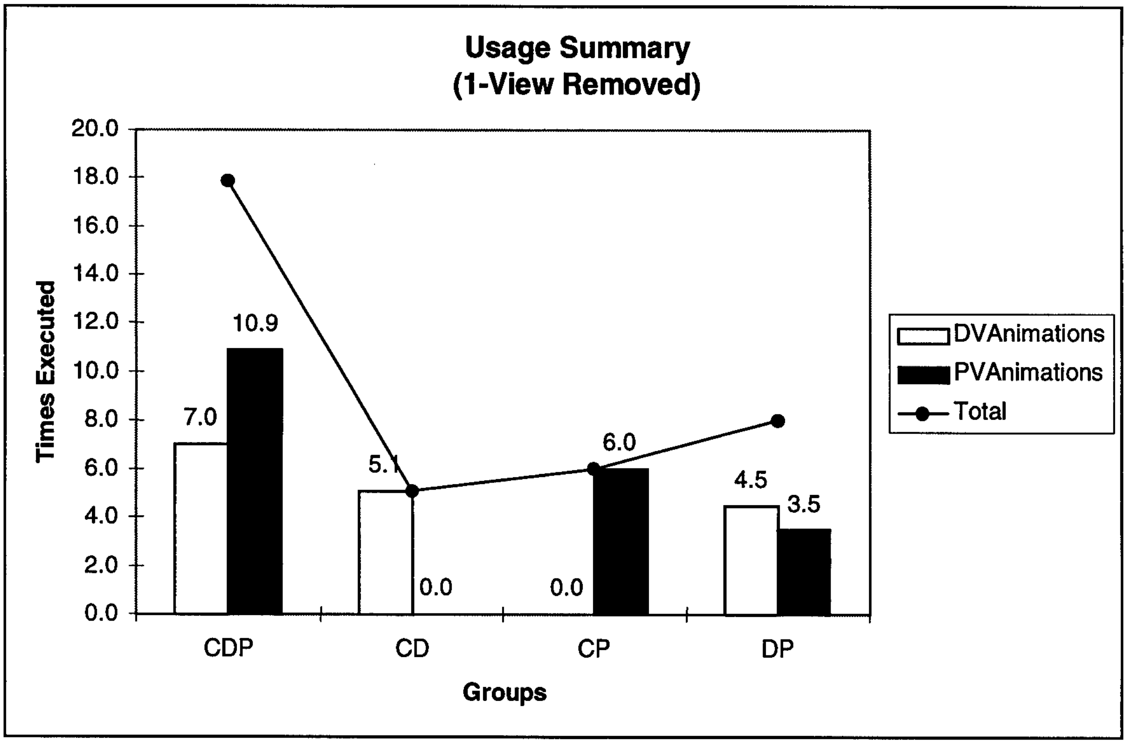


Figure 5.9. Animation Execution Summary for Experiment VI (1-View Removed)

5.3. EXPERIMENT VII: TWO VIEWS REMOVED

This experiment was designed to isolate each of the three views of our algorithm visualization framework to measure their respective impact and effectiveness. Our hypothesis was that each view was important to the framework but that the Detailed View would prove to be the single most valuable because of the amount of information and interaction it provided. We were uncertain how measurable the impact of the other views by themselves would be, since neither the Populated View not the Conceptual View contained the volume or depth of information available in the Detailed View.

Subjects

This experiment involved 27 undergraduate computer science students enrolled in a third year algorithm analysis course at Auburn University. Students were given extra credit for participating. Like the experiments described in the previous section, participants were ranked based on GPA and ACT/SAT scores, and randomly assigned to the following matched groups:

- Full (control) Group: This group received a fully-enabled version of HalVis and are referred to here as the CDP group (C = Conceptual View, D = Detailed View, P = Populated View).
- Conceptual-View-Only Group: This group interacted with a version of HalVis in which both the Detailed and Populated Views were disabled. We refer to them in this section as the CV (Conceptual View) group
- Detailed-View-Only Group: This group used a version of HalVis in which the Conceptual View and the Populated View were inaccessible. They are called the DV group.

- Populated-View-Only Group: This group used a version of HalVis with the Conceptual and Detailed Views removed. They are referred to here as the PV group

Table 5.7 shows the demographics of the groups involved in Experiment VII.

Group Name	Number of Subjects	ACT Average	GPA Average
Full (control) Group: "CDP"	6	28.5	2.8
No-Conceptual-View: "C"	7	27.6	2.8
No-Detailed-View: "D"	7	27.8	2.7
No-Populated-View: "P"	7	25.3	3.0

Table 5.7. Experiment VII Demographic Summary

Materials

Each of the groups interacted with the HalVis framework to learn Dijkstra's Shortest Path algorithm. Four versions of the HalVis system were created. The core information was the same in each version. The difference was in the number of screens that users were allowed to see, as indicated in Table 5.8. In effect, we ablated away two views and only provided subjects with a single view and limited supporting information. All groups were allowed to interact with the Fundamentals module and the description screen that contained a brief description of the algorithm and presented its basic pseudocode. All groups received the feedback questions. Otherwise, the views were disabled as indicated below.

	CDP Group (Control)	CV Group	DV Group	PV Group
Conceptual View	Y	Y	-	-
Detailed View	Y	-	Y	-
Populated View	Y	-	-	Y
Description Screen	Y	Y	Y	Y
Questions Module	Y	Y	Y	Y

Table 5.8. Views accessible to groups for Experiment VII (2-Views Removed)

A pre-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithm. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithms and the post-test measured changes resulting from the experimental conditions.

Procedure

As with the previous experiments, we timed the experiment to precede the course lectures that covered the subject of graph algorithms. Participants were asked to complete a pre-test that measured their prior knowledge about the Shortest Path algorithm. In addition to providing a baseline against which to compare subsequent changes, the pre-test results also helped verify that the four groups were balanced.

In the following week, the groups met in the same public computer laboratory on campus, but at different times. Each group received a brief, navigation-only orientation to the version of HalVis they were to use, then were assigned to a computer and instructed to interact with the visualization until they felt they understood the algorithm. The computers were

Pentium-class systems with 15 inch color monitors. There was no time limit imposed on any group, so when each subject indicated he/she was done, he/she was given a post-test to measure knowledge improvement. No student took more than 60 minutes for the entire experiment.

Results

Figure 5.10 shows the average improvements observed in each of the groups. As expected, the CDP group outperformed the others, followed closely by the DV group. Interestingly, the CV group outperformed the PV group by 21%. Equally interesting is how well the CV group did with the limited amount of information that they received.

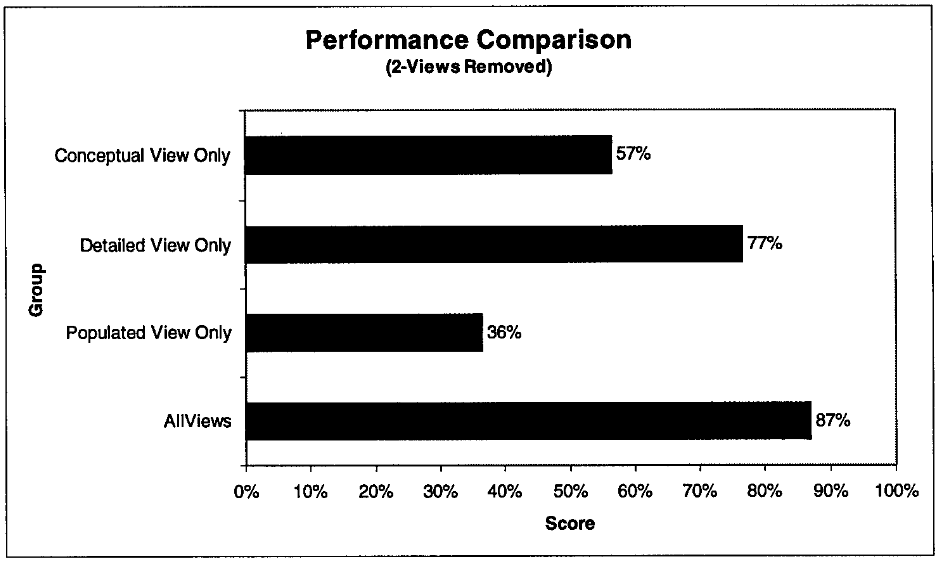


Figure 5.10. Improvement by Group for Experiment VII (2-Views Removed)

The box plots in Figure 5.11 provide deeper details into each group's performance, indicating the average (the dark vertical bar), the 25-75% quartile range (the box) and the total range of values (the lines extending from the quartile boxes). The CDP group shows the tightest and most normal-looking plot, mimicked closely by the DV group plot. The box plots for the CV and PV groups indicate a much wider range of post-test values.

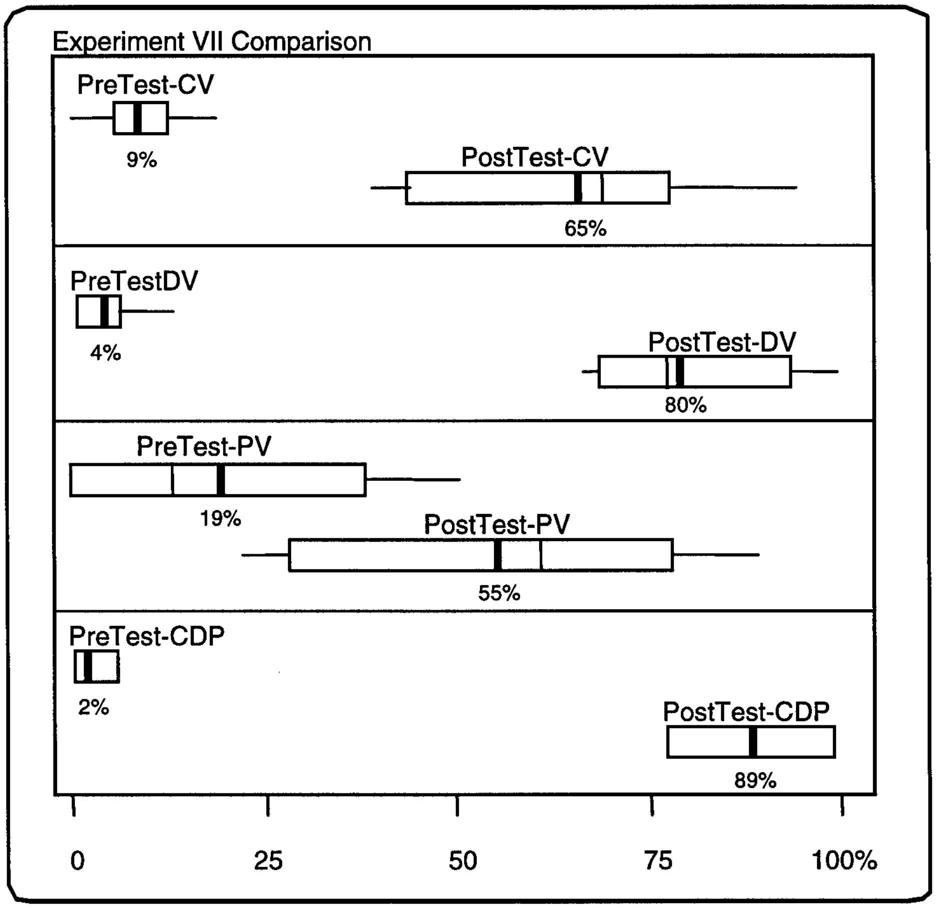


Figure 5.11. Experiment VII Box Plots

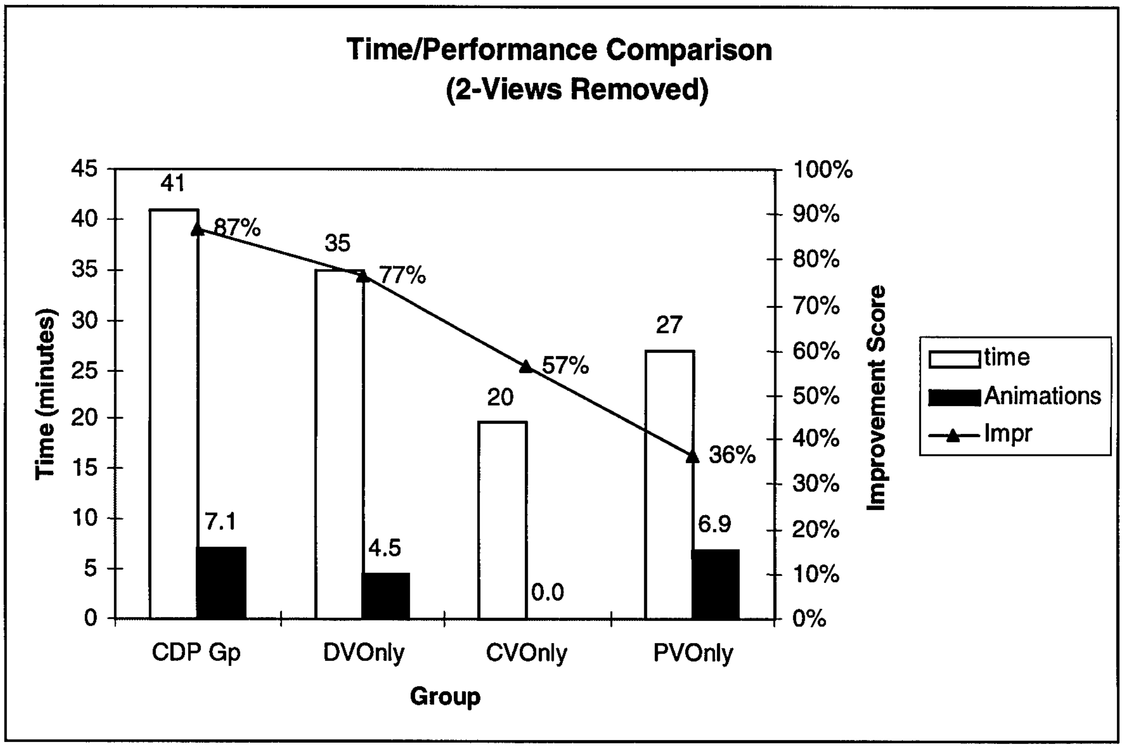
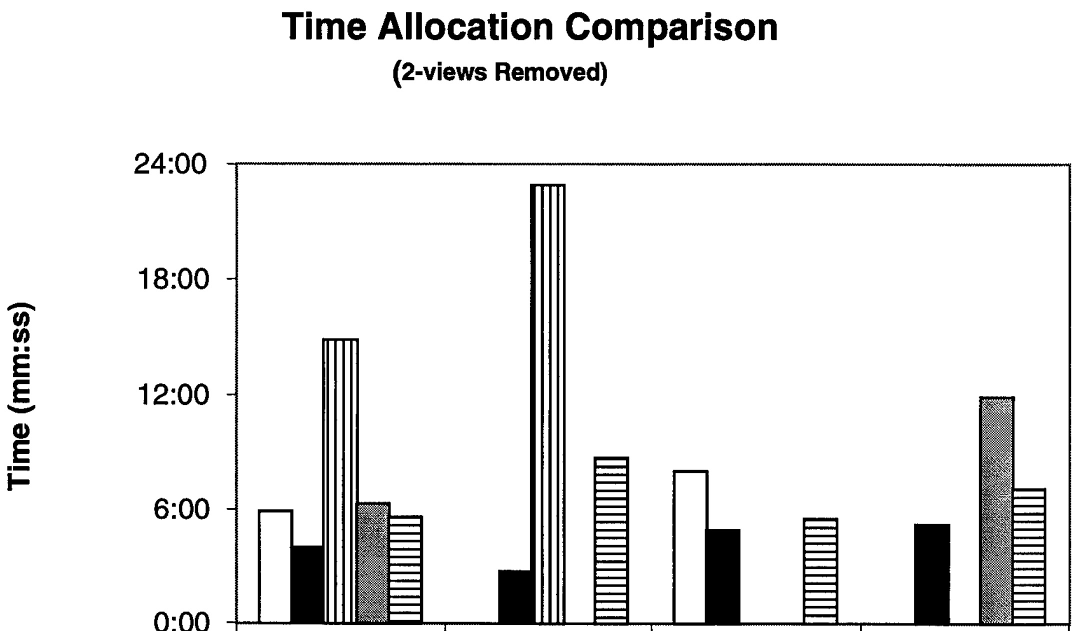


Figure 5.12. Time/Performance Comparison for Experiment VII (2-Views Removed)

Figure 5.12 shows the relationship between performance, number of PV and/or DV animations executed, and the amount of time users spent with the various version of the software. We were not surprised with the amount of time the different groups used to interact with HalVis. The most interaction occurs in the Detailed View, and the time figures for the CDP and DV groups reflect this design attribute. The group spending the least amount of time was the CV group, which did not surprise us, yet we find it interesting that they spent more time than we expected they would, given the limited amount of information and interaction provided in the Conceptual View. While it appears there is a loose correlation between group performance, amount of time spent using the system and the number of animations that the group executed on the average for the CDP, DV and CV groups, this trend is *not* present in the PV group. Despite running the animations nearly the same number of times as the CDP group and 50% more times than the DV group, the PV group achieved the *lowest* performance improvement. Based on animation executions alone, one could raise the question of the effectiveness of DV animations over PV animations. Also striking is the level of improvement of the CV group that viewed no animations, just interacted with an real-world analogy and simulation.

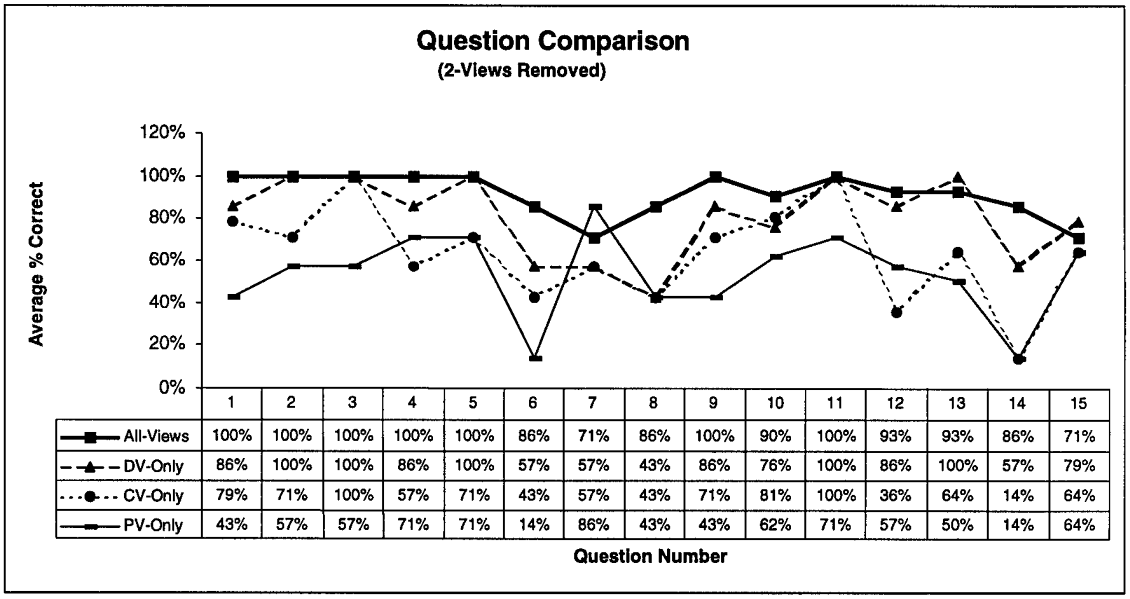
Figure 5.13 shows a detailed breakout of the amount of time subjects spent in each of the informational screens and views of the HalVis framework. As anticipated, the CV group spent the least amount of time and the CDP group spent the greatest amount of time interacting with the system. Furthermore, these data support the findings in the previous experiment that users tend to compensate for information that is omitted by spending time in related screens, as seen in the higher times for the DV group in the Detailed View, and the PV group in the Populated View.



	AllViews	DVOnly	CVOnly	PVOnly
□ ConceptualView	5:55	0:00	8:00	0:00
■ Description	3:58	2:44	4:53	5:08
▥ DetailedView	14:50	22:57	0:00	0:00
▒ PopulatedView	6:14	0:00	0:00	11:50
▤ Questions	5:32	8:38	5:27	7:02

Figure 5.13. Experiment VII Detailed Time Data for Each Screen

Figure 5.14 shows how the groups performed on each of the questions. It is interesting that the PV group performed much lower than the other groups except on the question that delves into assigning weights to edges (Q7) to force a particular Shortest Path sequence.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
—■— All-Views	100%	100%	100%	100%	100%	86%	71%	86%	100%	90%	100%	93%	93%	86%	71%
– –▲– – DV-Only	86%	100%	100%	86%	100%	57%	57%	43%	86%	76%	100%	86%	100%	57%	79%
···●··· CV-Only	79%	71%	100%	57%	71%	43%	57%	43%	71%	81%	100%	36%	64%	14%	64%
—▬— PV-Only	43%	57%	57%	71%	71%	14%	86%	43%	43%	62%	71%	57%	50%	14%	64%

Figure 5.14. Experiment VII Comparison of Question Responses

Table 5.9 summarizes the various statistical comparisons between the control group and groups with individual views (upper portion of the table), and between groups with individual views. Except for the CDP/D comparison, the results are all significant.

Statistical Summary					
CDP	0.87	CDP	0.87	CDP	0.87
C	0.57	D	0.77	P	0.36
F(1,11)	14.51	F(1,11)	2.55	F(1,11)	51.82
p	p<0.003	p	p<0.14	p	p<0.00001
C	0.57	D	0.77	C	0.57
D	0.77	P	0.36	P	0.36
F(1,12)	5.82	F(1,12)	28.29	F(1,12)	5.35
p	p<0.033	p	p<0.0002	p	p<0.039

Table 5.9. Statistical Summary for Experiment VII (2-Views Removed)

Discussion

This experiment indicates how much impact each of the views have on learning effectiveness. The importance of the Detailed View was confirmed, as was the value of the Conceptual View. We were somewhat surprised at the level of improvement observed in the group that only interacted with the Conceptual View, and this suggests that having a good analogy can produce surprisingly positive results. The performance of the Populated View group lagged behind the others. Yet in each of the experiments, the animations on this view were executed about the same number of times. Perhaps this serves as confirmation that some animations are merely 'candy for the eyes' in that they are entertaining to observe but seem to obscure the details, depth and mechanisms needed to engage the learner's mind. It should also be noted that the animation in the Populated View closely resembles algorithm animations created in previous research.

5.4. SUMMARY OF ABLATION EXPERIMENTS

This series of experiments was conducted to explore the components of the framework developed in this research that led to the significant results in learning effectiveness observed in Chapter 4. The first experiment removed features that were unique to our design and which we believed contributed to learning effectiveness. While the performance differences between the groups in this study were not statistically significant, there was a noticeable effect when each feature was removed. The absence of a feature did not lead to a linear decrease in performance. The effect manifested itself as decreased performance on the post-test and also as altered user behavior. Users appeared to be able to glean the information they needed by exploiting other sources in the system. For example, one group made up for the lack of chunking by rerunning the animation over and over, and the group that lacked collocated pseudocode on the Detailed View flipped back and forth between a screen that did show the pseudocode.

The second experiment explored the effect of removing a single view from the trio that form the core of the framework. Our initial expectations were not validated. We were surprised to find how much better the groups that interacted with the Conceptual View performed despite the greater amount of information portrayed in the Detailed and Populated Views. Probing this issue further involved the final experiment that provided a single view to the groups and measured their performance. Here, the value of the Detailed View was confirmed, but so was the impressive contribution of the Conceptual View. Interestingly, the students generally commented most favorably about the animations in the Populated View even though their performance was poorest in both sets of experiments. They thought they were learning more from the Populated View than was actually the case.

These results lead to several observations:

- The Conceptual View is important. While we initially believed that the analogy/real-world example was a minor educational addition and feature of our framework, it appears to have a pronounced effect in peaking student interest and priming them for learning from subsequent views of greater detail. We were not able to test our belief that using analogies might also contribute to long term retention.
- The Detailed View is important, but needs to be presented in context for the student to get the greatest potential from it. Providing the student with appropriate information makes the visualization more effective. Removing the pseudocode was not only an inconvenience, it led to poorer performance. The importance of user-provided data sets as inputs to the algorithms is not as clear, since very few users took advantage of this feature. Fully interactive algorithm animations may not be as necessary as originally

thought, and perhaps canned data sets are sufficient to illustrate representative algorithm behaviors to students.

- Questions are useful but not critical for effectiveness. Even though the subjects that were denied access to tickler questions in the Detailed View, predictions in the Populated View and the feedback-style queries in the Questions Screen did not perform significantly worse than subjects who received questions, we did note a decrease in their test scores and believe they are important to an educational system like HalVis.
- Chunking is important in helping students pace their learning and absorbing the subtleties of the algorithm. Even the best work-arounds that students attempted could not make up for the absence of this feature.
- The Populated View is less useful than originally thought. It is one of the most entertaining to watch, but appears to yield the least amount of understanding.

Of the components mentioned in Chapter 3, these experiments tested and provide valuable insight to the surprising potential of Bridging Analogies, Multiple Views, Semantic Chunking, and Purposeful Interaction, using a novel framework built using Multimodal Presentation techniques and the algorithm lessons were based on sound learning objectives. The next chapter investigates ways of disseminating the tools we have developed and empirically tested.

6. EXTENDING THE FRAMEWORK TO THE INTERNET

This section describes our efforts to make these results and products available for others to use for testing and research. First we describe how to access and execute the HalVis system which was used for the experiments reported in Chapters 4 and 5. Then we discuss the limitations of the current HalVis environment and describe our efforts to replicate the framework discussed in Chapter 3 using another algorithm animation system that addresses some of the HalVis shortcomings.

6.1. ACCESSING THE HALVIS PROTOTYPE

The HalVis prototype system is available for public downloading at http://www.eng.auburn.edu/department/cse/research/vi3rg. The self-extracting program creates a HalVis directory on the target system's hard drive and installs the runtime execution files needed to execute. This consists of the HalVis program and a group of runtime executable files. The system was developed using Asymetrix Toolbook, for which the runtime executable files are available *royalty-free* to the general public. HalVis works with the following versions of Microsoft Windows: 3.0, 3.1, 95, 98, and NT. Users will find the system fully functional and programmed with the six algorithms described in Appendix B. Unfortunately, extending the HalVis program to incorporate additional algorithm visualizations is not easily accomplished.

6.2. HALVIS LIMITATIONS

HalVis was designed as a dedicated system (see discussion in Chapter 2.2.4) similar to AACE, XSortLab and AlgoNet. The design philosophy behind HalVis was to use rapid prototyping techniques to create, test and validate the novel features of the framework described in Chapter 3. As a dedicated system, HalVis is not easily user-modifiable. Programming is required of researchers or instructors wanting to alter existing HalVis visualizations or to create new ones. This means having access to a licensed version of Asymmetrix Toolbook, and expertise in the Toolbook authoring language (called OpenScript). We offer and provide the already-coded visualizations and templates as examples to help. But, admittedly, this is a daunting task to a skilled programmer, and nearly unthinkable for the student wishing to create visualizations of his own.

Another limitation to the HalVis approach is it is only available for Microsoft Windows environment. While this is a very popular and prevalent operating environment, it does not support widespread, platform-independent execution. For these reasons, we investigated techniques to implement the HalVis framework using other tools.

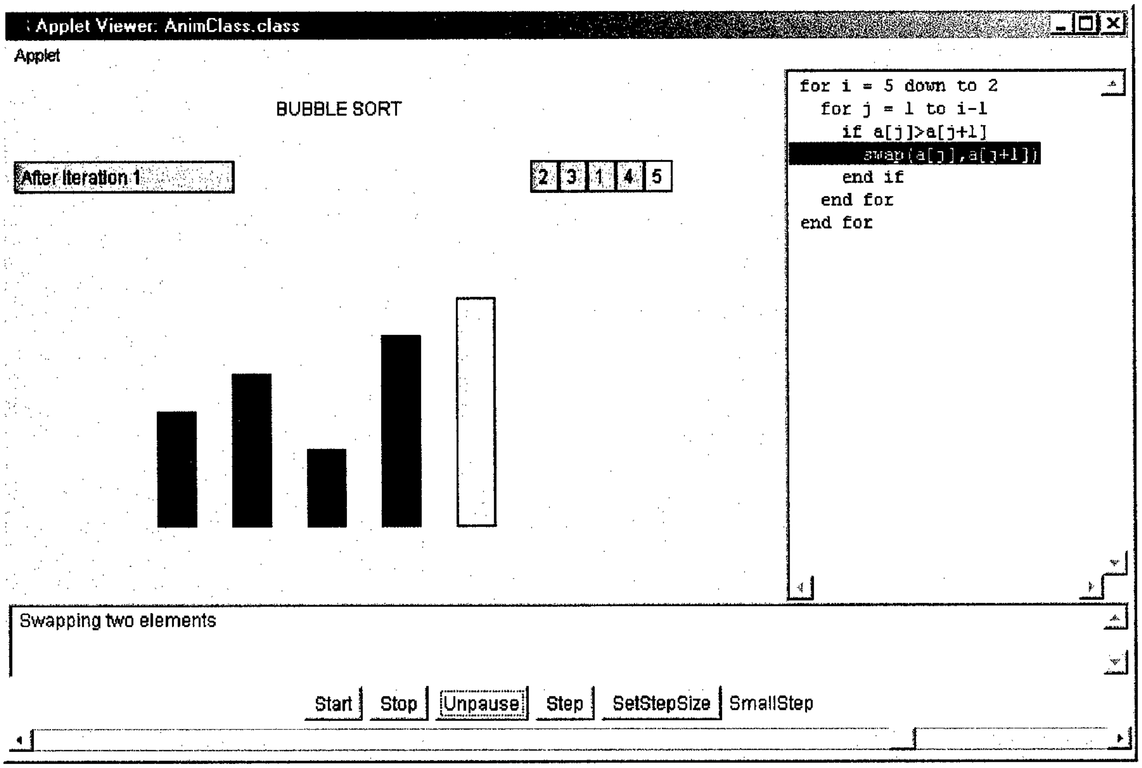
6.3. TOWARDS A GENERAL AUTHORING ENVIRONMENT

We envision the HalVis framework being implemented in a way that supports Internet access and execution, and written in a way that allows students to interact with already-prepared visualizations as well as supporting a more constructive approach of helping students create their own visualizations. To make the components of the framework available to a wider range of users desiring a broader delivery platform such as the Internet, a tool other than Asymmetrix Toolbook must be used. The rich properties of HTML and Java make porting some of the features of our framework possible. One shortcoming of the HalVis prototype was that it did not support user-developed animations without considerable knowledge and expertise in the Toolbook language. Fortunately, there are several systems available that support user-developed animations, built on the scripted 'interesting event' paradigm. XTango is the most popular interesting event animation tool, currently available for stand-alone execution in the Unix and Windows environments. The interesting event paradigm allows end users to create and modify animation scripts in addition to replaying and interacting with ones already prepared by others. Unfortunately, XTango is not available for Internet execution.

An attractive alternative tool is available. Researchers at Duke University (Pierson & Rodger, 1998) have created a system called JAWAA that implements the XTango command set using Java, which allows Internet delivery of animations. JAWAA supports the interesting event paradigm and allows user-created animation scripts. Like XTango, JAWAA commands facilitate creation, placement and manipulation of a wide variety of graphical objects like circles, lines, squares, and points. Furthermore, they have extended the XTango command set to facilitate animation of complex data structures, like trees, arrays, stacks and more, making it one of the best general purpose animation systems available. A description of the JAWAA command syntax is provided at the Internet address indicated above and, for the reader's convenience, at Appendix D. However, despite its superiority and wealth of desirable features, the JAWAA engine does not support some of the capabilities that were tested and found to be effective in the HalVis framework. Fortunately, the JAWAA system can be extended and wrapped in HTML script to provide most of the features we have described as being most significant.

We have named the modified software JAVIZ (Java-based Algorithm Visualization) tool. The key extensions we implemented are:

- **Contextual text messages**- JAVIZ provides authors the capability to post explanatory messages to the viewing screen to help users understand and interpret the actions occurring on the screen.
- **Pseudocode window** collocated with the animation- JAVIZ allocates a pane for the author to place pseudocode in any level of detail for the user to observe while the animation is executing.
- **Pseuducode highlighting**- JAVIZ provides the author with the ability to highlight specific lines in the pseudocode window, which helps the user identify at what step the animation might be during the course of its execution.
- **Chunking support**- JAVIZ provides a mechanism for authors to let the system and the user cooperatively control the semantic pauses in the execution of an animation script to allow time to think, interpret and absorb the information as it unfolds.



```
for i = 5 down to 2
  for j = 1 to i-1
    if a[j]>a[j+1]
      swap(a[j],a[j+1])
    end if
  end for
end for
```

Swapping two elements

Figure 6.1. The JAVIZ Screen Depicting the BubbleSort Algorithm

Figure 6.1 shows a screen capture of the JAVIZ depiction of the BubbleSort algorithm, showing the pseudocode window to the right, the animation pane to the left, and buttons to control the pace of the animation above the contextual text window located at the bottom of the screen. Note the similarity to the Detailed View of the HalVis framework (see Chapter 3.2.4).

6.4. JAVIZ LANGUAGE SYNTAX

This section describes the syntax of the scripting language commands. In most cases, the JAVIZ syntax is the same as in the JAWAA system, which these commands extend, and the XTANGO system, from which JAWAA evolved. Figure 2.15 shows a file of JAWAA scripted commands, and Appendix D contains a copy of the JAWAA command syntax that is also available at the Duke University Internet site.

The PostMessage command in Table 6.1 is used to place explanatory text in the contextual text window. The animation author would use this command to provide the student with a brief, one-line summary of that action being taken by the algorithm.

Command:	**PostMessage**
Parameter:	*string*
Example:	PostMessage This is a contextual explanation

Table 6.1. The PostMessage Command

The pair of commands in Table 6.2 provide the mechanism to post a delimited sequence of lines, generally pseudocode, into the text panel on the right hand side of the JAVIZ window. Note that the text lines must be enclosed in quotation marks. If the text is wider or longer than the panel provided, scroll bars will appear to allow the user to manipulate the text window. The example sequence shown in Table 6.2 would post the three lines representing a simple FOR loop in the JAVIZ pseudocode window.

