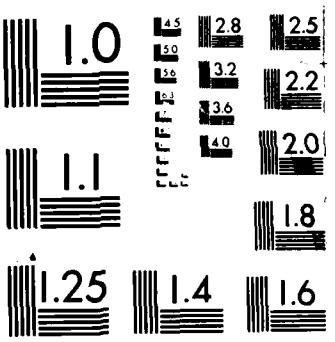
AD-A161 610 ELECTING A LEADER IN ASSYNCHRONOUS RING REVISION(U) 1/1
MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER
SCIENCE G N FREDERICKSON ET AL JUL 85
UNCLASSIFIED MIT/LCS/TM-277-REV N00014-75-C-0661 F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-277

ELECTING A LEADER IN A SYNCHRONOUS RING

Greg N. Frederickson and Nancy A. Lynch

July 1985

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MIT/LCS/TM-277	2. GOVT ACCESSION NO.: AD-A161 610
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE *(and Subtitle)*: Electing A Leader In A Synchronous Ring	5. TYPE OF REPORT & PERIOD COVERED: Update of previously published TM-277. March-July 1985
	6. PERFORMING ORG. REPORT NUMBER: MIT/LCS/TM-277
7. AUTHOR(s): Nancy A. Lynch & Greg N. Frederickson	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0661 and N00014-83-K-0125
9. PERFORMING ORGANIZATION NAME AND ADDRESS: MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: DARPA/DOD, 1400 Wilson Boulevard, Arlington, VA 22209	12. REPORT DATE: July 1985
	13. NUMBER OF PAGES: 22
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: ONR/Department of the Navy, Information Systems Program, Arlington, VA 22217	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for Public Release, Distribution is unlimited

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

Unlimited

18. SUPPLEMENTARY NOTES

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

Leader election, distributed algorithms, lower bounds, synchronous algorithms, message complexity and symmetry.

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

We consider the problem of electing a leader in a synchronous ring of n processors. We obtain both positive and negative results. On the one hand, if processor ID's are chosen from some countable set, then there is an algorithm which uses only O(n) messages in the worst case. Alternatively, if the number of rounds is required to be bounded by some t in the worst case, the ID's are chosen from any set having at least f(n,t) elements, for a certain very fast-growing function f, then any algorithm

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

requires $\Omega(n \log n)$ messages in the worst case.

ELECTING A LEADER IN A SYNCHRONOUS RING

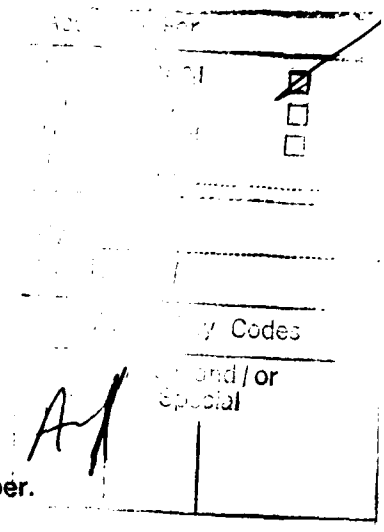
Greg N. Frederickson

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

and

Nancy A. Lynch

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139



July, 1985
Revision of MIT/LCS/TM-259 and March 1985 version of this paper.

ABSTRACT

We consider the problem of electing a leader in a synchronous ring of n processors. We obtain both positive and negative results. On the one hand, if processor ID's are chosen from some countable set, then there is an algorithm which uses only O(n) messages in the worst case. On the other hand, any algorithm that is restricted to use only comparisons of ID's requires $\Omega(n \log n)$ messages in the worst case. Alternatively, if the number of rounds is required to be bounded by some t in the worst case, and ID's are chosen from any set having at least f(n,t) elements, for a certain very fast-growing function f, then any algorithm requires $\Omega(n \log n)$ messages in the worst case.

Keywords: Leader election, distributed algorithms, lower bounds, synchronous algorithms, message complexity and symmetry.

The work of the first author was supported by the National Science Foundation under grants MCS-8201083 and DCR-8320124. The work of the second author was supported by NSF Grants MCS79-24370 and DCR-8302391, U.S. Army Research Office Contract # DAAG29-84-K-0058, Office of Naval Research Contract # N00014-85-K-0168, and Advanced Research Projects Agency of the Department of Defense Contracts # N00014-75-C-0661 and # N00014-83-K-0125.

1. Introduction

Communication in a network can be performed in either a synchronous or an asynchronous mode. How does the choice of communication mode affect the computational resources required to solve a problem? We examine this question by considering the problem of electing a leader in a ring-shaped network. In this problem there are n processors, which are identical except that each has its own unique identifier. At various points in time, one or more of the processors independently initiate their participation in an election to decide on a leader. The relevant resources for such a distributed computation are the total number of messages used and the amount of time expended from the time that the first processor wakes up.

The problem of electing a leader efficiently has been studied by a number of researchers [Bu, CR, DKR, GHS, HS, IR, L, P]. The best previous deterministic algorithms have used O(n log n) messages for either bidirectional rings [HS, GHS, Bu] or unidirectional rings [DKR, P]. These algorithms work for both the synchronous and asynchronous models, and use comparisons of ID's only. In addition, Burns has established a lower bound of $\Omega(n \log n)$ on the number of messages required if communication is asynchronous [Bu]. However, the proof in [Bu] does not extend to the case of synchronous communication. It is, therefore, quite natural to ask whether the $\Omega(n \log n)$ lower bound can be achieved in the synchronous case as well as the asynchronous, or whether there are algorithms that somehow make use of the synchrony to limit the number of messages transmitted.

We obtain both positive and negative answers to our question of whether synchrony helps. On the one hand, we show that if processor ID's are chosen from some countable set (such as the integers), then there is an algorithm which uses only O(n) messages in the worst case. The processors may initiate the algorithm at different rounds, and do not know the value of n. Our algorithm is thus an improvement on a probabilistic algorithm of [IR] that uses O(n) messages on average and assumes that the processors do know the value n. Unlike the earlier algorithms, our algorithm uses not only comparisons on ID's, but also the numerical value of the ID's to count rounds. However, the number of synchronous rounds used by our algorithm can be very large in the worst case. An algorithm similar to ours has been developed independently by Vitanyi [V].

On the other hand, we show that both the departure from the comparison model, and the possibility of using a large number of rounds, are necessary in order to obtain an algorithm of linear message complexity. More specifically, if the algorithm is restricted to use only comparisons of ID's, then we obtain an $\Omega(n \log n)$ lower

bound for the number of messages required in the worst case. To achieve this bound we generate an assignment of ID's to processors that exhibits a large amount of "replication symmetry" around the ring. We give a relatively simple assignment of values if n is a power of 2, and a somewhat more involved assignment for general values of n. More recently, a different assignment of ID's has been given in [ASW].

Alternatively, if the number of rounds is required to be bounded by some t in the worst case, then there is a (very fast-growing) function f(n, t) which has the following very interesting property. If ID's are chosen from any set T having at least f(n, t) elements, then any t-bounded algorithm requires $\Omega(n \log n)$ messages in the worst case. In particular, if t is a function of n, say t(n), then any t(n)-bounded algorithm for a set T with at least f(n,t(n)) elements exhibits the given lower bound on messages. We achieve this result by giving a transformation from any algorithm in what we call free form, over such a set T, to a comparison-based algorithm. The ideas for this transformation are derived from earlier work of Snir [S1]. Both of our lower bound results hold even in the case that the number of processors in the ring is known to each processor, and all the processors are known to start at the same round.

