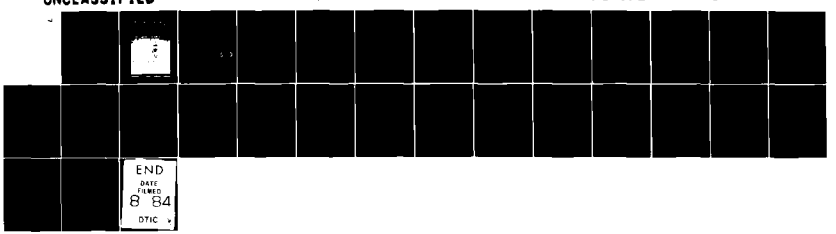
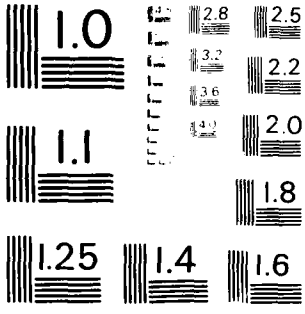
AD-A142 415 THE COMPLEXITY OF SORTING ON DISTRIBUTED SYSTEMS(U) 1/ 1
ILLINOIS UNIV AT URBANA APPLIED COMPUTATION THEORY
GROUP M C LOUI SEP 83 ACT-39 N00014-79-C-0424

UNCLASSIFIED F/G 9/2 NL



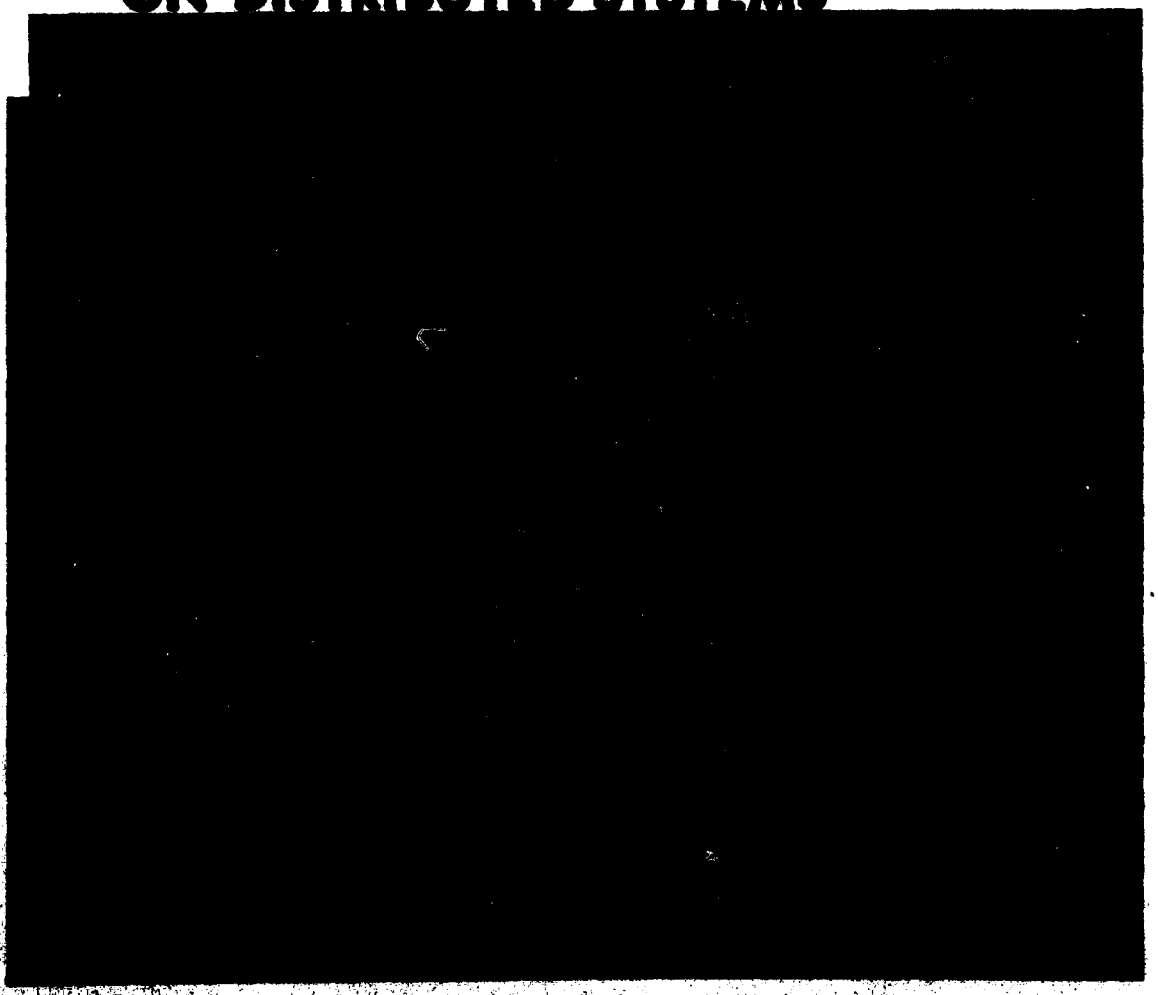
END
DATE FILMED
8 84
DTIC



ACT-39 SEPTEMBER 1983

CSL COORDINATED SCIENCE LABORATORY
APPLIED COMPUTATION THEORY GROUP

THE COMPLEXITY OF SORTING ON DISTRIBUTED SYSTEMS



UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. A142413	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE COMPLEXITY OF SORTING ON DISTRIBUTED SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER R-995,ACT-39;
7. AUTHOR(s) Michael C. Loui		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0424; MCS-8217445
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, IL 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program; National Science Foundation		12. REPORT DATE September 1983
		13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DTIC ELECTE JUN 25 1984 S D B

