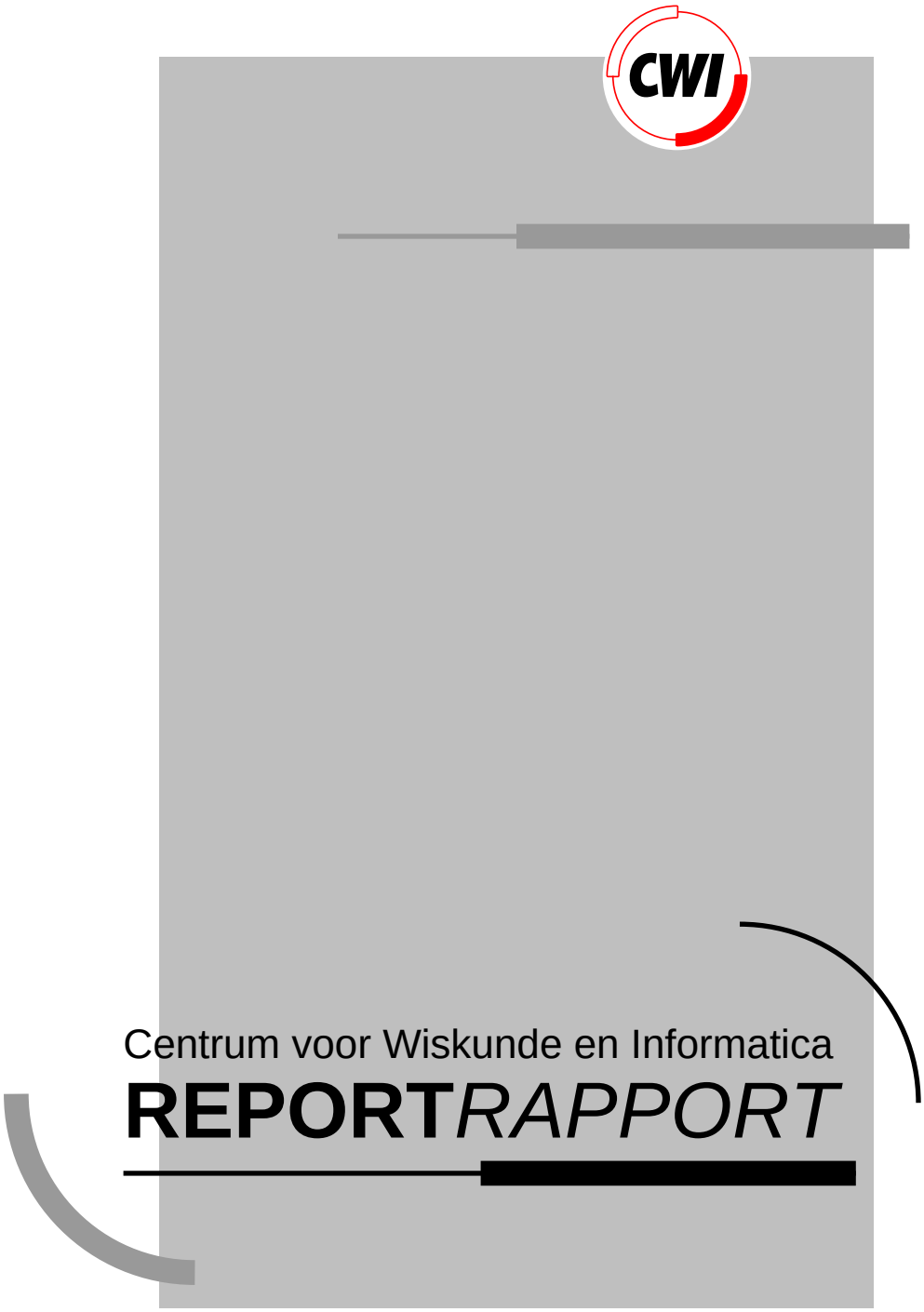

Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Optimal sorting in linear arrays with minimum global control

F. Abolhassan, J. Keller, D. Scheerer

Computer Science/Department of Algorithmics and Architecture

CS-R9244 1992

Optimal Sorting in Linear Arrays with Minimum Global Control

Ferri Abolhassan
Computer Science Department
Universität des Saarlandes, 6600 Saarbrücken, Germany