2. The Algorithm

In this section, we present an algorithm for electing a leader in a synchronous ring. The algorithm uses only O(n) messages, but may require a very large number of rounds. The elected processor, and only this processor, eventually enters one of a set of distinguished "elected" states. The total number of messages used, including any messages which might be sent after the winner is elected, is O(n). The algorithm presented is for a unidirectional ring, with communication assumed to be counterclockwise. Of course, essentially the same algorithm will work on a bidirectional ring. We assume that the unique ID of each processor is an integer. This assumption is reasonable if communication is implemented by transmitting packets of bits. In the description of the algorithm, we shall refer to the processor with ID i as "processor i".

The algorithm is initiated by individual processors deciding independently to wake up. The processors need not wake up at the same time, but no processor is allowed to wake after it has received a message from an awakened processor. When it wakes up, a processor (henceforth called a "participating processor") spawns a *message process*, which moves around the ring, carrying the ID of the originating processor. The message process is charged one message for each edge which it traverses.

Our algorithm uses two ideas. The first is that message processes that originate at different processors are transmitted at different rates: the message process carrying processor ID i travels at the rate of one message

transmission every 2^i rounds. (Specifically, each processor delays for $2^i - 1$ rounds before transmitting message process i.) Any slower message process that is overtaken by a faster message process is killed. Also, a message process carrying ID i arriving at processor j is killed if $j < i$ and processor j has also spawned a message process. A message process which returns to its originator causes that originator to become elected.

Suppose that all participating processors were to wake up at the same round. The above strategy would then guarantee that the total number of messages is O(n). To see this, consider the following. Let i be the smallest ID of any participating processor. Message process i traverses all edges, for a total cost of n. Consider any other message process, j. During message process i's circuit, either message process i overtakes message process j, or else message process j reaches processor i. In either case, message process j is killed by the time i's circuit is completed. Because of the different rates of travel, message process j could travel at most distance $n/(2^{j-i})$ during the time that message process i travels distance n. Summing over all message processes, the total number of messages expended would be less than 2n.

However, this variable rate of transmission scheme is by itself not enough to realize O(n) messages, in the case that not all participating processors wake up at the same time. The processors with smaller ID's could wake up correspondingly later, and spawn message processes that would chase and ultimately overtake the slower message processes, but not before $\Omega(n)$ messages had been expended by each of $\Omega(n)$ message processes.

The second idea is to have a preliminary phase for each message process, before the variable rate phase begins. In this phase, all message processes travel at the same rate, one message transmission per round. When a processor decides it wants to participate, it spawns its message process and sends it off to its neighbor. The message process is transmitted around the ring, until it encounters the next participating processor. At this point, the message process continues into the second phase, moving at its variable rate, and acting as previously described.

Lemma 1: There is an algorithm that elects a leader in a synchronous ring of n processors using fewer than 4n messages, and $O(n2^i)$ time, where i is the ID of the eventual winner.

Proof: We divide the messages into three categories, and bound each category separately. The categories are (1) the first phase messages, (2) the second phase messages sent before the eventual winner enters its second phase, and (3) the second phase messages sent after the eventual winner enters its second phase.

First consider (1). Since exactly one message from the first phase will be transmitted along each edge, the total number of first phase messages is exactly n. Next, consider (2). Every message

process that is activated will enter its second phase within n rounds of the time that the first of the processors awakens. Thus at most n rounds need to be considered. Furthermore, message process i sends no second phase messages during the rounds under consideration. Since the smallest ID that a winner can have is 0, the smallest possible ID for a processor which is not an eventual winner is 1. Thus the maximum number of second phase messages for message process j in these rounds is $n/2^j$, for $j>0$. Summing, the total number of messages sent for all the message processes in these rounds is less than n.

Finally, consider (3). The argument is similar to the one used for the case in which all processors awaken at the same round. That is, message process i makes a circuit, for a total cost of n. Any other message process j can send at most $n/2^{j-i}$ phase two messages during the time i travels distance n. As before, the total number of messages used in (3) is less than 2n. Thus the total number of messages for all categories is less than 4n.

Because of the variable transmission rate, the number of rounds required is $O(n\ 2^i)$, where i is the ID of the eventual winner. ∎

The bound of 4n messages for the algorithm above is reasonably tight. Consider the following example, where $f(n) = \log n \cdot \log \log n$. Let processor 1 be at distance 1 from processor 0, and let processor k, $k = 2,...,f(n)$, be at distance $k + \lfloor(2^{k-1} \cdot 2)n/(2^{k-1} \cdot 1)\rfloor$ from processor 0. Let processors 1 and 2 awaken at round 1, and each processor k, $k = 3,...,f(n)$, awaken the round before it would be visited by the first phase message from processor k-1. Similarly, let processor 0 awaken the round before it would be visited, which would be round $n \cdot f(n)$. Message process k, $k = 1,...,f(n)$, will start its second phase at round $2 + \lfloor(2^k \cdot 2)n/(2^k \cdot 1)\rfloor$. It will be killed when it reaches processor 0, or when it is overtaken near processor 0, and thus will traverse at least $n/(2^k \cdot 1) \cdot f(n)$ links before it is killed. There will be n first phase messages, at least $\Sigma_{k=1,...,f(n)} (n/(2^k \cdot 1) \cdot f(n))$ second phase messages for message processes $k = 1,...,f(n)$, and $n \cdot 1$ second phase messages for message process 0. Of the second phase messages for message process k, note that $[n/(2^k \cdot 1) \cdot f(n) \cdot 2]/2^k$ of them will fall under category (2), and the remainder under (3). For large n, the total is slightly more than 3.6n messages in all.

It is possible to achieve a tradeoff between the number of messages and the number of rounds by using powers of c, for any constant $c > 1$, rather than powers of 2. As before, there will be exactly n messages in category (1). In category (2), there will be fewer than $\Sigma_{j=1,...,\infty} n/c^j = n/(c \cdot 1)$ messages, while in category (3), there will be fewer than $\Sigma_{j=0,...,\infty} n/c^j = nc/(c \cdot 1)$ messages. Thus, we obtain an algorithm which elects a leader in a synchronous ring of n processors using fewer than $2cn/(c \cdot 1)$ messages, and using at most $O(n\ c^i)$ rounds, where i is the ID of the eventual winner. It is possible to retain the $2cn/(c \cdot 1)$ message bound, while reducing the time to $O(n\ c^i)$, where i is the minimum ID of all processors in the ring. The basic idea is to allow

each processor to awaken and begin its algorithm (spawning its message process) as soon as it receives any message from its neighbor, if it has not already awakened on its own. We thus obtain:

> **Theorem 2:** Let $c > 1$. There is an algorithm that elects a leader in a synchronous ring of n processors using fewer than $2cn/(c-1)$ messages, and $O(n\,c^i)$ time, where i is the smallest ID of the processors in the ring.

Note that the algorithm works correctly in the case where communication is purely asynchronous. It is only its complexity that depends on the synchrony. In the general, asynchronous, case, the algorithm is essentially the same as that of [CR], and so exhibits a worst-case message behavior which is $O(n^2)$.

3. Formal Model and Problem Statement

In this section, we describe the formal model we use for our lower bounds. The contents of this section are summarized from [FL], and the reader is referred to this paper for further details.