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

algorithm, computational complexity, distributed computation, sorting

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The sorting problem is no arrange N values in a distributed system of N processors into sorted order. Let the values be in $\{0,\ldots,L\}$. Every so ing algorithm requires $\Omega(N^2 \lg(L/N)/\lg N)$ messages on a bidirectional ring with N processors. Every sorting algorithm requires $\Omega(N^{3/2} \lg(L/N)/\lg N)$ messages on a square mesh with N processors. A novel sorting algorithm for unidirectional rings achieves the first lower bound.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

THE COMPLEXITY OF SORTING ON DISTRIBUTED SYSTEMS

Michael C. Loui*

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

September 1983

Abstract

The sorting problem is to arrange N values in a distributed system of N processors into sorted order. Let the values be in {0, ..., L}. Every sorting algorithm requires $\Omega(N^2 \lg (L/N)/\lg N)$ messages on a bidirectional ring with N processors. Every sorting algorithm requires $\Omega(N^{3/2} \lg (L/N)/\lg N)$ messages on a square mesh with N processors. A novel sorting algorithm for unidirectional rings achieves the first lower bound.

Key words and phrases: algorithm, computational complexity, distributed computation, sorting.

Running head: Sorting on Distributed Systems

*Supported by the Joint Services Electronics Program (U.S. Air Force, U.S. Army, U.S. Navy) under Contract N00014-79-C-0424 and by the National Science Foundation under Grant MCS-8217445.

1. Introduction

To cooperate to solve a problem, the processors in a distributed computing system must communicate among themselves. For both large computer networks and VLSI architectures, however, the inclusion of a shared memory to facilitate interprocessor communication is usually infeasible. The processors in these distributed systems can communicate only by sending messages via a network. Thus to exploit fully the potential efficiency of a distributed system, an efficient algorithm should minimize the message traffic in order to minimize the computation time.

The problem of finding an extremum -- also called electing a leader -- in a distributed system is well solved (Dolev et al., 1982; Matsushita, 1983; Peterson, 1982). Efficient distributed algorithms have also been proposed for determining medians (Frederickson, 1983; Matsushita, 1983; Rodeh, 1982; Santoro and Sidney, 1982), minimum spanning trees (Gallager et al., 1983), shortest paths (Chandy and Misra, 1982), and maximum flows (Segall, 1982).

It is natural to ask whether these algorithms achieve the smallest possible message traffic for each problem. Let lg denote the logarithm taken to base 2. For the extrema-finding problem Burns (1980) established a lower bound of 0.25 N lg N messages in the worst case on a bidirectional ring. Pachl et al. (1982) proved that 0.693 N lg N messages are necessary on the average on a unidirectional ring. Apparently, lower bounds for no other problems have been discovered.

In this paper I derive lower bounds on the number of messages required to arrange N values into sorted order. Let the values be in {0, ..., L}. Every sorting algorithm requires $\Omega(N^2 \frac{\lg (L/N)}{\lg N})$ messages on a bidirectional ring with N processors. Every sorting algorithm requires $\Omega(N^{3/2} \frac{\lg (L/N)}{\lg N})$ messages

on a square mesh with N processors. Evidently fewer messages are necessary if L is small than if L is large.

Furthermore, I present a simple sorting algorithm on rings that achieves the $\Omega(N^2 \frac{\lg (L/N)}{\lg N})$ lower bound. Within a constant multiplicative factor, this algorithm is optimal. The algorithm of Korach et al. (1982) uses $O(N^2)$ messages to rank the values in a network, but does not rearrange the values.

Section 2 defines the computational model and the sorting problem. Section 3 establishes the lower bounds on message complexity. Section 4 describes the optimal sorting algorithm for rings, and Section 5 presents other sorting algorithms.

2. Definitions

2.1. Computational Model

This paper adopts the model of distributed computation developed by Santoro (1981). The model is asynchronous, requires decentralized control, admits no shared memory, and permits data transfers only on a communication network.

The distributed computing system comprises N identical processors connected via a communication network. A link is an ordered pair of processors, and a network is a set of links. Processor x can send a message directly to processor y if and only if link (x, y) is in the network.

Every processor runs the same program. Initially, each processor knows only the links that involve it and the overall topology of the network -- for example, whether the network is a ring or a mesh.

Each of the processors has a distinct number representable with O(lg N) bits called its initial value. The processors exchange messages to compute a function of these values. At the end of the computation, every processor has a final value.

The transmission of a message incurs an unpredictable but finite delay, and the state of a processor changes whenever it receives a message. At processor y every message is placed on a queue when it arrives. Messages that arrive simultaneously are queued arbitrarily. Messages sent on the same link (x, y) arrive at y in the same order as they were sent.

To each processor assign an integer p, $0 \leq p < N$. For simplicity, to obviate the phrase mod N, also assign the integers $p + N$, $p + 2N$, ... to the

same processor p. The assignment of integers to processors is used only for clarity of exposition; since the processors are identical, processor p does not actually have immediate access to the number p. If integer p is assigned to processor x and q is assigned to processor y, then the link (x, y) will be written (p, q). The phrase "processor p" also denotes processor x.

I consider several topologies for the communication network. In a bidirectional ring, processor p can send messages only to processors p - 1 and p + 1. Formally, the bidirectional ring has links (p, p - 1) and (p, p + 1) for every p. In a unidirectional ring, processor p can send messages only to processor p + 1.