Begin Command:	**PostPseudocode**
Parameter(s):	*Line 1* *Line 2 ...* *Line n*
End Command:	**EndPostPseudocode**
Example:	PostPseudocode “for x = 1 to 10” “ y = y * x” “endfor” EndPostPseudocode

Table 6.2. The PostPseudocode Command Sequence

The command in Table 6.3 allows the animation author to highlight selected lines in the pseudocode panel as a means of attracting attention to and indicating the line of the algorithm currently being 'executed' in the animation. If a line number is given that exceeds the number of lines in the pseudocode window, the last line is highlighted. If another line has been highlighted as a result of a previous instance of this command, the other line is de-highlighted in favor of the new line being selected. The example below results in the second line in the pseudocode window being highlighted.

Command:	**HighlightLine**
Parameter:	*Line number (an integer)*
Example:	HighlightLine 2

Table 6.3. The HighlightLine Command

The command in Table 6.4 implements the chunking capability demonstrated in the HalVis framework. The animation author places these LogicalPause commands in the animation script to mark semantic chunks or logical pausing locations. The integer value provided as a parameter to the command indicates the pause level. The default value is 0, which is no logical pausing, meaning the animation proceeds to completion unless the user specifically intervenes by pressing the Pause or Stop buttons on the control panel. If the user presses the StepLevel button, the chunking level is changed to 1, 2 or 3, depending on how many times the button is pressed. Now, when the script encounters a LogicalPause, if the scripted pause level is less than the user selected threshhold, the animation pauses. Otherwise, the animation proceeds as though the LogicalPause was not in the script. This implements the chunking control feature that allows the author to mark logical sequence boundaries in an algorithm but gives the user control to skip across or pause at the boundaries. In the example below, if the user had selected StepLevel = 2, the LogicalPause command indicated below would cause the animation to pause until the user signaled he was ready to proceed by pressing the UnPause button. On the other hand, if the user had selected StepLevel = 1, the LogicalPause command indicated below would be ignored, and the animation would continue as though the command wasn't there.

Command:	**LogicalPause**
Parameter:	*Pause level (an integer)*
Example:	LogicalPause 2

Table 6.4. The LogicalPause Command

6.5. RUNNING A JAVIZ VISUALIZATION

This section describes the steps involved in creating and executing a JAVIZ animation. One of the strengths of this system is the ability to run animations from a remote site or locally-created script files. The JAVIZ code is available at http://www.eng.auburn.edu/departments/cse/vi3rg along with a limited number of animations to demonstrate the system capabilities. Users wishing to author their own animation script must create an HTML file, similar to the one shown below, that causes the JAVIZ program to seek the animation script file from the user's local Internet resource, identified using the *animLoc* parameter.

```
<APPLET         CODEBASE="http://www.eng.auburn.edu /department/research/vi3rg/JAVIZ"
                CODE="AnimClass.class"
                WIDTH=750
                HEIGHT=450
                ALIGN=CENTER>
<PARAM NAME=animLoc
                VALUE="http://www.eng.auburn.edu/department/research/vi3rg/JAVIZ/BubSort.anim">
<PARAM NAME=animName
                VALUE="Animation"></Applet>
```

Table 6.5. Sample HTML to Execute a Local Animation Script

The script file (an example is at Figure 2.15) containing the animation commands should be saved in ASCII format in a directory available on the Internet as a Web resource. The name of the file must match the name given in the *ParamName animLoc* entry of the HTML file as shown above. When the HTML file is accessed using a Web browser, the JAVIZ code will execute the script referenced and the animation will proceed as authored.

We will continue to enhance and improve the JAVIZ program as an ongoing research project.

7. CONCLUSION

This research has produced the first consistently significant results involving algorithm animation as a learning device. We have shown that subjects not only enjoy the hypermedia presentation of information, they learn more effectively interacting with algorithm-embedded hypermedia visualizations than from textbooks, lectures and animation-only approaches. Admittedly, our results are based on a small set of algorithms tested with smaller-than-ideal groups of subjects using a specially developed HalVis visualization system. Nevertheless, the results across a group of seven separate experiments give encouraging support to the potential benefits of using animation in algorithm education. Our results provide a general framework that others can adapt to individual circumstances as needed.

7.1. RESEARCH CONTRIBUTIONS

Our contributions fall into three general areas.

Theoretical: Drawing from prior research and blending some innovative ideas of our own, we have developed a framework for embedding animations into a knowledge-providing hypermedia structure for teaching abstract and dynamic concepts. The framework is built on a hierarchy of modules laced with embedded animations and hypermedia links that introduce topics with broad analogies, guide the user to learn detailed information and reinforce the learning objectives with multiple views. The framework underscores the importance of keeping the user engaged with a variety of interaction techniques, chunking, questioning, and encouraging self-explanation.

Empirical: We have conducted a series of experiments that validated our theoretical framework with statistically significant results. We also provide experiments to gain insight to the specific impact of various features and views of our framework. This work stands as the most comprehensive set of empirical experiments involving the use of animation to help students learn about algorithms that is available in current literature.

Software: We have created the HalVis system and programmed six algorithm tutorials. We have demonstrated that the more significant features of our framework can be ported to other algorithm animation environments by extending the capabilities of an existing Internet-based authoring program to create the JAVIZ system. Both HalVis and JAVIZ are available for public use.

7.2. FUTURE DIRECTIONS

In the course of most research projects, additional questions and topics are encountered, and our experience has been no different. Future research could be conducted in the following areas:

- Study the impact of the conceptual view, specifically characteristics of analogies. What about the analogies primed learning? How important are fidelity and interactivity to the analogy? How much can students learn from analogy alone? What about algorithms for which analogies don't readily exist?
- Study the effect of user/student-designed animations compared to expert-prepared ones using the framework discussed here. Will students learn more effectively if they take a more constructive approach and build their own animation depicting the algorithm's operations?
- Differentiate the effects of hypermedia visualization with different classes of subjects, such as different genders, different learning style, and so on.

- Conduct deeper research into other measures of success like long-term retention, speed and satisfaction. Our work just scratched the surface using these metrics. Does interacting with hypermedia aid long term retention? How useful is the analogy to helping users recall algorithmic details over the long term? Do hypermedia visualizations motivate students differently than using other teaching techniques? What would the effect be of establishing time limits on the comparative studies we conducted in Chapter 4?
- Explore the effects of color and sound on learning. Our work employed both, but on the intuitive notion that sound and color help highlight important concepts and events. What kind of audio is most effective to supplement and reinforce hypermedia modules? How can visual techniques such as color changes or flashing objects be used to attract attention to important items?
- Compile a list of guidelines to help authors create better algorithm animations. How big should chunks be? What is the best way to use questions such as the ticklers, predictions and feedback-style questions we employed?
- Conduct eye-tracking studies to explore where subjects look during a multi-pane animation. What patterns develop? Are some panes used more than others, or not at all? How could a designer place multiple panes for optimal use?
- Continue enhancing the authoring process. A major enhancement and convenience left for future development is the creation of an interactive scripting environment that would allow authors to visually place objects, generate the script and allow running the script from the same facility. Currently, one must create the script commands using a separate process, then run the JAVIZ engine to view the sequence. This approach works and is adopted by all other animation systems, but is not as convenient as an interactive environment could be.

- Conduct experiments to evaluate the HalVis framework in other settings. Does the HalVis framework accommodate harder and more complex algorithms? Can all algorithms be taught with visualization or are there ones that are inherently non-visual? Our experiments compared media against each other, and we hypothesize that combining visualization with textual, lecturing and other teaching techniques should enhance learning even more than any of these methods by themselves, but would that be the case?

Rather than abandoning the notion of employing algorithm animation as a learning aide, we have shown that there are contexts in which animation can be quite effective, and have raised many ideas and questions for future researchers to explore as a means of gathering deeper insight and understanding into how computers and hypermedia can help humans visualize and learn abstract concepts.

REFERENCES

Aho, A., Hopcroft, J., & Ullman, J. (1974). *The design and analysis of computer algorithms.* Reading, MA.: Addison-Wesley Pub. Co.

Aho, A., Hopcroft, J., & Ullman, J. (1983). *Data structures and algorithms.* Reading, MA.: Addison-Wesley Pub. Co.

Astrachan, O., Selby, T., & Unger, J. (1996). An object oriented, apprenticeship approach to data structures using simulation. In *Proceedings of IEEE 26th Frontiers in Education Conference* (pp 130-134). Los Alamitos, CA: IEEE CS Press.

Baase, S. (1988). *Computer algorithms : introduction to design and analysis.* Reading, MA: Addison-Wesley Pub. Co.

Badre, A., Beranek, M., Morris, J. M., & Stasko, J. T. (1991). Assessing program visualization systems as instructional aids. (Technical Report No. GIT-GVU-91-23). Atlanta, GA. Georgia Institute of Technology.

Baecker, R. (1981). *Sorting Out Sorting.* [Narrated color film]. Presented at ACM SIGGRAPH'81 and excerpted in ACM SIGGRAPH Video Review 7, 1983. Los Altos, CA: Morgan Kaufmann.

Baeker, R., DiGiano, C., & Marcus, A. (1997). Software visualization for debugging. *Communications of the ACM*, 40(4), 44-54.

Bagui, S. (1998). Reasons for increased learning using multimedia. *Journal of Educational Multimedia and Hypermedia*, 7(1), 3-18.

Bazik, J., Tamassia, R., Reiss, S., & van Dam, A. (1998). Software visualization in teaching at Brown University. In M. Brown, J. Domingue, B. Price, & J. Stasko (Eds.), *Software Visualization: Programming as a Multimedia Experience* (pp. 383-398). Cambridge, MA: The MIT Press.

Beall, J., Doppelt, A., & Hughes, J. (1997). Developing an interactive illustration: using java and the web to make it worthwhile. http://www.cs.brown.edu/research/graphics/projects/igi/spectrum, Brown University.

Bellamy, R. K. E. (1994). What does pseudo-code do? A psychological analysis of the use of pseudo-code by experienced programmers. *Human-Computer Interaction 9*, 225-246.

Bellamy, R. K. E. (1996). Designing educational technology: Computer-mediated change. In B. Nardi (Ed.), *Context and Consciousness: Activity Theory and Human-Computer Interaction* (pp. 123-146). Cambridge, MA: The MIT Press.

Birch, M., Boroni, C., Goosey, F., Patton, S., Poole, D., Pratt, C., & Ross, R. (1995). DYNALAB: a dynamic computer science laboratory infrastructure featuring program animation. In *Proceedings of the 26th SIGCSE Technical Symposium on Computer Science Education* (pp. 29-33). New York: ACM Press.

Blackwell, A.F. (1996). Metacognitive theories of visual programming: What do we think we are doing? In *Proceedings of the 1996 IEEE Symposium on Visual Languages* (pp. 240-246). Los Alamitos, CA: IEEE CS Press.

Bloom, B. (1956). *Taxonomy of educational objectives: the cognitive domain.* New York, NY: Longman, Inc.

Bødker, S. (1996). Applying activity theory to video analysis: How to make sense of video data in HCI. In B. Nardi (Ed.), *Context and consciousness: Activity Theory and Human-Computer Interaction* (pp. 147-175). Cambridge, MA: The MIT Press.

Boyle, T., Stevens-Wood, B., Feng, Z., & Tikka, A. (1996). Structured learning in a virtual environment. *Computers & Education*, 26(1-3): 41-49.

Brophy, S., & Schwartz, D. (1998). Interactive analogies. In *Proceedings of the 1998 International Conference of the Learning Sciences* (pp. 56-62). Charlottesville, VA: AACE Press.

Brown, M. H. (1988*a*). *Algorithm animation.* Cambridge, MA: The MIT Press.

Brown, M. H. (1988*b*). Perspectives on algorithm animation. In *Proceedings of the ACM SIGCHI '88 Conference on Human Factors in Computing Systems* (pp. 33-38). New York, NY: ACM Press.

Brown, M. H. (1988c). Exploring algorithms using Balsa-II. *Computer*, 21(5):14-36, 1988.

Brown, M. H. (1991). Zeus: a system for algorithm animation. In *Proceedings of the 1991 Workshop on Visual Languages* (pp. 4-9). Los Alamitos, CA: IEEE CS Press.

Brown, M. H. (1994). The 1994 SRC Algorithm Animation Festival. (Research report no. 126). Palo Alto, CA: Digital Equipment Corp.

Brown, M. H., & Hershberger, J. (1991). Color and sound in algorithm animation. *Proceedings of 1991 IEEE Workshop on Visual Languages* (pp. 10-17). Los Alamitos, CA: IEEE CS Press.

Brown, M., & Najork, M. (1996). Collaborative active textbooks: a web-based algorithm animation system for an electronic classroom. In *Proceedings of the 1996 Symposium on Visual Languages* (pp. 266-275). Boulder, CO: IEEE CS Press.

Brown, M., Najork, M., & Raisamo, R. (1997). A Java-based implementation of collaborative active textbooks. In *Proceedings of the 1997 IEEE Symposium on Visual Languages* (pp. 372-379). Capri, Italy: IEEE CS Press.

Brown, M. H., & Sedgewick, R. (1985). Techniques for algorithm animation. *IEEE Software 2*(1), 28-39.

Byrne, M., Catrambone, R., & Stasko, J. T. (1996). Do algorithm animations aid learning? (Technical Report No. GIT-GVU-96-18). Atlanta, GA: Georgia Institute of Technology.

Byrne, M., Guzdial, M., Ram, P., Catrambone, R., Ram, A., Stasko, J., Shippey, G., & Albrecht, F. (1995). The role of student tasks in accessing cognitive media types. In *Proceedings of the 1996 International Conference on the Learning Sciences*. Evanson, IL: AACE Press.

Casner, S. M., & Larkin, J. H. (1989). Cognitive efficiency considerations for good graphic design. In *Proceedings of the Annual Conference of the Cognitive Science Society* (pp. 275-282). Hillsdale, NJ: Erlbaum, Inc.

Catenazzi, N., Aedo, I., Diaz, P., & Sommaruga, L. (1997). The evaluation of electronic book guidelines from two practical experiences. *Journal of Educational Multimedia and Hypermedia*, 6(1):91-114.

Chi, M.T.H., Bassok, M., Lewis, M., Reimann, P., & Glaser, R. (1989). Self-explanations: how students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.

Cleveland, W. S., & McGill, R. (1986). An experiment in graphical perception. *International Journal of Man-Machine Studies, 25*, 491-500.

Cox, K. C., & Roman, G. C. (1992). Experiences with the Pavane program visualization system. (Technical report No. WUCS-92-40). St. Louis, MO: Washington University in St. Louis.

Crosby, M., & Stelovsky, J. (1995). From multimedia instruction to multimedia evaluation. *Journal of Educational Multimedia and Hypermedia*, 4(2/3):147-162.

Curtis, B. (1986). By the way, did anyone study any real programmers? In *Empirical Studies of Programmers* (pp. 256-262). Norwood, NJ: Ablex.

Daily, B. (1994). Multimedia and its impact on training engineers. *International Journal of Human-computer Interaction*, 6(2):191-204.

Dale, N., Lilly, S., & McCormick, J. (1996). *Ada plus data structures: an object-based approach.* Lexington, MA: D.C. Heath and Company.

De Lillo, N. (1993). *A first course in computer science with Ada.* Homewood, IL: Irwin Pub. Co.

Dershem, H., & Brummund, P. (1998). Tools for web-based sorting animation. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education* (pp. 222-226). Atlanta, GA: ACM Press.

Domingue, J., & Mulholland, P. (1997). Staging software visualizations on the web. In *Proceedings of 1997 IEEE Visual Languages*. Capri, Italy: IEEE CS Press.

Douglas, S.A., McKeown, D., & Hundhausen, C. (1993). Exploring human visualization of algorithms. (Technical Report No. TR CIS-TR-94-27). Eugene, OR: University of Oregon.

Douglas, S. A. (1995). Conversation analysis and human-computer interaction design. In P. Thomas (Ed.) *Social and Interactional Dimensions of Human-Computer Interfaces* . Cambridge: Cambridge University Press.

Douglas, S. A., Hundhausen, C. D., & McKeown, D. (1995). Toward empirically-based software visualization languages. In *Proceedings of the 1995 IEEE Symposium on Visual Languages* (pp. 342-349). Los Alamitos, CA: IEEE CS Press.

Douglas, S. A., Hundhausen, C. D., & McKeown, D. (1996). Exploring human visualization of computer algorithms. In *Proceedings 1996 Graphics Interface Conference* (pp. 9-16). Toronto, Canada: Canadian Graphics Society.

Duchastel, P. (1978). Illustrating instructional texts. *Educational Technology*, 18, 36-39.

Eck, D. (1998). XSortLab. _http://math.hws.edu/TMCM/java/labs/xSortLabLab.html, Brandeis University, New York.

Felder, R., & Silverman, L. (1988). Learning and teaching styles in engineering education. *Engineering Education*, 4, 674-681.

Ford, L. (1993). How programmers visualize programs. (Technical Report No. R 271). Exeter, U.K.: University of Exeter, Department of Computer Science.

Gentner, D. (1989). The mechanisms of analogical reasoning. In S. Vosniadou & A. Ortony (Eds.) *Similarity and Analogical Reasoning.* Cambridge, England: Cambridge University Press.

Gloor, P. (1992). AACE-algorithm animation for computer science education. In *Proceedings of the 1992 IEEE Workshop on Visual Languages* (pp. 25-31). Seattle, WA: IEEE CS Press.

Green, T., & Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*. 7, 131-174.

Gurka, J. S., & Citrin, W. (1996). Testing effectiveness of algorithm animation. In *Proceedings of the 1996 IEEE Symposium on Visual Languages* (pp. 182-189). Los Alamitos, CA: IEEE CS Press.

Guzdial, M. (1995). Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments*, 4(1), 1-44.

Guzdial, M., & Kehoe, C. (1998). Apprenticeship-based learning environments: A principled approach to providing software-realized scaffolding through hypermedia. (Technical Report). Atlanta, GA: Georgia Institute of Technology. Available at http://guzdial.cc.gatech.edu/papers/able/.

Hansen, S., Schrimpsher, D., & Narayanan, N. H. (1998a). From algorithm animations to animation-embedded hypermedia visualizations. (Technical Report No. CSE98-05). Auburn, AL: Auburn University.

Hansen, S., Schrimpsher, D., & Narayanan, N. H. (1998b). Empirical studies of animation-embedded hypermedia algorithm visualizations. (Technical Report No. CSE98-06). Auburn, AL: Auburn University.

Hansen, S., Schrimpsher, D., & Narayanan, N. H. (1998c). Learning algorithms by visualization: a novel approach using animation-embedded hypermedia. In *Proceedings of the 1998 International Conference on the Learning Sciences* (pp. 125-130). Charlottesville, VA: AACE Press.

Hansen, S., Schrimpsher, D., Narayanan, N. H. (1999). Helping learners visualize algorithms: Embedding analogies and animations in hypermedia. Paper accepted to the 1999 World Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA'99).

Harel, D. (1992). *Algorithmics : the spirit of computing.* Reading, MA.: Addison-Wesley Pub. Co.

Hartley, S. (1994). Animating operating systems algorithms with XTango. In *Proceedings of the 1994 SIGCSE Technical Symposium on Computer Science Education* (pp. 344-348). New York, NY: ACM Press.

Heath, M. T., & Etheridge, J. A. (1991). Visualizing the performance of parallel programs. *IEEE Software, 8*(5), 29-39.

Heath, M. T., Malony, A. D., & Rover, D. T. (1995). Parallel performance visualization: From practice to theory. *IEEE Parallel & Distributed Technology, 3*(4), 44-60.

Hegarty, M. (1992). Mental animation: inferring motion from static diagrams of mechanical systems. *Journal of Experimental Psychology: Learning, Memory and Cognition, 18*(5):1084-1102).

Hix, D., & Hartson, H. R. (1993). Formative evaluation: Ensuring usability in user interfaces. In L. Bass & J. Dewan (Eds.), *User Interface Software* (pp. 1-30). New York, NY: John Wiley & Sons.

Hmelo, C. E. & Guzdial, M. (1996). Of black and glass boxes: Scaffolding for learning and doing. In *Proceedings of the 1996 International Conference of the Learning Sciences* (pp. 128-134). Evanson, IL: AACE Press.

Horowitz, E., Sahni, S. (1978). *Fundamentals of computer algorithms.* Potomac, MD: Computer Science Press.

Hundhausen, C. D. (1995). A framework for semantics-based software visualization interaction. (Research Project Report). Eugene, OR: University of Oregon, Department of Computer and Information Science. Available at http://www.cs.uoregon.edu/~chundhau/research.

Hundhausen, C. (1996). A meta-study of software visualization effectiveness. Eugene, OR: University of Oregon. Available at http://www.cs.uoregon.edu/ ~chundhau/research.

Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *Journal of the Learning Sciences 4*(1), 39-103.

Kamada, T., & Kawai, S. (1991). A general framework for visualizing abstract objects and relations. *ACM Transactions on Graphics*, 10(1), 1-39.

Kehoe, C. M., & Stasko, J. T. (1996). Using animations to learn about algorithms: An ethnographic case study. (Technical Report No. GIT-GVU-96-20). Atlanta, GA: Georgia Institute of Technology.

Kimelman, D., Rosenburg, B., & Roth, T. (1994). Strata-Various: Multi-layer visualization of dynamics in software system behavior. In *Proceedings of Visualization '94* (pp. 172-178). Los Alamitos, CA: IEEE CS Press.

Kingston, J. (1998). *Algorithms and data structures : design, correctness, analysis.* / Reading, MA.: Addison-Wesley Pub. Co.

Kirsh, D. (1997). Interactivity and multimedia interfaces. *Instructional Sciences*, 1997.

Knuth, D. (1973). *The art of computer programming.* Reading, MA.: Addison-Wesley Pub. Co.

Kolodner, J. (1993). *Case-based Reasoning.* San Mateo, CA: Morgan Kaufmann Publishers.

Kozen, D. (1992). *The design and analysis of algorithms.* New York: Springer-Verlag Pub.

Kraemer, E., & Stasko, J. T. (1993). The visualization of parallel systems: An overview. *Journal of Parallel and Distributed Computing 18*(2), 105-117.

Lahtinen, S., Sutinen, E., & Tarhio, J. (1998). Automated animation of algorithms with Eliot. *Journal of Visual Languages and Computing*, 9, 337-349.

Larkin, J., & Simon, H. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11, 65-99.

Lave, J. (1988). *Cognition in Practice.* Cambridge, MA: Cambridge University Press.

Lavery, D., & Cockton, G. (1996). Heuristic evaluations for software visualization: usability evaluation materials. (FIDE Technical Report No. FIDE/96/10). Glasgow, Scotland: University of Glasgow.

Lawrence, A. W. (1993). Empirical studies of the value of algorithm animation in algorithm understanding. (Doctoral Dissertation, Georgia Institute of Technology, 1993).

Lawrence, A. W., Badre, A. N., & Stasko, J. T. (1994). Empirically evaluating the use of animations to teach algorithms. In *Proceedings of the 1994 IEEE Symposium on Visual Languages* (pp. 48-54). Los Alamitos, CA: IEEE CS Press.

Lester, J., Converse, S., Kahler, S., Barlow, T., Stone, B., & Bhogal, R. (1997). The persona effect: affective impact of animated pedagogical agents. In *Proceedings of CHI '97* (pp. 359-366). Atlanta, GA: ACM Press.

Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics 5*(2), 110-141.

Manber, U. (1989). *Introduction To Algorithms : A Creative Approach.* Reading, MA.: Addison-Wesley Pub. Co.

Matlin, M. (1989). *Cognition* (2nd ed). New York: Holt, Rinehard & Winston.

Mayer, R. (1989). Systematic thinking fostered by illustrations in scientific text. *Journal of Educational Psychology*, 81(2):240-246.

Mayer, R. & Anderson, R. (1991). Animations need narrations: An experimental test of a dual-coding hypothesis. *Journal of Educational Psychology 83*(4), 484-490.

Mayer, R., & Anderson, R. (1992). The instructive animation: helping students build connections between words and pictures in multimedia learning. *Journal of Educational Psychology*, 84(4):444-452.

Mayer, R., & Moreno, R. (1998) A cognitive theory of multimedia learning: Implications for design principles. In F. Durso (Ed.), *Handbook of Applied Cognition* (in press).

Mayer, R., & Sims, V. (1994) For whom is a picture worth a thousand words? Extensions of a Dual-Coding Theory of Multimedia Learning, *Journal of Educational Psychology*, 86(3):389-401.

McWhirter, J. (1996). Algorithm explorer: a student centered algorithm animation system. In *1996 IEEE Symposium on Visual Languages* (pp. 174-181). Los Alamitos, CA: IEEE Press.

Miller, G.A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Science*, 63:81-97.

Muldner, T., Muldner, K., & VanVeen, C. (1997). Experience from the design of an authoring environment. *Journal of Educational Multimedia and Hypermedia*, 6(1), 115-132.

Mukherjea, S., & Stasko, J. T. (1994). Toward visual debugging: Integrating algorithm animation capabilities within a source-level debugger. *ACM Transactions on Computer-Human Interaction 1*(3), 215-244.

Muthukumarasamy, J., & Stasko, J. T. (1995). Visualizing program executions on large data sets using semantic zooming. (Technical Report No. TR GIT-GVU-95-02). Atlanta, GA: Georgia Institute of Technology.

Myers, B. A. (1986). Visual programming, programming by example, and program visualization: a taxonomy. In *Proceedings of ACM CHI'86 Conference on Human Factors in Computing Systems* (pp. 59-66). New York: ACM Press

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing 1*(1), 97-123.

Mynatt, B., Leventhal, L., Instone, K., Farhat, J., & Rohlman, D. (1992). Hypertext or book: Which is better for answering questions? In *Proceedings CHI'92* (pp 19-25). New York: ACM Press.

Naps, T. (1990). Algorithm visualization in computer science laboratories. In *Proceedings of the 1990 SIGCSE Technical Symposium on Computer Science Education* (pp. 105-110). New York: ACM Press.

Naps, T., & Bressler, E. (1998). A multi-windowed environment for simultaneous visualization of related algorithms on the WWW. In *Proceedings of the 1998 SIGCSE Technical Symposium on Computer Science Education* (pp. 277-281). New York: ACM Press.

Naps, T., & Nance, D. (1995). Introduction to computer science: programming, problem solving and data structures. New York: West Pub. Co.

Narayanan, N. H., & Hegarty, M. (1998). On designing comprehensible interactive hypermedia manuals. *International Journal of Human-Computer Studies*, 48:267-301.

Narayanan, N. H., Hmelo, C., Petrushin, V., Newstetter, W., Guzdial, M., & Kolodner, J. (1995). Computational support for collaborative learning through generative problem solving. In *Proceedings of Computer-Support for Collaborative Learning (CSCL) '95.* Available at http://www-cscl95.indiana.edu/cscl95/narayanan.html.

Nardi, B. (Ed.). (1996). *Context and Consciousness: Activity Theory and Human-Computer Interaction.* Cambridge, MA: The MIT Press.

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. In *Proceedings of CHI'92 Conference on Human Factors in Computing Systems* (pp. 373-380). New York: ACM Press.

Oudshoorn, M. J., Widhaha, H., & Ellershaw, S. K. (1996). Aspects and taxonomy of program visualization. In P. Eades & K. Zhang (Eds.), *Software Visualization* (pp. 3-25). River Edge, NJ: World Scientific Publishing Co.

Paivio, A. (1986). *Mental Representations: A Dual Coding Approach.* Oxford, England: Oxford University Press.

Palmiter, S., & Elkerton, J. (1993). Animated demonstrations for learning procedural computer-based tasks. *Human-Computer Interaction 8*(3), 193-216.

Pane, J. F., Corbett, A. T., & John, B. E. (1996). Assessing dynamics in computer-based instruction. In *Proceedings CHI'96 Conference on Human Factors in Computing Systems* (pp. 197-204). New York, NY: ACM Press,

Petre, M. (1995). Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM 38*(6), 33-44.

Petre, M., Blackwell, A. F., & Green, T. R. G. (1998). Cognitive questions in software visualization. In M. Brown, J. Domingue, B. Price, & J. Stasko (Eds.), *Software visualization: Programming as a multimedia experience* (pp. 453-480). Cambridge, MA: The MIT Press.

Petre, M., & Green, T. R. G. (1993). Learning to read graphics: Some evidence that 'seeing' an information display is an acquired skill. *Journal of Visual Languages and Computing 4*, 55-70.

Pierson, W., & Rodger, S. (1998). Web-based animation of data structures using JAWAA. In *Proceedings of the 1998 SIGCSE Technical Symposium on Computer Science Education* (pp. 267-271). Atlanta, GA: ACM Press.

Price, B. A., Baecker, R. M., & Small, I. S. (1993). A principled taxonomy of software visualization. *Journal of Visual Languages and Computing 4*(3), 211-266.

Rappin, N., Guzdial, M., Realff, M., & Ludovice, P. (1997). Balancing usability and learning in an interface. In *Proceedings CHI'97* (pp. 479-486). Atlanta, GA: ACM Press.

Recker, M., Ram, A., Shikano, T., Li, G., & Stasko, J. T. (1995). Cognitive media types for multimedia information access. *Journal of Educational Multimedia and Hypermedia* 4(2/3):183-210.

Reed, S. (1985). Effect of computer graphics on improving estimates to algebra word problems. *Journal of Educational Psychology*, 77(3):285-296.

Reingold, E., & Hansen, W. (1983). *Data structures.* Boston, MA: Little, Brown Publishing.

Rieber, L. P. (1990) Using computer animated graphics in science instruction with children. *Journal of Educational Psychology*, 82(1):135-140.

Rieber, L. P., Boyce, M. J., & Assad, C. (1990). The effects of computer animation on adult learning and retrieval tasks. *Journal of Computer-Based Instruction, 17*(2), 46-52.

Robertson, P. (1991). A methodology for choosing data representations. *IEEE Computer Graphics and Applications*, May 1991, 55-66.

Rogers, P. Gaizauskas, R. Humphreys, K., & Cunningham, H. (1997). Visual execution and data visualization in natural language processing. In *Proceedings of the 1997 IEEE Symposium on Visual Languages* (pp. 338-343). Los Alamitos, CA: IEEE CS Press.

Roman, G. C., Cox, K. C., Wilcox, C. D., & Plun, J. Y. (1992). Pavane: A system for declarative visualization of concurrent computations. *Journal of visual languages and computing 3*(2), 161-193.

Roman, G. C., & Cox, K. C. (1993). A taxonomy of program visualization systems. *IEEE Computer 26*(12), 11-24.

Rowe, G. (1997). *Introduction to data structures and algorithms with C++*. New York: Prentice Hall.

Sanderson, P. M., & Fisher, C. (1994). Exploratory sequential data analysis: Foundations. *Human-Computer Interaction 9*(3-4), 251-318.

Sangwan, R., & Korsh, J. (1998). A system for program visualization in the classroom. In *Proceedings of the 1998 SIGCSE Technical Symposium on Computer Science Education* (pp. 272-276). Atlanta, GA: ACM Press.

Schneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction,* Addison-Wesley, Reading, MA, 1998.

Sedgewick, R. (1988). *Algorithms.* Reading, MA.: Addison-Wesley Pub. Co.

Shaffer, C. (1997). *A practical introduction to data structures and algorithm analysis.* Upper Saddle River, N.J.: Prentice Hall.

Shaffer, C., Heath, L., Nielsen, J., & Yang, J. (1996). SWAN: A student-controlled data structure visualization system. In *Proceedings of the 1996 World Conference on Educational Multimedia and Hypermedia (ED-MEDIA-96)* (pp. 632-637). Charlottesville, VA: AACE Press.

Shikano, T., Recker, M., & Ram, A. (1996*).* Evaluating organization of a hypermedia learning environment using GOMS model analysis. *Proceedings of the 1996 World Conference on Educational Multimedia and Hypermedia (ED-MEDIA-96).* Charlottesville, VA: AACE Press.

Shippey, G., Ram, A., Albrecht, F., Roberts, J., Guzdial, M., Catrambone, R., Byrne, M., & Stasko, J. T. (1996). Exploring interface options in multimedia educational environments. In *Proceedings of the 1996 International Conference of the Learning Sciences.* Evanston, IL: AACE Press.

Simpson, J. A. & Weiner, E.S.C. (Ed.). (1989). *The Oxford English Dictionary (2nd ed.).* Oxford: Oxford University Press.

Singh, B., & Naps, T. (1985). *Introduction to data structures.* New York: West Pub. Co.

Soloway, E., Jackson, S., Klein, J., Quintana, C., Reed, J., Spitulnik, J., Stratford, S., Studer, S., Jul, S., Eng, J., & Scala, N. (1996). Learning theory in practice: Case studies of learner-centered design. *Proceedings of CHI'96.* New York: ACM Press.

Stasko, J. (1990). TANGO: A framework and system for algorithm animation. *Computer,* Sept 1990, 27-39.

Stasko, J. (1992). Animating algorithms with XTango. *SIGACT News, 23*(2): 67-71.

Stasko, J. (1997). Using student-built algorithm animations as learning aids. In *Proceedings of the 1997 SIGCSE Technical Symposium on Computer Science Education* (pp. 25-29). New York: ACM Press.

Stasko, J., Badre, A., & Lewis, C. (1993). Do algorithm animations assist learning? an empirical study and analysis. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems* (pp. 61-66). New York, NY: ACM Press.

Stasko, J., Domingue, J., Brown, M., & Price, B. (1998). *Software Visualization: Programming as a Multimedia Experience.* Cambridge, MA: MIT Press.

Stasko, J. & Patterson, C. (1992). Understanding and characterizing software visualization systems. In *Proceedings of the IEEE Symposium on Visual Languages* (pp. 3-10). Los Alamitos, CA: IEEE Press.

Suchman, L. A. (1987). *Plans and Situated Actions: The Problem of Human-Computer Communication.* New York: Cambridge University Press.

Suni, I., & Ross, S. (1997). Iterative design and usability assessment of a materials science hypermedia document. *Journal of Educational Multimedia and Hypermedia,* 6(2), 187-199.

Tufte, E. R. (1983). *The Visual Display of Quantitative Information.* Cheshire, CT: Graphics Press.