3.1. Algorithms

We use the following model for ring algorithms. Each processor is assumed to be identical to every other one except for its own unique identifier, chosen from an *ID space* X, a totally ordered set. The processors all begin their identical election algorithms at the same time. Each processor behaves like an automaton as follows. Initially the state of the processor consists of its ID. At each round, the processor examines its state and decides whether to send a message to each of its neighbors, and what message to send. Then each processor receives any messages sent to it in that round. The processor uses its current state and these new messages to update its state. Certain of the states are designated as "elected" states.

It may be assumed, without loss of generality, that a ring algorithm is in a certain normal form. In this normal form, the state of each processor records exactly its initial ID and the history of messages received, and each message that is sent contains the entire state of the sending processor. We represent such history information by means of LISP S-expressions. The S-expressions that arise during computation are of a special type, which we will call *well-formed*. A *well-formed* S-expression over X is either: (1) an element of X, or (2) an expression of the form (s_1,s_2,s_3), where s_2 is a well-formed S-expression over X, and each of s_1 and s_3 is either a well-formed S-expression over X or the atom NIL. Let $\mathcal{S}(X)$ denote the set of well-formed S-expressions over X.

We refer to an algorithm in such a form as a *free algorithm* , and we restrict attention in this paper to algorithms which are free. An initial state of a processor will just be its ID. Each message will contain exactly

the state of the sending processor. When a processor in state s receives messages s_1 and s_2 from its counterclockwise and clockwise neighbors respectively, its new state will be the S-expression (s_1,s,s_2). If no message is received from a neighbor, the atom NIL is used in place of s_1 or s_2 as a placeholder. To complete the algorithm specification, we define a function which determines when messages are to be sent in either direction, and a designation of which states indicate that the processor has been elected. Thus, an *algorithm* over X is a pair (E,μ), where $E \subseteq \mathcal{S}(X)$ is the set of *elected states*, and μ, a mapping from $\mathcal{S}(X)$ x {clockwise, counterclockwise} to {yes,no}, is the *message generation function*. We assume that the set E of elected states is "closed", so that once a processor has been elected, it will remain elected.

3.2. Executions

To facilitate discussion, we index the processors in the ring clockwise, as 0,...,n-1. (For convenience we are switching from the naming convention which we used in Section 2. There, by "processor i" we meant "the processor with ID i", whereas for the rest of the paper we shall mean "the processor with index i". We count indices modulo n. A *ring* of size n over ID space X is an n-tuple of elements of X, giving the initial ID's of the processors 0,...,n-1, in order. A *configuration* of size n is an n-tuple of S-expressions in $\mathcal{S}(X)$, representing the states for the n processors. A *message vector* of size n is an n-tuple of ordered pairs of elements of $\mathcal{S}(X) \cup \{null\}$. It represents the messages sent counterclockwise and clockwise by each of the n processors.

An *execution* of an algorithm for ring R of size n is an infinite sequence of triples (C_1,M,C_2), where C_1 and C_2 are configurations and M is a message vector, all of size n. We require executions to satisfy several properties. First, the initial configuration must be R. Second, the second configuration in each triple must be the same as the first configuration in the next triple. Finally, each triple in an execution must describe correct message generation, as given by μ, and correct state changes, as described earlier. An *execution fragment* is any finite prefix of an execution.

We now define our complexity measures. We measure the number of messages sent and the number of rounds taken only up to the point where a processor becomes elected. (This convention only serves to strengthen our lower bound.) For any execution e, let *finishtime(e)* denote the number of the first round after which a processor has entered a state in E. Let *messages(e)* denote the number of messages sent during e, up to and including round finishtime(e).

3.3. Election of a Leader

Let X be an ID space with $|X| \geq n$. A ring algorithm over X is said to *elect a leader in rings of size n* provided that in each execution, e, of the algorithm, for a ring R of size n over X, exactly one processor eventually enters a state in E.

3.4. Comparison Algorithms

We next define algorithms whose only operation with respect to processor ID's is to compare them. We say that two S-expressions, s and s', over X are *order-equivalent* provided that they are structurally equivalent as S-expressions, and if two atoms from s satisfy one of the order relations $<$, $=$ or $>$, then the corresponding atoms from s' satisfy the same relation. An algorithm is a *comparison algorithm* provided that if s and s' are order-equivalent well-formed S-expressions over X, then processors with states s and s' transmit messages in the same direction or directions and have the same election status. That is, μ(s,clockwise) = μ(s',clockwise), μ(s,counterclockwise) = μ(s',counterclockwise), and s is in E exactly if s' is in E.

4. Chains

In this section, we describe the general theory needed for our lower bound proof for comparison algorithms. We introduce the concept of a "chain", which describes information flow during an execution of a ring algorithm. The notion of a "chain" used in this paper is a substantial generalization of the notion of a "chain" used for a similar purpose in [FL]. For comparison algorithms, we show that nonexistence of certain chains implies that certain processors in a ring remain indistinguishable.

4.1. Basic Definitions

A k-*segment* of a ring is a length k sequence of consecutive processors in the ring, in clockwise order. Let S and T be two k-segments in a ring, with first processors p and q respectively and last processors p' and q' respectively, and let e be an execution (or execution fragment) of an algorithm in the ring. Then a *clockwise chain* in e for (S,T) is a length k subsequence of the steps of e, e_{i_1}, e_{i_2}, ..., e_{i_k}, such that the following is true. In each step e_{i_j}, a message is sent either by processor $p + j - 2$ to processor $p + j - 1$, or else by processor q

+ j · 2 to processor q + j · 1. Thus, a clockwise chain for a pair of segments describes combined information flow clockwise in the two segments, from outside the two segments up to the last processors p' and q'. A *counterclockwise chain* in e for (S,T) is defined analogously, for information flow counterclockwise: in each step e_j, a message is sent either by processor p' · j + 2 to processor p' · j + 1, or else by processor q' · j + 2 to processor q' · j + 1.

Two length k vectors of X-elements are said to be *order-equivalent* provided that the elements in corresponding positions satisfy the same ordering relations in the two vectors. That is, if the two vectors are a and b, then a_i and a_j satisfy the same relation, <, = or >, as b_i and b_j. Two segments S and T are said to be *order-equivalent* in a particular ring R provided that the sequences of initial ID's of the processors in the two segments are order-equivalent.

Let e be an execution fragment. Then *maxcw(e)* is defined to be the maximum k for which there are order-equivalent length k segments S and T (possibly with S = T), such that e contains a clockwise chain for (S,T). The quantity *maxccw(e)* is defined analogously. Let *sum(e)* = maxcw(e) + maxccw(e).

4.2. Limitations on Chains

From the definitions of maxccw, maxcw, and sum, it follows that a length 0 execution e has maxcw(e) = maxccw(e) = sum(e) = 0. We show that chains cannot grow unreasonably quickly. The length of a longest chain can grow by at most 1 in any time step, and only if a message is sent in the appropriate direction.

Lemma 3: Let e and e' be execution fragments for a ring R, such that e' consists of all but the last step of e. Then (a) maxcw(e) $\leq$ maxcw(e') + 1, with maxcw(e) = maxcw(e') if no messages are sent clockwise at the last step of e, and (b) maxccw(e) $\leq$ maxccw(e') + 1, with maxccw(e) = maxccw(e') if no messages are sent counterclockwise at the last step of e.

Proof: We argue part (a). Part (b) is analogous. The second half of the claim is obvious. We argue the inequality maxcw(e) $\leq$ maxcw(e') + 1. We may assume that maxcw(e) $\geq$ 1, since otherwise the result is obvious.