The discrete torus is a square mesh with wrap-around connections. Let $N = M^2$. For each processor p, $0 \leq p < N$, write $p = i + jM$ such that $0 \leq i < M$ and $0 \leq j < M$. This equation defines a bijection between $\{0, \ldots, N - 1\}$ and pairs $\langle i, j \rangle$ in $\{0, \ldots, M - 1\}^2$. Processor p can also be called processor $\langle i, j \rangle$. In the discrete torus, for every i and j there are links $(\langle i, j \rangle, \langle i + 1, j \rangle)$, $(\langle i, j \rangle, \langle i - 1, j \rangle)$, $(\langle i, j \rangle, \langle i, j + 1 \rangle)$, $(\langle i, j \rangle, \langle i, j - 1 \rangle)$, where $i \pm 1$ and $j \pm 1$ are taken modulo M. For example, ILLIAC IV had the topology of a discrete torus with N = 64.

A fully interconnected network has the link (x, y) for every pair of processors x and y.

Each processor has O(lg N) bits of storage. This limit precludes trivial algorithms. For instance, on a fully interconnected network, if processor p had unbounded storage, then the other processors could ship their initial values to processor p, which could compute all the final values.

The limitation on storage implies that every message has O(lg N) bits. There are no other constraints on the form of messages; in particular, a message need not be one of the initial values. The limit on message length prohibits arbitrarily long messages. If messages of unbounded length were permitted, then for every solvable problem there would be an algorithm that used O(N) messages after it elects a leader. For example, on a unidirectional ring N long messages would suffice to send all the initial values to the leader, which would perform the computation, and N more long messages would suffice to distribute the final values from the leader to the other processors.

To evaluate the performance of a distributed algorithm, I assume that the processing time within a processor is negligible. Indeed, because computation within a processor generally proceeds much faster than transmission of messages, communication steps often dominate the running time of an algorithm (Lint and Agerwala, 1981). The two performance criteria used in this paper are expressed as functions of N. The message complexity of an algorithm is the maximum, over all problem instances, of the total number of messages passed among all the processors on that problem instance. This complexity measure provides a worst-case estimate of the communication time. Abelson (1980) and Papadimitiou and Sipser (1982) studied a similar measure for the number of transmitted bits. The ideal execution time is the maximum, over all problem instances, of the amount of time the computation would take on that problem instance if the processors were synchronized and if every message arrived one time unit after it was sent. This measure provides a lower bound on the communication time. In the terminology of Nassimi and Sahni (1980), the ideal execution time is the number of unit-routes.

2.2. The Sorting Problem

Initially, for every p, processor p has a distinct initial value IV(p). A sorting algorithm rearranges these values so that at the end of the computation, processor p has a final value FV(p) such that for some b,

$$FV(b + i) < FV(b + i + 1) \text{ for all } 0 \leq i < N-2.$$

Call processor b the base. The base processor has the smallest final value.

For a sorting algorithm A and a distribution of initial values, the destination of a value v is the processor p such that at the end of the computation of A, the final value at processor p is FV(p) = v. The destination of a value depends on which processor becomes the base.

3. Lower Bounds

3.1. Preliminaries

Let SBS(Q) denote the set of finite sequences of binary strings $(\beta_1, \ldots, \beta_k)$ in which every component β_i is a string of at most lg Q bits.

Lemma 1. There are fewer than $2(2Q)^k$ sequences in SBS(Q) that have at most k components.

Proof. Since each component in a sequence in SBS(Q) has at most lg Q bits, the number of possible components is

$$2 + \ldots + 2^{\lg Q - 1} + 2^{\lg Q} < 2Q.$$

It follows that the number of sequences in SBS(Q) with at most k components is smaller than

$$1 + 2Q + (2Q)^2 + \ldots + (2Q)^k \leq 2(2Q)^k . \quad \square$$

Lemma 2. Let S be a set of σ different sequences in SBS(Q). When each occurrence of a string is counted, the total number of strings among the sequences in S is at least

$$\frac{4}{5}\,\frac{\sigma \lg (\sigma/10)}{\lg (2Q)} .$$

Proof. Set

$$k = 1 + \left\lfloor \frac{\lg (\sigma/10)}{\lg (2Q)} \right\rfloor .$$

By definition,

$$\lg (\sigma/10) \geq (k - 1) \lg (2Q),$$

$$\sigma/5 \geq 2(2Q)^{k-1}.$$

Lemma 1 implies that at least 4/5 of the sequences in S have at least k components each. The total number of strings among these sequences is at least

$$\frac{4}{5}\,\sigma\,k \geq \frac{4\,\sigma\,\lg\,(\sigma/10)}{5\,\lg\,(2Q)}\ .\ \square$$

In a distributed computing system call the function $p \mapsto IV(p)$ a <u>distribution</u>. If P is a set of processors in the system, then the restriction of a distribution d to P is the <u>distribution for P induced by d.</u> Distributions d_1 and d_2 <u>agree</u> on P if their values on P are the same; equivalently, the restriction of d_1 to P is identical to the restriction of d_2 to P.

Consider a partition of the processors in a system into two sets P_1 and P_2. The <u>cut</u> C induced by this partition is the set of all links (x, y) for which either $x \in P_1$ and $y \in P_2$ or $x \in P_2$ and $y \in P_1$.

Let A be a distributed algorithm that uses messages of at most c lg N bits each. Let C be a cut. During the computation by A for a distribution d consider the sequence of messages transmitted on links in C in the order in which they were sent. To each message m of this sequence append a string of 2 lg N bits that identifies on which of the at most N^2 links in C message m was sent. Call the resulting sequence of binary strings the <u>signature</u> of A for d on C. The signature is in $SBS(N^{c+2})$.

<u>Lemma 3</u>. Let C be a cut induced by a partition of the processors into sets P_1 and P_2. Let D be a collection of distributions that agree on all processors in P_2. If algorithm A has fewer than $|D|$ different signatures on C for the distributions in D, then for two different distributions in D, algorithm A produces the same set of final values in P_2.