VanLehn, K., Jones, R., & Chi, M.T.H. (1992) A model of the self-explanation effect. *Journal of the Learning Sciences,* 2(1), 1-59.

Vessey, I. (1985). Expertise in debugging computer programs: a process analysis. *International Journal of Man-Machine Studies,* 23, 459-494.

Wang, D., & Arbib, M. (1993). Timing and chunking in processing temporal order. *IEEE Transactions on Systems, Man, and Cybernetics, 23*(4), 993-1110.

Weiss, M. A. (1993). *Data Structures and Algorithm Analysis using ADA.* Redwood City, CA: Benjamin Cummings Publishing Company.

Whale, G. (1996). A data structure animation tool for computer science laboratories. In *Proceedings of the 1996 World Conference on Educational Multimedia and Hypermedia (ED-MEDIA-96)* (pp. 697-702). Charlottesville, VA: AACE Press.

Whitney, R., & Urquhart, N. (1990). Microcomputers in the Mathematical Sciences: Effects on Courses, Students and Instructors. *Academic Computing*, Mar 90, 14-52.

Wilcox, E., Atwood, J., Burnett, M., Cadiz, J., & Cook, C. (1997). Does continuous visual feedback aid debugging in direct-manipulation programming systems? An empirical study. In *Proceedings of CHI'97* (pp. 258-265). Atlanta, GA: ACM Press.

Williams, C. J., & Brown, W. W. (1990). A review of research issues in the use of computer-related technologies for instruction: What do we know? *Journal of Instructional Media 17*(3), 213-225.

Wilson, J., Katz, I. R., Ingargiola, G., Aiken, R., & Hoskin, N. (1995). Students' use of animations for algorithm understanding. In *Proceedings CHI '95* (pp. 238-239). New York: ACM Press.

Wirth, N. (1986). *Algorithms and Data Structures.* Englewood Cliffs, NJ.: Prentice-Hall Publishing.

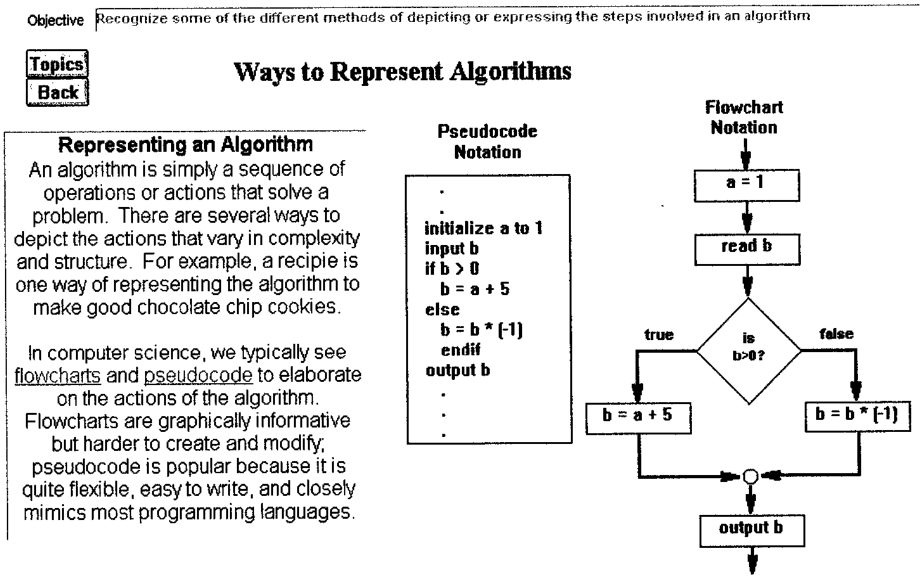
Wright, P., & Lickorish, A. (1983). Proof-reading texts on screen and paper. *Behaviour and Information Technology*, 2(3):227-235.

Yaverbaum, G., Kulkarni, M., & Wood, C. (1997). Multimedia projection: An exploratory study of student perceptions regarding interest, organization, and clarity. *Journal of Educational Multimedia and Hypermedia*, 6(2):139-153.

APPENDIX A: THE HALVIS VISUALIZATIONS

The sections that follow show all the screens of the HalVis prototype. Where possible, the screen captures were taken during execution rather than at the beginning or the end, to help the reader envision how the objects on the screen appeared to the user. The screens are ordered by algorithm and according to the natural navigational path the user was encouraged to pursue. The user was provided a flexible navigational interface that defaulted to the sequence of screens depicted in these pages, or allowed the user to deviate and navigate directly to desired topics. Most screens included a short audio welcoming and orientation track. The Detailed View and Populated View screens included audio event signals and end-of-execution prompts to encourage the user to try various other options available for enhancing the animation.

B.1 FUNDAMENTAL MODULE SCREENS



Objective Recognize some of the different methods of depicting or expressing the steps involved in an algorithm

Ways to Represent Algorithms

Representing an Algorithm

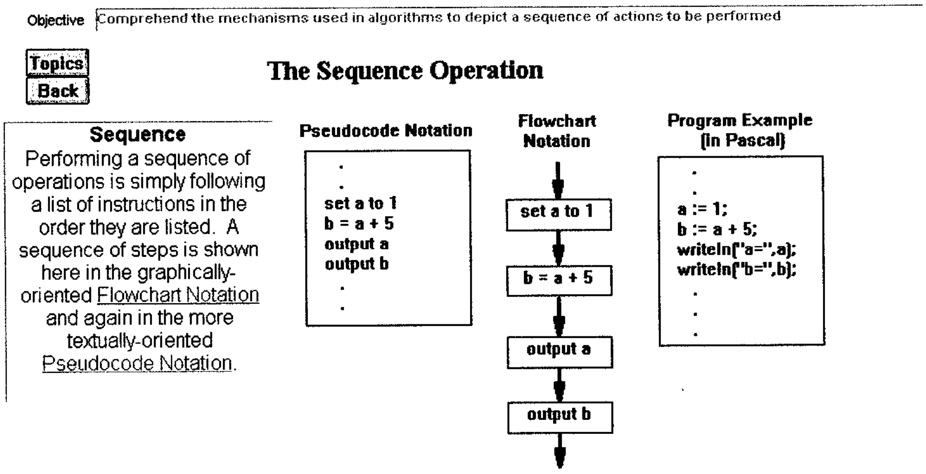
An algorithm is simply a sequence of operations or actions that solve a problem. There are several ways to depict the actions that vary in complexity and structure. For example, a recipie is one way of representing the algorithm to make good chocolate chip cookies.

In computer science, we typically see flowcharts and pseudocode to elaborate on the actions of the algorithm. Flowcharts are graphically informative but harder to create and modify; pseudocode is popular because it is quite flexible, easy to write, and closely mimics most programming languages.

Pseudocode Notation

```
.
.
initialize a to 1
input b
if b > 0
  b = a + 5
else
  b = b * (-1)
  endif
output b
.
.
.
```

Figure A.1.1. Fundamentals Module Screen- Representation



Objective Comprehend the mechanisms used in algorithms to depict a sequence of actions to be performed

The Sequence Operation

Sequence

Performing a sequence of operations is simply following a list of instructions in the order they are listed. A sequence of steps is shown here in the graphically-oriented Flowchart Notation and again in the more textually-oriented Pseudocode Notation.

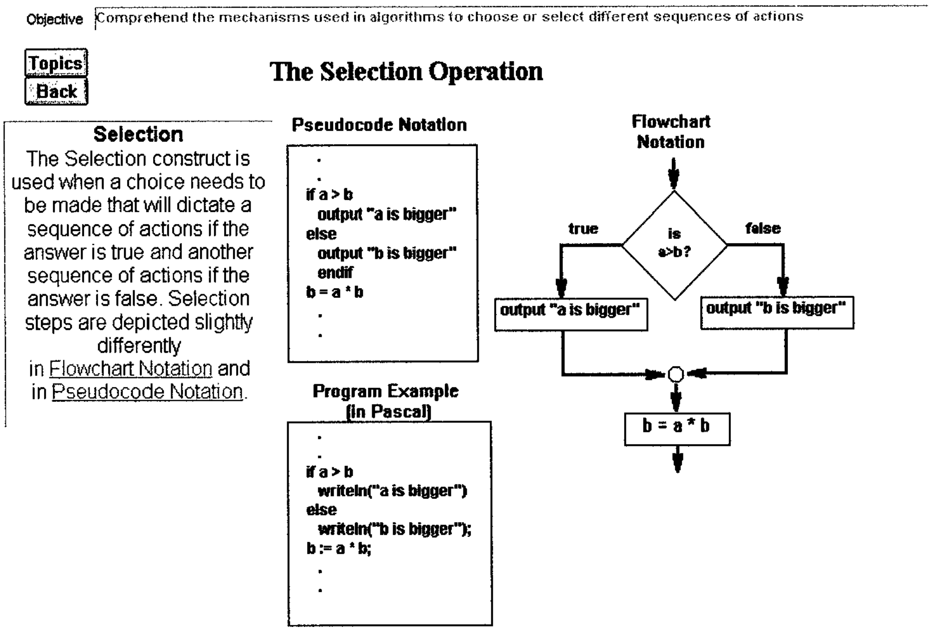
Pseudocode Notation

```
.
.
set a to 1
b = a + 5
output a
output b
.
.
```

Program Example (in Pascal)

```
.
.
a := 1;
b := a + 5;
writeln('a=',a);
writeln('b=',b);
.
.
.
```

Figure A.1.2. Fundamentals Module Screen- Sequence



Objective Comprehend the mechanisms used in algorithms to choose or select different sequences of actions

The Selection Operation

Selection

The Selection construct is used when a choice needs to be made that will dictate a sequence of actions if the answer is true and another sequence of actions if the answer is false. Selection steps are depicted slightly differently in Flowchart Notation and in Pseudocode Notation.

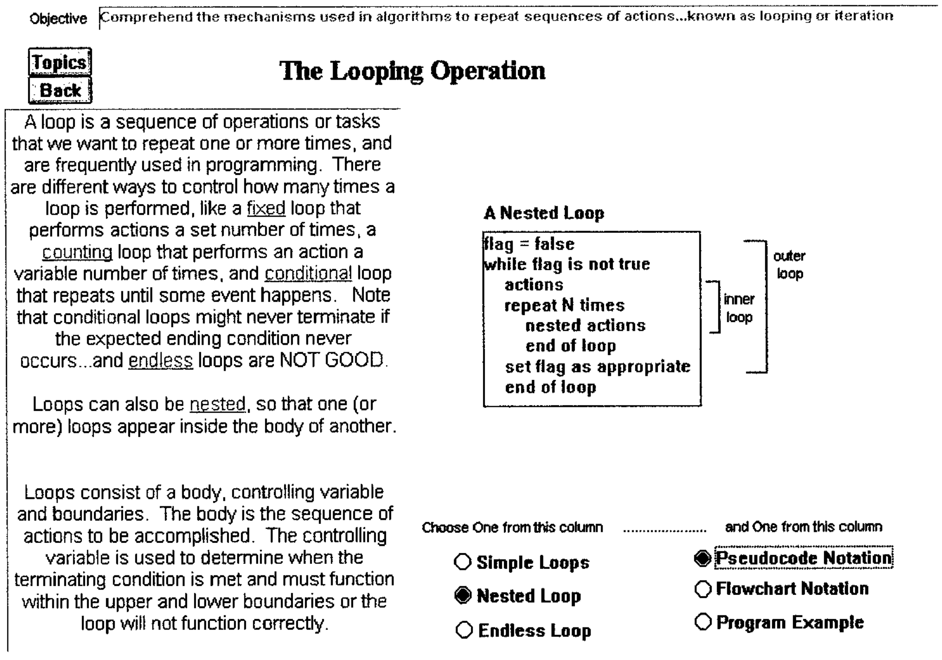
Pseudocode Notation

```
.
.
if a > b
  output "a is bigger"
else
  output "b is bigger"
  endif
b = a * b
  .
  .
```

Program Example (in Pascal)

```
  .
  .
if a > b
  writeln("a is bigger")
else
  writeln("b is bigger");
b := a * b;
  .
  .
```

Figure A.1.3. Fundamentals Module Screen- Selection



Objective Comprehend the mechanisms used in algorithms to repeat sequences of actions...known as looping or iteration

The Looping Operation

A loop is a sequence of operations or tasks that we want to repeat one or more times, and are frequently used in programming. There are different ways to control how many times a loop is performed, like a fixed loop that performs actions a set number of times, a counting loop that performs an action a variable number of times, and conditional loop that repeats until some event happens. Note that conditional loops might never terminate if the expected ending condition never occurs...and endless loops are NOT GOOD.

Loops can also be nested, so that one (or more) loops appear inside the body of another.

Loops consist of a body, controlling variable and boundaries. The body is the sequence of actions to be accomplished. The controlling variable is used to determine when the terminating condition is met and must function within the upper and lower boundaries or the loop will not function correctly.

A Nested Loop

```
flag = false
while flag is not true
   actions
   repeat N times
      nested actions
      end of loop
   set flag as appropriate
   end of loop
```

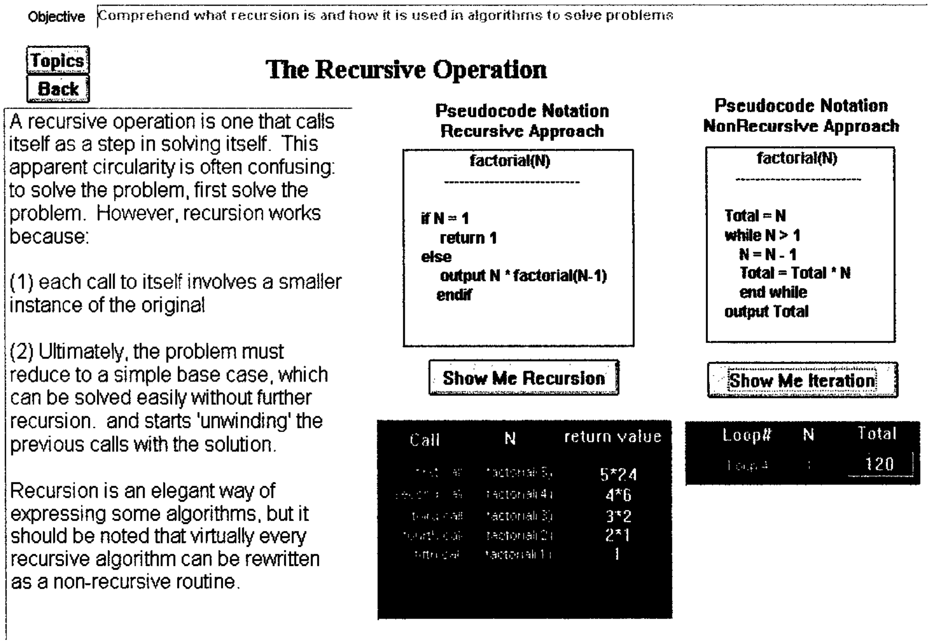
Choose One from this column .... and One from this column

○ Simple Loops ● Pseudocode Notation

● Nested Loop ○ Flowchart Notation

○ Endless Loop ○ Program Example

Figure A.1.4. Fundamentals Module Screen- Iteration



Objective Comprehend what recursion is and how it is used in algorithms to solve problems

The Recursive Operation

A recursive operation is one that calls itself as a step in solving itself. This apparent circularity is often confusing: to solve the problem, first solve the problem. However, recursion works because:

(1) each call to itself involves a smaller instance of the original

(2) Ultimately, the problem must reduce to a simple base case, which can be solved easily without further recursion. and starts 'unwinding' the previous calls with the solution.

Recursion is an elegant way of expressing some algorithms, but it should be noted that virtually every recursive algorithm can be rewritten as a non-recursive routine.

Pseudocode Notation
Recursivc Approach

```
factorial(N)
--------------------------

if N = 1
  return 1
else
  output N * factorial(N-1)
  endif
```

Pseudocode Notation
NonRecursive Approach

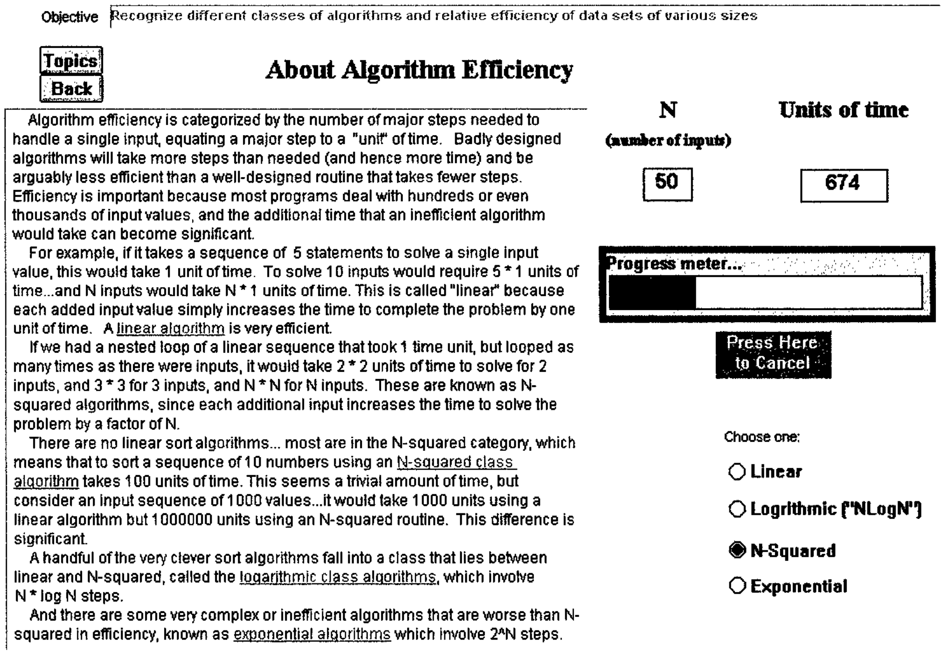
```
factorial(N)
--------------------------

Total = N
while N > 1
  N = N - 1
  Total = Total * N
  end while
output Total
```

Call	N	return value
first call	factorial(5)	5*24
second call	factorial(4)	4*6
third call	factorial(3)	3*2
fourth call	factorial(2)	2*1
fifth call	factorial(1)	1

Loop#	N	Total
Loop 4	1	120

Figure A.1.5. Fundamentals Module Screen- Recursion



Objective Recognize different classes of algorithms and relative efficiency of data sets of various sizes

About Algorithm Efficiency

Algorithm efficiency is categorized by the number of major steps needed to handle a single input, equating a major step to a "unit" of time. Badly designed algorithms will take more steps than needed (and hence more time) and be arguably less efficient than a well-designed routine that takes fewer steps. Efficiency is important because most programs deal with hundreds or even thousands of input values, and the additional time that an inefficient algorithm would take can become significant.

For example, if it takes a sequence of 5 statements to solve a single input value, this would take 1 unit of time. To solve 10 inputs would require 5 * 1 units of time...and N inputs would take N * 1 units of time. This is called "linear" because each added input value simply increases the time to complete the problem by one unit of time. A linear algorithm is very efficient.

If we had a nested loop of a linear sequence that took 1 time unit, but looped as many times as there were inputs, it would take 2 * 2 units of time to solve for 2 inputs, and 3 * 3 for 3 inputs, and N * N for N inputs. These are known as N-squared algorithms, since each additional input increases the time to solve the problem by a factor of N.

There are no linear sort algorithms... most are in the N-squared category, which means that to sort a sequence of 10 numbers using an N-squared class algorithm takes 100 units of time. This seems a trivial amount of time, but consider an input sequence of 1000 values...it would take 1000 units using a linear algorithm but 1000000 units using an N-squared routine. This difference is significant.

A handful of the very clever sort algorithms fall into a class that lies between linear and N-squared, called the logarithmic class algorithms, which involve N * log N steps.

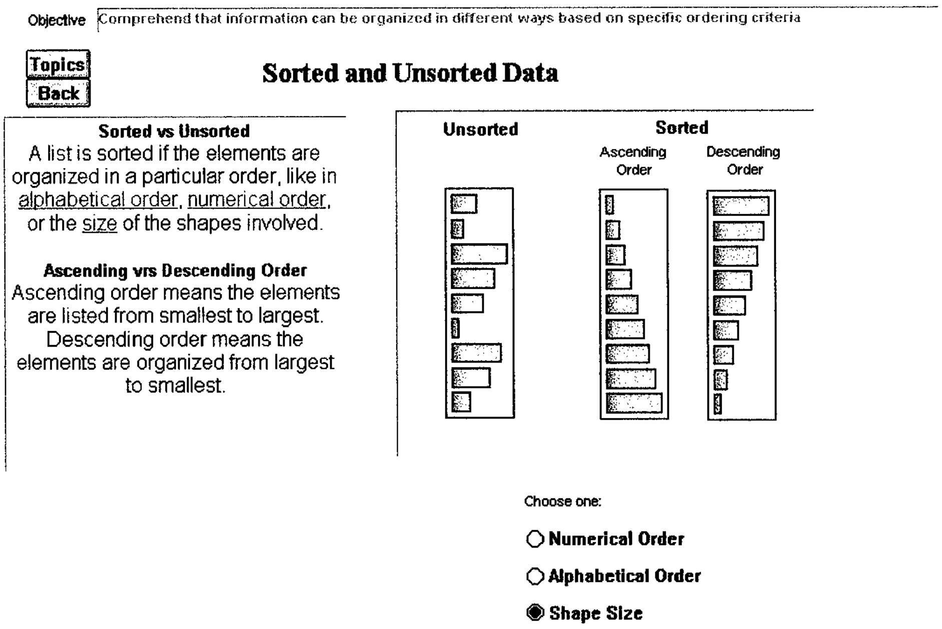
And there are some very complex or inefficient algorithms that are worse than N-squared in efficiency, known as exponential algorithms which involve 2^N steps.

N (number of inputs)	Units of time
50	674

Choose one:

○ Linear
○ Logrithmic ("NLogN")
● N-Squared
○ Exponential

Figure A.1.6. Fundamentals Module Screen- Efficiency



Objective Comprehend that information can be organized in different ways based on specific ordering criteria

Sorted and Unsorted Data

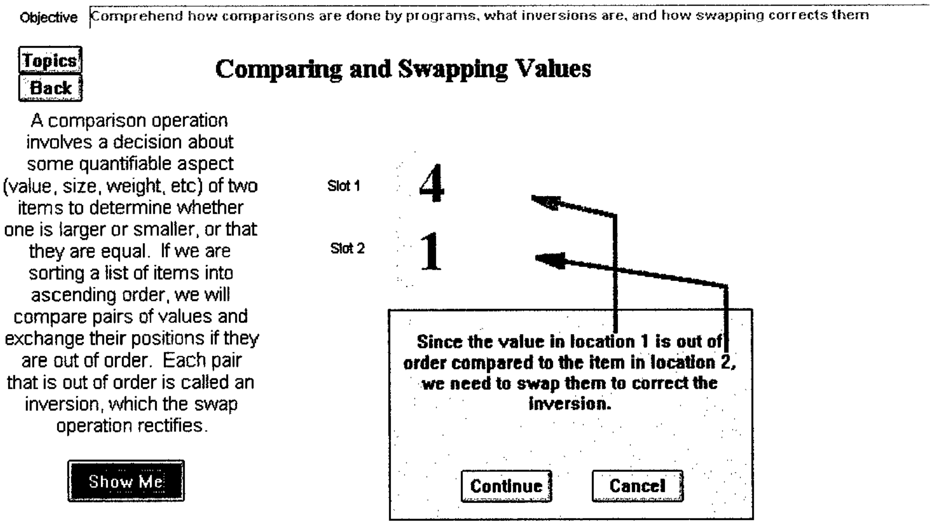
Sorted vs Unsorted
A list is sorted if the elements are organized in a particular order, like in alphabetical order, numerical order, or the size of the shapes involved.

Ascending vrs Descending Order
Ascending order means the elements are listed from smallest to largest. Descending order means the elements are organized from largest to smallest.

Choose one:

○ Numerical Order
○ Alphabetical Order
● Shape Size

Figure A.1.7. Fundamentals Module Screen- Sorted Data



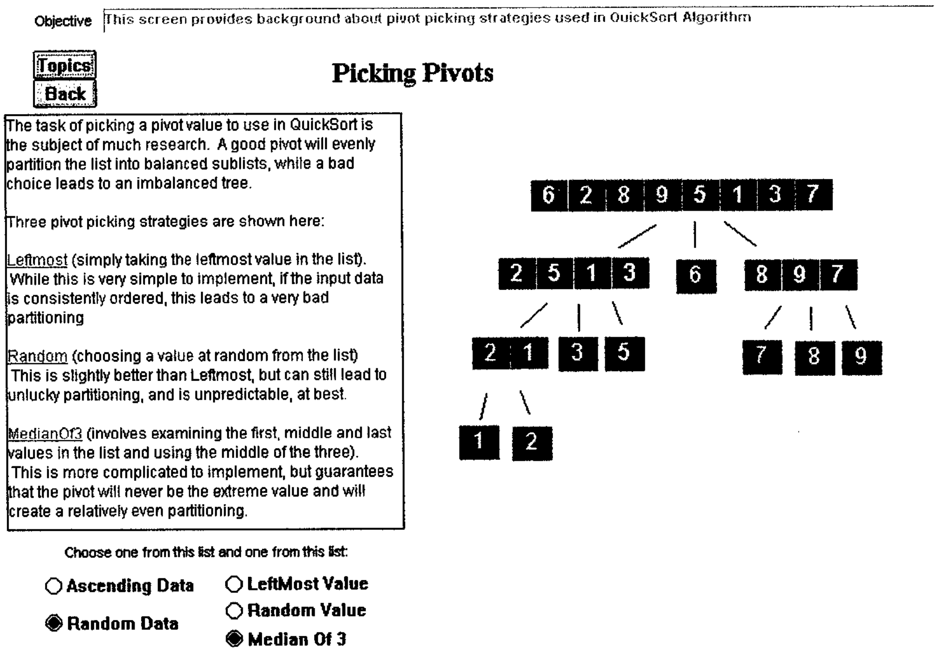
Objective Comprehend how comparisons are done by programs, what inversions are, and how swapping corrects them

Comparing and Swapping Values

A comparison operation involves a decision about some quantifiable aspect (value, size, weight, etc) of two items to determine whether one is larger or smaller, or that they are equal. If we are sorting a list of items into ascending order, we will compare pairs of values and exchange their positions if they are out of order. Each pair that is out of order is called an inversion, which the swap operation rectifies.

Since the value in location 1 is out of order compared to the item in location 2, we need to swap them to correct the inversion.

Figure A.1.8. Fundamentals Module Screen- Swapping



Objective This screen provides background about pivot picking strategies used in QuickSort Algorithm

Picking Pivots

The task of picking a pivot value to use in QuickSort is the subject of much research. A good pivot will evenly partition the list into balanced sublists, while a bad choice leads to an imbalanced tree.

Three pivot picking strategies are shown here:

Leftmost (simply taking the leftmost value in the list). While this is very simple to implement, if the input data is consistently ordered, this leads to a very bad partitioning

Random (choosing a value at random from the list) This is slightly better than Leftmost, but can still lead to unlucky partitioning, and is unpredictable, at best.

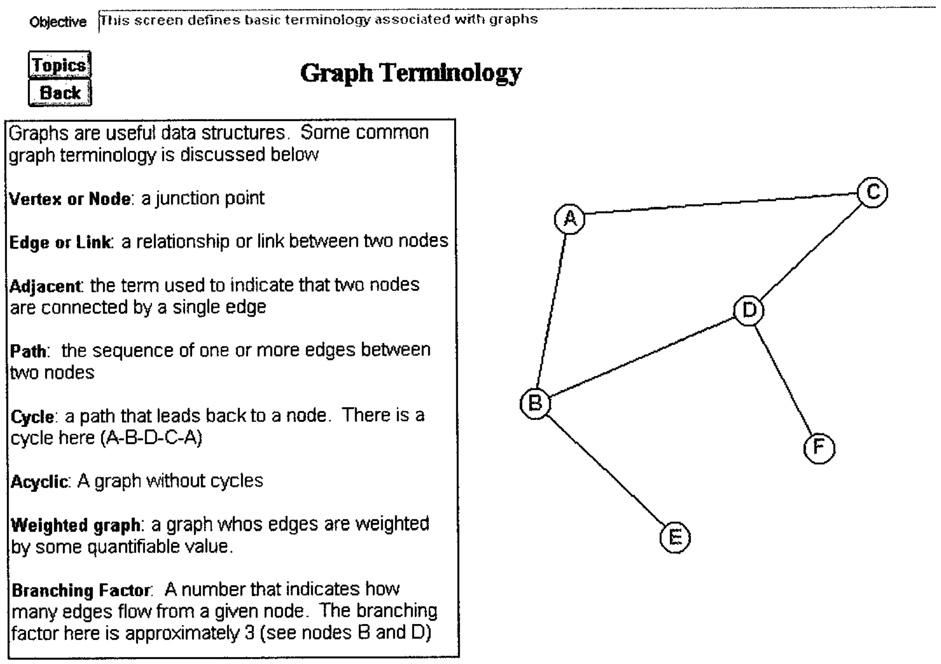
MedianOf3 (involves examining the first, middle and last values in the list and using the middle of the three). This is more complicated to implement, but guarantees that the pivot will never be the extreme value and will create a relatively even partitioning.

Choose one from this list and one from this list:

○ Ascending Data
● Random Data

○ LeftMost Value
○ Random Value
● Median Of 3

Figure A.1.9. Fundamentals Module Screen- Picking Pivots



Objective This screen defines basic terminology associated with graphs

Graph Terminology

Graphs are useful data structures. Some common graph terminology is discussed below

Vertex or Node: a junction point

Edge or Link: a relationship or link between two nodes

Adjacent: the term used to indicate that two nodes are connected by a single edge

Path: the sequence of one or more edges between two nodes

Cycle: a path that leads back to a node. There is a cycle here (A-B-D-C-A)

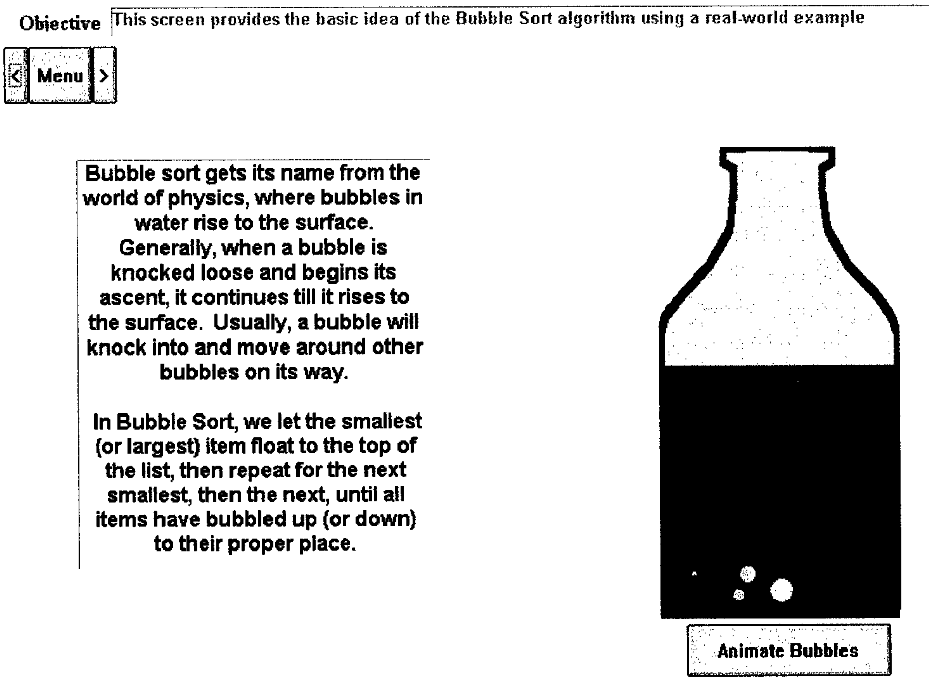
Acyclic: A graph without cycles

Weighted graph: a graph whos edges are weighted by some quantifiable value.

Branching Factor: A number that indicates how many edges flow from a given node. The branching factor here is approximately 3 (see nodes B and D)

Figure A.1.10. Fundamentals Module Screen- Graph Terms

B.2 BUBBLE SORT

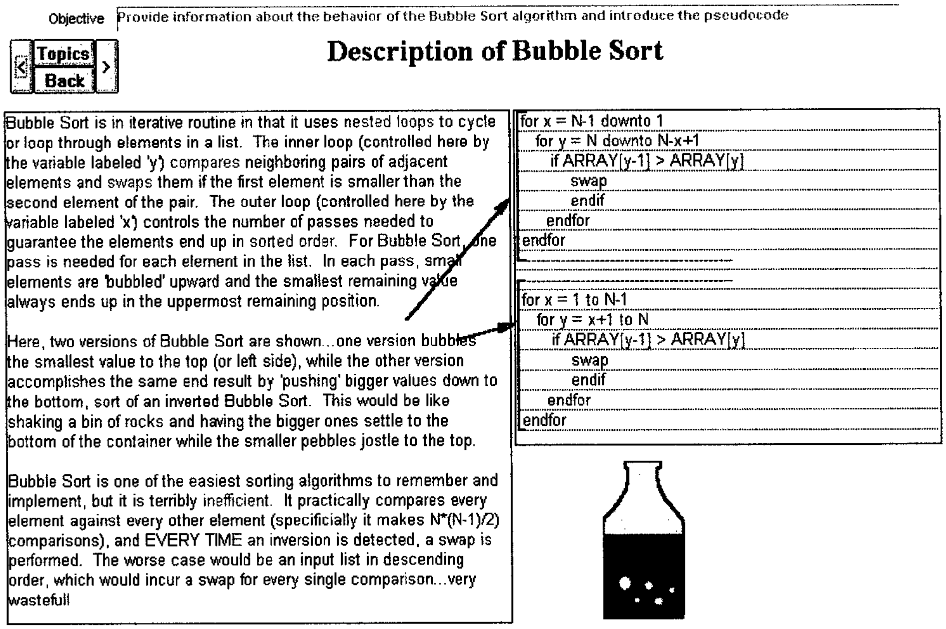


Objective This screen provides the basic idea of the Bubble Sort algorithm using a real-world example

Bubble sort gets its name from the world of physics, where bubbles in water rise to the surface. Generally, when a bubble is knocked loose and begins its ascent, it continues till it rises to the surface. Usually, a bubble will knock into and move around other bubbles on its way.

