MIT/LCS/TM-261

OPTIMAL DISTRIBUTED ALGORITHMS FOR SORTING AND RANKING

Shmuel Zaks

May 1984

OPTIMAL DISTRIBUTED ALGORITHMS FOR SORTING AND RANKING

Shmuel Zaks[1]

Laboratory for Computer Science
M.I.T.
Cambridge, MA 02139

April 1984

[1]On leave from the Department of Computer Science, Technion, Haifa, Israel; supported by NSF grant MCS-8302391.

ABSTRACT

We study the problems of sorting and ranking n processors that have initial values - not necessarily distinct - in a distributed system. Sorting means that the initial values have to move around in the network and be assigned to the processors according to their distinct identities, while ranking means that the numbers $1,2,...,n$ have to be assigned to the processors according to their initial values; ties between initial values can be broken in any chosen way. Assuming a tree network, and assuming that a message can contain an initial value, an identity or a rank, we present an algorithm for the ranking problem that uses, in the worst case, at most $1/2n^2 + O(n)$ such messages. The algorithm is then extended to perform sorting, using in the worst case at most $3/4n^2 + O(n)$ messages. Both algorithms are using a total of $O(n)$ space. The algorithms are extended to general networks. The expected behavior of these algorithms for three classes of trees are discussed. Assuming that the initial values, identities and ranks can only be compared within themselves, lower bounds of $1/2n^2$ and $3/4n^2$ messages are proved for a worst case execution of any algorithm to solve the ranking and sorting problems, correspondingly.

1 INTRODUCTION

A large effort is being recently put into the design and analysis of algorithms that are fully distributed. These algorithms are applicable in a network of processors, where no central controller is present and no common clock is available.

For these algorithms the model usually used contains a network of processors, each with a unique identity known in the beginning only to himself. Every processor has only a local knowledge of the network, and his only means of communication is exchanging messages with his neighbors in the network. The messages arrive in order and after a finite delay, but no a priori bound for that delay is known. We assume that a message can contain an initial value, an identity or a rank, that are of three distinct types, and can only be compared within themselves.

It is usually assumed that any non-empty subset of the processors start the algorithm (each processor has the same algorithm), and that at the end each processor has computed some function that is the result of that algorithm. There are usually many possible executions from a given starting point before all processors terminate. Assuming that the computation cost and the queueing cost in each processor is negligible compared to the communication cost, it is customary to measure the complexity of such algorithms by the total number of messages sent during any possible execution.

We study the problems of sorting and ranking n processors that have distinct identities and (not necessarily distinct) initial values. Sorting means that the initial values have to move around in the network and be assigned to the processors according to their distinct identities, while ranking means that the numbers $1,2,\ldots,n$ have to be assigned to the processors according to their initial values; ties between initial values can be broken in any chosen way.

Assuming a tree network, we first present an algorithm to solve the ranking problem, using, in the worst case, at most $1/2n^2 + O(n)$ messages, and show that any algorithm to solve the ranking problem must use, for its ranking part, in the worst case, at least $1/2n^2$ messages. The algorithm is then extended to a sorting algorithm, using in the worst case $3/4n^2 + O(n)$ messages, and a lower bound of $3/4n^2$ messages is shown; in other words, for a tree network these algorithms are best possible to within a $o(n^2)$ number of messages. Both algorithms are using a total of $O(n)$ space, by letting each node in the tree remember only an amount of information that is proportional to the number of his sons. The expected number of messages used by these algorithms is discussed for three classes of trees. For example, we show that for the class of n^{n-2} trees on n labeled nodes the expected numbers of messages used by the ranking and sorting algorithms are bounded by $2n^{3/2}(\pi/2)^{1/2} + O(n)$ and $3\ n^{3/2}(\pi/2)^{1/2} + O(n)$, correspondingly.

The extensions of these algorithms to general networks are done in the usual way, by first finding a spanning tree in the network and then applying the ranking and sorting algorithms designed for a tree network.

Section *2* presents the model and the problem, and discusses some related works. Section *3* describes the algorithms. Their proof of correctness and analysis are the subject of Sections *4* and *5*, correspondingly, and the lower bounds are proved in Section *6*.

2 THE MODEL, THE PROBLEMS AND RELATED WORKS

The model under investigation is a distributed network of n processors, with n distinct identities $id(1),id(2),\dots,id(n)$. The identities of the processors must be distinct, otherwise no deterministic distributed algorithm is possible, even when the distribution of initial values is not symmetric (see Gafni *et al* [1984]), and probability has to be introduced in order to break the symmetry (see Itai and Rodeh [1981]).

The processors are holding initial values $init(1),init(2),\dots,init(n)$, where $init(i)$ is the initial value held by processor i; these initial values are not necessarily distinct. Let $INIT = \{init(1),init(2),\dots,init(n)\}$ be the multiset of initial values, $ID = \{id(1),id(2),\dots,id(n)\}$ the set of identities, and $N = \{1,2,\dots,n\}$ the set of ranks.

Each processor is connected to some others by *communication lines*, and we assume that the *underlying graph* $G = (V,E)$ - where $V = \{1,2,\dots,n\}$ and $(i,j) \in E$ iff there is a communication line connecting processor i and processor j - is connected. A processor does not know the initial value of any of his neighbours in the network.

The communication between the processors is done by sending messages along the communication lines. A message can be any element from the set $ID \cup INIT \cup N \cup \{\lambda\}$. It is assumed that the initial values, identities and ranks are of three distinct types, and can only be compared within themselves. We use the element λ to denote an empty message. As will become clear from our algorithms, we can manage without this elememt, but we prefer to use it in order to simplify the presentation.

We assume that the messages arrive with no error after a finite but otherwise unpredicted delay, and are stored in a queue until processed.