Let S and T be order-equivalent segments of length maxcw(e) for which there is a clockwise chain in e. Let S' and T' be the segments of length maxcw(e) · 1 consisting of all but the last processor in S and T respectively. Then S' and T' are order-equivalent. Moreover, since only the last message in the chain could have been sent at the last step of e, it must be that e' contains a clockwise chain for (S',T'). Thus, maxcw(e') $\geq$ maxcw(e) · 1, as required. ∎

4.3. Bisegments

We next introduce notation that allows us to describe at the same time a counterclockwise chain and a clockwise chain leading to the same processor. If k_1 and k_2 are positive integers, a (k_1,k_2)-*bisegment* is defined to be a pair of segments, the first of size k_1 and the second of size k_2, which overlap in a single processor. (The last processor of the first segment is the first of the second segment.) The processor which appears in both segments is called the *center* of the bisegment. The *spanning segment* of a bisegment is the segment obtained by concatenating the two segments in the bisegment, and removing the duplicated center. Two bisegments are said to be *order-equivalent* in a particular ring provided their spanning segments are order-equivalent. Two processors p and q are (k_1,k_2)-*equivalent* in a particular ring provided that their (k_1,k_2)-bisegments (i.e. the (k_1,k_2)-bisegments centered at p and q) are order-equivalent.

Let $S = (S_1,S_2)$ and $T = (T_1,T_2)$ be two (k_1,k_2)-bisegments, and let e be an execution or execution fragment. Then a *clockwise chain* in e for (S,T) is a clockwise chain in e for (S_1,T_1), and a *counterclockwise chain* for (S,T) is a counterclockwise chain for (S_2,T_2). A *chain* in e for (S,T) is either a clockwise chain or a counterclockwise chain for (S,T).

4.4. Indistinguishability

In this subsection, we show that, for comparison algorithms, the absence of long enough chains implies that certain processors must remain "indistinguishable". The absence of these chains then also implies that a correspondingly large number of messages will be sent in the next round.

Our notion of "indistinguishability" is defined as follows. If S and T are two ID sequences, each of length k, and s and t are two S-expressions, then s is *congruent* to t with respect to (S,T) provided that s and t are structurally equivalent, and corresponding positions in s and t contains elements from corresponding positions of S and T, respectively. If S and T are two segments of a particular ring, then s and t are *congruent* with respect to (S,T) provided that s and t are congruent with respect to the corresponding sequences of ID's. Similarly, if S and T are two bisegments of a ring, we say that s and t are *congruent* with respect to S and T provided that they are congruent with respect to their spanning segments.

> **Lemma 4:** Let e be an execution fragment of a comparison algorithm for ring R. Let k_1 and k_2 be positive integers. Let p and q be any pair of (k_1,k_2)-equivalent processors in R, and let S and T be their respective (k_1,k_2)-bisegments. If there are no chains in e for (S,T), then at the end of e, the states of p and q are congruent with respect to (S,T).
>
> **Proof:** The proof is by induction on the length of e.

Base: $|e| = 0$. Neither p nor q has received any messages in e, so they will remain in states which are congruent with respect to (S,T).

Inductive step: $|e| > 0$. Assume as the induction hypothesis that the result holds for any execution fragment of length shorter than $|e|$ and any values of k_1 and k_2. Let e' denote e except for its last step. Then by inductive hypothesis, p and q remain in states which are congruent with respect to (S,T) up to the end of e'. Consider what happens at the last step. Let p' and q' be the respective counterclockwise neighbors of p and q, and p'' and q'' the respective clockwise neighbors.

Case 1: Both of the following hold: (a) Either p' and q' are in states which are congruent with respect to (S,T) just after e', or else neither p' nor q' sends a message clockwise at the last step of e. (b) Either p'' and q'' are in states which are congruent with respect to (S,T) just after e', or else neither p'' nor q'' sends a message counterclockwise at the last step of e.

In this case, it is easy to see that p and q remain in states which are congruent with respect to (S,T), after e. For if p' and q' are in states which are congruent with respect to (S,T) just after e', then since the algorithm is a comparison algorithm, they both make the same decision about whether or not to send a message clockwise at the last step of e. If they both send a message, Then the messages they send are just their respective states, which are congruent with respect to (S,T). A similar argument applies to p'' and q''. It follows that p and q remain in states which are congruent with respect to (S,T) after the last step of e.

Case 2: Processors p' and q' are in states which are not congruent with respect to (S,T) just after e', and at least one of them sends a message clockwise at the last step of e.

If $k_1 = 1$ (i.e. if p and q are at the counterclockwise ends of their respective bisegments), then a clockwise chain for (S,T) is produced by the message sent at the last step, a contradiction. So assume that $k_1 > 1$. Since p and q are (k_1,k_2)-equivalent, it follows that p' and q' are (k_1-1,k_2+1)-equivalent. Let S' and T' denote their respective (k_1-1,k_2+1)-bisegments. S' and T' contain exactly the same processors as S and T respectively, but are centered at p' and q' rather than p and q. Since the states of p' and q' just after e' are not congruent with respect to (S,T), they are also not congruent with respect to (S',T'). By the inductive hypothesis, there must be a chain in e' for (S',T'). If there is a counterclockwise chain in e' for (S',T'), then it is also a counterclockwise chain for (S,T), so there is a counterclockwise chain in e for (S,T). On the other hand, if there is a clockwise chain in e' for (S',T'), then since at least one of p' and q' sends a message clockwise at the last step of e, we obtain a clockwise chain in e for (S,T). Either case is a contradiction.

Case 3: Processors p'' and q'' are in states which are not congruent with respect to (S,T) just after e', and at least one of them sends a message counterclockwise at the last step of e. The argument is analogous to the one for Case 2. ∎

Thus, we have shown that absence of certain chains implies that certain processors must remain in congruent states. This lemma is actually stronger than we need for this paper, but this extra strength will probably be of use in handling other problems. In our subsequent analysis, we use as an upper bound on maxcw(e) simply the number of distinct rounds in which messages are sent clockwise, and similarly for maxccw(e). Thus, instead of the existence of a chain for (S,T), we could have substituted the condition that either there are k_1 rounds in which messages are sent clockwise or there are k_2 rounds in which messages are sent counterclockwise. Reorganized in this way, our proof would be substantially the same as it is now (in fact, marginally simpler), but the revised lemma would give less information about the communication that must occur for congruence to be broken.

Two corollaries which will be used in our lower bound proofs follow from this lemma. The first one says that, when chains are short and there are lots of equivalent processors, any message which gets sent has many corresponding messages sent at the same time by other processors.

Corollary 5: Let k be a positive integer. Assume ring R is such that every k-segment has at least i order-equivalent k-segments. Let e be any execution fragment of a comparison algorithm in R, e' be another fragment consisting of all but the last step of e, and assume that sum(e') < k. If some processor p sends a message clockwise (or counterclockwise) at the last step of e, then there are at least i processors that do the same.

Proof: Consider the case where p sends a message clockwise. The other case is analogous. Let k_1 = maxcw(e') + 1 and k_2 = maxccw(e') + 1. The (k_1,k_2)-bisegment for p has at most k elements, so that p has at least i (k_1,k_2)-equivalent processors. Let q be any one of these processors, and let S and T be the (k_1,k_2)-bisegments centered at p and q, respectively. Then there cannot be a chain in e' for (S,T), by the definitions of maxcw and maxccw. But then Lemma 4 implies that p and q remain congruent with respect to (S,T) at the end of e'; since the algorithm is a comparison algorithm, q also sends a message clockwise at the last step of e. ∎

Lemma 4 also has the following consequence for comparison algorithms to elect a leader. This corollary says that long chains must be generated in order to elect a leader, if certain equivalent processors exist.