<u>Proof</u>. By hypothesis, there are different distributions d_1 and d_2 in D for which A has the same signature on C. For both d_1 and d_2 the computation

by A sends the same messages on the same links in C in the same order. From the viewpoint of the processors in P_2 the computation by A for d_1 is the same as its computation for d_2. Consequently, at the end of both computations the final values in P_2 are the same. □

3.2. Rings

This section establishes a lower bound on the message complexity of sorting in a bidirectional ring. The lower bound applies *a fortiori* to unidirectional rings too.

Theorem 1. On a bidirectional ring of N processors with initial values in {0, ..., L}, every sorting algorithm has message complexity $\Omega(N^2 \frac{\lg (L/N)}{\lg N})$.

In particular, if L = 2N, then $\Omega(N^2/\lg N)$ messages are necessary. If L = N lg N, then $\Omega(N^2 \lg \lg N/\lg N)$ messages are necessary. If $L = N^e$ for a constant $e > 1$, then $\Omega(N^2)$ messages are necessary.

Proof. Consider an algorithm A that arranges values into sorted order using messages of length at most c lg N bits each. The main idea is the following: for some distribution of initial values, no matter which processor becomes the base, approximately N/4 initial values must migrate at least distance N/16 to their destinations. But the destination of a value depends the processor that becomes the base, which in turn depends on the initial values. The bulk of this proof overcomes this circularity.

Define R = L/N. Without loss of generality, assume that R is an integer and that N - 1 is divisible by 16. Define a collection of R^N distributions of initial values as follows. For p = 0, ..., N - 1 the initial value at processor p satisfies

$$(1)\qquad \begin{cases} (p/2)\ R \le IV(p) < (p/2 + 1)\ R & \text{if } p \text{ is even} \\ ((N + p)/2)\ R \le IV(p) < ((N + p)/2 + 1)\ R & \text{if } p \text{ is odd} \end{cases}$$

Example. (N = 17, R = 3)

IV(0) = 0	IV(5) = 33	IV(9) = 39	IV(13) = 45
IV(1) = 27	IV(6) = 9	IV(10) = 15	IV(14) = 21
IV(2) = 3	IV(7) = 36	IV(11) = 42	IV(15) = 48
IV(3) = 30	IV(8) = 12	IV(12) = 18	IV(16) = 24
IV(4) = 6	□		

Since the ring has N processors, there are only N possible bases. Therefore there is a base b such that for at least R^N/N of the distributions defined by (1), processor b becomes the base during the computation by algorithm A. Let D be this collection of R^N/N distributions.

Put

$$(2)\qquad \begin{cases} q = 6\ (N - 1)/16 + 1 + 2b \\ r = 10\ (N - 1)/16 + 2b \\ s = 11\ (N - 1)/16 + 1 + 2b \\ t = 5\ (N - 1)/16 + 2b \end{cases}$$

Let P_1 be the set of (N - 1)/4 processors q, q + 1, ..., r. Let P_2 be the set of 5(N - 1)/8 processors s, s + 1, ..., t. See Figure 1.

For the distributions in D processsor b becomes the base. Definition (1) implies that for every p the destination of IV(p) is

processor b + p/2 if p is even,

processor b + (N + p)/2 if p is odd.

It follows that for p = q, q + 1, ..., r, the destination of IV(p) is among processors

$$b + (N + q)/2 = s,\ s + 1,\ \ldots,\ b + r/2 = t.$$

Thus each of the initial values in P_1 must travel at least distance $1 + (N - 1)/16$ to its destination in P_2.

Let P'_1 be the set of $(3N + 3)/4$ processors that are not in P_1. There are $R^{(3N+3)/4}$ distributions for P'_1 consistent with (1). Consequently there is a distribution d_0 for P'_1 such that d_0 is induced by at least

$$\frac{R^{N}/N}{R^{(3N+3)/4}} = \frac{R^{(N-1)/4}}{N}$$

of the distributions in D. Let D' be this subset of at least $R^{(N-1)/4}/N$ distributions. The distributions in D' agree on P'_1. Let $\sigma = R^{(N-1)/4}/N$.

Consider the following $1 + (N - 1)/16$ pairwise disjoint cuts, which separate P_1 from P_2:

$$\begin{aligned}
&\{(q, q-1), (q-1, q), (r, r+1), (r+1, r)\},\\
&\{(q-1, q-2), (q-2, q-1), (r+1, r+2), (r+2, r+1)\}, \ldots\\
&\{(q - \tfrac{(N-1)}{16}, t), (t, q - \tfrac{(N-1)}{16}),\\
&\qquad (r + \tfrac{(N-1)}{16}, s), (s, r + \tfrac{(N-1)}{16})\}.
\end{aligned} \tag{3}$$

For each of the $1 + (N - 1)/16$ cuts C in (3) the number of different signatures of A on C must be at least $|D'| \geq \sigma$ because otherwise, by Lemma 3, there would be two different distributions in D' that would yield the same set of final values in P_2. Let $Q = N^{c+2}$. Let M(C, d) be the number of messages used by A on links in C for the initial distribution d. By Lemma 2, for each of the $1 + (N - 1)/16$ cuts C in (3),

$$\sum_{d \in D'} M(C,d) \geq \frac{4\ \sigma\ \lg\ (\sigma/10)}{5\ \lg\ (2Q)} ,$$

hence

$$\sum_{C \text{ in } (3)} \sum_{d \in D'} M(C,d) \geq (1 + \frac{N - 1}{16})\ \frac{4\ \sigma\ \lg\ (\sigma/10)}{5\ \lg\ (2Q)} .$$

Therefore there exists a d_1 in D' such that

$$\sum_{C \text{ in } (3)} M(C,d_1) \geq \left(\sum_{d \in D'} \sum_{C \text{ in } (3)} M(C,d)\right)/\sigma$$

$$\geq \left(1 + \frac{N-1}{16}\right) \frac{4 \lg (\sigma/10)}{5 \lg (2Q)}$$

$$= \frac{(N+15)}{16} \frac{(N-1) \lg R - 4 \lg (10N)}{5 \lg (2Q)}$$

$$\geq \frac{N^2 \lg R}{80 \lg (2N^{c+2})} - \frac{(N+15) \lg (10N)}{20 \lg (2N^{c+2})}$$

$$= \Omega\left(\frac{N^2 \lg R}{\lg N}\right).$$

Ergo, the message complexity of A is $\Omega(N^2 \lg R/\lg N)$. □

This proof resembles the proofs of Thompson (1979), who established time-space tradeoffs in VLSI. As Lipton and Sedgewick (1981) have observed, Thompson's technique is analogous to a crossing sequence argument for Turing machine complexity (Hopcroft and Ullman, 1979).