An algorithm consists of sending and receiving messages and doing local computations. It is assumed that any non-empty set of processors may start the algorithm. At the end of the algorithm every processor i has a value $F(i)$ of the function F computed by the algorithm.

We deal with distributed algorithms that will perform *sorting* or *ranking* on the initial values. An algorithm is solving the *sorting problem* if it terminates, and the function F satisfies:

1. The set $\{F(i) \mid 1 <= i <= n\}$ is a permutation of the set *INIT*, and

2. $id(i) < id(j)$ implies $F(i) < F(j)$.

and it is solving the *ranking problem* if the function F satisfies:

1. The set $\{F(i) \mid 1 <= i <= n\}$ is a permutation of the set N, and

2. $init(i) < init(j)$ implies $F(i) < F(j)$.

This ranking algorithm improves the one presented in Korach *et al* [1982], where an algorithm for a tree network that uses, in the worst case, $3/2\, n^2 + O(n)$ messages is suggested.

The problem of distinguishing a node (finding a leader, finding a maximum) became central in the literature of distributed algorithms. It is closely related to the problem of finding a spanning tree, since given a distributed algorithm to find a leader, one can easily design an algorithm to find a spanning tree with no more than $O(|E|)$ additional messages, and given an algorithm to find a spanning tree, one can easily design an algorithm to find a leader with no more than $O(n)$ additional messages.

Gallager *et al* [1983] construct a minimum spanning tree using $O(nlogn + |E|)$ messages, and this algorithm is usually used also to find any spanning tree in a network. In Korach *et al* [1983a and 1983b] distributed algorithms for a complete network are studied, and finding a spanning tree is shown to have upper and lower bounds of $O(nlogn)$ messages, while a lower bound of $\Omega(n^2)$ messages is shown for any algorithm for finding a minimum spanning tree. A leader in a network is found in

Gallager [1982] using an expected number of $O(nlogn)$ messages.

A leader in a ring is found in $O(nlogn)$ messages for the asynchronous bidirectional case in Hirschberg and Sinclair [1980] and for the asynchronous unidirectional case in Dolev *et al* [1982] and in Peterson [1982]. Lower bounds of $\Omega(nlogn)$ messages are proved in Burns [1980] for the worst case behaviour of any asynchronous algorithm for the bidirectional ring, in Pachl *et al* [1982] for the average behaviour of any asynchronous algorithm for the bidirectional ring, and in Frederickson and Lynch [1984] for the worst case behaviour of any synchronous algorithm for the bidirectional ring.

Our technique for proving the lower bounds borrows from the lower bound proof for the *distictness problem* in Gafni *et al*[1984] (in this problem it is required that every processor i will have the final result $F(i) = 1$ if all the initial values are distinct and $F(i) = 0$ otherwise), where lower bounds for various problems are studied.

Sorting is studied for various networks in Loui [1984]; for example, for the bidirectional ring of size n and initial values in the range $\{1,...,L\}$, upper and lower bounds of $O(n^2log(L/n)/logn)$ messages are shown.

3 THE ALGORITHM

3.1 The ranking algorithm - preprocessing phase

Given any network, we first find a spanning tree with a distinguished node as its root (this requires $O(n\log n + |E|)$ messages; see Gallager *et al* [1982]), and then find a center of the tree (this requires $O(n)$ messages; see Korach *et al* [1980]). Both algorithms can be applied by using messages of the required types. Actually, as will become clear from the analysis, we had better find a median in the tree, but this will not improve the worst-case complexity, and also seems to require other types of messages as well (as done in Korach *et al* [1980]).

From now on we deal with a tree rooted at its center, which we denote by v. We also assume that each node knows the smallest value in each of the subtrees rooted at its sons. This task can be easily incorporated into the center finding algorithm; this is done by letting each node i maintain a multiset $S(i)$ containing - in the beginning - his own initial value $init(i)$; starting at the leaves and climbing up the tree towards the root, each node i waits until he receives values from all of his sons (no waiting for the leaf nodes), adds them to $S(i)$, deletes the smallest value from $S(i)$ and passes it on to his father. The process terminates when the root v receives values from all his sons and adds them to $S(v)$. It is assumed that each node i knows, for every element in $S(i)$, from which of its sons it was sent. It is easy to prove by induction that each node transfers to his father the smallest value in the subtree rooted at him.

By updating these multisets as little as possible, we manage to keep the total communication in the second phase as low as possible. An efficiet implementation of these multisets can be done using heaps; for more details see Aho *et al* [1974].

3.2 The ranking algorithm - ranking phase

The distinguished node *v* starts now the second phase of the algorithm. He deletes the smallest element from his multiset *S(v)*, sends a message containing the rank *R = 1* to the son from which he received this smallest value, and increments *R*. this value (*R = 1*) is forwarded by each node to the son from whom he received this smallest value.

When eventually the node *u* having this smallest value (this is the unique node that transfered his own value to his father!) receives this message, he assignes *F(u) = 1* and deletes from *S(u)* the smallest element , that is transfered towards the root. The element λ is sent in case the multiset *S(u)* is empty. Every node *a* on the path from *u* to *v* adds the received value to *S(a)* (nothing is done in case λ is received), deletes from it the smallest element and forwards it to his father (again, λ is sent in case the multiset *S(a)* is empty, where no such deletion is possible). When the root *v* gets this value, he adds it to *S(v)* and continues this process until *S(v)* becomes empty.

If the smallest element is *S(v)* was received from *k>1* sons, then *k* ranks can be concurrently sent to all of these sons. This does not change the total number of messages, but improves the total time in a synchronous execution of the algorithm.

3.3 Example

We first demonstrate the algorithm. In this example *n = 9* and *L = 25*, and we assume that the first phase has just been completed. The relevant information is depicted in *Figure 1*. Each node's identity *i* is written within the corresponding circle, its initial value *init(i)* is written to its left, and the multiset *S(i)* is written to its right.