In Bubble Sort, we let the smallest (or largest) item float to the top of the list, then repeat for the next smallest, then the next, until all items have bubbled up (or down) to their proper place.

Figure A.2.1 Bubble Sort Algorithm Conceptual View Screen



Objective Provide information about the behavior of the Bubble Sort algorithm and introduce the pseudocode

Description of Bubble Sort

Bubble Sort is in iterative routine in that it uses nested loops to cycle or loop through elements in a list. The inner loop (controlled here by the variable labeled 'y') compares neighboring pairs of adjacent elements and swaps them if the first element is smaller than the second element of the pair. The outer loop (controlled here by the variable labeled 'x') controls the number of passes needed to guarantee the elements end up in sorted order. For Bubble Sort, one pass is needed for each element in the list. In each pass, small elements are 'bubbled' upward and the smallest remaining value always ends up in the uppermost remaining position.

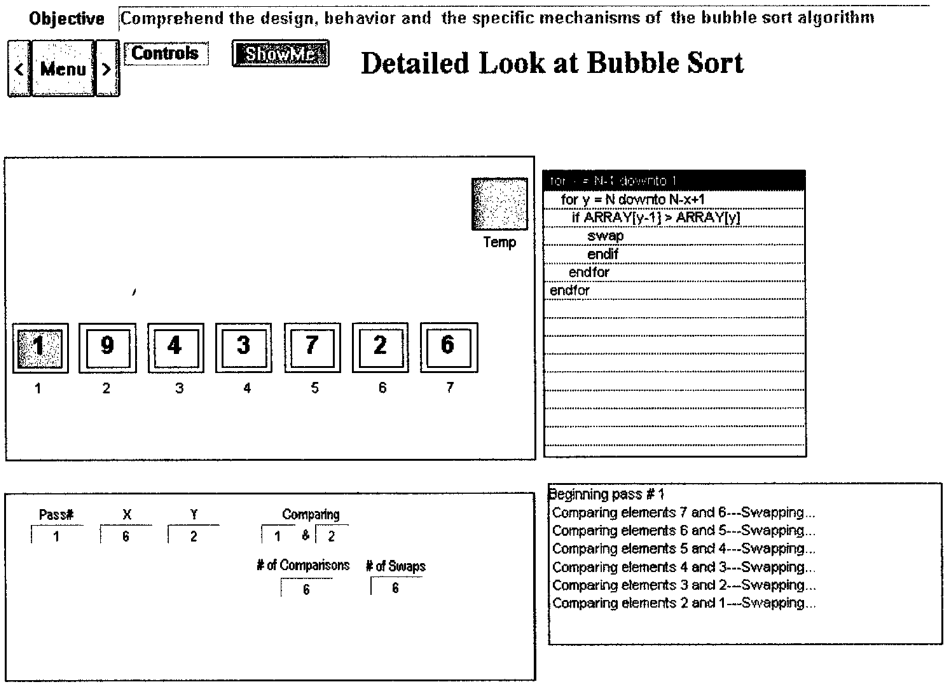
Here, two versions of Bubble Sort are shown...one version bubbles the smallest value to the top (or left side), while the other version accomplishes the same end result by 'pushing' bigger values down to the bottom, sort of an inverted Bubble Sort. This would be like shaking a bin of rocks and having the bigger ones settle to the bottom of the container while the smaller pebbles jostle to the top.

Bubble Sort is one of the easiest sorting algorithms to remember and implement, but it is terribly inefficient. It practically compares every element against every other element (specificially it makes N*(N-1)/2) comparisons), and EVERY TIME an inversion is detected, a swap is performed. The worse case would be an input list in descending order, which would incur a swap for every single comparison...very wastefull

```
for x = N-1 downto 1
  for y = N downto N-x+1
    if ARRAY[y-1] > ARRAY[y]
      swap
      endif
   endfor
endfor
```

```
for x = 1 to N-1
  for y = x+1 to N
    if ARRAY[y-1] > ARRAY[y]
      swap
      endif
   endfor
endfor
```

Figure A.2.2. Bubble Sort Algorithm Description Screen



Objective Comprehend the design, behavior and the specific mechanisms of the bubble sort algorithm

Detailed Look at Bubble Sort

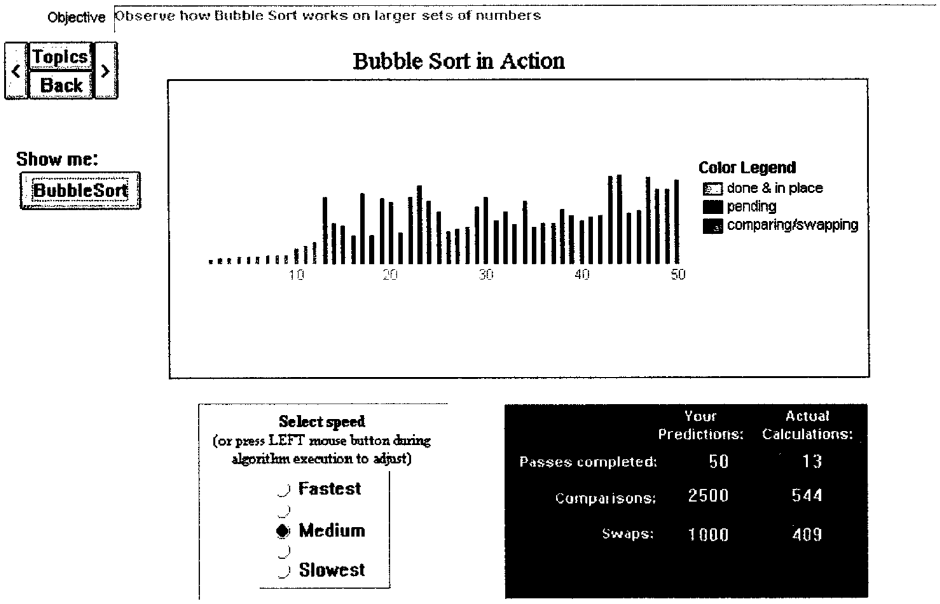
```
for x = N-1 downto 1
   for y = N downto N-x+1
      if ARRAY[y-1} > ARRAY[y]
         swap
         endif
   endfor
endfor
```

Pass# 1 X 6 Y 2 Comparing 1 & 2

# of Comparisons 6 # of Swaps 6

Beginning pass # 1
Comparing elements 7 and 6---Swapping...
Comparing elements 6 and 5---Swapping...
Comparing elements 5 and 4---Swapping...
Comparing elements 4 and 3---Swapping...
Comparing elements 3 and 2---Swapping...
Comparing elements 2 and 1---Swapping...

Figure A.2.3. Bubble Sort Algorithm Detailed View Screen



Objective Observe how Bubble Sort works on larger sets of numbers

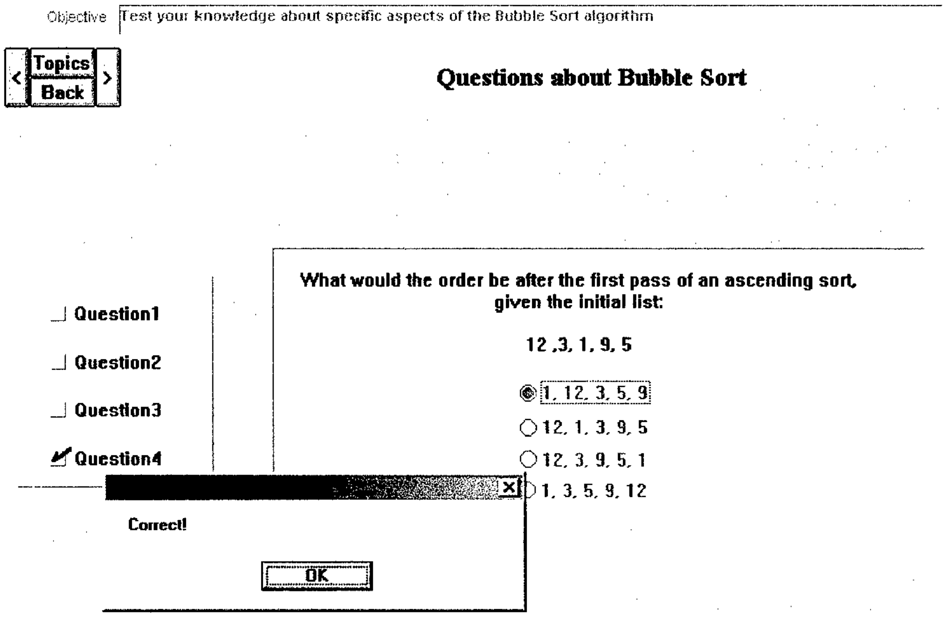
Bubble Sort in Action

Color Legend
▭ done & in place
■ pending
▨ comparing/swapping

Select speed
(or press LEFT mouse button during algorithm execution to adjust)
○ Fastest
○
● Medium
○
○ Slowest

	Your Predictions:	Actual Calculations:
Passes completed:	50	13
Comparisons:	2500	544
Swaps:	1000	409

Figure A.2.4. Bubble Sort Algorithm Populated View Screen



Objective: Test your knowledge about specific aspects of the Bubble Sort algorithm

Questions about Bubble Sort

What would the order be after the first pass of an ascending sort, given the initial list:

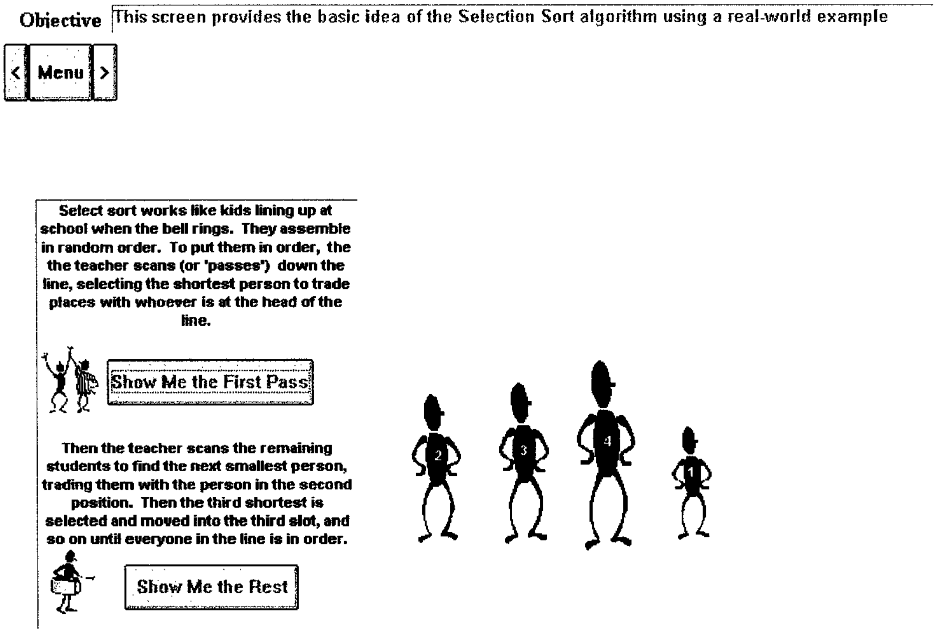
12 ,3, 1, 9, 5

● 1, 12, 3, 5, 9
○ 12, 1, 3, 9, 5
○ 12, 3, 9, 5, 1
○ 1, 3, 5, 9, 12

Correct!

Figure A.2.5. Bubble Sort Algorithm Question Screen

B.3 SELECTION SORT

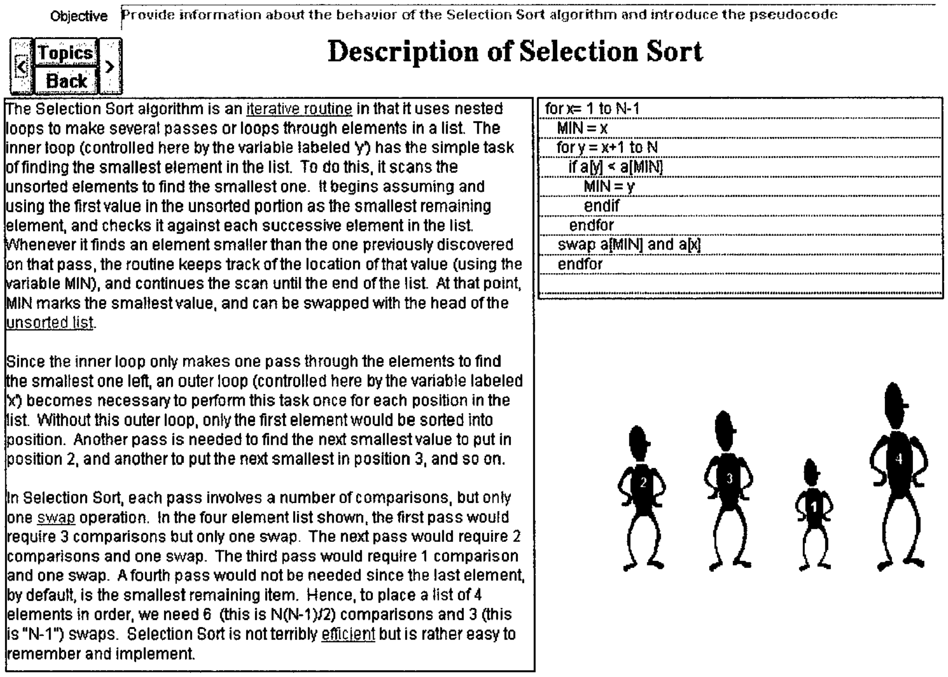


Objective This screen provides the basic idea of the Selection Sort algorithm using a real-world example

Select sort works like kids lining up at school when the bell rings. They assemble in random order. To put them in order, the the teacher scans (or 'passes') down the line, selecting the shortest person to trade places with whoever is at the head of the line.

Then the teacher scans the remaining students to find the next smallest person, trading them with the person in the second position. Then the third shortest is selected and moved into the third slot, and so on until everyone in the line is in order.

Figure A.3.1 Selection Sort Algorithm Conceptual View Screen



Objective Provide information about the behavior of the Selection Sort algorithm and introduce the pseudocode

Description of Selection Sort

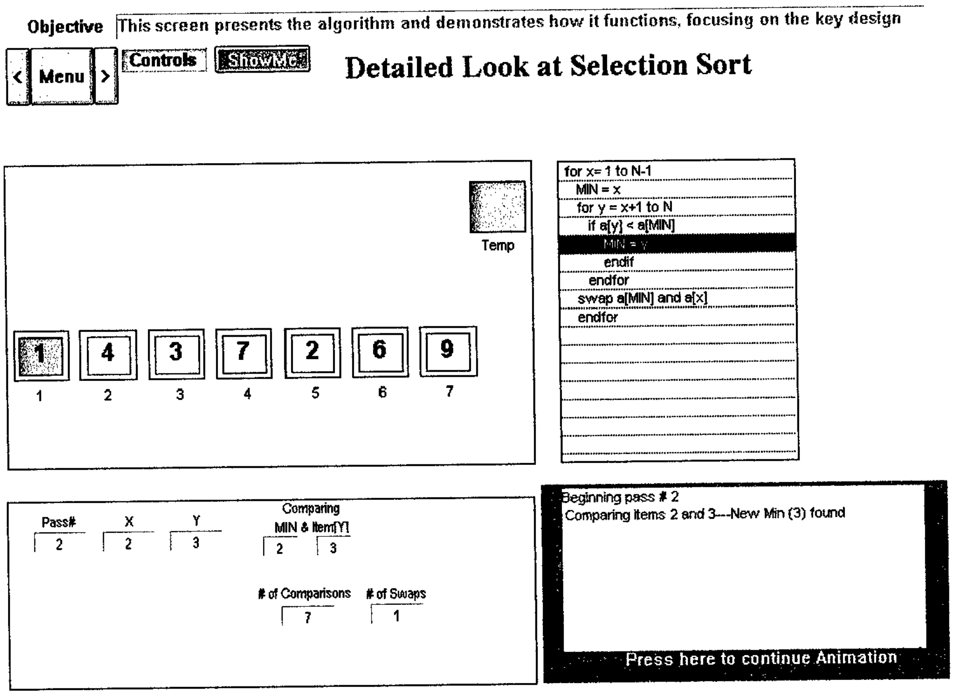
The Selection Sort algorithm is an iterative routine in that it uses nested loops to make several passes or loops through elements in a list. The inner loop (controlled here by the variable labeled 'y') has the simple task of finding the smallest element in the list. To do this, it scans the unsorted elements to find the smallest one. It begins assuming and using the first value in the unsorted portion as the smallest remaining element, and checks it against each successive element in the list. Whenever it finds an element smaller than the one previously discovered on that pass, the routine keeps track of the location of that value (using the variable MIN), and continues the scan until the end of the list. At that point, MIN marks the smallest value, and can be swapped with the head of the unsorted list.

Since the inner loop only makes one pass through the elements to find the smallest one left, an outer loop (controlled here by the variable labeled 'x') becomes necessary to perform this task once for each position in the list. Without this outer loop, only the first element would be sorted into position. Another pass is needed to find the next smallest value to put in position 2, and another to put the next smallest in position 3, and so on.

In Selection Sort, each pass involves a number of comparisons, but only one swap operation. In the four element list shown, the first pass would require 3 comparisons but only one swap. The next pass would require 2 comparisons and one swap. The third pass would require 1 comparison and one swap. A fourth pass would not be needed since the last element, by default, is the smallest remaining item. Hence, to place a list of 4 elements in order, we need 6 (this is N(N-1)/2) comparisons and 3 (this is "N-1") swaps. Selection Sort is not terribly efficient but is rather easy to remember and implement.

```
for x= 1 to N-1
  MIN = x
  for y = x+1 to N
    if a[y] < a[MIN]
      MIN = y
      endif
    endfor
  swap a[MIN] and a[x]
  endfor
```

Figure A.3.2. Selection Sort Algorithm Description Screen



Objective: This screen presents the algorithm and demonstrates how it functions, focusing on the key design

Detailed Look at Selection Sort

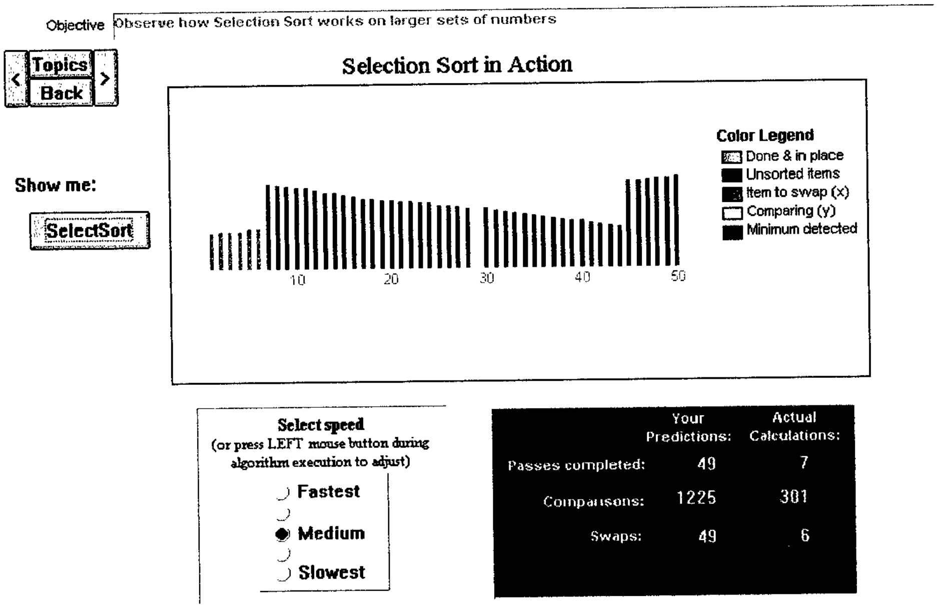
```
for x= 1 to N-1
  MIN = x
  for y = x+1 to N
    if a[y] < a[MIN]
      MIN = y
      endif
    endfor
  swap a[MIN] and a[x]
  endfor
```

Pass#	X	Y	Comparing MIN & Item[Y]	
2	2	3	2	3

# of Comparisons	# of Swaps
7	1

Beginning pass # 2
Comparing items 2 and 3---New Min (3) found

Figure A.3.3. Selection Sort Algorithm Detailed View Screen



Objective: Observe how Selection Sort works on larger sets of numbers

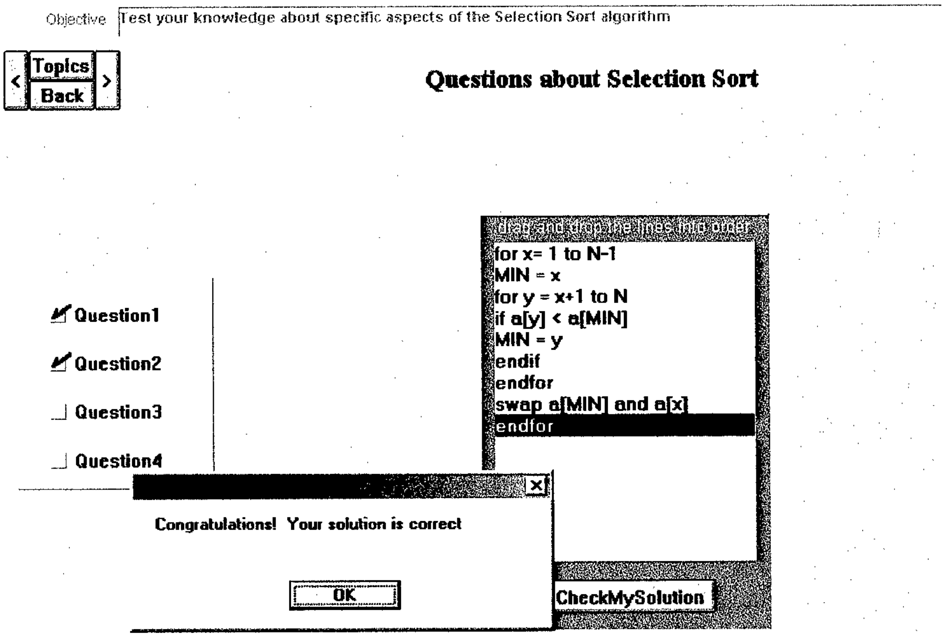
Selection Sort in Action

Select speed
(or press LEFT mouse button during algorithm execution to adjust)

○ Fastest
○
● Medium
○
○ Slowest

	Your Predictions:	Actual Calculations:
Passes completed:	49	7
Comparisons:	1225	301
Swaps:	49	6

Figure A.3.4. Selection Sort Algorithm Populated View Screen



Objective: Test your knowledge about specific aspects of the Selection Sort algorithm

Questions about Selection Sort

☑ Question1

☑ Question2

□ Question3

□ Question4

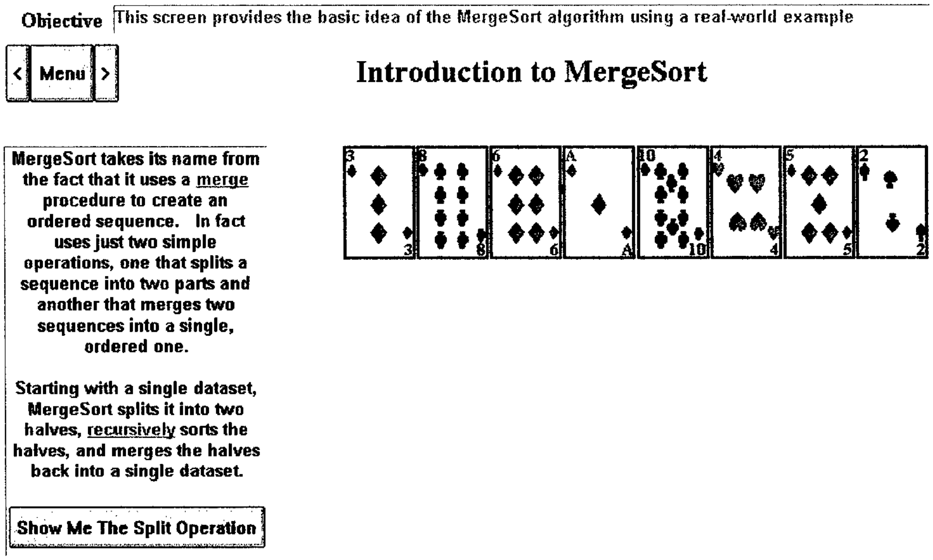
drag and drop the lines into order

```
for x= 1 to N-1
MIN = x
for y = x+1 to N
if a[y] < a[MIN]
MIN = y
endif
endfor
swap a[MIN] and a[x]
endfor
```

Congratulations! Your solution is correct

Figure A.3.5. Selection Sort Algorithm Question Screen

B.4 MERGE SORT



Objective This screen provides the basic idea of the MergeSort algorithm using a real-world example

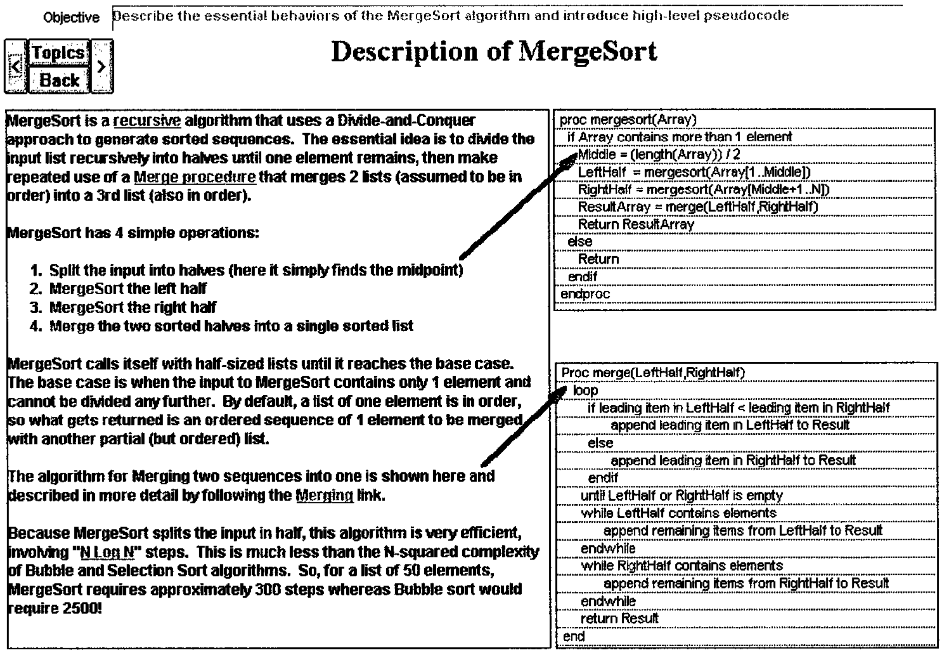
Introduction to MergeSort

MergeSort takes its name from the fact that it uses a merge procedure to create an ordered sequence. In fact uses just two simple operations, one that splits a sequence into two parts and another that merges two sequences into a single, ordered one.

Starting with a single dataset, MergeSort splits it into two halves, recursively sorts the halves, and merges the halves back into a single dataset.

Show Me The Split Operation

Figure A.4.1 MergeSort Algorithm Conceptual View Screen



Objective Describe the essential behaviors of the MergeSort algorithm and introduce high-level pseudocode

Description of MergeSort

MergeSort is a recursive algorithm that uses a Divide-and-Conquer approach to generate sorted sequences. The essential idea is to divide the input list recursively into halves until one element remains, then make repeated use of a Merge procedure that merges 2 lists (assumed to be in order) into a 3rd list (also in order).

MergeSort has 4 simple operations:

1. Split the input into halves (here it simply finds the midpoint)
2. MergeSort the left half
3. MergeSort the right half
4. Merge the two sorted halves into a single sorted list

MergeSort calls itself with half-sized lists until it reaches the base case. The base case is when the input to MergeSort contains only 1 element and cannot be divided any further. By default, a list of one element is in order, so what gets returned is an ordered sequence of 1 element to be merged with another partial (but ordered) list.

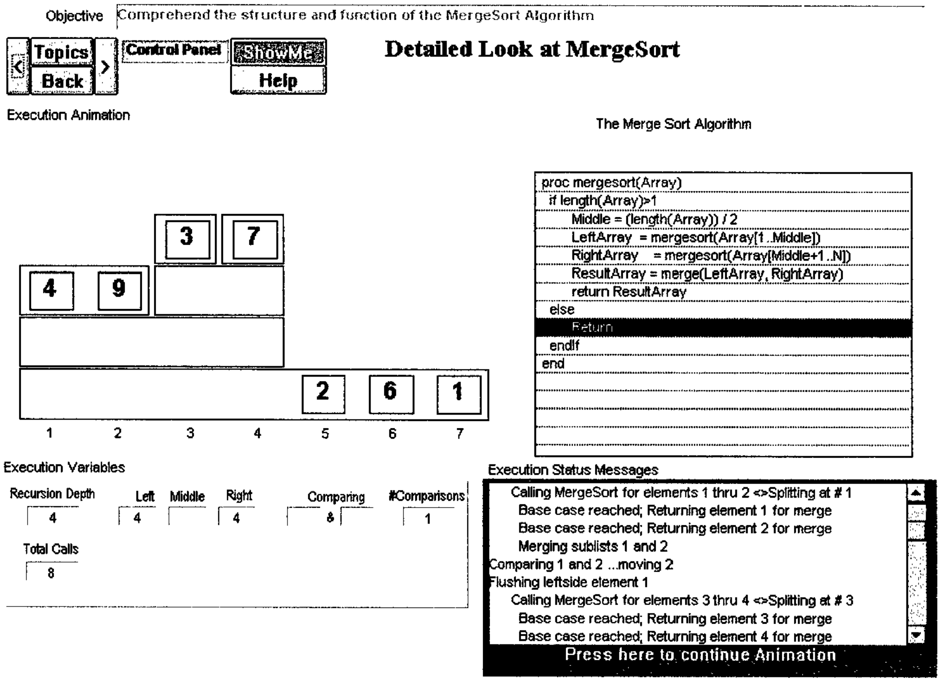
The algorithm for Merging two sequences into one is shown here and described in more detail by following the Merging link.