Corollary 6: Let k be a positive integer. Let R be a ring in which every k-segment S has another order-equivalent k-segment T. Let e be any execution fragment of a comparison algorithm which elects a leader in R, such that a leader gets elected in e. Then sum(e) $\geq$ k.

Proof: Assume the opposite, that sum(e) = maxcw(e) + maxccw(e) < k. Let k_1 = maxcw(e) + 1 and k_2 = maxccw(e) + 1. The (k_1,k_2)-bisegment for the processor p that gets elected leader has at most k elements, so that p has a (k_1,k_2)-equivalent processor q ≠ p; let S and T be the (k_1,k_2)-bisegments centered at p and q, respectively. Then there cannot be a chain in e for (S,T), by the definition of maxcw and maxccw. But then Lemma 4 implies that p and q remain congruent

with respect to (S,T); since the algorithm is a comparison algorithm, p and q cannot be distinguished as to leadership. This is a contradiction. ∎

5. Lower Bound for Comparison Algorithms When n is a Power of 2

In this section, we restrict attention to algorithms which use comparisons only, and to rings in which the number of processors is a power of 2. We present a lower bound of n/2 (log n + 1) for the number of messages required for a comparison algorithm to elect a leader in this case. We handle the case of powers of 2 first because the assignment of ID's to processors that realizes the lower bound is simpler than for general values of n, and also because the constant of proportionality in the lower bound is larger than we have been able to achieve for general n.

5.1. Replication Symmetry

We first generate a labelling of the processors in a ring which has a large amount of replication symmetry. Let $\langle n \rangle$ denote $\{0,...,n-1\}$. We assume that n is a power of 2, and let X^* be the ID space consisting of the set $\langle n \rangle$, with the usual ordering.

For $j \in \langle n \rangle$, let *reverse(j)* denote the integer whose binary representation is the reverse of the binary representation of j. We assign processor ID's so that processor j has ID reverse(j), for $j \in \langle n \rangle$. We call this pattern of ID's Q_n. We note that if a segment of Q_n is of length at most 2^i, then all ordering information about the ID's of processors in the segment is determined solely by the i high-order bits.

Lemma 7: Let S be any segment of Q_n of length at most 2^i, where $i < \log n$. Then there are at least $n/2^i$ segments of Q_n that are order-equivalent to S, including S itself.

Proof: For each $i < \log n$, the processor ID's repeatedly cycle through the 2^i possible arrangements of i high-order bits. Thus in a segment of length at most 2^i, each ID differs from any other in its i high-order bits. Any segment that is order-equivalent to S will have its first processor at a distance that is any integral multiple of 2^i from the first processor in S. There are $n/2^i$ such segments, including S itself. ∎

5.2. Lower Bound

We can now prove the lower bound for comparison algorithms when n is a power of 2. We make use of the following observation about comparison algorithms. Suppose X and X' are arbitrary ID spaces, and n is any integer. If $\mathcal{A}$ is a comparison algorithm over X which elects a leader in a ring of size n and uses at most s messages, then there exists a comparison algorithm $\mathcal{A}'$ over X' which elects a leader in a ring of size n and uses at most s messages. Thus a lower bound result over ID space X^* translates directly into a lower bound

result for any arbitrary ID space X.

Theorem 8: Assume n is a power of 2. Let $\mathcal{A}$ be a comparison algorithm over an arbitrary ID space, X, which elects a leader in a synchronous ring of size n. Then there is an execution, e, of $\mathcal{A}$ for which messages(e) $\geq (n/2)(\log n + 1)$.

Proof: It suffices to consider $X = X^*$. Let e be the execution fragment on Q_n, which terminates just when the elected processor enters an "elected" state. By Lemma 7, every segment of length n/2 has at least one other order-equivalent segment in Q_n. Thus by Corollary 6, execution e must progress from having a sum of 0 to having a sum of at least n/2.

Consider any step of e at which the sum first stops being at most k, for any $k < 2^i$. By Lemma 3, the sum increases by at most 2 at this step. Moreover, if no messages are sent clockwise (resp., counterclockwise) at this step, then the sum increases by at most 1.

Let e' be the prefix of e preceding this step. Then sum(e') $< 2^i$. Lemma 7 implies that any segment of length 2^i has at least $n/2^i$ order-equivalent segments in Q_n. Thus by Corollary 5, if any messages are sent clockwise at this step, then at least $n/2^i$ messages are sent clockwise, and similarly for messages sent counterclockwise. Thus, if the sum increases by 1 at this step, at least $n/2^i$ messages are sent, while if the sum increases by 2 at this step, then at least twice that number of messages are sent. It follows that the cost of increasing the sum from 0 to at least n/2 can be apportioned as a cost of at least $n/2^i$ for each increase from k to k + 1, where $k < 2^i$.

We now total up the number of messages sent in e. Grouping increases by powers of 2, we see that the number of messages sent must be at least

$$n + \Sigma_{1,\ldots,\log(n/2)} n/2^i (2^i \cdot 2^{i-1})$$

$$= n + \Sigma_{1,\ldots,\log(n/2)} n/2$$

$$= n/2 (\log n + 1). \blacksquare$$

6. Lower Bound for Comparison Algorithms for General n

In the last section we generated an assignment of ID's to processors in the case that n was a power of 2. The assignment possessed a large amount of replication symmetry, which allowed us to achieve the $\Omega(n \log n)$ lower bound. It does not appear possible to take our pattern Q_n, and then try to extend it in some way to accommodate extra processors. Such a strategy would introduce special treatment for the extra processors, which might change the behavior of the algorithm entirely, perhaps allowing some processor to become elected easily. Instead, we generate a pattern P_n for any general value of n, such that a ring assigned ID's from P_n possesses a large amount of replication symmetry. We then show that this replication symmetry causes the ring to require a large number of messages for election of a leader.

6.1. Hierarchical Organization of Processors

Fix a particular ring size $n \geq 1$. We generate a pattern P_n of ID's, the elements of which are then assigned to processors 0 through n-1, respectively. To achieve considerable replication symmetry, the construction of P_n uses a hierarchical grouping of processors. The idea is that on any level of the hierarchy, two groups of processors should receive order-equivalent sequences of ID's. To have the construction work for general n, one type of group is not enough, so that at every level there will be two types of groups. We describe the grouping using a derivation tree of a context-free grammar. Later, we will use the structure of the derivation tree to assign ID's to the n leaves of the tree and thereby produce pattern P_n.

Define the context-free grammar G as follows. The nonterminals, representing groups of processors, are A_i and B_i, $1 \leq i \leq d$, plus B_0. There is just one terminal symbol, p, representing a processor. The start symbol is B_0. The productions are:

$B_i \rightarrow B_{i+1}A_{i+1}A_{i+1}B_{i+1}B_{i+1}A_{i+1}A_{i+1}B_{i+1}B_{i+1}$, for $0 \leq i \leq d-1$,

$A_i \rightarrow A_{i+1}B_{i+1}B_{i+1}A_{i+1}A_{i+1}B_{i+1}B_{i+1}A_{i+1}A_{i+1}$, for $1 \leq i \leq d-1$,

$B_d \rightarrow p^{(b_d)}$, and $A_d \rightarrow p^{(a_d)}$.