3.3. The Discrete Torus

A modification of the proof of Section 3.2 yields an $\Omega\left(N^{3/2} \frac{\lg (L/N)}{\lg N}\right)$ lower bound on the message complexity of sorting on the discrete torus.

Consider a discrete torus with N processors. Let $M = N^{1/2}$. Suppose the initial value at every processor p satisfies (1) in Section 3.2. For the q, r, s, and t defined by (2), the values among processors q, q + 1, ..., r must migrate to their destinations at processors s, s + 1, ..., t. Let P_1 be the set of processors q, q + 1, ..., r. Let P_2 be the set of processors s, s + 1, ..., t. It is easy to find a set of $\lceil (N-1)/(16M) \rceil$ pairwise disjoint cuts that separate P_1 and P_2. As in Section 3.2, for every sorting algorithm, there is some distribution for which the algorithm uses at least

$$\left(\frac{N-1}{16M}\right)\frac{4\lg(\sigma/10)}{5\lg(2N^{c+2})} = \left(\frac{N-1}{16M}\right)\frac{(N-1)\lg R - 4\lg(10N)}{5\lg(2N^{c+2})}$$

$$\geq \frac{(N^2-2N)\lg R}{80M\lg(2N^{c+2})} - \frac{(N-1)\lg(10N)}{20M\lg(2N^{c+2})}$$

$$= \Omega\left(\frac{N^{3/2}\lg R}{\lg N}\right).$$

messages.

Theorem 2. On a discrete torus of N processors with initial values in $\{0, \ldots, L\}$, every sorting algorithm has message complexity $\Omega\left(N^{3/2}\frac{\lg(L/N)}{\lg N}\right)$.

4. Optimal Sorting

Section 4 and Section 5 present algorithms for the sorting problem. All algorithms first employ an extrema-finding algorithm to elect a leader, namely, the processor whose initial value is smallest. Message complexities and ideal execution times for the extrema-finding problem are as follows:

	message complexity	ideal execution time
unidirectional ring (Peterson, 1982)	1.44 N lg N + O(N)	2N - 1
discrete torus (Matsushita, 1983)	0.72 N lg N + O(N)	$6N^{1/2} - 3$
fully interconnected network (Matsushita, 1983)	4.4 N	2.88 lg N + O(1)

4.1. Representing a Sorted Subset

Let $S = \{a_1, \ldots, a_k\}$ be a nonempty subset of $\{0, \ldots, L\}$. Index the elements of S so that $a_1 < a_2 < \ldots < a_k$. Let $a_0 = 0$. The set S can be represented by the sequence $(a_1 - a_0, a_2 - a_1, \ldots, a_k - a_{k-1})$. Encode this sequence as follows. Write each $a_j - a_{j-1}$ in binary; then replace simultaneously

0 by 00 1 by 01

, by 10 (and) by 11.

Call this encoded result E(S). The length of E(S) is

$$\sum_{j=1}^{k} (2 \lg (a_j - a_{j-1}) + O(1)) \text{ bits.}$$

By Jensen's inequality,

$$\frac{1}{k} \sum_{j=1}^{k} \lg (a_j - a_{j-1}) \leq \lg \left(\frac{1}{k} \sum_{j=1}^{k} (a_j - a_{j-1})\right).$$

Thus the length of E(S) is at most

$$2 \sum_{j=1}^{k} \lg (a_j - a_{j-1}) + O(k) \leq 2k \lg \left(\frac{1}{k} \sum_{j=1}^{k} (a_j - a_{j-1})\right) + O(k)$$

$$= 2k \lg (a_k/k) + O(k) \leq 2k \lg (L/k) + O(k).$$

If every a_j were written out in binary, then S would be encoded with $k \lg L + O(k)$ bits. When k is large, E(S) has fewer bits.

This encoding permits efficient insertion of a new value into S and efficient deletion of the smallest value from S. To insert a value b such that $a_i < b < a_{i+1}$, replace the encoding of $a_{i+1} - a_i$ by the encoding of the subsequence $b - a_i$, $a_{i+1} - b$. To delete the smallest value a_1, replace the encoding of the subsequence a_1 , $a_2 - a_1$ at the beginning of E(S) by the encoding of a_2.

4.2. A Sorting Algorithm on Unidirectional Rings

Consider a unidirectional ring of N processors with initial values is in {0, ..., L}. This section presents an algorithm that sorts these values by successive insertions with $O(N^2 \lg (L/N)/\lg N)$ messages. By Theorem 1, this algorithm is optimal.

The algorithm employs the encoding E defined in Section 4.1. Let $S \subseteq$ {0, ..., L} have k values. The encoding E(S) is transmitted as a sequence of messages, each of length $c \lg N$, where c is a constant. Thus the number of messages used to transmit E(S) is

$$\left\lceil \frac{2k \lg (L/k) + O(k)}{c \lg N} \right\rceil \leq \frac{2N \lg (L/N) + O(N)}{c \lg N}$$

since $k \leq N \leq L$.

The algorithm comprises three phases.

During the first phase, the processors elect a leader. Without loss of generality, assume that processor 0 is the leader.