Because MergeSort splits the input in half, this algorithm is very efficient, involving "N Log N" steps. This is much less than the N-squared complexity of Bubble and Selection Sort algorithms. So, for a list of 50 elements, MergeSort requires approximately 300 steps whereas Bubble sort would require 2500!

```
proc mergesort(Array)
 if Array contains more than 1 element
   Middle = (length(Array)) / 2
   LeftHalf  = mergesort(Array[1 ..Middle])
   RightHalf = mergesort(Array[Middle+1 ..N])
   ResultArray = merge(LeftHalf,RightHalf)
   Return ResultArray
 else
   Return
 endif
endproc
```

```
Proc merge(LeftHalf,RightHalf)
  loop
     if leading item in LeftHalf < leading item in RightHalf
          append leading item in LeftHalf to Result
     else
          append leading item in RightHalf to Result
     endif
   until LeftHalf or RightHalf is empty
   while LeftHalf contains elements
        append remaining items from LeftHalf to Result
   endwhile
   while RightHalf contains elements
        append remaining items from RightHalf to Result
   endwhile
   return Result
end
```

Figure A.4.2. MergeSort Algorithm Description Screen



Objective: Comprehend the structure and function of the MergeSort Algorithm

Detailed Look at MergeSort

The Merge Sort Algorithm

```
proc mergesort(Array)
  if length(Array)>1
      Middle = (length(Array)) / 2
      LeftArray  = mergesort(Array[1..Middle])
      RightArray    = mergesort(Array[Middle+1..N])
      ResultArray = merge(LeftArray, RightArray)
      return ResultArray
  else
      Return
  endif
end
```

Execution Variables

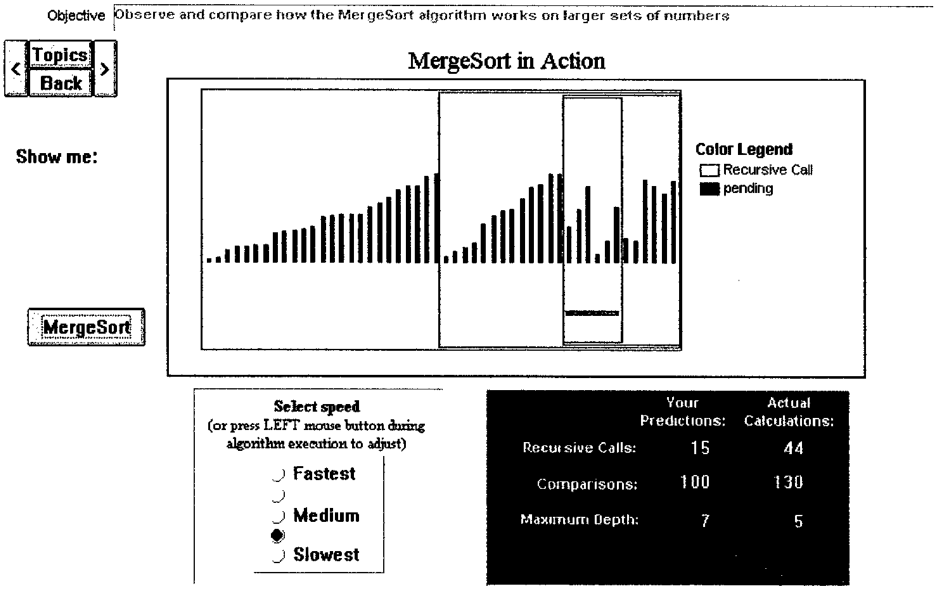
Recursion Depth	Left	Middle	Right	Comparing	#Comparisons
4	4		4	&	1

Total Calls: 8

Execution Status Messages

```
  Calling MergeSort for elements 1 thru 2 <>Splitting at # 1
    Base case reached; Returning element 1 for merge
    Base case reached; Returning element 2 for merge
    Merging sublists 1 and 2
Comparing 1 and 2 ...moving 2
Flushing leftside element 1
  Calling MergeSort for elements 3 thru 4 <>Splitting at # 3
    Base case reached; Returning element 3 for merge
    Base case reached; Returning element 4 for merge
```

Figure A.4.3. MergeSort Algorithm Detailed View Screen



Objective: Observe and compare how the MergeSort algorithm works on larger sets of numbers

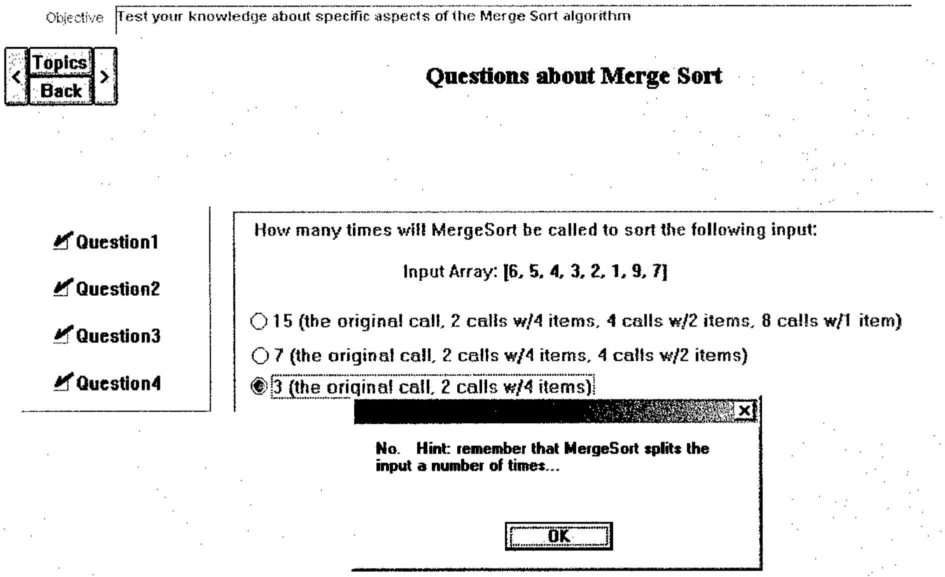
MergeSort in Action

Select speed
(or press LEFT mouse button during algorithm execution to adjust)

○ Fastest
○
○ Medium
●
○ Slowest

	Your Predictions:	Actual Calculations:
Recursive Calls:	15	44
Comparisons:	100	130
Maximum Depth:	7	5

Figure A.4.4. MergeSort Algorithm Populated View Screen



Test your knowledge about specific aspects of the Merge Sort algorithm

Questions about Merge Sort

How many times will MergeSort be called to sort the following input:

Input Array: [6, 5, 4, 3, 2, 1, 9, 7]

○ 15 (the original call, 2 calls w/4 items, 4 calls w/2 items, 8 calls w/1 item)

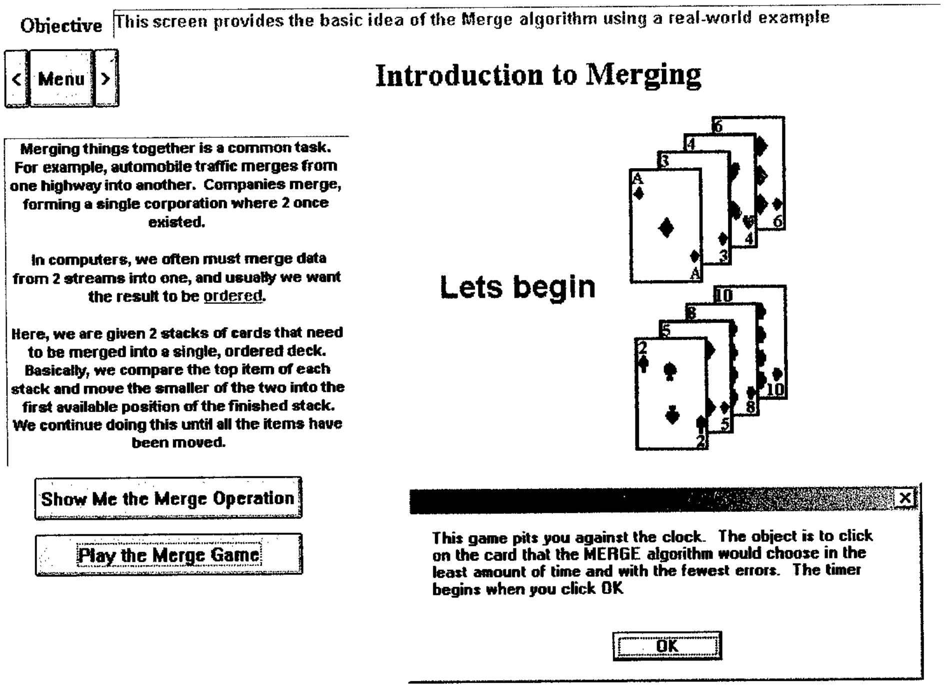
○ 7 (the original call, 2 calls w/4 items, 4 calls w/2 items)

● 3 (the original call, 2 calls w/4 items)

No. Hint: remember that MergeSort splits the input a number of times...

Figure A.4.5. MergeSort Algorithm Question Screen

B.5 MERGE ALGORITHM



Objective This screen provides the basic idea of the Merge algorithm using a real-world example

Introduction to Merging

Merging things together is a common task. For example, automobile traffic merges from one highway into another. Companies merge, forming a single corporation where 2 once existed.

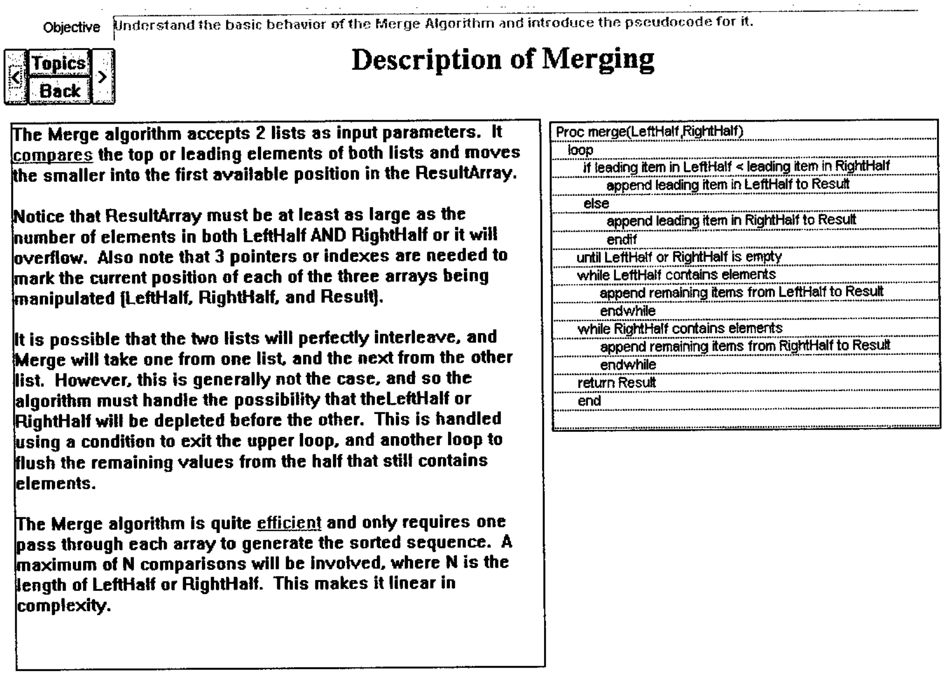
In computers, we often must merge data from 2 streams into one, and usually we want the result to be ordered.

Here, we are given 2 stacks of cards that need to be merged into a single, ordered deck. Basically, we compare the top item of each stack and move the smaller of the two into the first available position of the finished stack. We continue doing this until all the items have been moved.

Lets begin

This game pits you against the clock. The object is to click on the card that the MERGE algorithm would choose in the least amount of time and with the fewest errors. The timer begins when you click OK

Figure A.5.1 Merge Algorithm Conceptual View Screen



Objective Understand the basic behavior of the Merge Algorithm and introduce the pseudocode for it.

Description of Merging

The Merge algorithm accepts 2 lists as input parameters. It compares the top or leading elements of both lists and moves the smaller into the first available position in the ResultArray.

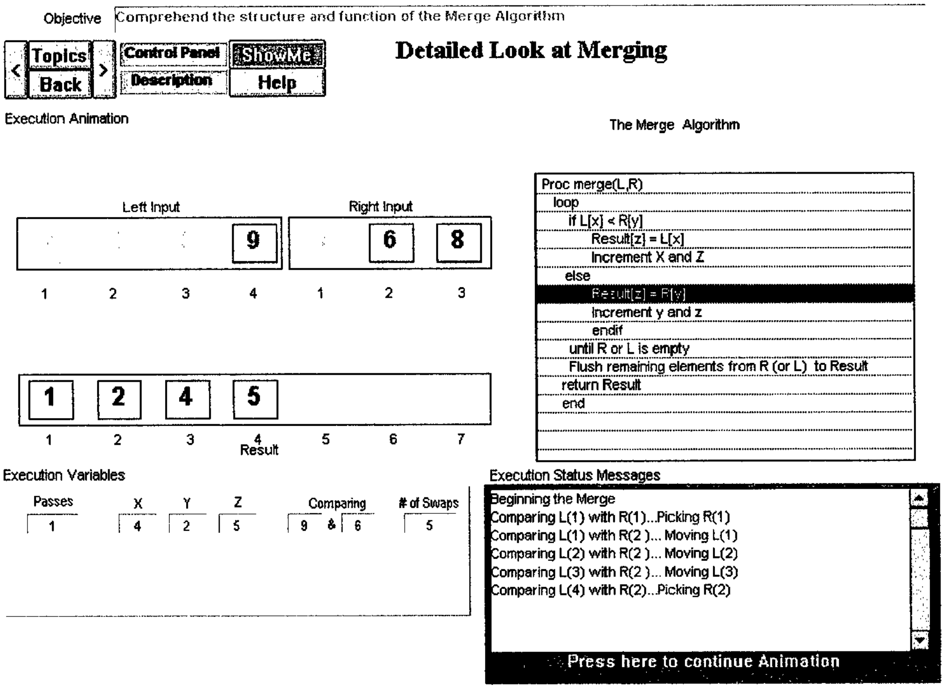
Notice that ResultArray must be at least as large as the number of elements in both LeftHalf AND RightHalf or it will overflow. Also note that 3 pointers or indexes are needed to mark the current position of each of the three arrays being manipulated (LeftHalf, RightHalf, and Result).

It is possible that the two lists will perfectly interleave, and Merge will take one from one list, and the next from the other list. However, this is generally not the case, and so the algorithm must handle the possibility that theLeftHalf or RightHalf will be depleted before the other. This is handled using a condition to exit the upper loop, and another loop to flush the remaining values from the half that still contains elements.

The Merge algorithm is quite efficient and only requires one pass through each array to generate the sorted sequence. A maximum of N comparisons will be involved, where N is the length of LeftHalf or RightHalf. This makes it linear in complexity.

```
Proc merge(LeftHalf,RightHalf)
  loop
     if leading item in LeftHalf < leading item in RightHalf
         append leading item in LeftHalf to Result
     else
         append leading item in RightHalf to Result
         endif
  until LeftHalf or RightHalf is empty
  while LeftHalf contains elements
        append remaining items from LeftHalf to Result
        endwhile
  while RightHalf contains elements
        append remaining items from RightHalf to Result
        endwhile
  return Result
  end
```

Figure A.5.2. Merge Algorithm Description Screen



Objective: Comprehend the structure and function of the Merge Algorithm

```
Proc merge(L,R)
 loop
   if L[x] < R[y]
      Result[z] = L[x]
      Increment X and Z
   else
      Result[z] = R[y]
      Increment y and z
      endif
   until R or L is empty
   Flush remaining elements from R (or L) to Result
 return Result
 end
```

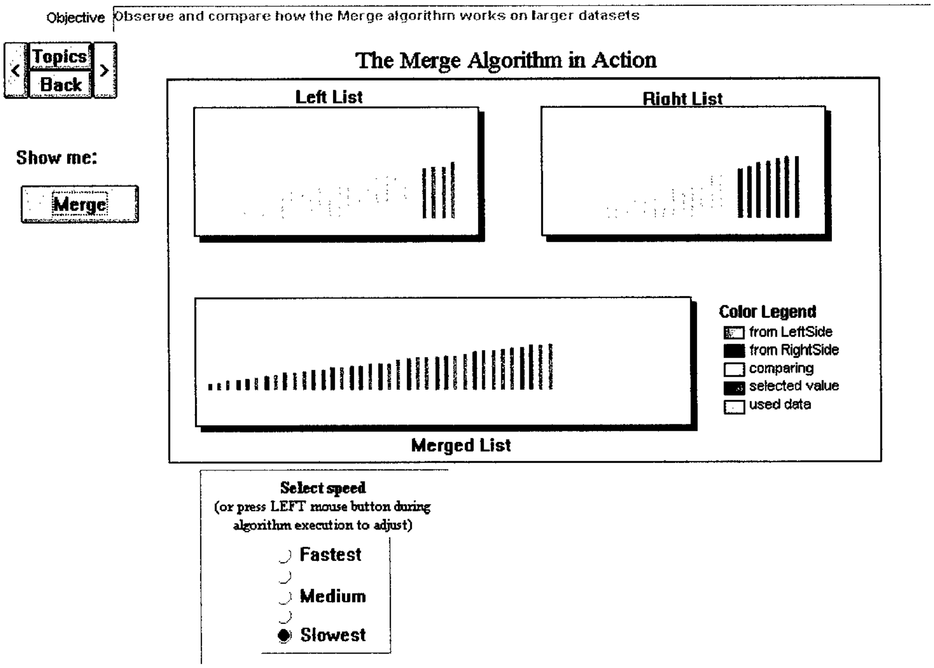
Execution Variables

Passes	X	Y	Z	Comparing	# of Swaps
1	4	2	5	9 & 6	5

Execution Status Messages

```
Beginning the Merge
Comparing L(1) with R(1)...Picking R(1)
Comparing L(1) with R(2 )... Moving L(1)
Comparing L(2) with R(2 )... Moving L(2)
Comparing L(3) with R(2 )... Moving L(3)
Comparing L(4) with R(2)...Picking R(2)
```

Figure A.5.3. Merge Algorithm Detailed View Screen



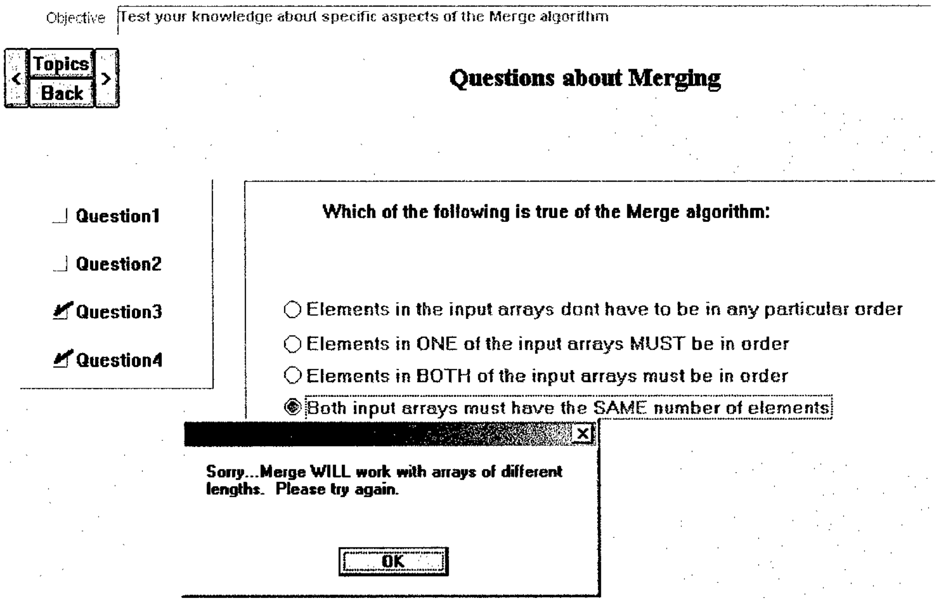
Objective: Observe and compare how the Merge algorithm works on larger datasets

Color Legend
- ▨ from LeftSide
- ■ from RightSide
- □ comparing
- ▩ selected value
- □ used data

Select speed
(or press LEFT mouse button during algorithm execution to adjust)

○ Fastest
○
○ Medium
○
● Slowest

Figure A.5.4. Merge Algorithm Populated View Screen



Objective: Test your knowledge about specific aspects of the Merge algorithm

Questions about Merging

Which of the following is true of the Merge algorithm:

○ Elements in the input arrays dont have to be in any particular order

○ Elements in ONE of the input arrays MUST be in order

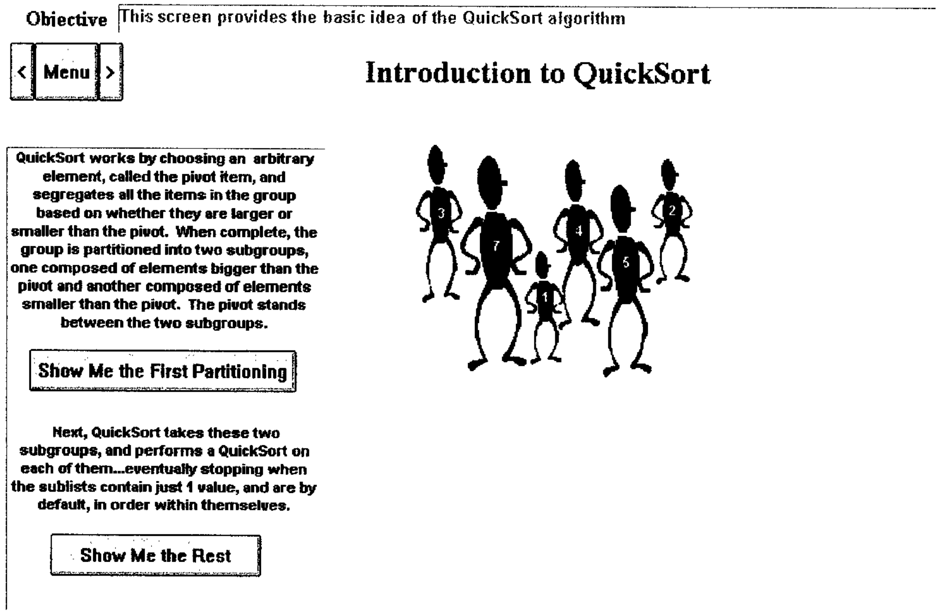
○ Elements in BOTH of the input arrays must be in order

● Both input arrays must have the SAME number of elements

Sorry...Merge WILL work with arrays of different lengths. Please try again.

Figure A.5.5. Merge Algorithm Question Screen

B.6 QUICKSORT



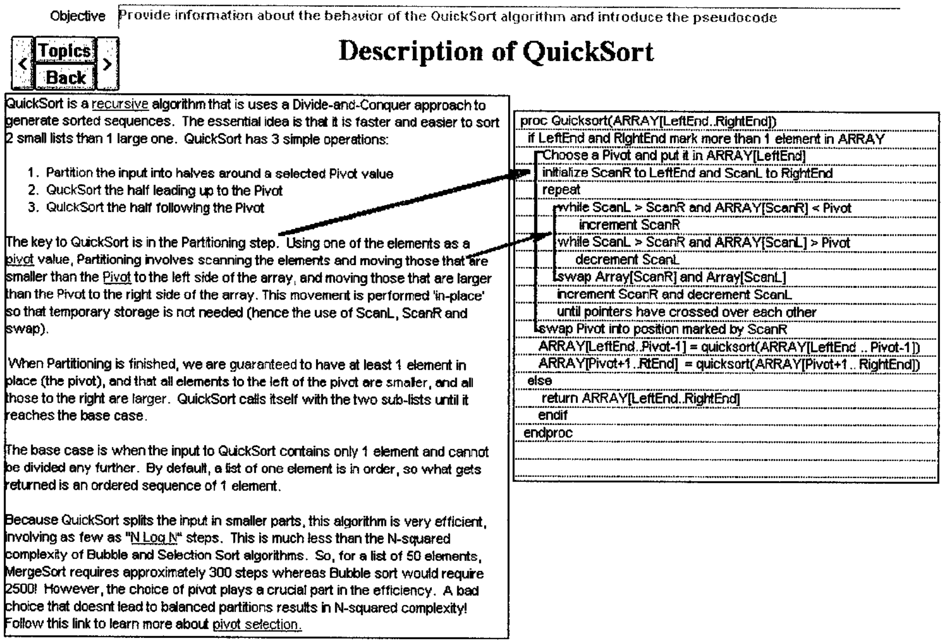
Objective This screen provides the basic idea of the QuickSort algorithm

Introduction to QuickSort

QuickSort works by choosing an arbitrary element, called the pivot item, and segregates all the items in the group based on whether they are larger or smaller than the pivot. When complete, the group is partitioned into two subgroups, one composed of elements bigger than the pivot and another composed of elements smaller than the pivot. The pivot stands between the two subgroups.

Next, QuickSort takes these two subgroups, and performs a QuickSort on each of them...eventually stopping when the sublists contain just 1 value, and are by default, in order within themselves.

Figure A.6.1 QuickSort Algorithm Conceptual View Screen



Objective Provide information about the behavior of the QuickSort algorithm and introduce the pseudocode

Description of QuickSort

QuickSort is a recursive algorithm that is uses a Divide-and-Conquer approach to generate sorted sequences. The essential idea is that it is faster and easier to sort 2 small lists than 1 large one. QuickSort has 3 simple operations:

1. Partition the input into halves around a selected Pivot value
2. QuckSort the half leading up to the Pivot
3. QuickSort the half following the Pivot

The key to QuickSort is in the Partitioning step. Using one of the elements as a pivot value, Partitioning involves scanning the elements and moving those that are smaller than the Pivot to the left side of the array, and moving those that are larger than the Pivot to the right side of the array. This movement is performed 'in-place' so that temporary storage is not needed (hence the use of ScanL, ScanR and swap).

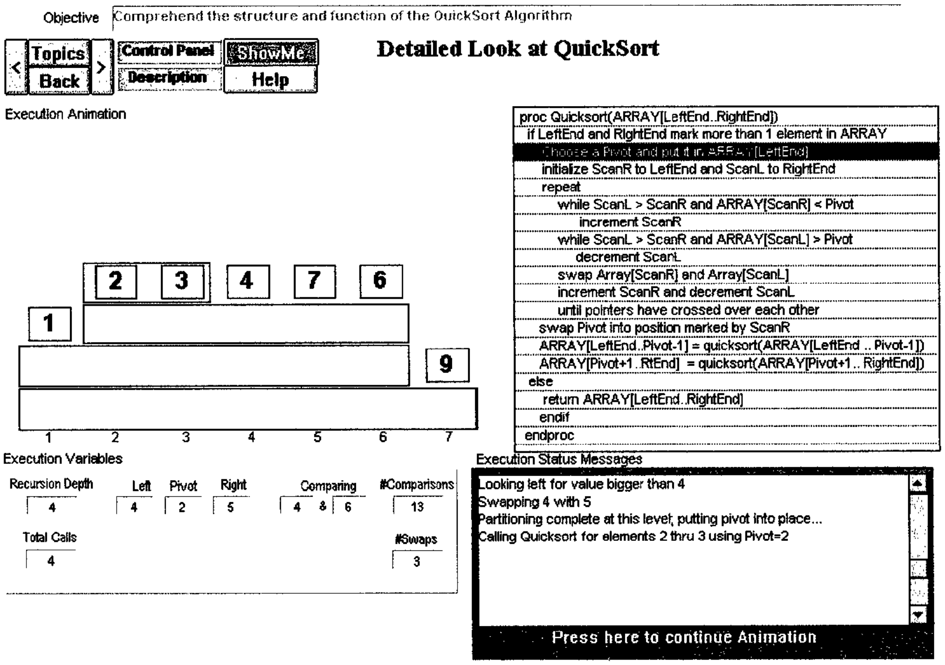
When Partitioning is finished, we are guaranteed to have at least 1 element in place (the pivot), and that all elements to the left of the pivot are smaller, and all those to the right are larger. QuickSort calls itself with the two sub-lists until it reaches the base case.

The base case is when the input to QuickSort contains only 1 element and cannot be divided any further. By default, a list of one element is in order, so what gets returned is an ordered sequence of 1 element.

Because QuickSort splits the input in smaller parts, this algorithm is very efficient, involving as few as "N Log N" steps. This is much less than the N-squared complexity of Bubble and Selection Sort algorithms. So, for a list of 50 elements, MergeSort requires approximately 300 steps whereas Bubble sort would require 2500! However, the choice of pivot plays a crucial part in the efficiency. A bad choice that doesnt lead to balanced partitions results in N-squared complexity! Follow this link to learn more about pivot selection.

```
proc Quicksort(ARRAY[LeftEnd..RightEnd])
 if LeftEnd and RightEnd mark more than 1 element in ARRAY
   Choose a Pivot and put it in ARRAY[LeftEnd]
   initialize ScanR to LeftEnd and ScanL to RightEnd
   repeat
     while ScanL > ScanR and ARRAY[ScanR] < Pivot
         increment ScanR
     while ScanL > ScanR and ARRAY[ScanL] > Pivot
        decrement ScanL
     swap Array[ScanR] and Array[ScanL]
     increment ScanR and decrement ScanL
     until pointers have crossed over each other
   swap Pivot into position marked by ScanR
   ARRAY[LeftEnd..Pivot-1] = quicksort(ARRAY[LeftEnd .. Pivot-1])
   ARRAY[Pivot+1 ..RtEnd]  = quicksort(ARRAY[Pivot+1 .. RightEnd])
 else
   return ARRAY[LeftEnd..RightEnd]
   endif
endproc
```

Figure A.6.2. QuickSort Algorithm Description Screen



Objective: Comprehend the structure and function of the QuickSort Algorithm

Detailed Look at QuickSort

```
proc Quicksort(ARRAY[LeftEnd..RightEnd])
  if LeftEnd and RightEnd mark more than 1 element in ARRAY
    Choose a Pivot and put it in ARRAY[LeftEnd]
    initialize ScanR to LeftEnd and ScanL to RightEnd
    repeat
      while ScanL > ScanR and ARRAY[ScanR] < Pivot
        increment ScanR
      while ScanL > ScanR and ARRAY[ScanL] > Pivot
        decrement ScanL
      swap Array[ScanR] and Array[ScanL]
      increment ScanR and decrement ScanL
      until pointers have crossed over each other
    swap Pivot into position marked by ScanR
    ARRAY[LeftEnd..Pivot-1] = quicksort(ARRAY[LeftEnd .. Pivot-1])
    ARRAY[Pivot+1..RtEnd]  = quicksort(ARRAY[Pivot+1 .. RightEnd])
  else
    return ARRAY[LeftEnd..RightEnd]
    endif
endproc
```

Execution Variables

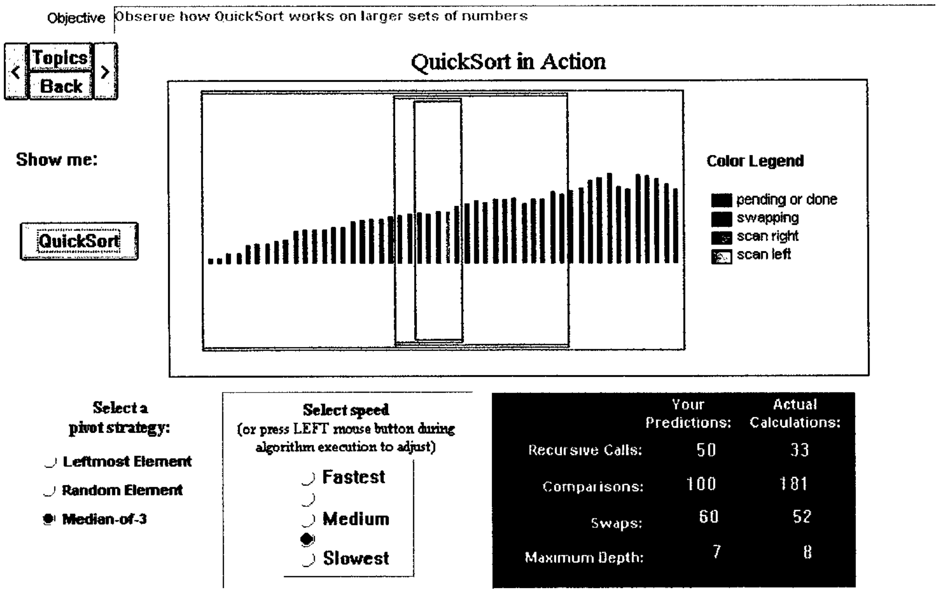
Recursion Depth	Left	Pivot	Right	Comparing	#Comparisons
4	4	2	5	4 & 6	13

Total Calls	#Swaps
4	3

Execution Status Messages

```
Looking left for value bigger than 4
Swapping 4 with 5
Partitioning complete at this level, putting pivot into place...
Calling Quicksort for elements 2 thru 3 using Pivot=2
```

Figure A.6.3. QuickSort Algorithm Detailed View Screen



Objective: Observe how QuickSort works on larger sets of numbers

QuickSort in Action

Select a pivot strategy:

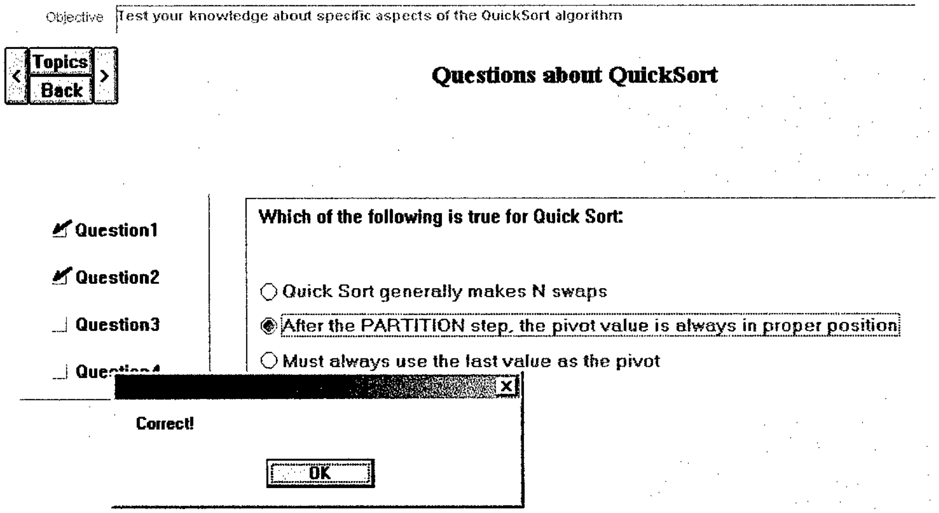
○ Leftmost Element
○ Random Element
● Median-of-3

Select speed (or press LEFT mouse button during algorithm execution to adjust)

○ Fastest
○
○ Medium
●
○ Slowest

	Your Predictions:	Actual Calculations:
Recursive Calls:	50	33
Comparisons:	100	181
Swaps:	60	52
Maximum Depth:	7	8

Figure A.6.4. QuickSort Algorithm Populated View Screen



Objective: Test your knowledge about specific aspects of the QuickSort algorithm

Questions about QuickSort

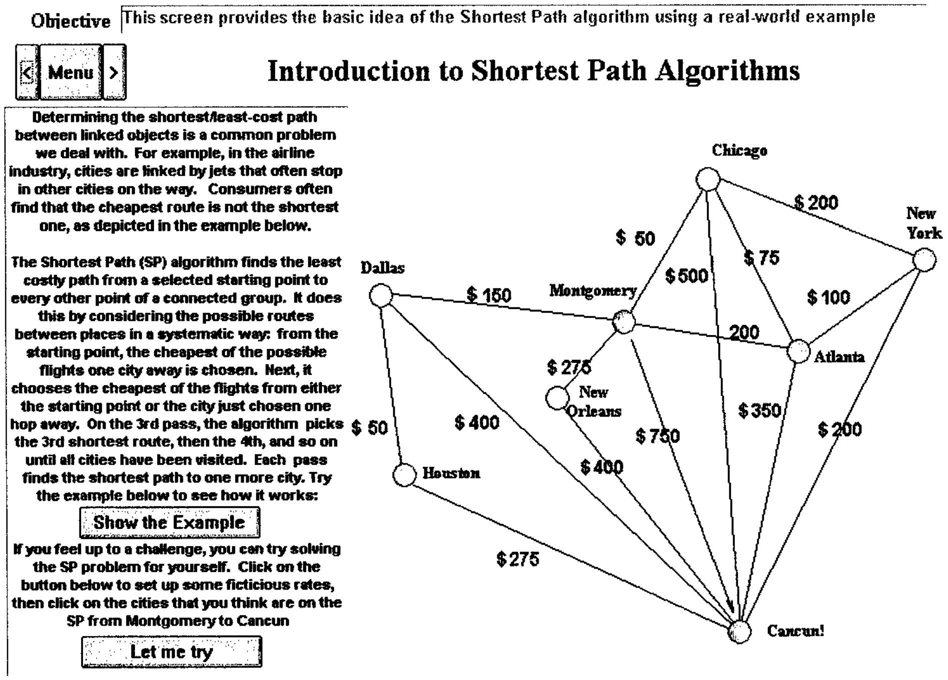
Which of the following is true for Quick Sort:

○ Quick Sort generally makes N swaps
● After the PARTITION step, the pivot value is always in proper position
○ Must always use the last value as the pivot

Correct!