The depth d of the hierarchy is defined as $d = \lfloor (\log_9 n)/2 \rfloor$. Note that in the last two productions, B_d generates a string consisting of b_d p symbols, and analogously for A_d. The quantities a_d and b_d will be defined later, in such a way as to guarantee that the length of the unique sentence generated by G is n.

For each i, $0 \leq i \leq d$, define the *level i sentential form* of G to be the unique string over $\{A_i,B_i\}$ derivable in G. There are exactly 9^i nonterminal symbols in the level i sentential form. Moreover, for each i, the number of symbols A_i is exactly one less that the number of symbols B_i.

> **Lemma 9:** In the level i sentential form of G, $0 \leq i \leq d$, the number of symbols A_i is $\lfloor 9^i/2 \rfloor$, and the number of symbols B_i is $\lceil 9^i/2 \rceil$.
>
> **Proof:** By induction on i. ∎

All A_i nodes derive a terminal string of the same length; we call this length a_i. Similarly, all B_i nodes derive a terminal string of the same length, which we will call b_i. Let $c_i = \min(a_i,b_i)$, for all i, $1 \leq i \leq d$.

We next describe how to select the values a_d and b_d. They are chosen in such a way that the total length of the unique sentence derived in G is exactly n, and so that $|b_d - a_d|$ is small. We use the following.

Lemma 10: Let $m, n \geq 0$ be integers. Then there are integers a and b such that $n = am + b(m + 1)$ and $|b - a| \leq m$.

Proof: Fix m. If $m = 0$, then $a = b = n$ suffices, so assume that $m \geq 1$. We proceed by induction on n.

Basis: $n = 0$. Then $a = b = 0$ suffices.

Inductive step: Assume that $n = am + b(m + 1)$ and $|b - a| \leq m$. We will produce a' and b' such that $n + 1 = a'm + b'(m + 1)$ and $|b' - a'| \leq m$. There are two cases:

Case 1. $b - a \leq m - 2$. Then let $a' = a - 1$ and $b' = b + 1$. The equation is satisfied, and $b' - a' = b - a + 2$. Then $b' - a' \geq b - a \geq -m$, and $b' - a' \leq (m - 2) + 2 = m$, as needed.

Case 2. $b - a \geq m - 1$.

Then let $a' = a + m$ and $b' = b - m + 1$. The equation is satisfied, and $b' - a' = b - a - 2m + 1$. Then $b' - a' \geq m - 1 - 2m + 1 = -m$, and $b' - a' \leq b - a$ since $m \geq 1$. Thus, $b' - a' \leq m$, as needed. ∎

Let $m = \lfloor 9^d/2 \rfloor$. It is easy to see that m is $\Theta(n^{1/2})$, and in particular, that $m \leq n^{1/2}/2$. Using Lemma 10, choose a_d and b_d to be integers such that $n = a_d m + b_d(m + 1)$ and $|b_d - a_d| \leq m$. We must show that a_d and b_d are nonnegative: if either of a_d and b_d is negative, then $\max(a_d, b_d) \leq m - 1$, so $n = a_d m + b_d(m + 1) \leq 2(m^2) \leq n/2$, a contradiction.

Lemma 11: The length of the unique sentence generated by G is n.

Proof: By Lemma 9, there are exactly $\lfloor 9^d/2 \rfloor = m$ symbols A_d, and exactly $\lceil 9^d/2 \rceil = m + 1$ symbols B_d in the level d sentential form of G. Since $n = a_d m + b_d(m + 1)$, the result holds. ∎

We have already noted that m is $\Theta(n^{1/2})$. Since a_d is nonnegative, we have that $n \geq b_d(m+1)$. Using the lower bound on m, we see that b_d is $O(n^{1/2})$.

The final lemma of this subsection gives the exact value of the difference $c_i - c_{i+1}$, which we will use in the analysis of the lower bound.

Lemma 12: The difference $c_i - c_{i+1} = 4 \cdot 9^{d-(i+1)}(n - b_d)/m$, for $0 \leq i \leq d - 1$.

Proof: Note that $c_i = \min(a_i, b_i) = \min(5a_{i+1} + 4b_{i+1}, 4a_{i+1} + 5b_{i+1}) = 4a_{i+1} + 4b_{i+1} - \min(a_{i+1}, b_{i+1}) = 4(a_{i+1} + b_{i+1}) - c_{i+1}$. Thus $c_i - c_{i+1} = 4(a_{i+1} + b_{i+1})$.

From the choice of a_d and b_d, we have $a_d + b_d = (n - b_d)/m$. It follows that $a_{i+1} + b_{i+1} = g^{d-(i+1)}(n - b_d)/m$. Substituting into the expression for $c_i - c_{i+1}$ gives the desired result. ∎

6.2. Labelling of Processors

Let X be the ID space consisting of all strings of length d + 1 whose elements are nonnegative integers, with the strings ordered lexicographically. X is the ID space from which the pattern P_n will be constructed.

We define P_n by describing an assignment of ID's to n processors, corresponding to the leaves of the derivation tree of G. In order to do this, we associate *labels* with the nodes of the derivation tree. The label of the root of the tree is the null string. If a node with a corresponding nonterminal A_i or B_i, $0 \leq i \leq d - 1$, is labelled by the string w, then the labels of its nine children are respectively w0, w1, w2, w3, w8, w7, w6, w5, w4. If a node with a corresponding nonterminal A_d is labelled by the string w, then the labels of its a_d children are respectively w0, w1,..., w(a_d - 1). If a node with a corresponding nonterminal B_d is labelled by the string w, then the labels of its b_d children are respectively w0, w1,..., w(b_d - 1). Processor ID's are generated by interpreting the labels of the leaves as elements of X, i.e. as length d + 1 strings of nonnegative integers, ordered lexicograpically.

In the level i sentential form of G, define an ordered pair of nonterminal symbols to be "of type A>A" provided that it consists of the two symbols A_iA_i, and the label of the node of the first nonterminal is lexicographically greater than that of the second. We use analogous definitions for types A<A, A>B, A<B, B>A, B<A, B>B and B<B. We now show that the level i sentential form has equal numbers of consecutive pairs of nonterminals of the eight possible types.

Lemma 13: In the level i sentential form of G, $0 \leq i \leq d$, the number of occurrences of consecutive pairs of each of the eight types A>A, A<A, A>B, A<B, B>A, B<A, B>B and B<B is exactly $\lfloor 9^i/8 \rfloor$.

Proof: It suffices to show that the numbers of occurrences of the eight types of pairs are equal, since the total number of pairs is exactly $9^i - 1 = 8 \lfloor 9^i/8 \rfloor$. We proceed by induction on i. For the basis, i = 0, the result is vacuously true. Assume that the result is true for i, and consider the level i + 1 sentential form. There are two kinds of pairs of level i + 1 nonterminals: those in which both elements are derived from the same level i nonterminal node, and those in which the two elements are derived from two different level i nonterminal nodes. Each level i nonterminal node generates a length 9 sequence of level i + 1 nonterminals, in which each of the eight types of pairs has exactly one occurrence. Therefore, there are equal numbers of the eight possible types among the pairs which are derived from the same level i nonterminal node. Also, each pair which is derived from two different level i nonterminal nodes is of the same type as the corresponding pair of parent nodes; the inductive hypothesis implies that there are equal numbers of the eight

possible types among these pairs, as well. The result follows. ∎

In any level i sentential form, note that the pair consisting of the last nonterminal node followed by the first nonterminal node, is of type B>B.