To initiate the second phase, the leader sends E{IV(0)} to processor 1. For p = 0, ..., N - 1, define S(p) = {IV(0), ..., IV(p)}. In general, during the second phase, processor p receives E(S(p - 1)) from processor p - 1 and sends E(S(p)) to processor p + 1. Since E(S(p - 1)) is an encoding of a sorted set, processor p need not store all of E(S(p - 1)). Rather, processor p inserts IV(p) into S(p - 1) at the appropriate point, as described at the end of Section 4.1. At the end of the second phase the leader receives E(S(N - 1)).

During the third phase, the processors successively remove the smallest value from S(N - 1). For p = 0, ..., N - 2, processor p receives an encoding E(S) from processor p - 1. It defines FV(p) to be the smallest value in S and sends E(S - {FV(p)}) to processor p - 1. Section 4.1 shows that the encoding E supports efficient deletion of the smallest value in the set. Processor N - 1 receives the largest value.

Theorem 3. On a unidirectional ring of N processors with initial values in {0, ..., L}, suppose the election problem can be solved with $\mu(N)$ messages in ideal execution time $\tau(N)$. Then the sorting problem can be solved with $O(N^2 \lg (L/N)/\lg N) + \mu(N)$ messages and ideal execution time $2N + \tau(N) - 1$.

Proof. Every processor transmits an encoding of a set during the second phase and another encoding during the third phase. Therefore the algorithm uses at most

$$2N \frac{2N \lg (L/N) + O(N)}{c \lg N} = O(N^2 \lg (L/N)/\lg N)$$

messages after it elects a leader.

During both the second and third phases, every processor p can transmit a message to processor p + 1 as soon as it receives a message from processor p - 1. The second phase runs in ideal time N. The third phase runs in ideal time N - 1. Consequently the ideal execution time is 2N - 1 after the leader has been elected. □

5. Other Sorting Algorithms

5.1. The Bidirectional Ring

Although the algorithm of Section 4.2 is optimal for a wide range of initial values, the odd-even transposition sort (Knuth, 1973) can be implemented easily on a bidirectional ring of N processors with message complexity $O(N^2)$. The implementation has three phases.

In the first phase the processors elect a leader. Without loss of generality, assume that processor 0 is the leader and that N is even. In the second phase the leader initiates a message around the ring to deliver the number N and to inform each processor whether its position p is odd or even.

In the third phase each processor executes the following program fragment. At processor p, the initial value is IV, the final value FV. The procedure SEND (+; J, V) sends the two-part message (J, V) to processor p + 1, and SEND (−; J, V) sends the message (J, V) to processor p − 1. Procedure RECEIVE (J; V) waits until a message whose first part is J has entered the message queue; the second part of this message is assigned to the variable V.

```
FV := IV;
if p is odd then
        for J := 1 to N/2 do
                begin   SEND (+; 2J - 1, FV);  RECEIVE (2J - 1, V);
                        if V < FV then FV := V;
                        SEND (-; 2J, FV);  RECEIVE (2J, V);
                        if V > FV then FV := V
                end
else if p is even and p ≠ 0 then
        for J := 1 to N/2 do
                begin   SEND (-; 2J - 1, FV);  RECEIVE (2J - 1, V);
                        if V > FV then FV := V;
                        SEND (+; 2J, FV);  RECEIVE (2J, V);
                        if V < FV then FV := V
                end
else (* Program fragment for the leader *)
        for J := 1 to N/2 do
```

```
begin   SEND (-; 2J - 1, ∞);  RECEIVE (2J - 1, V);
        SEND (+; 2J, -∞);  RECEIVE (2J, V)
end
```

Sending 2J - 1 and 2J keeps the messages properly ordered. For example, processor 3 may send a message with 2J = 10 to processor 2 before processor 2 receives a message with 2J - 1 = 9 from processor 1.

Theorem 4. On a bidirectional ring with N processors suppose the election problem can be solved with $\mu(N)$ messages in ideal execution time $\tau(N)$. Then the sorting problem can be solved with $N(N + 1) + \mu(N) - 1$ messages and ideal execution time $2N + \tau(N) - 1$.

Proof. The second phase uses N - 1 messages. In the third phase every processor sends N messages, hence this phase uses N^2 messages. Thus the algorithm uses $N^2 + N - 1$ messages after electing the leader.

The second phase runs in ideal time N - 1, and the third phase runs in ideal time N. Therefore the ideal execution time of the algorithm is $2N + \tau(N) - 1$. □

5.2. The Fully Interconnected Network

The well known merge-sort enjoys a straightforward implementation on a fully interconnected network with N processors. For convenience assume that N is a power of 2.

First, the processors elect a leader. Without loss of generality, assume that processor 0 is the leader. Let P_0 be the set of processors 0, ..., N/2 - 1, and let P_1 be the set of processors N/2, ..., N - 1. Using one message, the leader designates processor N/2 the temporary leader of P_1. Processor N/2 initiates the merge-sort recursively to sort the initial values in P_1;

simultaneously processor 0 initiates the merge-sort recursively tc sort the initial values in P_0. When each recursive invocation of this algorithm is completed, the final values in P_0 and P_1 are in ascending order. Processor N/2 sends the leader a message when P_1 is sorted.

Next, the leader controls the merging of the values in P_0 and P_1. Each processor k has a temporary value TV(k) that will become its new final value. The leader executes the following algorithm, which successively compares the final values now at processors i and j and sends the smaller to processor k.

```
i := 0; j := N/2;
DONE0 := false;  DONE1 := false;
Obtain FV(N/2) from processor N/2;
for k := 0 to N-1 do
        if FV(i) < FV(j) or DONE1 then
                begin   Send FV(i) to processor k, which sets TV(k) := FV(i);
                        i := i + 1;
                        if i < N/2 then Obtain FV(i) from processor i
                                else DONE0 := true
                end
        else if FV(i) > FV(j) or DONE0 then
                begin   Send FV(j) to processor k, which sets TV(k) := FV(j);
                        j := j + 1;
                        if j < N then Obtain FV(j) from processor j
                                else DONE1 := true
                end
```

Finally, the leader sends a message to every processor p to set FV(p) := TV(p).