Figure A.6.5. QuickSort Algorithm Question Screen

B.7 SHORTEST PATH



Objective This screen provides the basic idea of the Shortest Path algorithm using a real-world example

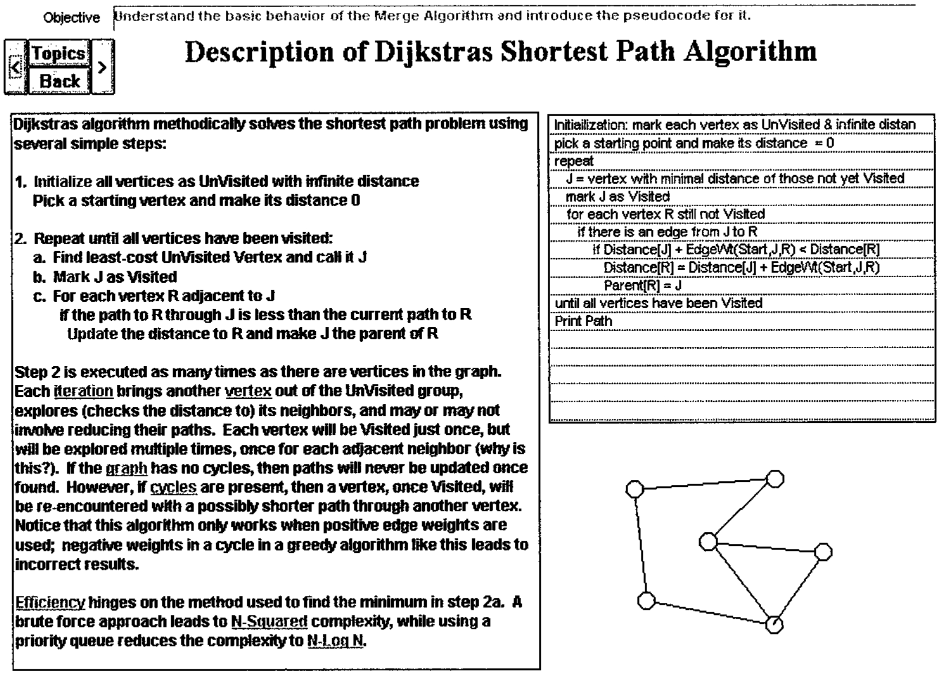
Introduction to Shortest Path Algorithms

Determining the shortest/least-cost path between linked objects is a common problem we deal with. For example, in the airline industry, cities are linked by jets that often stop in other cities on the way. Consumers often find that the cheapest route is not the shortest one, as depicted in the example below.

The Shortest Path (SP) algorithm finds the least costly path from a selected starting point to every other point of a connected group. It does this by considering the possible routes between places in a systematic way: from the starting point, the cheapest of the possible flights one city away is chosen. Next, it chooses the cheapest of the flights from either the starting point or the city just chosen one hop away. On the 3rd pass, the algorithm picks the 3rd shortest route, then the 4th, and so on until all cities have been visited. Each pass finds the shortest path to one more city. Try the example below to see how it works:

If you feel up to a challenge, you can try solving the SP problem for yourself. Click on the button below to set up some ficticious rates, then click on the cities that you think are on the SP from Montgomery to Cancun

Figure A.7.1 Shortest Path Algorithm Conceptual View Screen



Objective Understand the basic behavior of the Merge Algorithm and introduce the pseudocode for it.

Description of Dijkstras Shortest Path Algorithm

Dijkstras algorithm methodically solves the shortest path problem using several simple steps:

1. Initialize all vertices as UnVisited with infinite distance
   Pick a starting vertex and make its distance 0

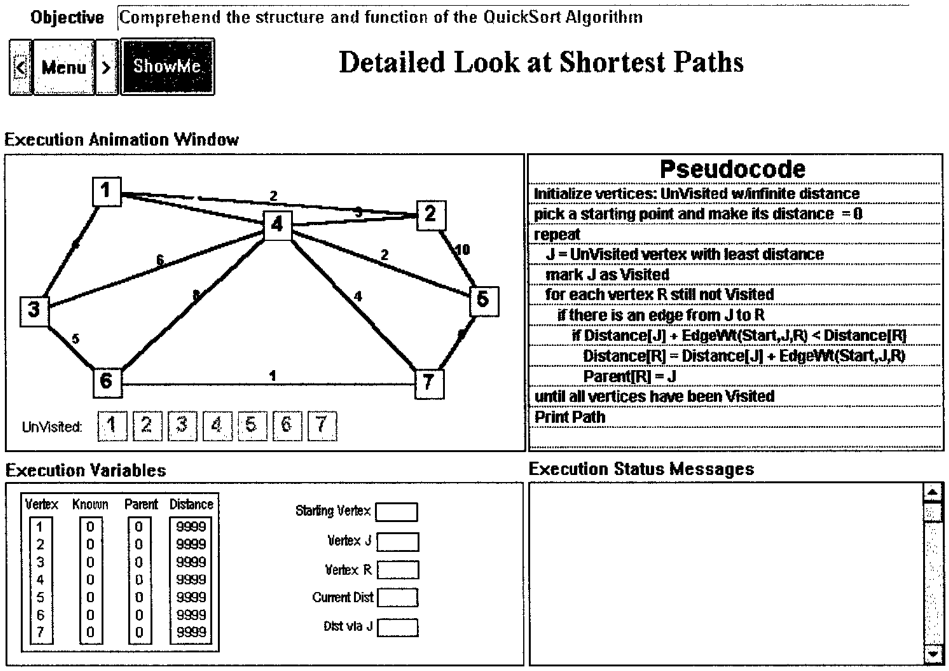
2. Repeat until all vertices have been visited:
   a. Find least-cost UnVisited Vertex and call it J
   b. Mark J as Visited
   c. For each vertex R adjacent to J
      if the path to R through J is less than the current path to R
      Update the distance to R and make J the parent of R

Step 2 is executed as many times as there are vertices in the graph. Each iteration brings another vertex out of the UnVisited group, explores (checks the distance to) its neighbors, and may or may not involve reducing their paths. Each vertex will be Visited just once, but will be explored multiple times, once for each adjacent neighbor (why is this?). If the graph has no cycles, then paths will never be updated once found. However, if cycles are present, then a vertex, once Visited, will be re-encountered with a possibly shorter path through another vertex. Notice that this algorithm only works when positive edge weights are used; negative weights in a cycle in a greedy algorithm like this leads to incorrect results.

Efficiency hinges on the method used to find the minimum in step 2a. A brute force approach leads to N-Squared complexity, while using a priority queue reduces the complexity to N-Log N.

```
Initiailization: mark each vertex as UnVisited & infinite distan
pick a starting point and make its distance  = 0
repeat
  J = vertex with minimal distance of those not yet Visited
  mark J as Visited
  for each vertex R still not Visited
    if there is an edge from J to R
      if Distance[J] + EdgeWt(Start,J,R) < Distance[R]
        Distance[R] = Distance[J] + EdgeWt(Start,J,R)
        Parent[R] = J
until all vertices have been Visited
Print Path
```

Figure A.7.2. Shortest Path Algorithm Description Screen



Objective: Comprehend the structure and function of the QuickSort Algorithm

Detailed Look at Shortest Paths

Execution Animation Window

UnVisited: 1 2 3 4 5 6 7

Pseudocode

```
Initialize vertices: UnVisited w/infinite distance
pick a starting point and make its distance  = 0
repeat
  J = UnVisited vertex with least distance
  mark J as Visited
  for each vertex R still not Visited
    if there is an edge from J to R
      if Distance[J] + EdgeWt(Start,J,R) < Distance[R]
        Distance[R] = Distance[J] + EdgeWt(Start,J,R)
        Parent[R] = J
until all vertices have been Visited
Print Path
```

Execution Variables

Vertex	Known	Parent	Distance
1	0	0	9999
2	0	0	9999
3	0	0	9999
4	0	0	9999
5	0	0	9999
6	0	0	9999
7	0	0	9999

Starting Vertex

Vertex J

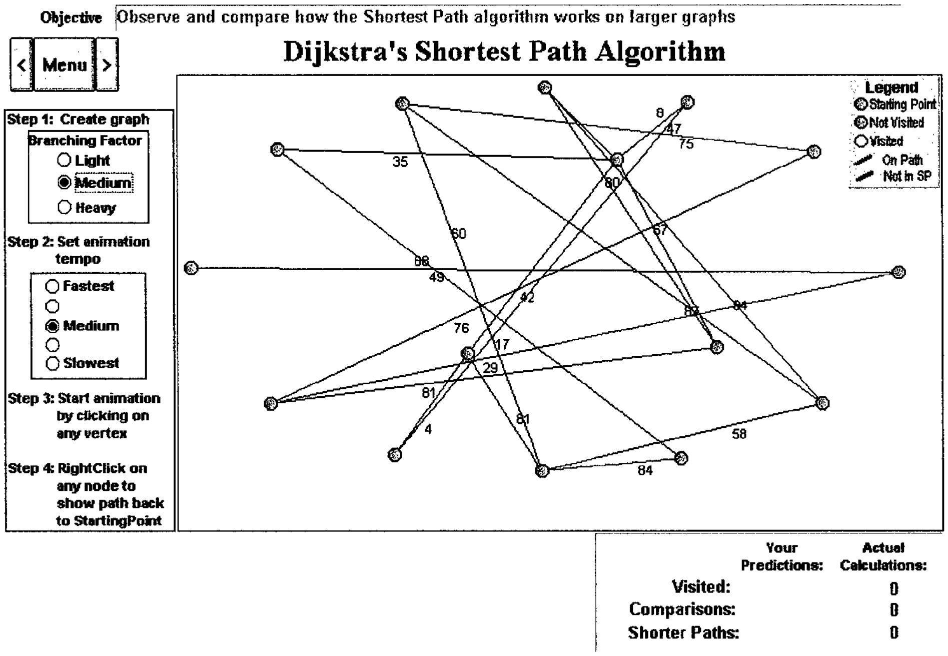
Vertex R

Current Dist

Dist via J

Execution Status Messages

Figure A.7.3. Shortest Path Algorithm Detailed View Screen



Objective: Observe and compare how the Shortest Path algorithm works on larger graphs

Dijkstra's Shortest Path Algorithm

Step 1: Create graph

Branching Factor

○ Light
● Medium
○ Heavy

Step 2: Set animation tempo

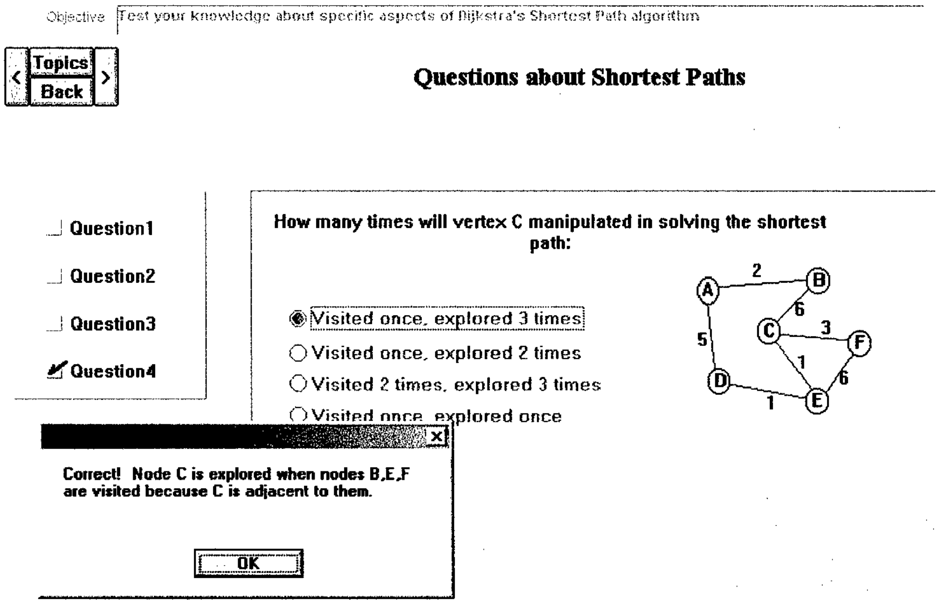
○ Fastest
○
● Medium
○
○ Slowest

Step 3: Start animation by clicking on any vertex

Step 4: RightClick on any node to show path back to StartingPoint

	Your Predictions:	Actual Calculations:
Visited:		0
Comparisons:		0
Shorter Paths:		0

Figure A.7.4. Shortest Path Algorithm Populated View Screen



Objective: Test your knowledge about specific aspects of Dijkstra's Shortest Path algorithm

Questions about Shortest Paths

How many times will vertex C manipulated in solving the shortest path:

● Visited once, explored 3 times

○ Visited once, explored 2 times

○ Visited 2 times, explored 3 times

○ Visited once, explored once

Correct! Node C is explored when nodes B,E,F are visited because C is adjacent to them.

Figure A.7.5. Shortest Path Algorithm Question Screen

APPENDIX B: MATERIALS USED DURING EXPERIMENTS

This appendix contains examples of materials used for the experiments described in this dissertation. For brevity's sake, only the post-tests are shown; the pre-tests for each experiment had the same form and type of question as the post-test, but with different data or slightly lower complexity.

D.1. CONSENT FORM

The consent form that each subject read and completed is shown on the following page.

Auburn University

Auburn University, Alabama 36849-5347
College of Engineering

Computer Science and Engineering
107 Dunstan Hall

Telephone (334) 844-4330

Research Project Information

"Rethinking Algorithm Animations"

Thank you for volunteering to participate in our research project at Auburn University. This project will explore the effectiveness of several methods of teaching algorithms. In the long term, your inputs might help us show that the ways we currently teach algorithms and other abstract concepts could be significantly improved, not only for better retention but for higher student satisfaction. In the short term, your participation will give you exposure to exciting new teaching techniques as well as a good foundation in several popular computer science algorithms.

As a participant, you will be asked to do the following:

- take a written, prior-knowledge survey (about 15 minutes)
- participate in an observation session with a written post-test (about 60 minutes)
- take a follow-up survey (about 20 minutes)

Additionally, we ask that you not not discuss this session with other people (to avoid influencing their possible participation in any way).

Please remember that we are evaluating OUR SYSTEM AND NOT YOU! The tests you take and the data we collect during the observation session will be totally anonymous; you won't be identified as an individual in any of the data we collect, nor named in any of the reports we develop after this research. You cannot fail any part of this session. Your participation is needed to help us identify usability problems with our system and if our animation system helps a student learn more effectively. Besides making a significant contribution to computer science research and learning more about selected popular algorithms, participation will provide homework credit for COMP0220 and COMP0360 students.

There are no known risks associated with this experiment. During the experiment, you may request that your data not be used in our analysis, and we'll destroy it from our files. You may withdraw from the session at any time, although we hope you'll stick with it through to the end.

We truly appreciate your time and willingness to participate in this computer system evaluation. Remember, we are not evaluating you, but our system, and therefore, you cannot fail any part of this session. If you have any questions, please contact any of the individuals listed below:

Principal Investigator	Dr. N. Hari Narayanan	(334) 844-6312	narayan@Eng.Auburn.edu
Research Assistant	Steven Hansen	(334) 270-9795	hansensr@mindspring.com
Research Assistant	Dan Schrimpshier	(334) 844-2158	schridj@Eng.Auburn.edu

For more information regarding your rights as a participant you may contact Ms. Jeanna Sasser (844-5966) or Dr. Leanne Lamke (844-3231) at the Office of Human Subjects.

Your session is scheduled on ________________ at ______________ in ________________.
date time location

Thank you for your participation!

D.2. DEMOGRAPHIC SURVEY

The demographic survey shown on the next page was used to gather information from prospective participants. This information was used to help create the randomly-assigned matched groups used in each of the experiments of this study.

Demographic Information

Disclaimer: The following information is requested to allow us to form lab groups that have similar backgrounds and capabilities. This information will not be disseminated in any way and will not affect your standing in class.

Name:	Email address:	Your anonymous ID: (any 3-digit code that we can use to identify your results)

Please indicate the approximate score you obtained on any of the tests below that you may have taken:		
ACT	SAT.	GRE

What is your approximate overall GPA?						
○	○	○	○	○	○	
4.0	3.5	3.0	2.5	2.0	1.5	
What is your approximate GPA in computer science classes?						
○	○	○	○	○	○	
4.0	3.5	3.0	2.5	2.0	1.5	
What was your approximate grade in CSE200 (or its equivalent)?						
○	○	○	○	○	○	
A	B	C	D	F	Didnt Take It	
How many years of college have you completed?						
○	○	○	○	○	○	○
1	2	3	4	5	6	7 +

What is the highest degree you have earned?		
○	○	○
High School	Bachelor's Degree	Master's Degree
What degree are you attempting to earn now?		
○	○	○
Bachelor's Degree	Master's Degree	Doctoral Degree

Have you participated in Algorithm Visualization Research at Auburn University before?		
○	○	○
No, never	Yes, during Fall 97	Yes, during Winter 98

If yes:

What did you like about it?	
What didn't you like?	

The Algorithm Visualization Lab sessions will probably take place in 2-hour sessions on the last two Fridays in May. Please indicate the times during which you have a conflict and **could not attend**

○	○	○	○	○	○	○
11-1	12-2	1-3	2-4	3-5	4-6	5-7

D.3. USER SATISFACTION SURVEY

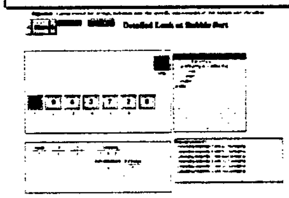
The survey shown on the following pages provided subjective feedback about student impressions of the HalVis system.

Satisfaction and General Impressions

Disclaimer: The following information is requested to allow researchers to information to improve the program with which you interacted recently. This information will not be disseminated on an individual basis and will not affect your class standing.

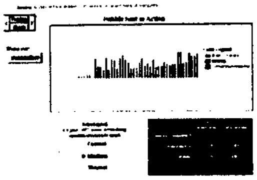
Please rate each of the items listed below in terms of how effective you thought it was

	Very Positive/ Effective	Positive	Negative	Extremely Negative	No Opinion/ Dont recall
Use of textual descriptions (pop-ups)	O	O	O	O	O
Use of audio	O	O	O	O	O
Use of color to highlight actions	O	O	O	O	O
Use of analogies and examples to introduce algorithms	O	O	O	O	O
Use of hyperlinking to find related information	O	O	O	O	O
Explanation of learning objectives on each screen	O	O	O	O	O
Use of pop-up questions	O	O	O	O	O
Clarity of button and menu names	O	O	O	O	O



The following questions ask you to rate aspects of the screens that described the algorithm specifics, containing the animations and the pseudocode (shown in the thumbnail diagram to the left)

	Very Positive/ Effective/ Helpful	Positive	Negative	Extremely Negative/ Useless/ Confusing	No Opinion/ Dont recall
Effectiveness of the blue textual introduction explaining the algorithm prior to the animation	O	O	O	O	O
Use of textual explanations (event messages) during animation	O	O	O	O	O
Highlighting the pseudocode statements during the animation	O	O	O	O	O
Counters that reflected the progress of the algorithm	O	O	O	O	O
Ability to alter the speed of the animation	O	O	O	O	O
Ability to rerun the animation	O	O	O	O	O
Comment on the amount of information presented	Excessive		Perfect		Sparse
Did you try changing the data?	Yes				No
How many times did you run the animation?	4+	3	2	1	0



The following questions ask you to rate aspects of the screens that illustrated the performance of the algorithms using the animated bars (shown in the thumbnail diagram to the left)

	Very Positive/ Effective/ Helpful	Positive	Negative	Extremely Negative/ Useless/ Confusing	No Opinion/ Dont recall
What was your impression of this screen	○	○	○	○	○
Counters that reflected the progress of the algorithm	○	○	○	○	○
Ability to predict the performance of the algorithm	○	○	○	○	○
Ability to rerun the animation	○	○	○	○	○
Ability to alter the speed of the animation	○	○	○	○	○
Use of colors to illustrate the actions of the algorithm	○	○	○	○	○
Comment on the amount of information presented	Excessive		Perfect		Sparse
How many times did you run the animation?	4+	3	2	1	0
Circle the data case you used most	best case		average case		worst case

Please provide your overall impression					
What was your impression of this program?	Very Positive/ Helpful	Positive	Negative	Extremely Negative/ Confusing	No Opinion/ Dont recall
What was your impression of the user interface?	Very Easy to Learh	Easy to Learn	Hard to Learn	Rigid	No Opinion/ Dont recall
What was your impression of the navigation?	Very Intuitive	Intuitive	Difficult	Frustrating	No Opinion/ Dont recall

If you had to learn more algorithms in the future, which technique would you prefer to use, if it was available:

○ This Animation System

○ Traditional Text

○ No Opinion

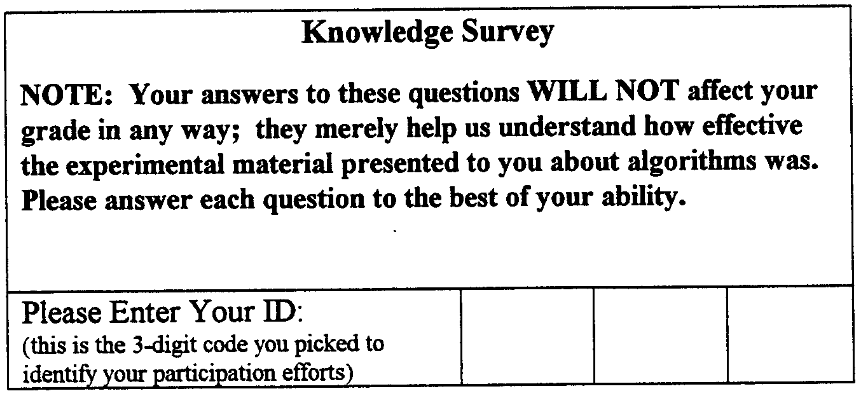
Please comment on what you liked the most:

Please comment on anything you didn't like about this program:

D.4. EXPERIMENTS WITH TEXT

D.4.1. POSTTEST FOR EXPERIMENT IA

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:



Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

Please Enter Your ID: (this is the 3-digit code you picked to identify your participation efforts)			

1a. Does the pseudocode here re-order the elements in ARRAY into descending or descending order?

1b. Make the necessary changes to the pseudocode above to produce an ascending sequence (unless it already does)

```
1- Procedure MergeSort(ARRAY)
2-   if length(ARRAY) > 1
3-     Middle = length(ARRAY)/2
4-       LeftArray  = MergeSort(Array[1..Middle])
5-       RightArray = MergeSort(Array[Middle+1..N])
6-       ResultArray = Merge (LeftArray,RightArray)
7-       Endif
8-   Return ResultArray
9-   EndMergeSort

10- Proc Merge(LeftHalf,RightHalf)
11-   loop
12-       if LeftHalf[x] > RightHalf[y]
13-           Result[z] = LeftHalf[x]
14-           Increment X and Z
15-     else
16-           Result[z] = RightHalf[y]
17-           Increment y and z
18-           endif
19-     until RightHalf or LeftHalf is empty
20-     Flush remaining elements from RightHalf
              (or LeftHalf)  to Result
21-    return Result
22-    endMerge
```

3. Using the algorithm above on the data shown below, indicate how many calls to MergeSort would be needed to place the data into ascending order.

ARRAY:

Before	**52**	**91**	**7**	**41**	**34**	**10**	**55**

of calls:

4. If the values [52,91,7,41,34,10,55] are passed to MysterySort above, show:

the value(s) passed as parameters into MergeSort when line 4 is executed the **<u>first</u>** time:	
the value(s) passed as parameters into MergeSort when line 5 is executed the **<u>first</u>** time:	

5. If the values [52,91,7,41,34,10,55] are passed to MergeSort, show the value(s) returned in ResultArray following <u>each</u> recursive call to MERGE on line 6

Call #	ResultArray

6. The MergeSort algorithm will be used to sort an array of 1000 test scores. Which of the following is true:

   (a) The sort is fastest if the original scores are in random order

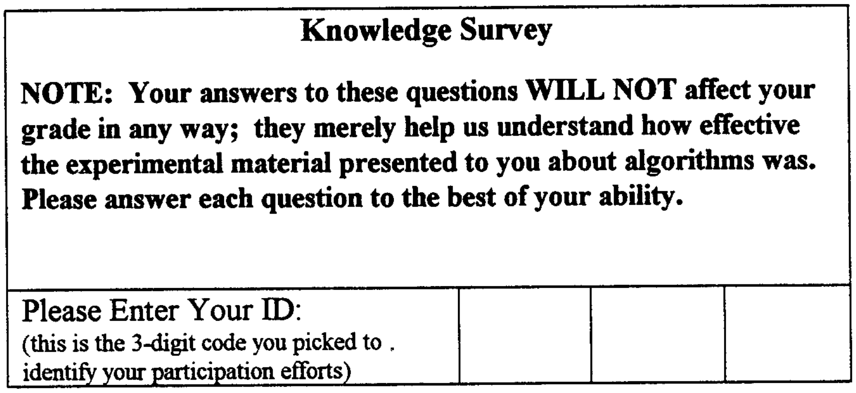
   (b) The sort is fastest if the original scores are ordered from smallest to largest

   (c) The sort is fastest if the original scores are ordered from largest to smallest

   (d) The sort is the same, no matter what the order of the original elements.

D.4.2. POSTTEST FOR EXPERIMENT IB

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:



Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

Please Enter Your ID: (this is the 3-digit code you picked to . identify your participation efforts)			

```
1- Procedure MergeSort(ARRAY)
2-  if length(ARRAY) > 1
3-    Middle = length(ARRAY)/2
4-     LeftArray   = MergeSort(Array[1..Middle])
5-     RightArray = MergeSort(Array[Middle+1..N)
6-     ResultArray = Merge (LeftArray,RightArray)
7-     Endif
8-  Return ResultArray
9-  EndMergeSort

10- Proc Merge(LeftHalf,RightHalf)
11-   loop
12-       if LeftHalf[x] > RightHalf[y]
13-            Result[z] = LeftHalf[x]
14-            Increment X and Z
15-      else
16-            Result[z] = RightHalf[y]
17-            Increment y and z
18-            endif
19-      until RightHalf or LeftHalf is empty
20-      Flush remaining elements from RightHalf
              (or LeftHalf)  to Result
21-     return Result
22-     endMerge
```

B1a. Does the pseudocode above re-order the elements in ARRAY into ascending or descending order?

B1b. Make the necessary changes to the pseudocode above to produce an ascending sequence (unless it already does)

B2. The MergeSort algorithm will be used to sort an array of 1000 test scores. Which of the following is true:

(a) The sort is dramatically faster if the original scores are in random order

(b) The sort is dramatically faster if the original scores are ordered from smallest to largest

(c) The sort is dramatically faster if the original scores are ordered from largest to smallest

(d) The sort is about the same, no matter what the order of the original elements.

B3. If the following data was passed to the Merge procedure on line #10, indicate what would be returned in the array called Results. Use the unaltered version of the algorithm.

Left Array

5	26	34

Right Array

9	27	30	33	71

Result Array

B4. Using algorithm above on the data shown below, indicate how many calls to MergeSort would be needed to place the data into ascending order.

ARRAY:

Before	**52**	**91**	**7**	**41**	**34**	**10**	**55**

of calls:

B5. If the values [52,91,7,41,34,10,55] are passed to MergeSort, produce a diagram showing the sequence of calls. Use the unaltered version. Be sure to indicate the values passed as inputs to each call and the results that are passed back upon completion of the call.

e.g. ProcedureName INPUT:(52, 91, 7,41, 34, 10, 55)
RETURNS (??, ??, ??, ??, ??, ??, ??)

```
Procedure QuickSort(ARRAY[L..R])
  if length(ARRAY) > 1
    Select Pivot and put in ARRAY[L]
    initialize ScanR = L-1  and ScanL = R
    repeat
      while ScanR > ScanL and ARRAY[ScanR] < Pivot
          increment ScanR
      while ScanR > ScanL and ARRAY[ScanL] > Pivot
          decrement ScanL
      swap ARRAY[ScanR] and ARRAY[ScanL]
      increment ScanR and decrement ScanL
      until ScanL <= ScanR
    swap Pivot in ARRAY[L] with ARRAY[ScanR]
    ARRAY[L..ScanR-1] = QuickSort(ARRAY[L.. ScanR-1])
    ARRAY[ScanR+1..R]= QuickSort(ARRAY[ScanR+1.. R)
  else
    return ARRAY[L..R]
  endif
endproc
```

C1a. Does the pseudocode above re-order the elements in ARRAY into ascending or descending order?

C2. Make changes to the pseudocode above to produce an ascending sequence (unless it already does)

C3. What can be said after *k* (some arbitrary number) of calls to QuickSort :

(a) About k^2 comparisons will have been made

(b) At least *k* elements will always be in place regardless of the order of the input

(c) Exactly *k-1* Merge operations will have been performed

(d) No more than *k* swaps will have been made.

(e) None of the above

C4. If the following data was passed to the QuickSort procedure, indicate the contents of the ARRAY after the first execution of the loop depicted in lines #5-13. Use the unaltered version of the algorithm in question #C1a.

<table>
<tr><th colspan="7">Original Input
ARRAY[L..R]</th></tr>
<tr><td><b>33</b></td><td><b>26</b></td><td><b>44</b></td><td><b>9</b></td><td><b>71</b></td><td><b>21</b></td><td><b>6</b></td></tr>
</table>

<table>
<tr><th colspan="3">Result after 1st Partitioning (lines #5-13)</th></tr>
<tr><td>ARRAY[L..ScanR-1]</td><td>Pivot</td><td>ARRAY[ScanR+1..R]</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

C5. Using the algorithm above on the data shown below, indicate how many calls to QuickSort would be needed to place the data in the array below into order. Use the unaltered version in question #C1.

ARRAY:

Before	**33**	**26**	**44**	**9**	**71**	**21**	**6**

of calls:

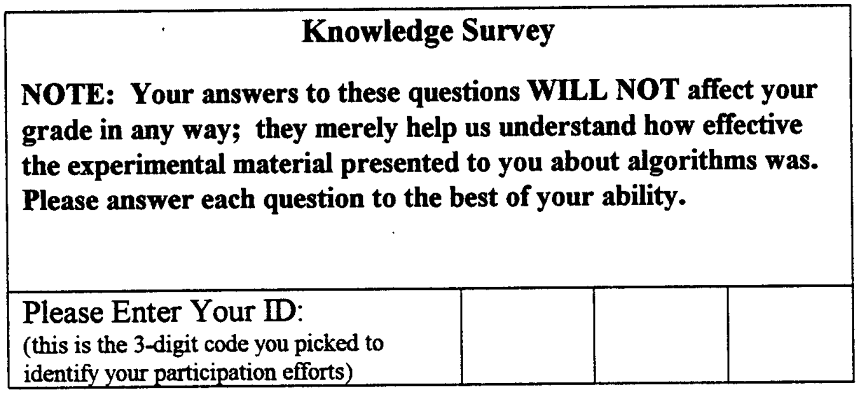
C6. If the values [33, 26, 44, 9, 71, 21, 6] are passed to QuickSort, produce a diagram showing the sequence of calls. Use the unaltered version. Be sure to indicate the values passed as inputs to each call and the results that are passed back upon completion of the call.

e.g. ProcedureName INPUT:(33, 26, 44, 9, 71, 21, 6)
RETURNS (??, ??, ??, ??, ??, ??, ??)

D.5. EXPERIMENTS WITH TEXT AND EXERCISES

D.5.1. POSTTEST FOR EXPERIMENT II

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:



Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

Please Enter Your ID: (this is the 3-digit code you picked to identify your participation efforts)			

1. Show the order of elements in ARRAY after the first pass of an ascending Selection Sort algorithm:

ARRAY

Before	**4**	**2**	**1**	**5**	**3**
After 1 pass					

2. How many swap operations would occur in the problem described above (first pass of ascending Selection Sort)?

□

3. In a sentence or two, describe the basic behavior of the Selection Sort algorithm

4. Show the order of elements in ARRAY after the first pass of an ascending Bubble Sort algorithm:

ARRAY

Before	**Don**	**Carl**	**Ann**	**Eric**	**Bev**
After 1 pass					

5. How many swap operations would occur in the problem described above (first pass of ascending Bubble Sort)?

□

6. In a sentence or two, describe the basic behavior of the Bubble Sort algorithm

7. The psueudocode to the right implements which popular sort algorithm?

```
for x = N-1 downto 1
   for y = N downto N-x+1
      if ARRAY[y] < ARRAY[y-1]
         swap ARRAY[y] and ARRAY[y-1]
      endif
   endfor
endfor
```

8a. Does the pseudocode above re-order the elements in ARRAY into descending order?

8b. If not, make pen and ink changes to alter the pseudocode above to produce a descending sequence

9. Using the algorithm shown in problem 7 above, how many comparisons and swaps would be made to produce a sorted sequence given the input [1,2,3,4]

Comparisons	Swaps

10. Using the modified algorithm below, how many comparisons and swaps would be needed to produce a sorted sequence given the input [1,2,3,4]?

Comparisons	Swaps

```
for x = N-1 downto 1
   SORTED = 1
   for.y = N downto N-x+1
      if ARRAY[y] < ARRAY[y-1]
         SORTED = 0
          swap ARRAY[y] and ARRAY[y-1]
      endif
   endfor
   if SORTED = 1
          exit
   endif
endfor
```

11. This psueudocode to the right implements which popular sort algorithm?

```
for x= 1 to N-1
  CHOICE = x
  for y = x+1 to N
    if ARRAY[y] > ARRAY[CHOICE]
       CHOICE = y
    endif
  endfor
  swap ARRAY[CHOICE] and ARRAY[x]
endfor
```

12a. Does the pseudocode above re-order the elements in ARRAY into descending order?

12b. If not, make pen and ink changes to alter the pseudocode above to produce a descending sequence

13. Using the algorithm above, how many comparisons and swaps would be made to produce a sorted sequence given the input [1,2,3,4]

Comparisons	Swaps

14. Using the modified algorithm below, how many comparisons and swaps would be needed to produce a sorted sequence given [1,2,3,4]?

Comparisons	Swaps

```
for x= 1 to N-1
  CHOICE = x
  for y = x+1 to N
    if ARRAY[y] > ARRAY[CHOICE]
       CHOICE = y
    endif
  endfor
  if CHOICE not = x
    swap ARRAY[CHOICE] and ARRAY[x]
  endif
endfor
```

15. Circle the item(s) below that are true:

A On average, Bubble Sort makes fewer swaps than Selection Sort

B On average, Bubble Sort makes more swaps than Selection Sort

C On average, Bubble Sort makes the same number of swaps as Selection Sort

16. After *k* (some arbitrary number) passes, the first *k* items are always in the proper place for:

A Bubble Sort

B Selection Sort

C Both Bubble Sort and Selection Sort

D Neither Bubble Sort nor Selection Sort

17. Organize the values [1,2,3,4,5] into an input sequence that causes the standard Bubble Sort algorithm to make fewer swap operations than the Selection Sort algorithm.