Now, the root node *5* starts the second phase: the smallest element *2* is deleted from *S(5)*, the value *R = 1* is being sent to node *8* through node *2*, and is then being incremented to *2*. When node *8* receives this message, he knows it was intended to him (since he previously transfered his own value *2* to his father; he knows it because *init(8)* $\notin$ *S(8))*, so he sets *F(8) = 1*, deletes *4* from *S(8)* and sends it to node *2*; node *2* adds it to *S(2)*, deletes from it the smallest element *3* and sends it to the root, who then adds it to *S(5)*. The current configuration is depicted in *Figure 2*.

Now there are two smallest elements in *S(5)* (*init(2) = init(9) = 3*). The rank *R = 2* is sent towards node *2* and towards node *9*, and is being incremented by *2*. Node *2*, upon receiving the message, updates his final value *F(2) = 2*, deletes *4* from *S(2)* and sends it to the root, who adds it to *S(5)*, while node *9*, upon receiving the message, updates his final value *F(9) = 3* and sends λ to node *6*, who then deletes *4* from *S(6)* and sends it to the root, who adds it to *S(5)*. The current configuration is depicted in *Figure 3*.

Now there are three smallest elements in *S(5)*, so corresponding rank messages are being concurrently sent towards the three corresponding nodes *1,3* and *6*, and so on. The configuration at the end is shown in *Figure 4*.