Observe that the leader needs to store only one value from P_1 and only one value from P_0 other than its own. Thus the number of bits of storage required by the leader is O(lg N). Indeed, every processor needs only O(lg N) bits of storage.

Let M(N) be the number of messages used by this algorithm on a fully interconnected network with N processors, after the leader has been elected. Then

```
M(N) = 2 M(N/2)   for recursive invocations of the algorithm
     + 2          to begin and end the recursive invocations
     + (N - 1)    messages from the leader to each processor
                      p ≠ 0 to obtain FV(p)
     + (N - 1)    messages sending FV(p) to the leader
     + (N - 1)    messages from the leader to each processor
                      k ≠ 0 to set TV(k)
     + (N - 1)    messages from the leader to each processor
                      p ≠ 0 to set FV(p) := TV(p)
```

Thus

$$M(1) = 0,$$
$$M(N) = 2\,M(N/2) + 4N - 1;$$

hence

$$M(N) \leq 4\,N\,\lg N.$$

Let T(N) be the ideal execution time of the algorithm. Then

```
T(N) = T(N/2)     for recursive invocations of the algorithm
     + 2          to begin and end the recursive invocations
     + (N - 1)    for messages from the leader to each processor
                      p ≠ 0 to obtain FV(p)
     + (N - 1)    for messages sending FV(p) to the leader
     + (N - 1)    for messages from the leader to each processor
                      k ≠ 0 to set TV(k)
     + 1          for messages from the leader to each processor
                      p ≠ 0 to set FV(p) := TV(p)
```

Thus

$$T(1) = 0,$$
$$T(N) = T(N/2) + 3N;$$

hence

$$T(N) < 6\,N.$$

Theorem 5. On a fully interconnected network with N processors suppose the election problem can be solved with $\mu(N)$ messages in ideal execution time $\tau(N)$. Then the sorting problem can be solved with at most $4\,N\,\lg N + \mu(N)$ messages and ideal execution time less than $6N + \tau(N)$.

The algorithm uses many messages to initiate and end recursive invocations. These messages would be unnecessary if the system were synchronous.

The odd-even transposition sorting algorithm of Section 5.1 also runs on a fully interconnected network. It has a smaller ideal execution time, but

uses more messages.

5.3. The Discrete Torus

The algorithm of Nassimi and Sahni (1979), when implemented asynchronously, uses $O(N^{3/2})$ messages because each of the N processors sends at most $N^{1/2}$ messages. Therefore, when the initial values are in $\{0, \ldots, N^e\}$ for some constant $e > 1$, this algorithm is optimal within a constant multiplicative factor.

References

Abelson, H. (1980), Lower bounds on information transfer in distributed systems, J. Asso. Comput. Mach. 27, 384-392.

Burns, J.E. (1980), "A formal model for message passing systems," Tech. Rep. 91, Comput. Sci. Dept., Indiana Univ. at Bloomington, May 1980.

Chandy, K.M., and Misra, J. (1982), Distributed computation on graphs: Shortest path algorithms, Commun. Asso. Comput. Mach. 25, 833-837.

Dolev, D., Klawe, M., and Rodeh, M. (1982), An O(n log n) unidirectional distributed algorithm for extrema finding in circles, J. Algorithms 3, 245-260.

Frederickson, G.N. (1983), Tradeoffs for selection in distributed networks, in "Proc. 2nd ACM Symp. on Principles of Distributed Computing," Association for Computing Machinery, New York.

Gallager, R.G., Humblet, P.A., and Spira, P.M. (1983), A distributed algorithm for minimum-weight spanning trees, ACM Trans. Prog. Lang. Syst. 5, 66-77.

Hopcroft, J.E., and Ullman, J.D. (1979), "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, Mass.

Knuth, D.E. (1973), "The Art of Computer Programming," vol. 3: "Sorting and Searching," Addison-Wesley, Reading, Mass.

Korach, E., Rotem, D., and Santoro, N. (1982), Distributed algorithms for ranking the nodes of a network, in "Proc. 13th Southeast Conf. on Combinatorics, Graph Theory, and Computing," 235-246.

Lint, B., and Agerwala, T. (1981), Communication issues in the design and analysis of parallel algorithms, IEEE Trans. Softw. Eng. SE-7, 174-188.

Lipton, R.J., and Sedgewick, R. (1981), Lower bounds for VLSI, in "Proc. 13th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 300-307.

Matsushita, T.A. (1983), "Distributed algorithms for selection," Tech. Rep. T-127 (ACT-37), Coordinated Science Lab., Univ. Illinois at Urbana-Champaign, July 1983.

Nassimi, D., and Sahni, S.K. (1979), Bitonic sort on a mesh-connected parallel computer, IEEE Trans. Comput. C-28, 2-7.

Nassimi, D., and Sahni, S. (1980), An optimal routing algorithm for mesh-connected parallel computers, J. ACM 27 6-29.

Papadimitriou, C.H., and Sipser, M. (1982), Communication complexity, in "Proc. 14th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 196-200.

Peterson, G.L. (1982), An O(n log n) unidirectional algorithm for the circular extrema problem, ACM Trans. Prog. Lang. Syst. 4, 758-762.

Pachl, J., Korach, E., and Rotem, D. (1982), A technique for proving lower bounds for distributed maximum-finding algorithms, in "Proc. 14th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 378-382.