Answer:					

D.5.2. HANDOUT FOR TEXT GROUP OF EXPERIMENT II

The following pages are the handout we crafted from our review of the sections of sorting algorithms from 19 commercially available textbooks.

Sorting Algorithms

Learning Objectives:

- **Understand the basic concepts of sorting algorithms**
- **Learn the construction and behavior of specific sorting algorithms**
- **Predict how sort algorithms will operate on a given data set**
- **Be able to modify and find errors in sorting algorithms**

Introduction

Sorting is one of the more interesting topics in computer science and in the study of algorithms, not only because sorting is a common and useful problem but also because there are many different ways one can sort a list. The various approaches are interesting to study and understand. They represent different ways of solving a similar problem. Some approaches are easier to understand than others, some are more take less time, some use less space and some are better in situations where the order of the lists to be sorted is known in advance.

The input to a sorting problem is an unordered list of elements, typically numbers or letters. The task is to produce the list in a particular order, either ascending or descending, based on each element's lexicographic value. A dictionary and a phone book are examples of alphabetically ordered lists in ascending sequence. Examples of numerical sequences are shown below, depicted as both horizontal and vertical lists of numbers.

Unordered	Ascending Order	Descending Order
6	1	10
4	4	8
8	6	6
10	8	4
1	10	1

Unordered	6	4	8	10	1
Ascending Order	1	4	6	8	10
Descending Order	10	8	6	4	1

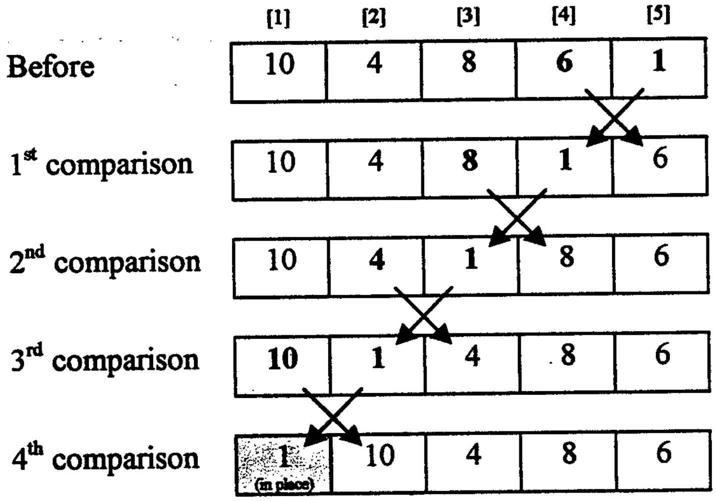
Bubble Sort

The bubble sort algorithm uses a simple scheme. Each iteration puts the smallest unsorted element in its correct place, changing places with other elements in the list. The first iteration puts the smallest element in the first position. Starting with the last element, we compare successive pairs of elements, swapping whenever the bottom element of the pair is smaller than the one above it. In this way, the smallest element "bubbles" up to the top or front of the list. The next iteration puts the smallest element in the unsorted part of the list into the second position, using the same technique.

The figures below walk through sorting a 5-element list. Each row represents a comparison of the items in bold print, and arrows are used to show items that were exchanged or not. Each pass is indicated separately for easier reading. Note that in addition to putting one element in its proper place, each iteration causes some intermediate changes in the order of the other elements also.

The first traversal puts the value 1 into place at the head of the list, making 4 comparisons and 4 swaps in the process. Note that this first traversal does not guarantee the entire list is sorted—it only ensures that the first element is.

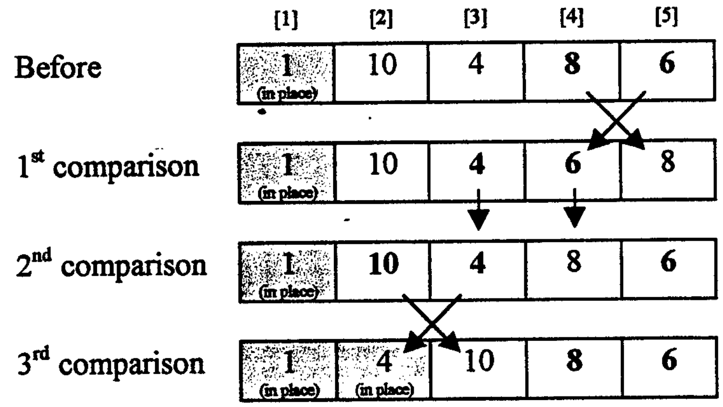
First Pass



	[1]	[2]	[3]	[4]	[5]
Before	10	4	8	**6**	**1**
1st comparison	10	4	**8**	**1**	6
2nd comparison	10	**4**	**1**	8	6
3rd comparison	**10**	**1**	4	8	6
4th comparison	1 (in place)	10	4	8	6

The second traversal will bring the second largest element to its proper resting place. Notice that only 3 comparison operations were needed. A fourth was not required since the first element is already in position. Also notice that only 2 swap operations were needed, since the elements in the second comparison were not out of order as a pair.

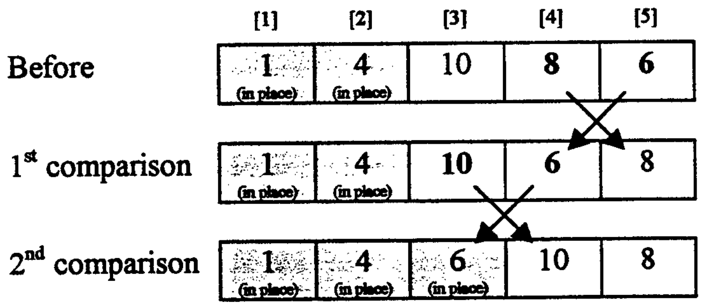
Second Pass



	[1]	[2]	[3]	[4]	[5]
Before	1 (in place)	10	4	8	6
1st comparison	1 (in place)	10	4	6	8
2nd comparison	1 (in place)	10	4	8	6
3rd comparison	1 (in place)	4 (in place)	10	8	6

The third traversal will bring the third largest element to its proper resting place, requiring 2 comparison operations and 2 swap operations.

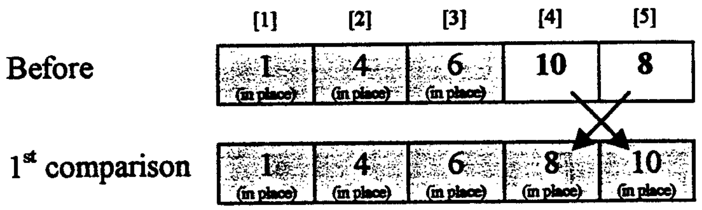
Third Pass



	[1]	[2]	[3]	[4]	[5]
Before	1 (in place)	4 (in place)	10	8	6
1st comparison	1 (in place)	4 (in place)	10	6	8
2nd comparison	1 (in place)	4 (in place)	6 (in place)	10	8

The fourth traversal brings the fourth element into position, making 1 comparison and 1 swap.

Fourth Pass



	[1]	[2]	[3]	[4]	[5]
Before	1 (in place)	4 (in place)	6 (in place)	10	8
1st comparison	1 (in place)	4 (in place)	6 (in place)	8 (in place)	10 (in place)

Before writing down the algorithm in more detail, it should be pointed out that the second traversal need not extend to the first element, since by the time the second traversal starts, the first position in the list already contains its rightful tenant: the smallest value in the list. Similarly, the third traversal need not consider the first 2 elements, etc. This leads to an algorithm that carries out N-1 such traversals (why not N?) to produce the final list. On each pass or traversal, the algorithm need only compare N-1 elements in its first traversal, N-2 elements in the second, N-3 in its third, and so on. Thus the bubble sort algorithm involves two nested loops. The outer loop controls the number of (successively smaller) passes or traversals through the array. The inner loop controls the pairs of adjacent entries being compared.

```
for x = N-1 downto 1
  for y = N downto N-x+1
    if A[y] < A[y-1]
       swap the values in A[y] and A[y-1]
       endif
    endfor
endfor
```

Selection Sort

The basic idea of selection sort is to make repeated selections from a list of values, moving the selected value into its proper position in the list. On the first pass, find the smallest number in the list and exchange it with the one in the first position(A[1]). On the second pass, find the smallest number from the values in positions 2 on down and exchange it with A[2]. On the third pass, find and place the smallest remaining value into the third position, and so on until there are no more values in the unsorted portion of the list. Each pass puts one element into proper order, and reduces by one the number of elements in the unsorted portion.

The figures below walk through sorting a 5-element list. Each row is labeled as a comparison between items in bold print or a swap, wiht arrows showing items that were exchanged. Each pass is indicated separately for easier reading. Note that in addition to putting one element in its proper place, each iteration causes some intermediate changes in the order of the other elements also.

The first pass involves comparing each of the values to find the smallest, keeping track of its position (call it MIN) until the last value has been considered. Then that value indicated by MIN is swapped with the item in position 1. Here, 4 comparisons are made, and one swap. Note that this first traversal does not guarantee the entire list is sorted—it only ensures that the first element is. The first pass produces:

First Pass

	[1]	[2]	[3]	[4]	[5]
Before	10	4	8	6	1
1st comparison	**10**	**4** min	8	6	1
2nd comparison	10	**4** min	**8**	6	1
3rd comparison	10	**4**	8	**6** min	1
4th comparison	10	4	8	6	1 min
Swap	1 (in place)	4	8	6	10

The second pass can ignore the value in the first position and begin comparing values in positions 2 through 5. Notice that no swap is needed, since the value "4" was already in its proper position (in A[2]), yet 3 comparisons are made.

Second Pass

	[1]	[2]	[3]	[4]	[5]
Before	1 (in place)	4	8	6	10
1st comparison	1 (in place)	**4** min	**8**	6	10
2nd comparison	1 (in place)	**4** min	8	**6**	10
3rd comparison	1 (in place)	**4** min	8	6	**10**
Swap	1 (in place)	4 (in place)	8	6	10

The third pass will locate the smallest value in positions 3-5 and place it in position 3. This requires 2 comparison operations and one swap.

Third Pass

	[1]	[2]	[3]	[4]	[5]
Before	1 (in place)	4 (in place)	8	6	10
1st comparison	1 (in place)	4 (in place)	**8**	**6** min	10
2nd comparison	1 (in place)	4 (in place)	8	**6** min	**10**
Swap	1 (in place)	4 (in place)	6 (in place)	8	10

The fourth and final pass yields the sorted list, using one comparison but not needing any swap operations:

Fourth Pass

	[1]	[2]	[3]	[4]	[5]
Before	1 (in place)	4 (in place)	6 (in place)	8	10
1st comparison	1 (in place)	4 (in place)	6 (in place)	8 min	10
Swap	1 (in place)	4 (in place)	6 (in place)	8 (in place)	10 (in place)

Before writing the algorithm for this sorting procedure, note the following:

1. If the array is of length N, then it takes N-1 steps to put it into order
2. We must be able to find the smallest number. Numbers that are equal are not considered smaller than each other.
3. We need to exchange appropriate array components that are out of order (inverted)

```
for j = 1 to N-1
        for j = i to N
                MIN = the index of the smallest value encountered
                endfor
        swap the values in A[J] and A[MIN]
```

Questions

Use the data set below to consider answers for questions 1-5:

	[1]	[2]	[3]	[4]	[5]
Initial Values	8	4	26	2	7

		Bubble Sort	Selection Sort
1	Which element will move into the leftmost position on the first pass?		
2	How many comparisons will be needed to complete the first pass?		
3	How many swaps will occur in the first pass?		
4	How many passes will it take until the remaining values are in place?		
5	Write the order of the elements as they would appear at the completion of the second pass		

6. The lines to the Bubble Sort algorithm are out of sequence below. From memory, try to number them to represent the correct sequence

Line #	Algorithm Statement
	endfor
	for i = N-1 downto 1
	compare A[J] with A[J+1], exchanging if necessary
	for j = N downto N-i
	endfor

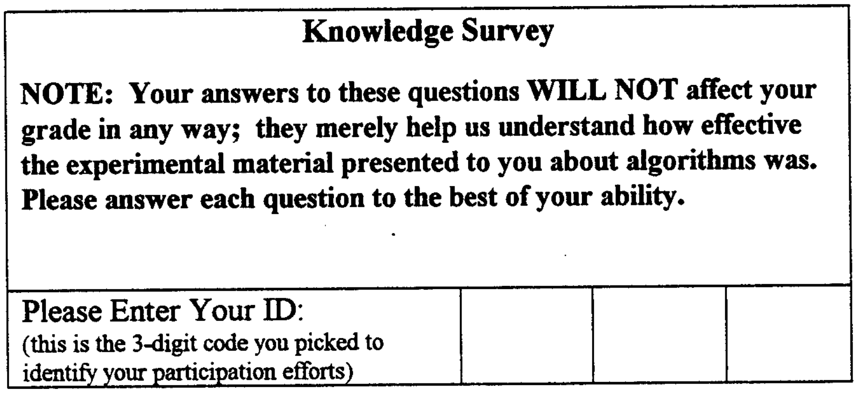
7. Do the same as problem #5 for the Selection Sort algorithm below:

Line #	Algorithm Statement
	for j = i to N
	endfor
	exchange the values of A[J] and A[MIN]
	MIN = the index of the smallest value encountered
	for j = 1 to N-1
	endfor

D.6. EXPERIMENTS WITH LECTURES

D.6.1. POSTTEST FOR EXPERIMENT III

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:



Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

Please Enter Your ID: (this is the 3-digit code you picked to identify your participation efforts)			

A1. The psueudocode to the right implements which popular sort algorithm?

A	Quicksort
B	Mergesort
C	Bubble Sort
D	Insertion Sort
E	Selection Sort
F	I Dont Know

```
for x= 1 to N-1
  for y = x+1 to N
     if ARRAY[y-1] > ARRAY[y]
        swap ARRAY[y-1] and ARRAY[y]
     endif
   endfor
endfor
```

A2. The psueudocode to the right implements which popular sort algorithm?

A	Quicksort
B	Mergesort
C	Bubble Sort
D	Insertion Sort
E	Selection Sort
F	I Dont Know

```
for x= 1 to N-1
   CHOICE = x
   for y = x+1 to N
      if ARRAY[y] < ARRAY[CHOICE]
         CHOICE = y
      endif
   endfor
   if CHOICE not = x
     swap ARRAY[CHOICE] and ARRAY[x]
   endif
   endfor
```

A3. Show the order of elements in ARRAY after the following number of passes have occurred, using the Selection Sort algorithm. Remember that a single pass is the execution of all statements in the body of the loop controlled by variable *x*.

Input ARRAY	*1*	*2*	*3*	*5*	*4*
After 1 pass (i.e when *x* = 1)					
After all passes (i.e. after *x* has looped from 1 to N-1)					

#Comparisons	**#Swaps**

A4. After 3 passes, which of the following are true about the Selection Sort algorithm:

A We are guaranteed that no more than 2 swap operations will be required
B We are guaranteed that the first 3 items will always be in proper position
C We cannot make any guarantees until we know the order of the input
D All of the above are true
E None of the above is true

A5. If the data is already sorted into the proper order, can you think of a way for the Selection Sort algorithm to recognize this and prevent unnecessary comparisons and work?

A6. If you read the section in your textbook that covers the Selection Sort algorithm, please indicate about how many minutes you spent doing it:

B1. What would happen if line #15 was altered to read:

if LeftHalf[x] > RightHalf[y]

A ResultArray would be in ascending order

B ResultArray would have scrambled results

C ResultArray would be in descending order

D None of the above

```
1- Procedure MergeSort(ARRAY)
 2-  if length(ARRAY) > 1
 3-    Middle = length(ARRAY)/2
 4-     LeftArray  = MergeSort(Array[1..Middle])
 5-     RightArray = MergeSort(Array[Middle+1..N])
 6-     ResultArray = Merge (LeftArray,RightArray)
 7-     Return ResultArray
 8-  Else
 9-     Return Array
10-  Endif
11-   EndMergeSort

12- Proc Merge(LeftHalf,RightHalf)
13- initialize x, y, z to 1
14- loop
15-      if LeftHalf[x] < RightHalf[y]
16-           Result[z] = LeftHalf[x]
17-           Increment X and Z
18-      else
19-           Result[z] = RightHalf[y]
20-           Increment y and z
21-           endif
22-      until RightHalf or LeftHalf is empty
23-      Flush remaining elements from RightHalf
              (or LeftHalf)  to Result
24-     return Result
25-     endMerge
```

B2. This MergeSort algorithm will be used to sort an array of test scores. Which of the following is true:

(a) The sort will take longer and work harder if the original scores are ordered from smallest to largest

(b) The sort will take longer and work harder if the original scores are in random order

(d) The sort will take longer and work harder if the original scores are ordered from largest to smallest

(d) The sort will do the same amount of work and time no matter the order of the original elements.

(e) None of the above are true

B3. If the following data was passed to the Merge procedure on line #10, indicate what would be returned in the array called Results. Use the unaltered version of the algorithm.

LeftArray

51	57	74

RightArray

19	27	30	33	71

ResultArray

B4. Using algorithm above on the data shown below, indicate how many calls to the MergeSort and Merge procedures would be needed to place the data into ascending order, and the contents of the array when the call(s) are completed:

ARRAY:

Before	**Evan**	**Moe**	**Al**	**Dirk**	**Carl**	**Zed**	**Art**
After							

# of calls to MergeSort:	# of calls to Merge:

B5. If you read the section in your textbook that covers the MergeSort algorithm, about how many minutes do you remember it took you?

Please list some of the things you liked about the Algorithm Visualization Program:

Please list some of the things you didn't like, or provide some suggestions we could use to improve the program:

THANK YOU FOR PARTICIPATING—
WE TRULY APPRECIATE YOUR TIME AND SUGGESTIONS!

D.7. EXPERIMENTS WITH OTHER ALGORITHM ANIMATION SYSTEMS

D.7.1. POSTTEST FOR EXPERIMENT IV

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:

Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

Please Enter Your ID: (this is the 3-digit code you picked to identify your participation efforts)			

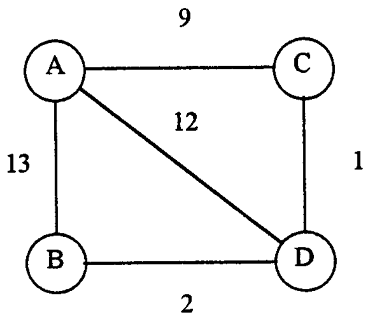
```
Procedure DijkstraShortestPath
1-  Initiailization: mark each vertex in table as UnKnown
        with infinite distance and no parent
2-  Pick a Starting Vertex and make its distance in table = 0
3-  Repeat
4-    J = vertex with minimal distance among UnKnown vertices
    in table
5-    mark J as Known in table
6-    for each vertex R still UnKnown in table
7-      if there is an edge from J to R
8-        if Distance[J] + EdgeWeight(J,R) < Distance[R]
9-            Distance[R] = Distance[J] + EdgeWeight(J,R)
10-           Parent[R] = J
11-     until all vertices are Known in table
```

Vertex id	Known?	Distance (d_v)	Parent (p_v)
id-1	UnKnown	∞	--
id-2	UnKnown	∞	--
id-3	UnKnown	∞	--
id-4	UnKnown	∞	--

This table represents the data structure used by Dijkstra's Shortest Path algorithm

1. What drives the order in which vertices are marked as Known?

Line #1 of the algorithm above on the graph below leads to a table initialized as indicated:



Vertex	Known?	Distance (d_v)	Parent (p_v)
A	UnKnown	∞	--
B	UnKnown	∞	--
C	UnKnown	∞	--
D	UnKnown	∞	--

2. Indicate the order that nodes would be marked as 'Known' if **vertex A** was the starting point (ex: A-B-C-D)

3. Show the final contents of the data structure in the table below (right) if **vertex A** was the Starting Point

Before Execution

Vertex	Known?	Distance (d_v)	Parent (p_v)
A	UnKnown	0	--
B	UnKnown	∞	--
C	UnKnown	∞	--
D	UnKnown	∞	--

Upon Completion

Vertex	Known?	Distance (d_v)	Parent (p_v)
A		0	
B			
C			
D			

4. How many times will the algorithm consider/examine the distance to **vertex D** in solving problem #3

5. What can be said after k (some arbitrary number) iterations of lines 3-11:
   - (f) About k^2 comparisons will have been made
   - (g) Approximately *log k* comparisons will have been made
   - (h) Exactly k vertices will have been visited
   - (i) k unique parents will have been identified.
   - (j) At least k solution paths will have been found

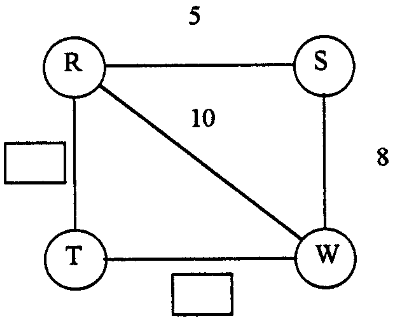
6. Dijkstra's algorithm reaches line 4 with the following results (the graph is not shown for this data):

Vertex	Known?	Distance (d_v)	Parent (p_v)
1	Known	0	
2	Known	1	1
3	UnKnown	3	2
4	UnKnown	4	1

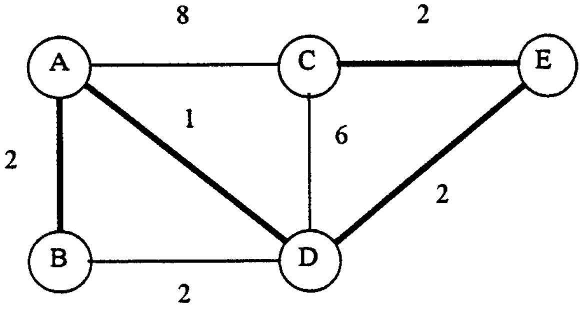
What will J be when line 4 is executed?

7. What would the weights have to be in order to force the algorithm to mark the nodes as Known in the order:

R-T-W-S

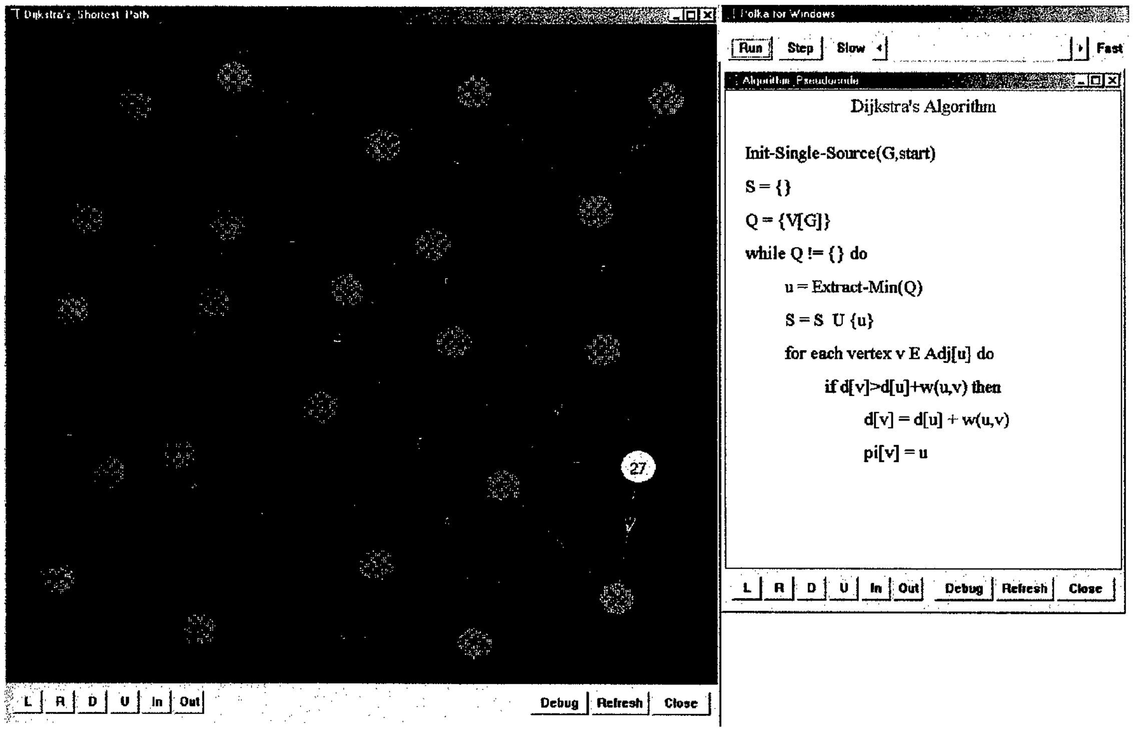


8. Given the following graph, fill in the appropriate values in the table as they would appear after the algorithm finished executing. The bold lines indicate the shortest paths from source vertex A to each of the other nodes that the algorithm found.



Vertex	Known ?	Distance (d_v)	Parent (p_v)
A		0	
B			
C			
D			
E			

D.7.2. SCREEN CAPTURE OF VISUALIZATION USED BY AA GROUP



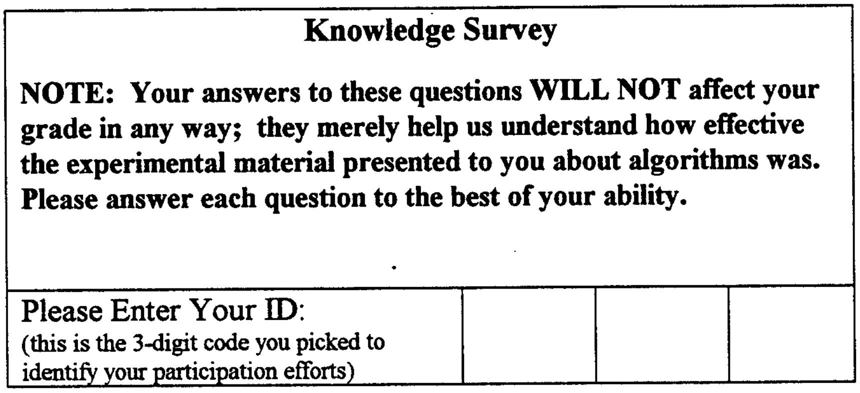
Dijkstra's Algorithm

```
Init-Single-Source(G,start)
S = {}
Q = {V[G]}
while Q != {} do
    u = Extract-Min(Q)
    S = S U {u}
    for each vertex v E Adj[u] do
        if d[v]>d[u]+w(u,v) then
            d[v] = d[u] + w(u,v)
            pi[v] = u
```

D.8. EXPERIMENT: ABLATION OF FEATURES

D.8.1. POSTTEST FOR FEATURES ABLATION STUDY

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:



Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

Please Enter Your ID: (this is the 3-digit code you picked to identify your participation efforts)			

1. Does the pseudocode to the right sort the elements in ARRAY into ascending or descending order?

```
1- Procedure QuickSort(ARRAY[L..R])
2-  if length(ARRAY) > 1
3-    initialize VALUE = ARRAY[L]
4-    initialize LPtr = L+1  and  RPtr = R
5-    repeat
6-      while RPtr > LPtr and ARRAY[LPtr] < VALUE
7-         increment LPtr
8-      while RPtr > LPtr and ARRAY[RPtr] > VALUE
9-         decrement RPtr
10-     swap ARRAY[RPtr] and ARRAY[LPtr]
11-     increment LPtr and decrement RPtr
12-     until RPtr <= LPtr
13-   swap VALUE in ARRAY[L] with ARRAY[RPtr]
14-   ARRAY[L..RPtr-1] = QuickSort(ARRAY[L.. RPtr-1])
15-   ARRAY[RPtr+1..R]= QuickSort(ARRAY[RPtr+1.. R)
16- else
17-   return ARRAY[L..R]
18-   endif
19- endproc
```

2. Make pen and ink changes to the pseudocode so that it produces a sorted sequence that is the opposite of what it currently generates.

3. What can be said after *k* (some arbitrary number) calls to QuickSort :

(k) Exactly *k-1* Merge operations will have been performed

(l) No more than *k* swaps will have been made.

(m) About k^2 comparisons will have been made

(n) At least *k* elements will always be in place regardless of the original order of the input

4. If the following data was passed to the QuickSort procedure, indicate the contents of the ARRAY after the first execution of the loop depicted in lines #5-13. Use the unaltered version of the pseudocode in question 1.

Original Input						
35	14	26	9	71	55	33

Result after 1st execution of lines 5-13		
ARRAY[L...RPtr-1]	ARRAY[RPtr]	ARRAY[RPtr+1..R]

5. The effect(s) of the statements on lines 3-13 of Quick Sort include (circle all that apply):

(a) Always places at least one value into its final sorted position

(b) Migrates smaller elements to the left and large elements to the right

(c) Places all elements in the right hand side into sorted order

(d) Divides the array into equal sized partitions for sorting

(e) All of the above

(f) None of the above

6a. If the values [21,14,40,25,55,20] were passed to QuickSort, what pair of numbers would be exchanged first?

(a) 14 and 21

(b) 14 and 40

(c) 20 and 40

(d) 21 and 40

(e) none of the above

6b. What pair would be exchanged second?

(a) 14 and 20

(b) 14 and 21

(c) 20 and 21

(d) 25 and 55

(e) none of the above

7. QuickSort reaches line 14 with the following results:

After lines 1-13:		
ARRAY[L..RPtr-1]	ARRAY[RPtr]	ARRAY[RPtr+1..R]
4, 1, 6, 3, 7	**8**	**18, 29, 11, 20**

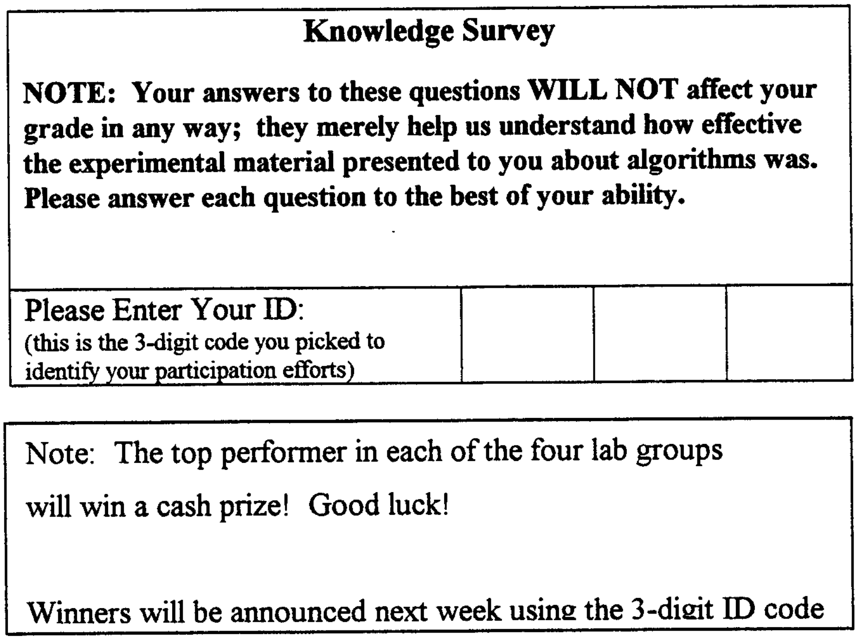
The next time line 3 is reached, what will VALUE contain?

What two numbers would be the next to be swapped?

D.9. EXPERIMENT: DUAL VIEW ABLATION

D.9.1. POSTTEST FOR DUAL VIEW ABLATION

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:



Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

Please Enter Your ID: (this is the 3-digit code you picked to identify your participation efforts)			

Note: The top performer in each of the four lab groups will win a cash prize! Good luck!

Winners will be announced next week using the 3-digit ID code

1. Does the pseudocode to the right sort the elements in ARRAY into ascending or descending order?

```
1- Procedure QuickSort(ARRAY[L..R])
2-   if length(ARRAY) > 1
3-     initialize VALUE = ARRAY[L]
4-     initialize LPtr = L+1  and   RPtr = R
5-     repeat
6-       while RPtr > LPtr and ARRAY[LPtr] < VALUE
7-           increment LPtr
8-        while RPtr > LPtr and ARRAY[RPtr] >= VALUE
9-           decrement RPtr
10-       swap ARRAY[RPtr] and ARRAY[LPtr]
11-       increment LPtr and decrement RPtr
12-       until RPtr <= LPtr
13-     swap VALUE in ARRAY[L] with ARRAY[RPtr]
14-     ARRAY[L..RPtr-1] = QuickSort(ARRAY[L.. RPtr-1])
15-     ARRAY[RPtr+1..R]= QuickSort(ARRAY[RPtr+1.. R])
16-  else
17-     return ARRAY[L..R]
18-     endif
19- endproc
```

2. Make pen and ink changes to the pseudocode so that it produces a sorted sequence that is the opposite of what it currently generates.

3. The effect(s) of the statements on lines 3-13 of Quick Sort include (circle **all** that apply):

(a) Migrates smaller elements to the left and large elements to the right

(b) Always places at least one value into its final sorted position

(c) Places all elements in the right hand side into sorted order

(d) Divides the array into equal sized partitions for sorting

(e) None of the above

4. If the following data was passed to the QuickSort procedure, indicate the contents of the ARRAY after the first execution of the loop depicted in lines #5-13. Use the unaltered version of the pseudocode in question 1.

Original Input

35	**14**	**26**	**9**	**71**	**55**	**33**

Result after 1st execution of lines 5-13

ARRAY[L...RPtr-1]	ARRAY[RPtr]	ARRAY[RPtr+1..R]

5. What can be said after *k* (some arbitrary number) calls to QuickSort :

(a) Exactly *k-1* Merge operations will have been performed

(b) No more than *k* swaps will have been made.

(c) About k^2 comparisons will have been made

(d) Approximately *k* elements will always be in place regardless of the order of the input

6a. If the values [21,14,40,25,55,7,20] were passed to QuickSort, what pair of numbers would be exchanged first?

□ and □

6b. What pair would be exchanged second?

□ and □

6c. What pair would be exchanged third?

(a) 7 and 21

(b) 14 and 21

(c) 20 and 21

(d) There won't be a third exchange

6d. How many calls to QuickSort will it take to complete the sort from start to finish, using the data in #6a?