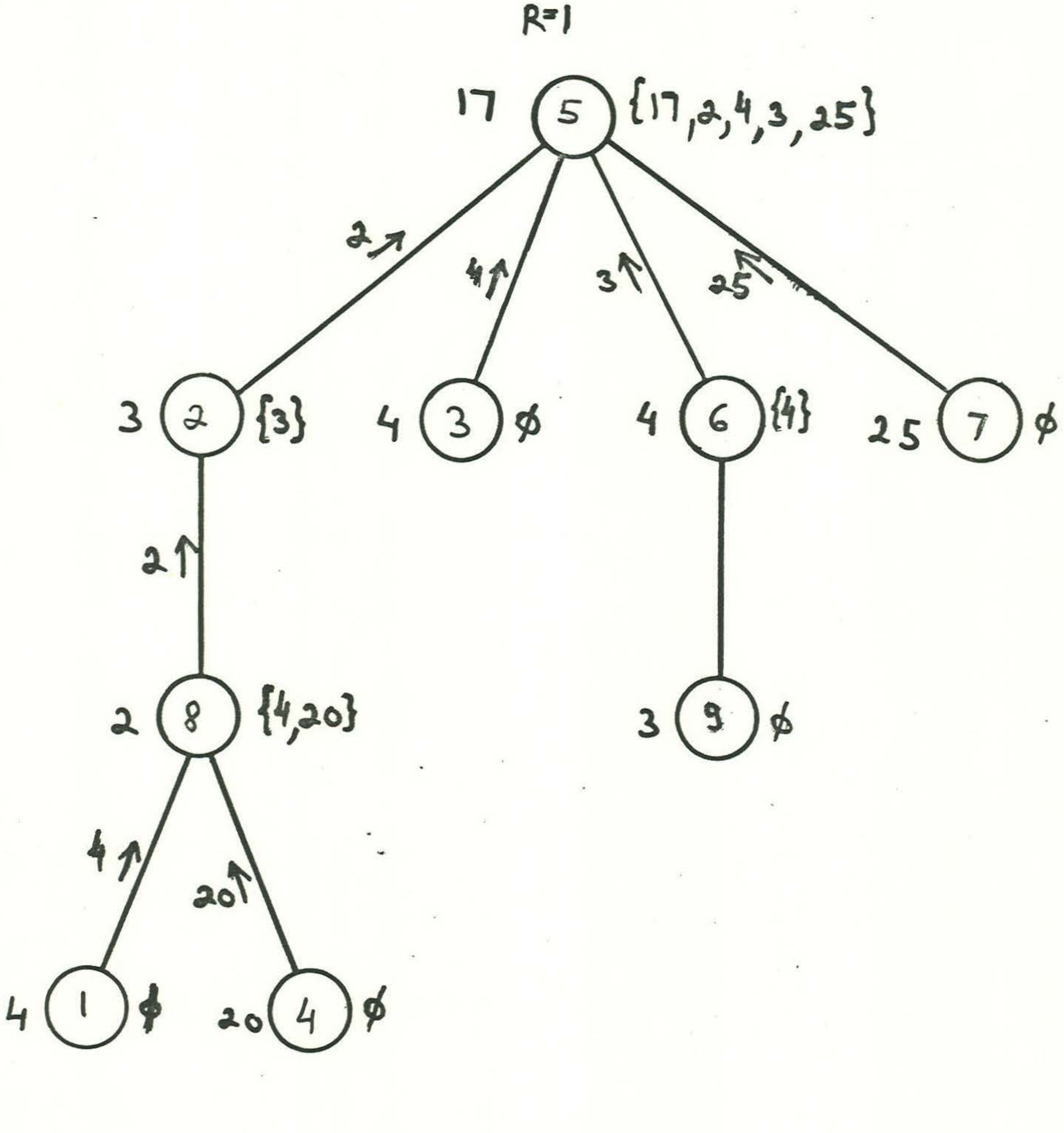

Figure 1: Configuration at the end of the first phase

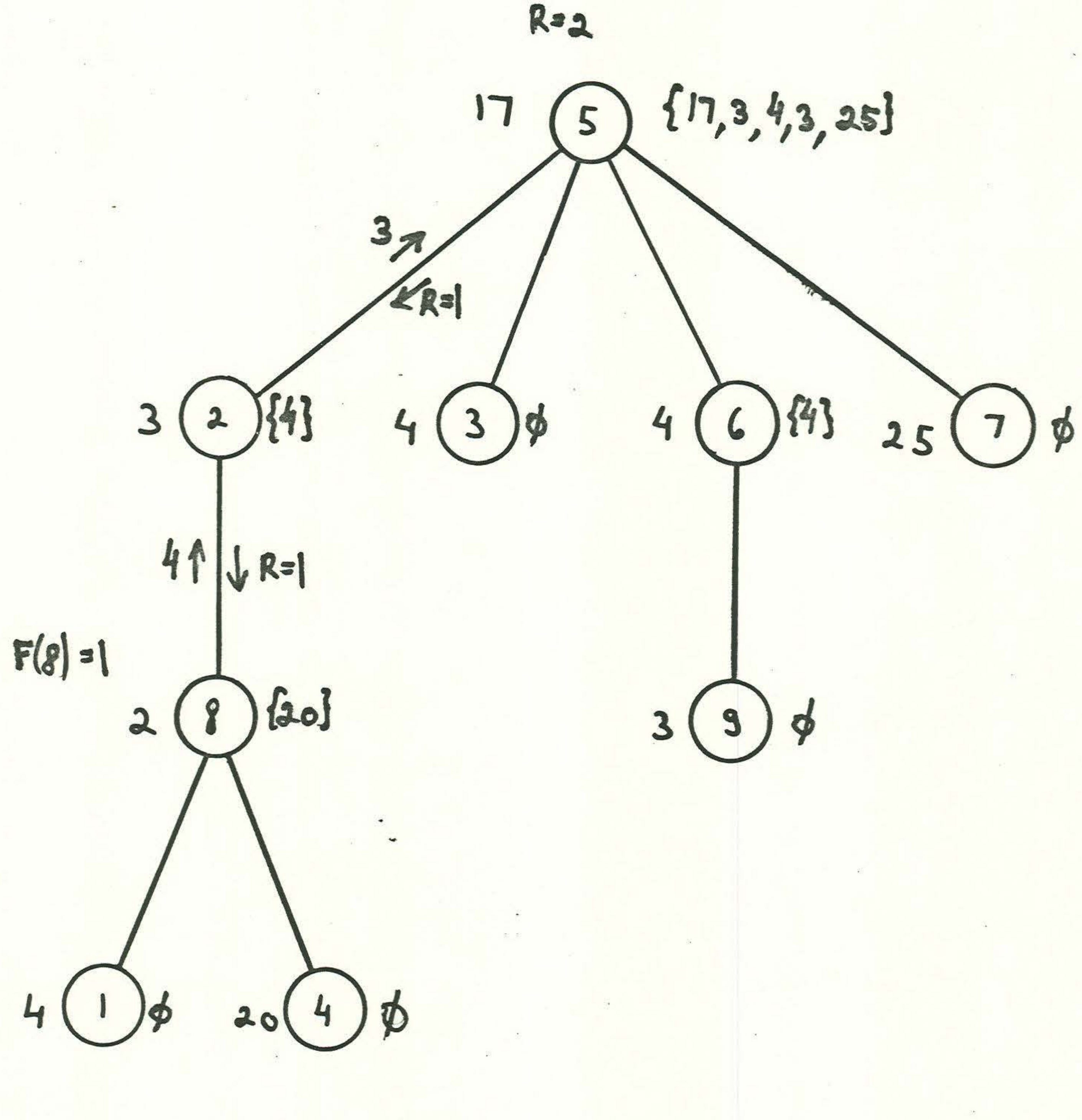

Figure 2: Configuration after *R = 1* is treated

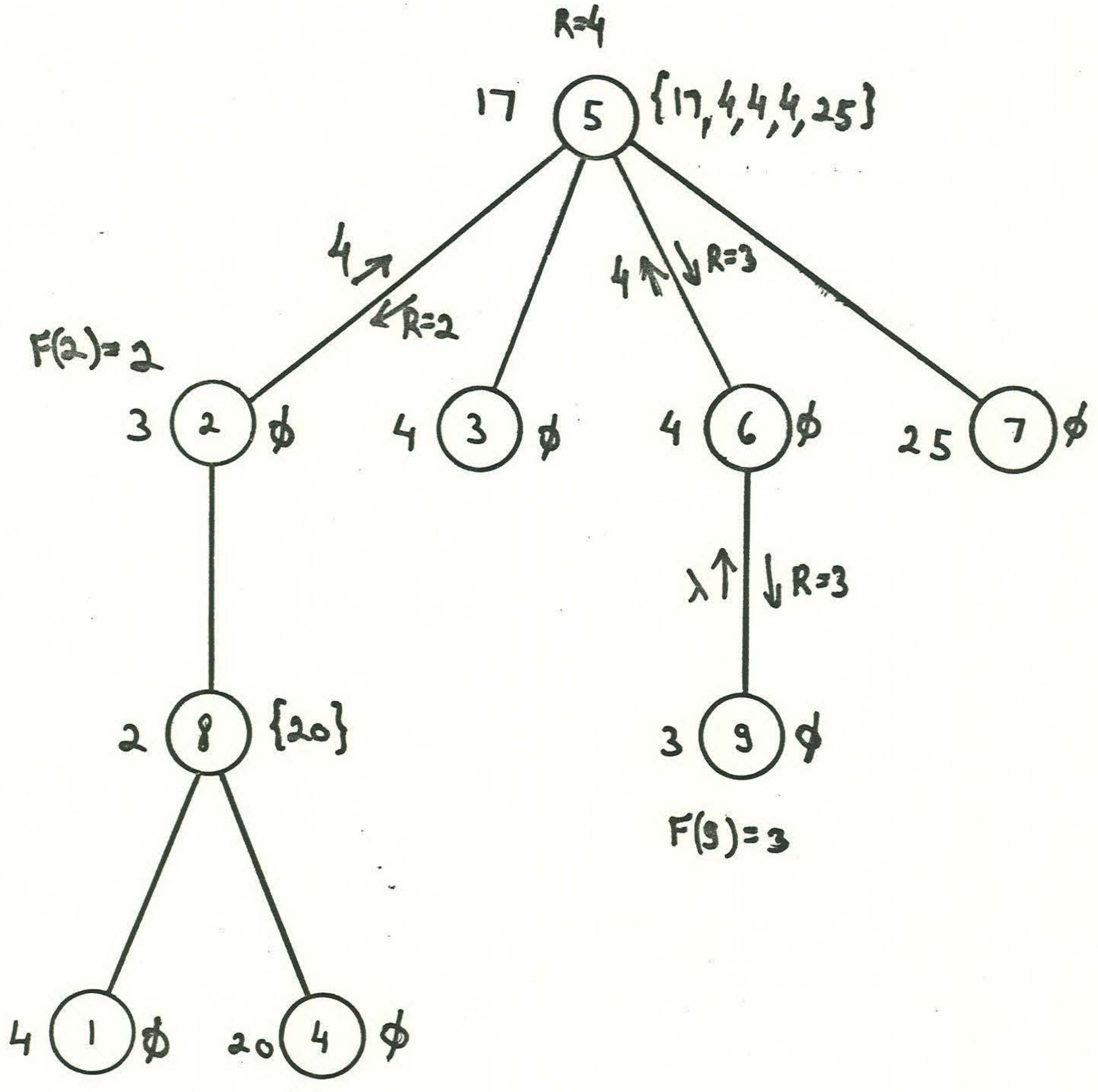

Figure 3: Configuration after R = 2,3 are treated

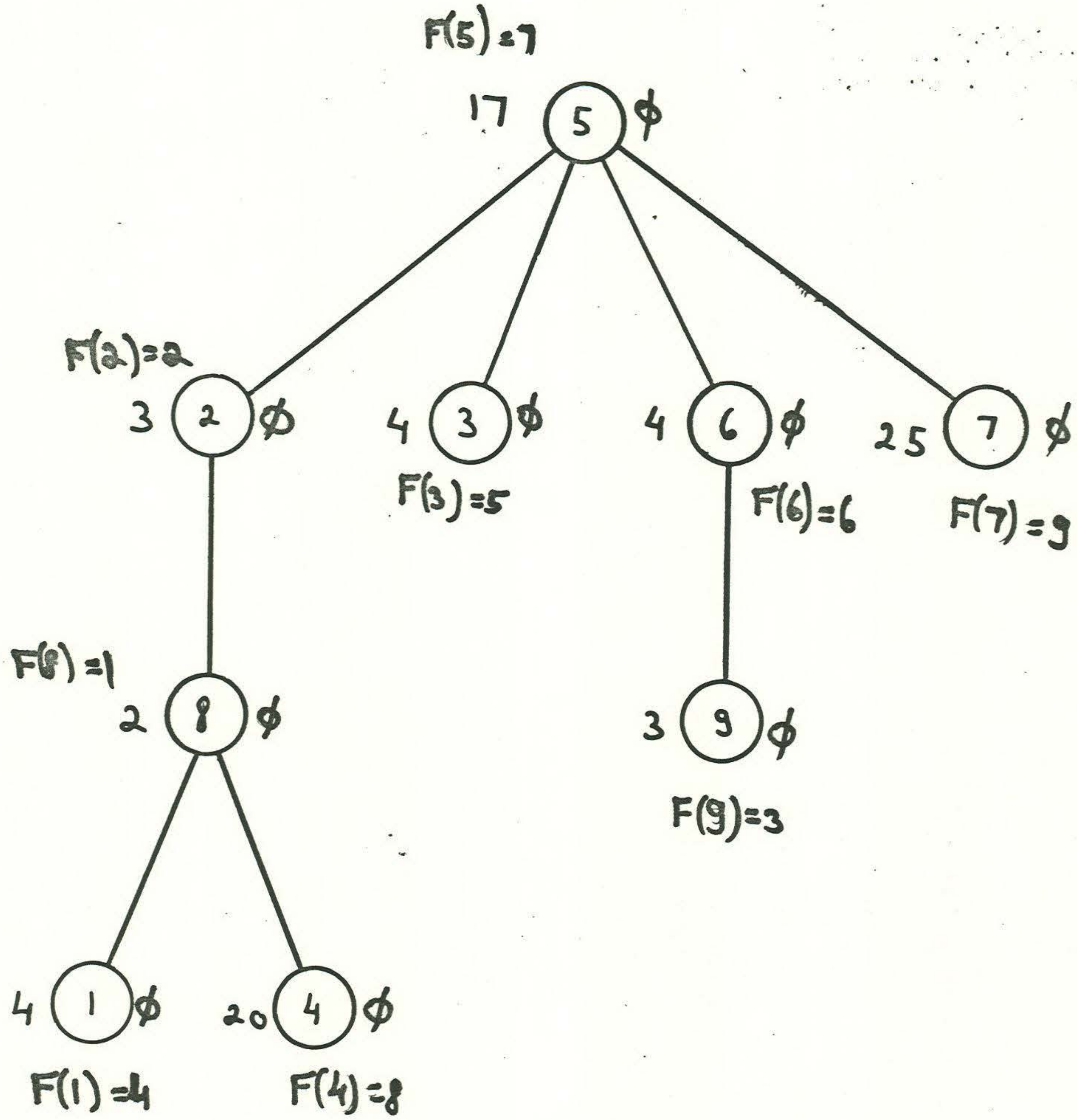

Figure 4: Configuration at the end of the second phase

3.4 The sorting algorithm

In this case each processor i maintains a multiset $S(i)$ (as before) and a subset $R(i)$ of ID. In the beginning of the process of finding a center, we let $R(i) = \{id(i)\}$; starting at the leaves and climbing up the tree towards the root, each node i waits until he receives identities from all of his sons (no waiting for the leaf nodes), adds them to $R(i)$, deletes the smallest identity from $R(i)$ and passes it on to his father. This process continues until the root receives values from all of his sons.