Rodeh, M. (1982), Finding the median distributively, J. Comput. Syst. Sci. 24, 162-166.

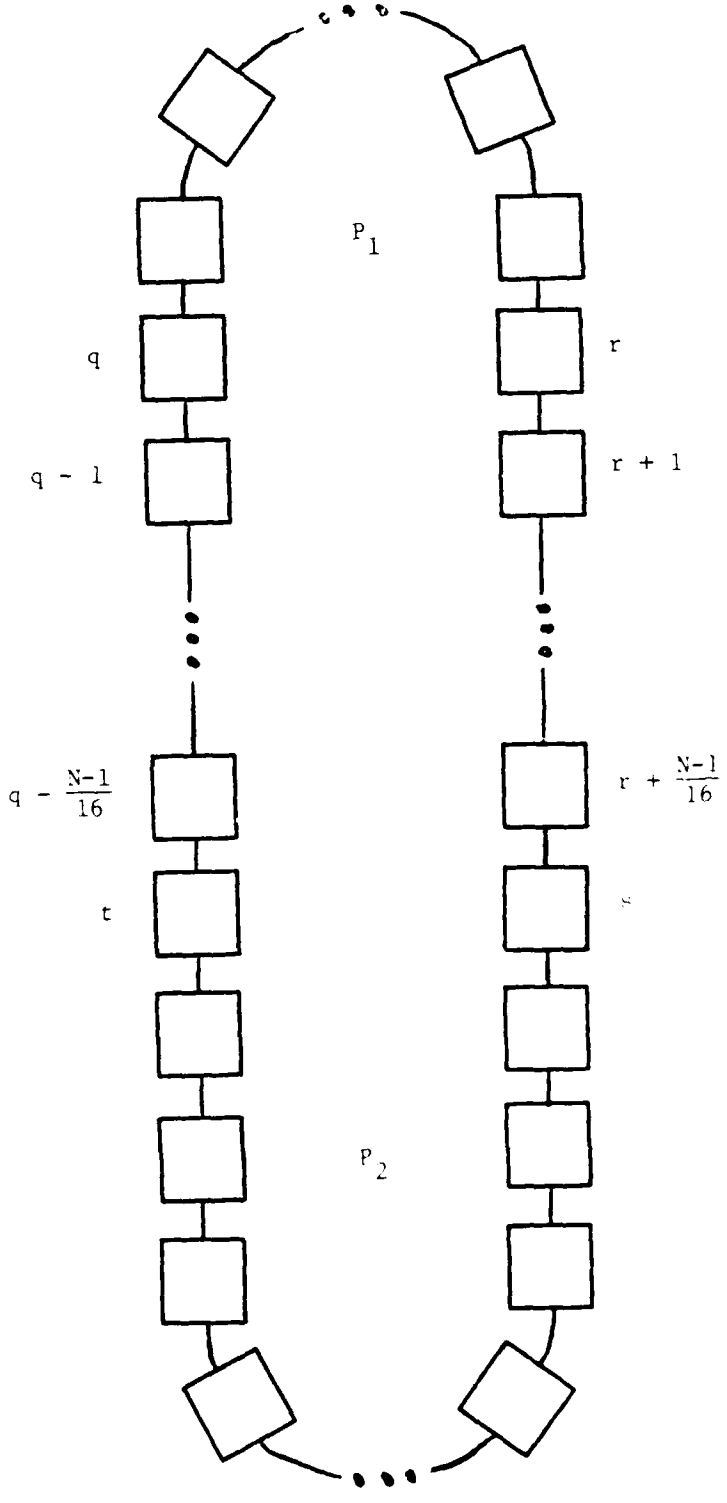
Santoro, N. (1981), Distributed algorithms for very large distributed environments: New results and research directions, in "Proc. Canad. Inform. Processing Soc.," Waterloo, 1.4.1 - 1.4.5.

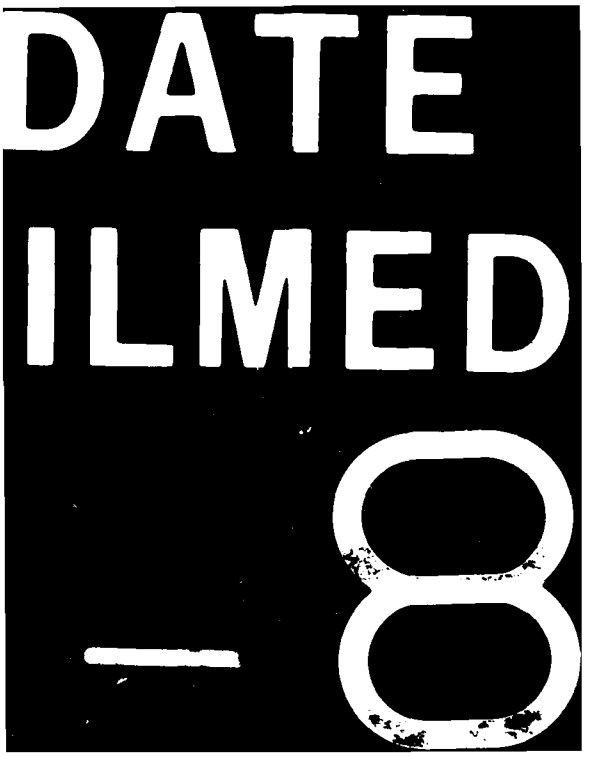
Santoro, N., and Sidney, J.B. (1982), Order statistics on distributed sets, in "Proc. 20th Ann. Allerton Conf. on Communication, Control, and Computing," Univ. Illinois at Urbana-Champaign, 251-256.

Segall, A. (1982), Decentralized maximum-flow protocols, Networks 12, 213-220.

Thompson, C.D. (1979), Area-time complexity for VLSI, in "Proc. 11th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 81-88.

Figure 1.




DATE
ILMED
8