(a) 3

(b) 4

(c) 7

(d) 11

(e) None of the above

7. A different execution of QuickSort reaches line 14 with the following results:

After lines 1-13:		
ARRAY[L..RPtr-1]	ARRAY[RPtr]	ARRAY[RPtr+1..R]
4, 1, 6, 3, 7, 2	**8**	**18, 29, 11, 20**

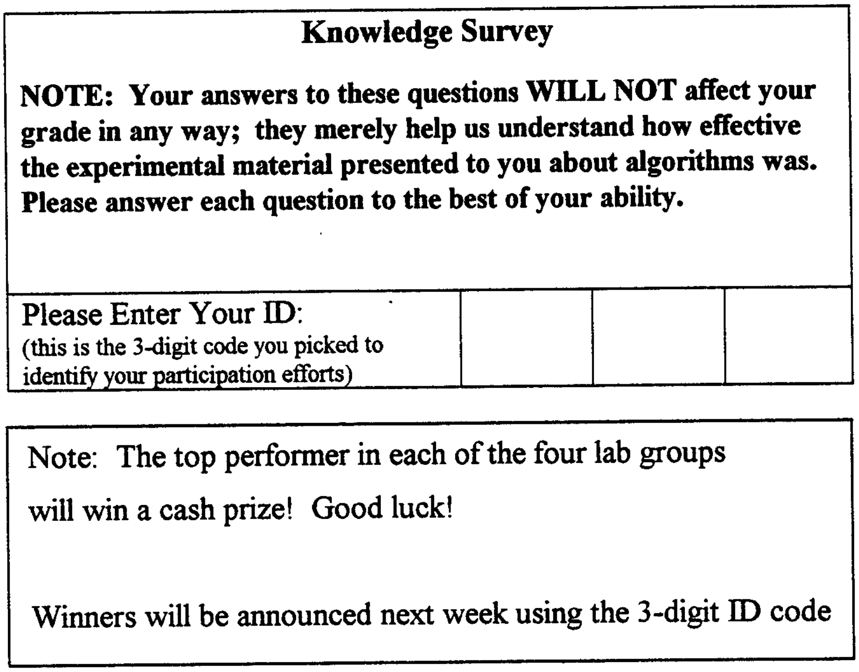
The next time line 3 is reached, what will VALUE contain? □

What two numbers would be the next to be swapped? □ □

D.10. EXPERIMENT: SINGLE VIEW ABLATION

D.10.1. POSTTEST FOR SINGLE VIEW ABLATION STUDY

Each test included a cover page with space for the subject to enter his or her identification code (a 3 digit code that gave anonymity to each participant but allowed us to track performance through the phases of the experiment), and the disclaimer shown below:



Knowledge Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

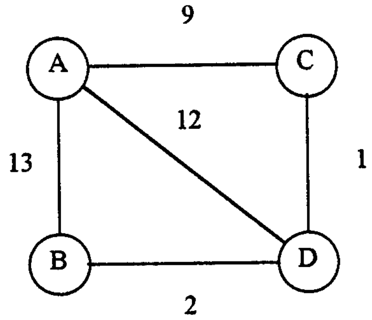
Please Enter Your ID: (this is the 3-digit code you picked to identify your participation efforts)			

Note: The top performer in each of the four lab groups will win a cash prize! Good luck!

Winners will be announced next week using the 3-digit ID code

1. In Dijkstra's Shortest Path algorithm, what drives the order in which vertices are marked as Visited?

Based on the example graph below, the initialization step of the Shortest Path algorithm leads to a table initialized as indicated:



Vertex	Visited?	Distance (d_v)	Parent (p_v)
A	UnVisited	∞	--
B	UnVisited	∞	--
C	UnVisited	∞	--
D	UnVisited	∞	--

2. Indicate the order that nodes would be marked as 'Visited' if **vertex A** was the starting point (ex: A-B-C-D)

3. Show the final contents of the data structure in the table below (right) if **vertex A** was the Starting Point

Before Execution

Vertex	Visited?	Distance (d_v)	Parent (p_v)
A	UnVisited	0	--
B	UnVisited	∞	--
C	UnVisited	∞	--
D	UnVisited	∞	--

Upon Completion

Vertex	Visited?	Distance (d_v)	Parent (p_v)
A		0	
B			
C			
D			

4. How many times will the algorithm consider/examine the distance to **vertex D** in solving problem #3

5. What can be said after *k* (some arbitrary number) iterations/loops of the basic ShortestPath algorithm:

(a) About k^2 comparisons will have been made

(b) Approximately *log k* comparisons will have been made

(c) Exactly *k* vertices will have been visited

(d) *k* unique parents will have been identified.

(e) At least *k* solution paths will have been found

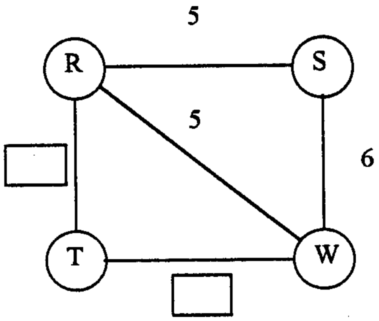
6. Dijkstra's algorithm is about to begin another iteration, with interim results as shown in the table below (note the graph is **not** shown for this data):

Vertex	**Visited?**	**Distance (d_v)**	**Parent (p_v)**
1	Visited	0	
2	Visited	1	1
3	UnVisited	3	2
4	UnVisited	4	1

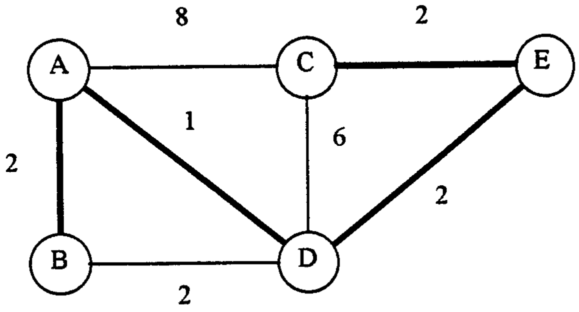
Based on these interim results, what Vertex will be the next one to be marked as Visited?

7. What would the weights have to be in order to force the algorithm to mark the nodes as Visited in the order:

R-T-W-S



8. Given the following graph, fill in the appropriate values in the table as they would appear after the algorithm finished executing. The bold lines indicate the shortest paths from source **vertex A** to each of the other nodes that the algorithm found.



Vertex	**Visited?**	**Distance (d_v)**	**Parent (p_v)**
A		0	
B			
C			
D			
E			

9. The statements below represent the steps involved in the Shortest Path algorithm, but they are obviously out of order. Number the statements below to indicate their proper order. Note that statement #1 is already marked for you. As a convenience, the table used by the algorithm is shown to the right.

Line #	Algorithm Statements
	EndRepeat
	if Distance[J] + EdgeWeight(J,R) < Distance[R]
	mark J as Visited in table
	Parent[R] = J
	J = vertex with minimal distance among UnVisited vertices in table
	Endif
	Pick a Starting Vertex and make its distance in table = 0
	Distance[R] = Distance[J] + EdgeWeight(J,R)
	if there is an edge from J to R
	for each vertex R still UnVisited in table
	Endif
	until all vertices are Visited in table
1	mark each vertex in table as UnVisited with infinite distance and no parent
	Repeat

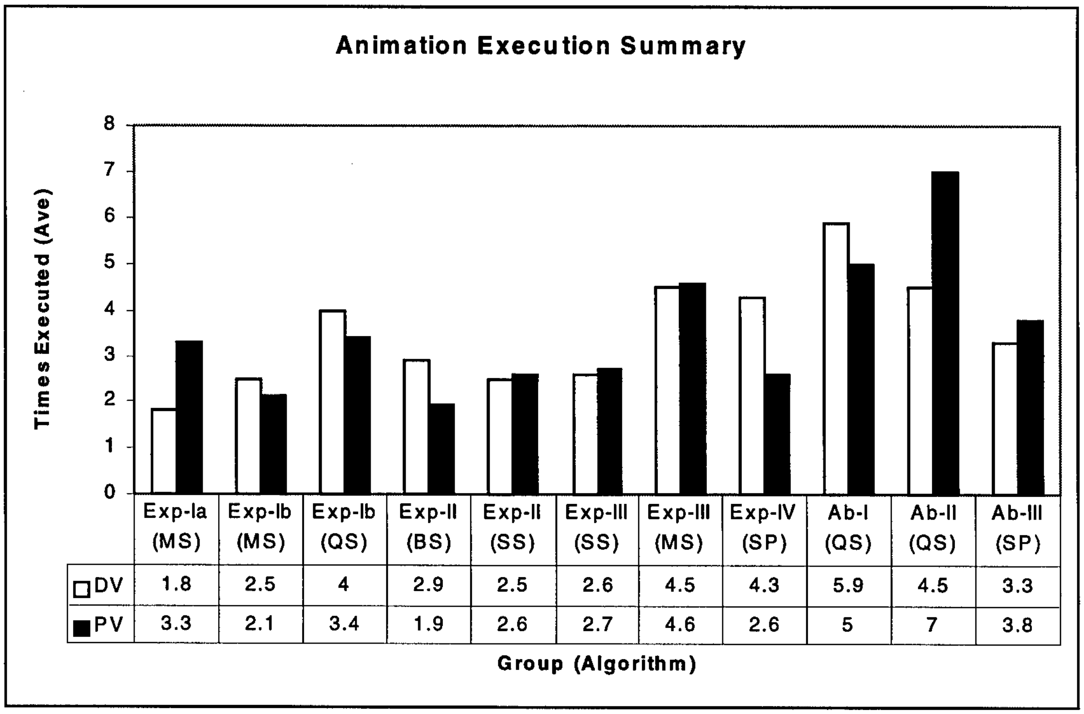
Vertex id	Visited?	Dist (d_v)	Parent (p_v)
id-1	UnVisited	∞	–
id-2	UnVisited	∞	–
id-3	UnVisited	∞	–
id-4	UnVisited	∞	–

This table represents the data structure used by Dijkstra's Shortest Path algorithm

APPENDIX C: USAGE AND INTERACTION DATA

This section contains figures, tables and data gathered from subjective surveys that subjects completed in two of the experiments, and from user interaction profiles that HalVis logged during the experiments.

The figure below shows the number of times the animations on the Detailed View (DV) and Populated View (PV) were executed by students for each of the experiments reported in this document. In the case of the ablation studies where selected features or views were removed, only the totals for the group receiving all views are provided.



	Exp-Ia (MS)	Exp-Ib (MS)	Exp-Ib (QS)	Exp-II (BS)	Exp-II (SS)	Exp-III (SS)	Exp-III (MS)	Exp-IV (SP)	Ab-I (QS)	Ab-II (QS)	Ab-III (SP)
□DV	1.8	2.5	4	2.9	2.5	2.6	4.5	4.3	5.9	4.5	3.3
■PV	3.3	2.1	3.4	1.9	2.6	2.7	4.6	2.6	5	7	3.8

Figure C.1. Summary of Animation Executions for All Experiments

C.1 EXPERIMENT II USAGE AND SATISFACTION DATA

A satisfaction survey was provided to all subjects to assess their perception of the HalVis system and their general attitudes formed during interacting with it. The scale for the satisfaction survey ranged from (-2) for very bad to (+2) for very good. The columns Max and Min indicate the highest and lowest rating of all the responses.

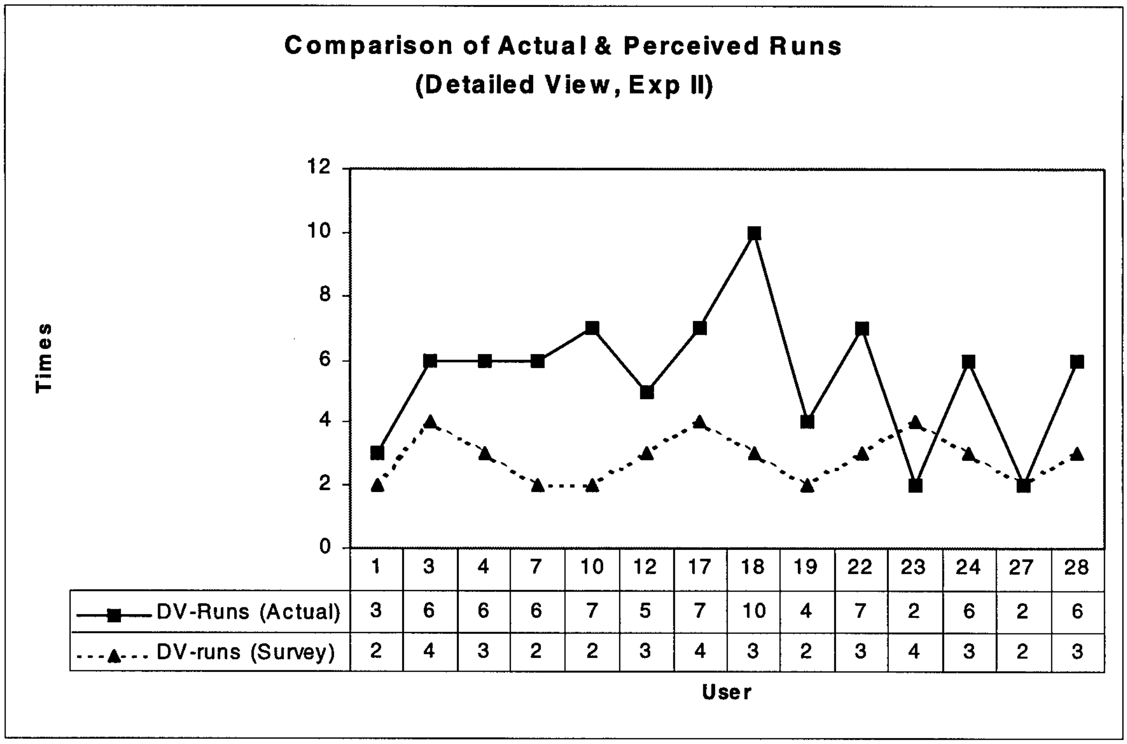
Survey Item	Average	Max	Min
Use of textual descriptions (pop ups)	1.3	2	1
Use of audio	0.8	2	-1
Use of color to highlight actions	1.5	2	0
use of analogies/examples to introduce algorithms	1.6	2	0
Use of hyperlinking to find related information	1.4	2	0
Explanation of learning objectives on each screen	1.1	2	0
Use of pop-up questions	1.0	2	-1
Clarity of button and menu names	1.4	2	-1
DETAILED VIEW ITEMS			
Effectiveness of textual description of algorithm	1.2	2	0
Use of contextual explanations during animation	1.3	2	0
Highlighting pseudocode statements during animation	1.5	2	0
Counters that reflect the progress of the algorithm	1.4	2	1
Ability to alter the speed of the animation	1.8	2	1
Ability to rerun the animation	1.9	2	1
Did you try changing the data (1=Yes, 0=No)	0.6	1	0
How many times did you run the animation	2.9	4	2
Actual figure from usage logs:	*5.5*	*10*	*2*
POPULATED VIEW ITEMS			
What was your impression of the Populated View	1.3	2	0
Counters that reflect the progress of the algorithm	1.4	2	1
Ability to predict the performance of the algorithm	1.1	2	-1
Ability to rerun the animation	1.6	2	1
Ability to alter the speed of the animation	1.9	2	1
Use of colors to illustrate actions of the algorithm	1.4	2	0
How many times did you run the animation	2.6	4	1
Actual figure from usage logs:	*4.1*	*7*	*2*
Data case used most (3=BestCase,2=Random,1=WorseCase)	1.9	3	1
OVERALL IMPRESSIONS			
What was your impression of this program	1.4	2	1
What was your impression of the user interface	1.3	2	0
What was your impression of the navigation	1.3	2	1

Table C.1. Experiment II Satisfaction Survey Data

Other observations from the survey:

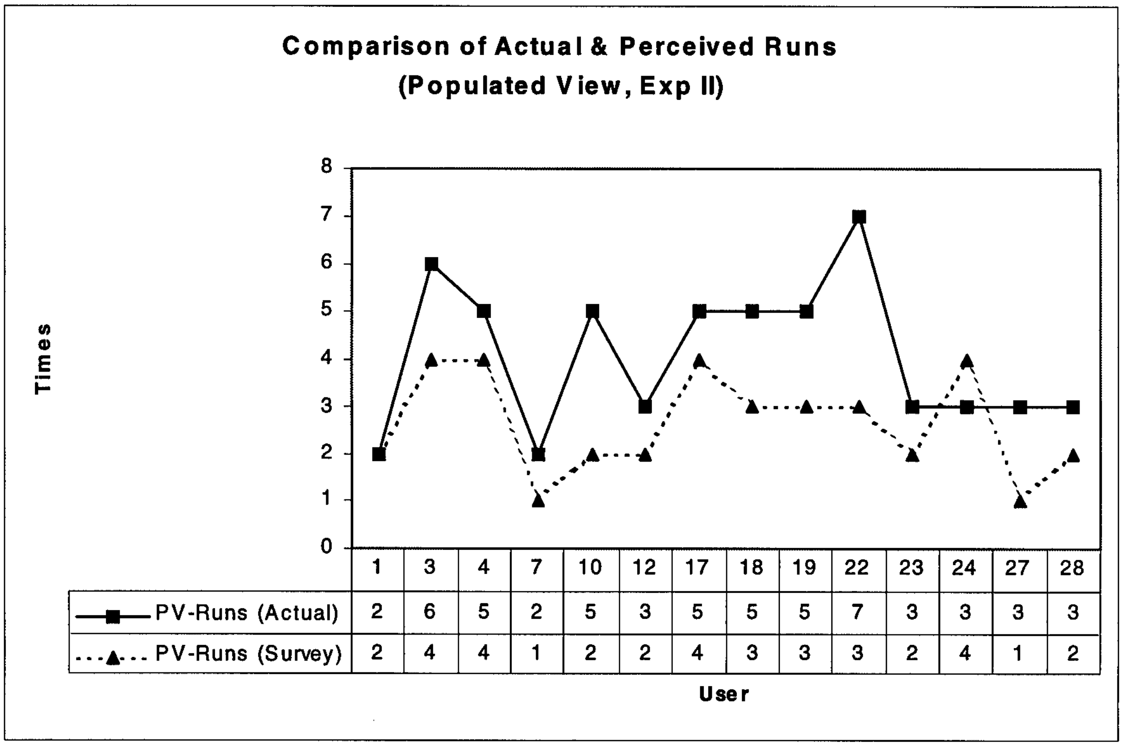
- 64% gave use of analogies to introduce algorithms the highest rating (2)
- 100% rated use of analogies #1 or #2
- 86% understated the number of times they thought they ran DV animations compared to actual times they ran it. 86% also understated the number of times they ran PV animations
- 71% used random ordering for the populated view. 12% used worse case ordering once.
- 57% accessed the control panel on the DV
- Only 10% tried changing the animation data (90% didn't) in the Detailed View.
- 84% said they would choose to use the HalVis system to learn other algorithms if given the choice. The other 16% indicated no specific choice.

The figure below compares the number of times that students remembered running the animations, as reported in the subjective survey, to the actual number of times as reported in the HalVis usage logs. Except for student #23, everyone believed they ran the animations fewer times than they actually did.



User	1	3	4	7	10	12	17	18	19	22	23	24	27	28
—■— DV-Runs (Actual)	3	6	6	6	7	5	7	10	4	7	2	6	2	6
···▲··· DV-runs (Survey)	2	4	3	2	2	3	4	3	2	3	4	3	2	3

Figure C.2. Actual and Perceived Detailed View Animation Executions, Experiment II



User	1	3	4	7	10	12	17	18	19	22	23	24	27	28
—■— PV-Runs (Actual)	2	6	5	2	5	3	5	5	5	7	3	3	3	3
···▲··· PV-Runs (Survey)	2	4	4	1	2	2	4	3	3	3	2	4	1	2

Figure C.3. Actual and Perceived Populated View Animation Executions, Experiment II

Positive Comments from Experiment II Subjective Survey	
I liked: the animations the examples and analogies the questions & making predictions the step-by-step progress on algorithm pseudocode seeing efficiency…("I didn't realize there was that much difference") the voice explanations during the animation	6 comments 3 comments 3 comments 2 comments 1 comment 1 comment
The software: made the algorithm easier to understand was much better than reading a book made learning algorithms more interesting	2 comments 1 comment 1 comment

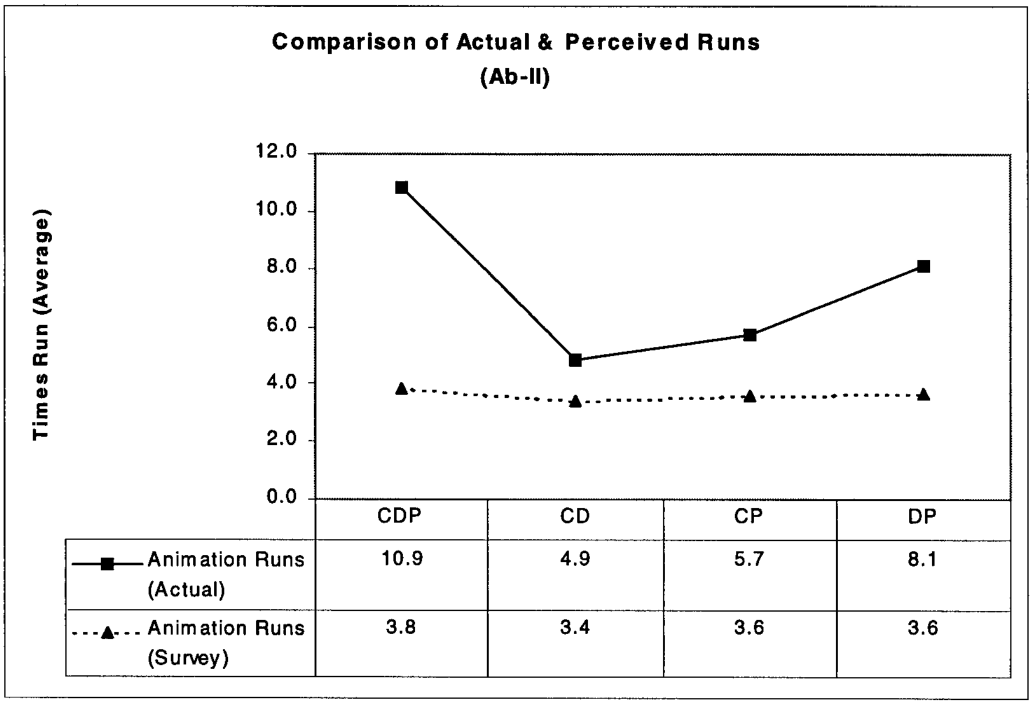
Negative Comments from Experiment II Subjective Survey	
Sound: The sound was annoying It could use more sound	1 comment 2 comments
Text: The text was too long and dull Needs more detail in the textual explanations	2 comments 1 comment
The system was too simplistic…almost like we weren't supposed to know anything about sorting	

Table C.2. Summary of Student Remarks and Comments Following Experiment II

C.2 EXPERIMENT VI USAGE AND SATISFACTION DATA

Survey Item	CDP	CD	CP	DP
Use of textual descriptions (pop ups)	1.0	1.1	1.1	1.4
Use of audio	0.1	-0.1	0.0	-0.1
Use of color to highlight actions	1.3	1.4	1.3	1.3
Use of hyperlinking to find related information	1.3	1.6	0.9	1.6
Explanation of learning objectives on each screen	1.3	1.0	0.8	0.8
Use of pop-up questions	1.3	0.3	1.2	0.8
Clarity of button and menu names	1.1	1.1	0.6	1.1
DETAILED VIEW ITEMS				
Ability to alter the speed of the animation	1.1	1.1	0.8	1.5
Ability to rerun the animation	1.9	1.3	1.3	1.4
Did you try changing the data (1=Y, 0=N)	1.0	1.0	0.7	0.5
How many times did you run the animation	3.9	3.5	3.6	3.6
OVERALL IMPRESSIONS				
What was your impression of this program	1.3	0.4	1.3	1.3
What was your impression of the user interface	1.1	1.3	1.3	0.9
What was your impression of the navigation	1.0	1.4	1.2	0.5

Table C.3. Experiment VI Satisfaction Survey Data



	CDP	CD	CP	DP
—■— Animation Runs (Actual)	10.9	4.9	5.7	8.1
---▲--- Animation Runs (Survey)	3.8	3.4	3.6	3.6

Figure C.4. Actual and Perceived Animation Executions, Experiment VI.

Positive Comments from Experiment VII Subjective Survey	
I liked:	
the animations ("made understanding faster and easier…")	15 comments
the examples and analogies	2 comments
the questions & making predictions ("the question about rearranging the	3 comments
out-of-order pseudocode was very helpful")	2 comments
the step-by-step progress shown by highlighting algorithm pseudocode	3 comments
the Efficiency screen was quite revealing")	1 comment
the text explanations	2 comments
being able to enter my own data	5 comments
The software:	
made the algorithm easier to understand	2 comments
was much better than reading a book	1 comment
made learning algorithms more interesting	1 comment
The hyperlinks were helpful	1 comment

Negative Comments from Experiment VII Subjective Survey	
The text:	
There was too much text	2 comments
The text was small and hard to read	1 comment
There wasn't enough text to explain the steps	3 comments
The software:	
Needed more sound	1 comment
Needed less color	1 comment
The question about rearranging the out-of-order pseudocode was hard	6 comments
The animation:	
Needed a reverse button	1 comment
Was hard to stop once it was started	3 comments
The highest speed was not fast enough	2 comments
The populated view mooved too quickly, even at the slowest speed	1 comment
setting	1 comment
Should only be allowed to advance by mouse clicks	
I didn't like the algorithms expressed as pseudocode	1 comment

Table C.4. Summary of Student Remarks and Comments Following Experiment VI

APPENDIX D. THE JAWAA COMMANDS

Researchers at Duke University (Pierson & Rodger, 1998) have created a system called JAWAA that implements the XTango command set using Java, which allows Internet delivery of animations. They have extended the XTango command set to facilitate animation of complex data structures, making it one of the best general purpose animation systems available. This section is provided for the convenience of the reader to include a description of the JAWAA script commands that the JAVIZ system also recognizes (because of its JAWAA heritage). This is a replica of the command syntax summary available at http://www.cs.duke.edu/~wcp/commands.html.

D.1. GRAPHIC OBJECT CREATION COMMANDS

D.1.1. PRIMITIVE OBJECTS

• **circle**	name	*string uniquely identifying this object*
	x	*x coordinate (int)*
	y	*y coordinate (int)*
	diameter	*(int)*
	color	*color of circle drawn*
	bkgrd	*color indicating background*

Example: circle c1 20 20 30 black transparent

This creates a circle with the upper left corner at coordinate 20, 20, with a diameter of 20 and with the circle black and the interior transparent

- **line**

name	*string uniquely identifying this object*
x1	*starting x coordinate (int)*
y1	*starting y coordinate (int)*
x2	*ending x coordinate (int)*
y2	*ending y coordinate (int)*
color	*color of line drawn*

Example: line l1 20 20 40 40 30 black

This will create a black line extending from 20,20 to 40,40

- **text**

name	*quoted string uniquely identifying this object*
x	*x coordinate (int)*
y	*y coordinate (int)*
text	*string of text, in quotes*
color	*color of text*

Example text t1 40 40 "HELLO" red

This will write the string "HELLO" at 40,40 in red

- **rectangle**

name	*string uniquely identifying this object*
x1	*topleft x coordinate (int)*
y1	*topleft y coordinate (int)*
x2	*bottomright x coordinate (int)*
y2	*bottomright y coordinate (int)*
color	*color of rectangle drawn*
bkgrd	*background color*

Example: rectangle r1 10 20 100 120 black red

This will create a rectangle with its upper left corner at 10,20 and lower right corner at 100,120. It will be outlined in black and red in the interior

• **polygon**	name	*string uniquely identifying this object*
	length	*number of points in coordinate list*
	x1, y1 ... xn, yn	*list of coordinates connected by lines to form the polygon*
	color	*color of polygon drawn*
	bkgrd	*color of polygon background*

Example polygon p1 3 15,10 30,5 40,20 yellow blue

This will create a triangle with vertices at (15,10), (30,5), and (40,20). The edge will be yellow with blue on the interior

D.2. DATA STRUCTURE OBJECT CREATION COMMANDS

D.2.1. TREE OBJECTS

The tree command will create a binary tree based on the connections listed in the command. All node names must be integers less than 100. Connections made between nodes will be created automatically using names in the form "startNode"-"endNode" where " startNode" has been replaced with the name of the node from which the connection orginates.

• **tree**	name	*string uniquely identifying this object*
	x	*x coordinate of root node(int)*
	y	*y coordinate of root node(int)*
	width	*(int)*
	(start1, end1) ... (startn, endn)	*list of connections between nodes. The startNode must have already been mentioned as an endNode or must be the root node.*

Example tree t1 20 20 300 (1,2) (1,3) (2,4)

This will create a tree with width 300 at 20,20 with the connections described

• **addNode**	name	*name of tree*
	(start-i, end-i)	*connection to add to tree*

Example addNode t1 (3,5)

This will add the connection (3,5) to tree t1

D.2.2. GRAPH OBJECTS

- **node**

name	*string uniquely identifying this object*
x	*x coordinate (int)*
y	*y coordinate (int)*
diameter	*int*
color	*color of circle drawn*
bkgrnd	*color indicating background*

Example node n1 40 30 20 black transparent

This will create a node at 40,30 with diameter 20 using a black outline and a transparent interior.

- **connectNodes**

name	*string uniquely identifying this object*
node1	*string name of the node arc is coming from*
node2	*string name of the node arc is going to*
color	*color of arc*
animateDraw	*``true" or ``false" indicating whether to animate the arc draw*

Example connectNodes a1 2 3 black true

This command will animate the connection of node 2 to node 3.

- **marker**

name	*string uniquely identifying this object*
nodeName	*node where marker starts*
diameter	*radius of marker (int)*
color	*color of marker edge*
bkgrnd	*color of background*

Example marker m1 2 10 black red

This will create a black marker with red interior located at node 2

- **moveMarker**

name	*string uniquely identifying this object*
node1	*string identifying the node to leave*
node2	*string identifying the destination node*
connectionNm	*string identifying the arc/edge to follow*
color	*color of marker edge*
bkgrd	*color of background*

Example moveMarker m1 2 3 a1 black blue

Animates the marker's movement from node 2 to 3 along arc "a1"

- **graph**

name	*name of graph to create*
x	*x coordinate of upper left corner of graph*
y	*y coordinate of upper left corner of graph*
type	*string indicating drawing style, can be "CIRCLE" or "NORMAL"*
width	*width of graph*
(start1, end1)	*list of connections that make up the graph*
...	
(startn, endn)	

Example

graph g1 1 1 NORMAL 300 (1,2) (1, 6) (6, 2) (6,3) (2,3) (5, 1)

Creates a normal graph at 1,1 with width 300 using the listed connections to build the connections. As with trees, the connections created are named with the form: "startNode"-"endNode". Note that NORMAL graphs must be connected, while CIRCLE graphs can have unconnected nodes

- **addNode**

name	*name of graph*
node	*name of node to add*

Example

addNode g1 10

Adds node 10 to graph g1. Note that this command can only be used with graphs of type CIRCLE

- **addEdge**

name	*name of graph*
(start-i, end-i)	*connection between two nodes*

Example

addNode g1 (10,11)

Adds a connection between nodes 10 and 11. Note that if used with a CIRCLE graph neither node must already be in the graph, but when used with NORMAL graph one of the nodes must already be in the graph

D.2.3. ARRAY OBJECTS

- **array**

name	*name of array*
x	*x coordinate*
y	*y coordinate*
length	*number of cells in array*
value1- valuen	*list of cell values*
orientation	*can be "horz" or "vert"*
color	*color of array outline*
bkgrd	*color of background*

Example array 20 30 3 4d " " hello horz black red

Creates a horizontal array at 20,30 with 3 cells. This first cell contains "4d", the second is blank, and the third contains "hello". All cells in arrays can be accessed individually as rectangles. After creating an array called ``a1" we could access the fourth cell by referring to an object called ``a1[3]". For example if we wanted to change the color of this cell to blue we would use the command: changeParam a1[3] bkgrd blue

D.2.4. STACK OBJECTS

- **stack**

name	*name of stack*
x	*x coordinate*
y	*y coordinate*
length	*length of stack contents*
value1-valuen	*list of stack values, beginning with top item*
color	*color of stack*

Example stack s1 30 40 3 1 fred 56 black

Creates a stack with the elements 1, "fred", and 56

- **push**

name	*name of stack to push on*
value	*value to push on stack*

Example push s1 34

Pushes the value "34" on stack s1

• **pop**	name	*name of stack to pop from*
Example	pop s1	
	Pops the top item off stack s1	

D.2.5. QUEUE OBJECTS

• **queue**	name	*name of queue*
	x	*x coordinate*
	y	*y coordinate*
	length	*current length of queue contents*
	value1-valueN	*list of queue values beginning with first item*
	color	*color of stack*
Example	queue q1 30 40 3 1 fred 56 black	
	Creates a queue with the elements 1, "fred", and 56	

• **enqueue**	name	*name of queue to push on*
	value	*value to put on the queue*
Example	enqeue q1 34	
	Puts the value "34" on the end of queue q1	

• **dequeue**	name	*name of queue to dequeue*
Example	deqeue q1	
	Removes the first value of queue q1	

D.3. ACTION COMMANDS

- **delay**

length	*int indicating length of pause in milliseconds*

Example

delay 100

This will delay the animation for 1/10 a second

- **changeParam**

name	*string indicating which object will be altered*
ParamName	*name of parameter to change*
NewValue	*new value of parameter*

Example

changeParam c1 color red
This will change the color of c1's outline to red.

Example

changeParam c1 bkgrd blue
This will change the color of c1's interior to blue

Example

changeParam t1 text "HELLO"
The text object t1 will now display "HELLO"

- **moveRelative**

name	*object to move*
x	*amount to add to x position*
y	*amount to add to y position*

Example

moveRelative r1 30 40
Animates the movement of r1 to a point above and to the left of its current position.

- **groupObjects**

name	*name of aggregate of several objects*
name1-nameN	*list of objects to group together*

Example

groupObjects group1 c1 r1 b2

This will create a new object that contains the objects c1, r1, and b2 which can then be treated as one object. For example, they could be moved together.

- **delete**

name	*name of object to delete*

Example

delete g1

This will remove g1 from the animation