Therefore, at the beginning of the second phase, the root node knows the smallest initial value x and the smallest identity y, so he deletes x from $S(v)$ and y from $R(v)$, and sends x towards node y. This message will propogate down the tree, until it reaches node y (like in the ranking algorithm, node y will know the message was intended to him since $id(y) \notin R(y)$). Node y will then set $F(y) = x$, delete the smallest elements from $S(y)$ and $R(y)$ and send them back to his father. The rest of the details for this modification of the ranking algorithm are left to the reader.

3.5 A code for the ranking algorithm

delete(a,A) deletes the element **a** from the multiset **A**. **add(a,A)** adds the element **a** to the multiset **A**. **send(m,u)** sends a message **m** to node **u**. **receive(m,u)** receives a message **m** from node **u**. **min(A)** is the smallest element in the set **A**. **f(a)** is the father of node **a**. **g(u)** denotes the last value received from the (son) node **u**. **best(u)** is the son of node **u** from which he received the smallest value that has not yet been ranked (this is the latest element that was forwarded by **u** to **f(u)**); the details of the updating of these values throughout the algorithms should be clear from the discussion, and are omitted in the code for the algorithm.

The algorithm for the root node v

```
begin
   R:=1;
   while S(v) $\neq \Phi$ do
          begin
             x:=min(S(v));
             A:={u| u is a son of v and g(u)=x};
             k:=|A|;
             for i:=1 to k do
                begin
                  delete(x,S(v));
                  let a be any element in A;
                  delete(a,A);
                  send(R,a);
                  R:=R+1
                end;
             for i:=1 to k do
                begin
                  receive(x,a); (a is a son of v)
                  if $x \neq \lambda$ then add(x,S(v))
                end
          end
end.
```

The algorithm for node i $\neq$ v

```
begin
    receive(R,f(i));
    if init(i) $\notin$ S(i)
    then
       begin
          F(i):=R;
          if S(i) = $\Phi$
          then send ($\lambda$,f(i))
          else
             begin
               x:=min(S(i));
               delete(x,S(i));
               send (x,f(i))
             end
       end
     else
       begin
          send(R,best(i));
          receive(x,best(i));
          if x $\neq$ $\lambda$ then add(x,S(i));
          if S(i) = $\Phi$
          then send($\lambda$,f(i))
          else
             begin
                x:=min(S(i));
                delete(x,S(i));
                send(x,f(i))
             end
       end
end.
```

3.6 Comparison with previous work

Our algorithms differs from the one in Korach *et al* [1982] mainly in its second phase. The distinguished node found in their algorithm - not necessarily a center in the tree - starts the second phase by sending the rank *R = 1* towards the node with the smallest initial value; this node receives the message, and then initiates a sequence of messages towards the node with the second smallest value, who then initiates a sequence of messages towards the node with the third smallest value, and so on. Each node has to maintain and update a certain information, and to achieve a proper behaviour of the algorithm they send in each message two initial values and one rank. The total number of messages is, in the worst case, $n^2/2$ for the ranking phase, each carrying three entities, which amounts to a total of $3/2n^2$ messages. Moreover, their algorithm is sequential in nature, even in the case where repetitions are allowed in the set of initial values,whereas in our algorithm some messages are sent concurrently when repetitions are allowed.

4 PROOF OF CORRECTNESS

The correctness of the algorithms follows from the following discussion. The first lemma is straightforward, and its proof is omitted.

Lemma 1: At the beginning of the second phase of the ranking algorithm, each initial value *init(i)* resides in exactly one multiset *S(j)*, the number of elements in *S(i)* is $|s(i)|-1$ (*s(i)* denotes the set of sons of *i*), and the smallest initial value is in *S(v)*. At the beginning of the second phase of the sorting algorithm, each initial value *init(i)* resides in exactly one multiset *S(j)* and each identity *id(i)* resides in exactly one set *R(j)*, the number of elements in *S(i)* and in *R(i)* is $|s(i)|-1$, the smallest initial value is in *S(v)* and the smallest identity is in *R(v)*.

Lemma 2: In the second phase of the ranking algorithm, a node other than the root sends the element λ to his father only if all the nodes in his subtree have already been ranked.

(A similar result holds for the sorting algorithm)

Proof: Suppose node *a* sends the element λ to his father. We prove by induction on the largest distance *t* of this node from any node in the subtree rooted at *a* that all the nodes in that subtree have already been ranked. We denote this subtree by *t(a)*.

For $t = 0$: in this case the node *a* is a leaf. In the beginning of the ranking phase $S(a) = \Phi$ (by Lemma *1*). *a* wants to send a value to his father because he received a rank from him (see the algorithm for node $i = v$). Then $init(a) \notin S(a)$, so *F(a)* is been updated, and then λ is sent back to *f(a)*. In other words, *a* is sending λ to his father after all the nodes in *t(a)* have been ranked.

Assume it holds for nodes of distance $< t$, and let *a* be a node with a maximal distance of *t* from any node in *t(a)*, that is sending a λ message to his father.

In the beginning of the ranking phase, *S(a)* contains *|s(a)|-1* elements. For each rank that was received from *f(a)* and was forwarded to one of his sons, a value was received from that son, and only if that value was λ the size of *S(a)* was decreased by one, otherwise it was not changed. This means that each node in *s(a)* has already sent a λ message to his father *a*. By the inductive hypothesis, this was done because all the nodes in *t(b)*, for each $b \in s(a)$, have already been ranked. Moreover, there was a first time when a rank was received from *f(a)* and $init(a) \notin S(a)$ (since $init(a) \in S(a)$ in the beginning and $init(a) \notin S(a)$ at the end), and in that first time *a* was ranked. Therefore, when *a* sent λ to *f(a)* all the nodes in *t(a)* have already been ranked, which completes the proof.