Having assigned the ID's in pattern P_n to the processors of the ring, we state a lemma which describes the replication symmetry of the ring. This lemma will be used in the next subsection, to yield our lower bound for the number of messages required by a comparison algorithm to elect a leader.

Lemma 14: Consider a ring labelled with P_n. Let $1 \leq i \leq d$. Let S be any segment of length at most $c_i + 1$. Then there are at least $\lfloor 9^i/8 \rfloor$ segments that are order-equivalent to S, including S itself.

Proof: S is contained in the subtrees of at most two nonterminal nodes at level i. These two are either two consecutive nonterminal nodes, or else the last and first nonterminals in the sentential form. Let t be the type of this ordered pair of nonterminal nodes.

By Lemma 13, there are at least $\lfloor 9^i/8 \rfloor$ instances of type t consecutive pairs of nonterminal nodes in the level i sentential form. Each of these instances of a pair of type t contains a segment which is order-equivalent to S. ∎

6.3. Lower Bound

In this section, we state and prove our lower bound for the number of messages required by a comparison algorithm to elect a leader. We use the pattern P_n constructed in the previous subsection, and the two corollaries from Section 4.

Theorem 15: Let $\mathcal{A}$ be a comparison algorithm over an arbitrary ID space, X, which elects a leader in a synchronous ring of size n. Then there is an execution, e, of $\mathcal{A}$ for which messages(e) $\geq \Omega(n \log n)$.

Proof: Assume n is fixed, and at least 9^4. This ensures that the depth $d = \lfloor (\log_9 n)/2 \rfloor$ is at least 2. It suffices to consider the ID space X consisting of length d + 1 strings of nonnegative integers, ordered lexicographically. Assume the pattern P_n is used to label the ring. Let e be the execution fragment for the ring that terminates just when the elected processor enters an "elected" state. By Lemma 14, every segment of length $c_2 + 1$ has at least one other order-equivalent segment in the ring. (The Lemma actually implies that there are at least nine others, but we do not require this fact here.) Thus, by Corollary 6, execution e must progress from having a sum of 0 to having a sum of at least $c_2 + 1$.

Consider any step of e at which the sum first stops being at most k, for any $k \leq c_1$. By Lemma 3, the sum increases by at most 2 at this step. Moreover, if no messages are sent clockwise (resp., counterclockwise) at this step, then the sum increases by at most 1.

Let e' be the prefix of e preceding this step. Then sum(e') $< c_i + 1$. Lemma 14 implies that any segment of length $c_i + 1$ has at least $\lfloor 9^i/8 \rfloor$ order-equivalent segments in the ring. Thus by Corollary 5, if any messages are sent clockwise at this step, then at least $\lfloor 9^i/8 \rfloor$ messages are sent clockwise, and similarly for messages sent counterclockwise. Thus, if the sum increases by 1 at this step, at least $\lfloor 9^i/8 \rfloor$ messages are sent, while if the sum increases by 2 at this step, then at least twice that number of messages are sent. It follows that the cost of increasing the sum from 0 to at least $c_2 + 1$ can be apportioned as a cost of at least $\lfloor 9^i/8 \rfloor$ for each increase from k to k + 1, where $k \leq c_i$.

We now total up the number of messages sent in e. Grouping increases according to level, we see that the number of messages sent must be at least

$$\Sigma_{i = 2,\ldots,d-1} \lfloor 9^i/8 \rfloor (c_i - c_{i+1}).$$

By Lemma 12, this quantity is equal to

$$\Sigma_{i = 2,\ldots,d-1} \lfloor 9^i/8 \rfloor (4 \cdot 9^{d-(i+1)} (n - b_d)/m)$$

$$= 4 ((n - b_d)/m) \Sigma_{i = 2,\ldots,d-1} \lfloor 9^i/8 \rfloor 9^{d-(i+1)}$$

$$\geq 4 ((n - b_d)/m) [\Sigma_{i = 2,\ldots,d-1} (9^i/8) 9^{d-(i+1)} - \Sigma_{i = 2,\ldots,d-1} 9^{d-(i+1)}].$$

The first summation evaluates to $(d-2)\ 9^{d-1}/8$, while the second is bounded above by $9^{d-2}/8$. Thus, the message bound is at least

$$4 ((n - b_d)/m) [(d-2) 9^{d-1}/8 - 9^{d-2}/8].$$

Since $m = \lfloor 9^d/2 \rfloor \leq 9^d/2$, this is at least

$$8 ((n - b_d)/ 9^d) [(d-2) 9^{d-1}/8 - 9^{d-2}/8]$$

$$= (n - b_d) [(d-2)/9 - 1/81] = (n - b_d) [d/9 - O(1)].$$

Since b_d is $O(n^{1/2})$, the message bound is at least

$$= (n - O(n^{1/2})) [d/9 - O(1)]$$

$$= (n - O(n^{1/2})) [((1/2)\log_9 n)/9 - O(1)]$$

$$= n ((1/2)\log_9 n)/9 - O(n)$$

$$= (n \log_2 n)/(18 \log_2 9) - O(n). \blacksquare$$

7. Lower Bound for Time-Bounded Algorithms

In this section, we prove our lower bound for time-bounded algorithms. We use the lower bound for comparison algorithms to do this. First, we show how to map from time-bounded algorithms to comparison algorithms. This result, presented in the paracomputer model, is due to Snir [S1]. (Snir [S2] credits Yao [Y] with inspiration for this result.) For completeness, we present a careful proof in our setting, even though a similar proof appears in [S1]. We then infer the lower bound for time-bounded algorithms.

7.1. Definitions

In order to map from time-bounded to comparison algorithms, we require definitions describing the behavior of an algorithm within a bounded amount of time. We say that a free algorithm is a *t-comparison algorithm* provided that both of the following conditions hold.

(1) If s and s' are order-equivalent S-expressions of parenthesis depth at most t-1, then μ'(s,clockwise) = μ'(s',clockwise) and μ'(s,counterclockwise) = μ'(s',counterclockwise).

(2) If s and s' are order-equivalent S-expressions of depth at most t, and a $\in$ A, then s is in E exactly if s' is in E.

During execution of a free algorithm, the S-expressions which appear as states at the end of any round t have depth exactly t. Thus, this definition says that the algorithm behaves as a comparison algorithm up to the end of the first t rounds. We also add the qualifier "on inputs from U" to this definition, provided that the appropriate conditions hold for those S-expressions which use atoms chosen from the set U.

7.2. Mapping a Time-Bounded Algorithm to a Comparison Algorithm

In this subsection, we show how to convert a time-bounded algorithm to a comparison algorithm. The first step is to show that any free algorithm behaves as a comparison algorithm on a subset of its inputs. For the first lemma, we use a particular fast-growing function f(n,t). The precise definition of f depends on Ramsey's Theorem, and is implicit in the proof of the lemma.

Lemma 16: Fix n, t. Let $\mathcal{A}$ be any free algorithm over ID space X, where X has at least f(n,t) elements. Then there exists a subset U of X, of size at least n, such that $\mathcal{A}$ is a t-comparison algorithm, on inputs from U.