By a similar induction of the height of the tree, the following theorem can be proved (its analogue for the sorting algorithm is omitted):

Theorem 3: The algorithm at each node *a* terminates with $S(a) = \Phi$, with *F(a)* containing the correct rank.

5 ANALYSIS

We turn now to the analysis of the algorithms. It is based on the following lemma, the proof of which is by a straightforward induction on the number of nodes in the tree:

> **Lemma 4:** In the second phase of the ranking algorithm, each element *init(i)* is sent exactly once along the path from *i* to *v*, and each rank *i* is sent exactly once along the path from *v* to *i*. In the second phase of the sorting algorithm, each of the elements *init(i)* and *id(i)* are sent exactly once along the path from *i* to *v*, and each element *id(i)* is sent exactly once along the path from *v* to its unique destination. In addition to this, in both algorithms each node except for the root sends exactly one λ message to his father.

Let $d(i,j)$ denote the distance (= number of edges) on the (unique) path from node *i* to node *j*. Denote

$$\sigma = \sum_u d(v,u) .$$

Then - by Lemma *4* - the total number of messages sent in the second phase of the ranking and sorting algorithms is bounded by $2\sigma + n-1$ and $3\sigma + n-1$, correspondingly.

But we have:

> **Lemma 5:** Let $T = (V,E)$ be a tree, with $|V| = n$, and let *v* be a center in *T*. Then
>
> $$\sigma = \sum_u d(v,u) <= n^2/4$$
>
> **Proof:** Denote by *r* the distance from the center node *v* to any node. There are two leaves *a* and *b* such that the path between them goes through *v*,
>
> $$d(v,a) = r <= n/2, \text{ and}$$
> $$d(v,b) = r \text{ or } r-1.$$
>
> Let *a'* be the father of *a*, and let *V'* be the set of nodes on the path connecting *a* and *b*. Any node *x* in *V-V'* satisfies
>
> $$d(v,x) <= r,$$
>
> so if we move it to be a son of *a'* its new distance *d'(v,x)* from the root does not decrease. in the case when $d(v,b) = r$ this yields

$\Sigma d(v,u) <= 2(1+2+\ldots+r) + (n-2r-1)r$

$= (r+1)r + (n-2r-1)r = (n-r)r <= n^2/4$.

The same result can be derived for the case when $d(v,b) = r-1$. This completes the proof.

From the above discussion we have:

Theorem 6: For a tree network, the ranking algorithm uses in its second phase at most $1/2n^2 + n-1$ messages, and the sorting algorithm uses in its second phase at most $3/4n^2 + n-1$ messages.

Because every node maitains a data structure that is proportional to the number of his sons, the following theorem holds:

Theorem 7: The total space used by each of the ranking and the sorting algorithms is of $O(n)$.

We conclude this section by studying the expected behavior of these algorithms for three classes of trees (we assume the trees in each class to be equally probable).

1. The n^{n-2} trees on n labeled nodes (see Moon [1970]). As was pointed out by Moon [1984], it follows from Riordan and Sloane [1969] (see also Meir and Moon [1970]) that the expected value of $\Sigma d(v,u)$ is bounded by $n^{3/2}(\pi/2)^{1/2}$ (we do not use here the fact that v is a center in the tree); therefore, the expected number of messages sent by the ranking and sorting algorithms are bounded by $2n^{3/2}(\pi/2)^{1/2} + O(n)$ and $3n^{3/2}(\pi/2)^{1/2} + O(n)$, correspondingly. The expected number of messages used by the ranking algorithm suggested in Korach *et al* [1982] is bounded by $3n^{3/2}(\pi/2)^{1/2} + O(n)$.

2. The $1/(n+1)\binom{2n}{n}$ ordered trees with n edges (see Knuth [1968]). The expected height of a node in these trees is approximately $1/2(\pi n)^{1/2}$ (see Dershowitz and Zaks [1981]). It follows that the expected number of messages used by the ranking and sorting algorithms are bounded by $n^{3/2}\pi^{1/2} + O(n)$ and $3/2n^{3/2}\pi^{1/2} + O(n)$, correspondingly.

3. The $1/(n+1)\binom{2n}{n}$ binary trees with n internal nodes. The expected height of a node in these trees is approximately $(\pi n)^{1/2}$ (see Knuth [1968]). It follows that the expected number of messages used by the ranking and sorting algorithms are bounded by $2n^{3/2}\pi^{1/2} + O(n)$ and $3n^{3/2}\pi^{1/2} + O(n)$, correspondingly.

The expected behavior of these algorithms for other classes of trees, and for the case when repetitions in the set of initial values are allowed, are yet to be studied.

6 LOWER BOUNDS

We present here proofs for the lower bounds.

Theorem 8: Any algorithm for the ranking problem uses, in the worst case, at least $1/2n^2$ messages.

Proof: Let $T = (V,E)$ be a tree, where $V = ID = \{v_1, v_2, \ldots, v_n\}$ and $E = \{(v_i, v_{i+1}) | 1 <= i < n\}$. We assume that n is even, and let the initial values be as follows:

$$init(v_i) = 2i-1 \quad \text{for} \quad 1 <= i <= n/2$$

$$init(v_i) = 2i-n \quad \text{for} \quad n/2 + 1 <= i <= n.$$

In order for processors v_i and $v_{i+n/2}$ to know their correct ranks at the end of the algorithm, their values must be compared at some node, which takes at least $n/2$ messages. Otherwise, suppose that the initial values of v_j and $v_{j+n/2}$, for some $1 <= j <= n/2$, are not compared; then, by running the algorithm with exchanging these two initial values (without changing any other initial value), it will terminate with the same final values for all nodes as before this exchange took place, a contradiction. Moreover, the result of that comparison has to reach both nodes in order for them to correctly determine their ranks, which amounts to additional $n/2$ messages. Here we are using the assumption that no operation other than comparisons is allowed on the set of ranks; this prevents one from coding few rank messages in one. We conjecture that the same lower bound holds even without this restriction.