Proof: Let Y and Z be two n-subsets of X, and let Y = $(y_1, y_2, ..., y_n)$ and Z = $(z_1, z_2, ..., z_n)$ be their representations in increasing order. Define Y and Z to be decision-equivalent if for every S-expression of depth at most t over Y, the corresponding S-expression over Z (generated by substituting z_i for y_i, i = 1,...n), gives rise to the same combination of choices: whether a message

is sent counterclockwise, whether a message is sent clockwise, and whether or not the expression is in E. Decision-equivalence partitions the n-subsets of X into finitely many equivalence classes. By Ramsey's Theorem [Be], there is a function f(n,t) such that if X is of cardinality at least f(n,t), then there is a subset C of X of cardinality 2n-1 such that all n-subsets of C belong to the same equivalence class. Then take U to be the set of the n smallest elements of C.

That U is the desired subset of X is shown as follows. Consider two m-subsets Y' and Z' of U, where m<n. The sets Y' and Z' can be extended to sets Y and Z, each of size n, by including the n-m largest elements of C. Thus an S-expression over Y' (and thus over Y) will be decision-equivalent to the corresponding S-expression over Z' (and thus over Z). ∎

The next lemma gives the mapping from free time-bounded algorithms to comparison algorithms.

Lemma 17: Fix n and t. Let $\mathcal{A}$ be a free algorithm over ID space X and alphabet A, where X has at least f(n,t) elements.

If $\mathcal{A}$ elects a leader in t rounds, using at most s messages in the worst case, then there exists a comparison algorithm $\mathcal{A}'$, which elects a leader in t rounds, using at most s messages in the worst case.

Proof: By Lemma 16, there is a subset U of X of size at least n such that $\mathcal{A}$ is a t-comparison algorithm on inputs of U. Consider any S-expression, L, of depth less than t, with atoms in X. Define the value of the message decision function of $\mathcal{A}'$ on this expression to be that of the message decision function of $\mathcal{A}$ on any S-expression, L', with atoms from U, which is order-equivalent to L. Similarly, for any S-expression of depth at most t, with atoms in X, define membership in E for $\mathcal{A}'$ according to membership in E for $\mathcal{A}$ of any order-equivalent S-expression with atoms in U. We define the message generation and decision functions so that $\mathcal{A}'$ sends no messages after round t-1 and does not change any election status after round t.

Clearly algorithm $\mathcal{A}'$ is a comparison algorithm. Since it simulates $\mathcal{A}$ on a sufficiently large subset of X, it can be seen to elect a leader in the same number of rounds, and with at most the same number of messages. ∎

This lemma appears to be of much wider applicability than just to this work and Snir's. This result, or variants, should be very useful for the study of other order-invariant problems on many different kinds of computation models. For example, see [MMS].

7.3. Lower Bound

Finally, we present our lower bound for time-bounded algorithms.

Theorem 18: Fix n and t. Let X be an arbitrary ID space with at least f(n,t) elements. Let $\mathcal{A}$ be any algorithm over X which elects a leader in a synchronous ring of size n, using at most time t. Then there is an execution, e, of $\mathcal{A}$ for which messages(e) is $\Omega(n \log n)$.

Proof: From Theorem 15, we know that there are constants a and b such that a comparison algorithm will have messages(e) $\geq$ an log n + bn for some execution e. Assume that there exists

an algorithm $\mathcal{A}$ over X which elects a leader in a synchronous ring of size n, using no more than time t, and using fewer than an log n + bn messages in the worst case. Then Lemma 17 implies that there exists a comparison algorithm which elects a leader in t rounds and uses fewer than an log n + bn messages in the worst case. This is a contradiction. ∎

8. Remaining Questions

The general $\Omega(n \log n)$ bound which we have proved has a very small constant, $1/(18 \log_2 9)$. In contrast, the best constant known for an upper bound is around 1.4 [P,DKR]. It remains to close this gap. For certain values of n, powers of 2, we do have a narrower gap. It is possible that there are certain properties of the number n, e.g. properties of its prime factorization, that affect the size of the constant. It would be interesting to understand these relationships.

Acknowledgements:

The authors thank Cynthia Dwork for pointing out the results of Snir, Maria Klawe for her encouragement in our attempts to obtain rings with replication symmetry, and Mark Tuttle for his comments on early versions of the manuscript. Thanks also go to Mike Fischer and the referees for several suggestions on improving the presentation.

References:

[A] D. Angluin, Local and Global Properties in Networks of Processors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, Los Angeles (1980) 82-93.

[ASW] C. Attiya, M. Snir, and M. Warmuth, The cost of symmetry - n log n lower bounds for synchronous rings, abstract (1985).

[Be] C. Berge, Graphs and hypergraphs, North-Holland, Amsterdam, 1973.

[Bu] J. E. Burns, A formal model for message passing systems, TR-91, Indiana University (September 1980).

[CR] E. Chang and R. Roberts, An improved algorithm for decentralized extrema-finding in circular configurations of processes, *Comm. ACM 22* (1979) 281-283.

[DKR] D. Dolev, M. Klawe and M. Rodeh, An *O (n* log *n)* unidirectional distributed algorithm for extrema finding in a circle, *J. Algorithms 3,3* (September 1982) 245-260.

[FL] G. N. Frederickson and N. A. Lynch,
The impact of synchronous communication on the problem of electing a leader in a ring,
Proceedings of the 16th Annual ACM Symposium on Theory of Computing, Washington, D.C., (April 1984), 493-503.

[GHS] R. G. Gallager, P. A. Humblet and P. M. Spira, A distributed algorithm for minimum-weight spanning trees, *ACM Trans. Prog. Lang. Sys. 5*, 1 (January 1983) 66-77.

[GLTWZ] E. Gafni, M. Loui, P. Tiwari, D. West and S. Zaks
Lower bounds on common knowledge in distributed algorithms, abstract (1984).

[HS] D. S. Hirschberg and J. B. Sinclair, Decentralized extrema-finding in circular configurations of processes, *Comm. ACM 23* (November 1980) 627-628.

[IR] A. Itai and M. Rodeh,
Symmetry breaking in distributive networks,
Proceedings of 22nd Symposium on Foundations of Computer Science, Nashville, Tennessee (October 1981), 150-158.

[L] G. LeLann, Distributed systems - toward a formal approach,
Information Processing 77, North Holland, Amsterdam (1977) 155-160.

[P] G. L. Peterson, An *O (n* log *n)* unidirectional algorithm for the circular extrema problem, *Trans. Prog. Lang. Sys. 4*, 4 (1982) 758-762.

[S1] M. Snir, On parallel searching, Hebrew University of Jerusalem, Department of Computer Science, RR 83-21 (June 1983).

[S2] M. Snir, Personal communication (1983).

[V] P. Vitanyi, Distributed elections in an Archimedean Ring of Processors,
Proceedings of the 16th Annual ACM Symposium on Theory of Computing, Washington, D.C., (April 1984), 542-547.

[Y] A. Yao, Should tables be sorted? *J. ACM 28*, 3 (July 1981) 615-628.

OFFICIAL DISTRIBUTION LIST

1985

Director
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

2 Copies

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. R. Grafton, Code 433

2 Copies

Director, Code 2627
Naval Research Laboratory
Washington, DC 20375

6 Copies

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314

12 Copies

National Science Foundation
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

2 Copies

Dr. E.B. Royce, Code 38
Head, Research Department
Naval Weapons Center
China Lake, CA 93555

1 Copy

Dr. G. Hopper, USNR
NAVDAC-OOH
Department of the Navy
Washington, DC 20374

1 Copy

END

FILMED

1-86

DTIC