This amounts to using, in the worst case, at least $n^2/2$ messages.

Theorem 9: Any algorithm for the sorting problem uses, in the worst case, at least $3/4n^2$ messages.

Proof: We take the same tree as before, and assume $v_i < v_j$ for $i < j$. Let the initial values be as follows:

$$init(v_i) = 2i \quad \text{for} \quad 1 <= i <= n/2$$

$$init(v_i) = 2i-n-1 \quad \text{for} \quad n/2 + 1 <= i <= n.$$

For each pair of nodes v_i and $v_{i+n/2}$, both their identities and initial values must be compared (otherwise an excange in either their identities or initial values couldn't be detected, like in the previous proof). After these comparisons are made, these initial values have to be switched between these two nodes, which amounts to using an additional *n/2* messages.

This amounts to using, in the worst case, at least $3/4n^2$ messages.

Acknowledgements: I would like to thank Nancy Lynch for help in clarifying the lower bounds arguments, Nissim Francez for first making me aware of the distributed sorting problem, and John Moon for references used in the average-case analysis.

REFERENCES

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman [1974], **The Design and Analysis of Computer Algorithms**, Addison-Wesley.

2. J. E. Burns [1980], A FORMAL MODEL FOR MESSAGE PASSING SYSTEMS, TR-91, Indiana University.

3. N. Dershowitz and S. Zaks [1981], APPLIED TREE ENUMERATIONS, Proceedings of the 6th CAAP conference, Genoa, Italy, pp. 180-193.

4. D. Dolev, M. Klawe and M. Rodeh [1982], AN O(NLOGN) UNIDIRECTIONAL DISTRIBUTED ALGORITHM FOR EXTREMA FINDING IN A CIRCLE, Journal of Algorithms, 3, pp. 245-260.

5. G. N. Frederickson and N. A. Lynch [1984], THE IMPACT OF SYNCHRONOUS COMMUNICATION ON THE PROBLEM OF ELECTING A LEADER IN A RING, proceedings of the 16th Annual ACM Symp. on Theory of Computing, Washington D.C., to appear.

6. E. Gafni. M. C. Loui, P. Tiwari, D. B. West and S. Zaks [1984], LOWER BOUNDS ON COMMON KNOWLEDGE IN DISTRIBUTED ALGORITHMS, WITH APPLICATIONS, in preparation.

7. R. G. Gallager [1982], CHOOSING A LEADER IN A NETWORK, unpublished memorandum, M.I.T.

8. R. G. Gallager, P. A, Humblet and P. M. Spira [1983], A DISTRIBUTED ALGORITHM FOR MINIMUM SPANNING TREE, Transactions of Programming Languages and Systems, 5, 1, pp. 66-77.

9. D. S. Hirschberg and J. B. Sinclair [1980], DECENTRALIZED EXTREMA FINDING IN CIRCULAR CONFIGURATIONS OF PROCESSES, Communications of the ACM 23, pp. 627-628.

10. A. Itai and M. Rodeh [1981], SYMMETRY BREAKING IN DISTRIBUTED NETWORKS, IEEE 22nd Symposium on Foundations of Computer Science, pp. 150-158.

11. D. E. Knuth [1968], **The Art of Computer Programming, Vol 1: Fundamental Algorithms**, Addison-Wesley, Reading, MA.

12. E. Korach, S. Moran and S. Zaks [1983], TIGHT LOWER AND UPPER BOUNDS

FOR SOME DISTRIBUTED ALGORITHMS FOR A COMPLETE NETWORK OF PROCESSORS, TR no. 124, IBM Scientific Center, Haifa, Israel.

13. E. Korach, S. Moran and S. Zaks [1983], FINDING A MINIMUM SPANNING TREE CAN BE HARDER THAN FINDING A SPANNING TREE IN A DISTRIBUTED NETWORK, TR no. 126, IBM Scientific Center, Haifa, Israel.

14. E. Korach, D. Rotem and N. Santoro [1980], DISTRIBUTED ALGORITHMS FOR FINDING CENTERS AND MEDIANS IN NETWORKS, RR CS-80-44, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

15. E. Korach, D. Rotem and N. Santoro [1982], DISTRIBUTED ALGORITHMS FOR RANKING THE NODES OF A NETWORK, Proceedings of the 13th Southeastern Conference on Combinatorics, Graph Theory and Computing, pp. 235-246.

16. M. C. Loui [1983], THE COMPLEXITY OF SORTING ON DISTRIBUTED SYSTEMS, TR R-995 (ACT-39), Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Illinois, Urbana 61801.

17. A. Meir and J. W. Moon [1970], THE DISTANCE BETWEEN POINTS IN RANDOM TREES, Journal of Combinatorial Theory 8, pp. 99-103.

18. J. W. Moon [1970], **Counting Labelled Trees**, Canadian Mathematical Monographs, No. 1.

19. J. W. Moon [1984], private communication.

20. J. Pachl, E. Korach and D. Rotem [1982], A TECHNIQUE FOR PROVING LOWER BOUNDS FOR DISTRIBUTED MAXIMUM-FINDING ALGORITHMS, proceedings of the 14th Annual ACM Symp. on Theory of Computing, San Francisco, CA, pp. 378-382.

21. G. L. Peterson [1982], AN O(NLOGN) UNIDIRECTIONAL ALGORITHM FOR THE CIRCULAR EXTREMA PROBLEM, Transactions of Programming Languages and Systems, 4, pp. 758-762.

22. J. Riordan and N. J. A. Sloane [1969], THE ENUMERATION OF ROOTED TREES BY TOTAL HEIGHT, Journal of the Australian Mathematical Society 10, pp. 278-282.