Jörg Keller
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Dieter Scheerer
Computer Science Department
Universität des Saarlandes, 6600 Saarbrücken, Germany

Abstract

We investigate linear sorting arrays with a single global control wire. We show that the lower bound on the necessary time to sort n numbers in such an array is $2n$, in contrast to $3n$ in arrays without global control. We present a two–phase sorting algorithm for arrays with a global control wire that matches the time bound of $2n$. As $2n$ is a lower bound for sorting arrays with sequential input and output, this implies that further global control cannot lead to any improvements as long as sequential I/O is assumed. The algorithm needs an array of minimum length $\lceil n/2 \rceil$. The scheme can be easily extended to eliminate duplicate numbers. One possible application of the algorithm is in packet routing for algorithms that need presorted inputs.

1991 Mathematics Subject Classification: 68P10, 68Q10, 68Q22, 68Q25
1991 CR Categories: B.6.1, F.1.3, F.2.2
Keywords and Phrases: Sorting, linear array, global control, lower bound, prefix computation
Note: Part of this work was done while the second author was working at the Univ. des Saarlandes, Saarbrücken, Germany. This work is partially supported by the German Science Foundation (DFG) under SFB 124 – D4 and by the Dutch Science Foundation (NWO) through NFI Project ALADDIN under Contract number NF 62–376.

1 Introduction

Sorting is a very important and often required task. Much effort has been put in research to speed up sorting by parallel computation. The time to sort n numbers in parallel has been reduced to $O(\log n)$ [2], which is optimal for comparison–based sorting. In many cases however, the numbers to be sorted are not available all together but are generated sequentially. This leads to a lower bound of $O(n)$ on the time to sort. A simple network that allows to sort in time $O(n)$ is a *linear array*.

Linear arrays consist of p array elements or *nodes* where node $i, 0 \leq i < p$ only is connected to node $i-1$ (if $i > 0$) and to node $i+1$ (if $i < p-1$). Node $i-1$ is called the *left neighbour* of i, $i+1$ is called the *right neighbour*. The links are bidirectional. All inputs are sent to node 0, all outputs come from there. Each node takes its decisions depending on its own state and the informations sent by its

neighbours. The only global control consists of informing all nodes that the last input has reached node 0. This definition of the linear array is the same as in [3] except for the global wire.

The time to sort on a linear array with global control is at least $2n$ steps because n numbers have to be put into the network sequentially, and the sorted sequence has to be put out of the network again. Without the global information about the end of the input phase, the lower bound is $3n$ [3]. Input of n numbers takes time n. Forwarding information about the end of the input phase to all nodes takes time n as well, and output of n numbers takes the last n steps.

Each node in the array contains at least two registers as it must be able to store and compare two numbers. Therefore one needs an array of length at least $p = \lceil n/2 \rceil$ to sort n numbers. The algorithm presented in [3] needs arrays of length n. The nodes in both algorithm are similar. Each step of a node consists of comparing two numbers, accepting an input and sending one number.

To our knowledge all other work on networks with global information transmit deals with busses. These are global links that allow to broadcast one data word per clock tick. That model therefore is more powerful but also more expensive than the single global control wire that we use, because in networks with n nodes the word size normally is considered to be at least $\log n$. These networks also do not take care of input and output, all data is assumed to be in that network at system start.

A typical example is [5] (see also the references therein). There an exact bound of $\Theta(2n/3)$ is derived for sorting n numbers on a linear array of size n with an additional bus.

In section 2 we will present an algorithm to sort n numbers on a linear array. In section 3 we will show that the required length of the array is minimum, we will prove the correctness and we will show that the algorithm runs in minimum time. The fact that the runtime of the algorithm matches the lower bound also implies that further extension of global control cannot lead to any runtime improvements as long as sequential input and output is assumed. In section 4 we describe how the algorithm can be extended to eliminate duplicate numbers without performance loss and how it can be used for prefix computations. Section 5 contains some concluding remarks.

2 Algorithm

The algorithm works in two phases. Phase 1 lasts until the last number has entered the array. In this phase each node receives numbers from its left neighbour and sends numbers to its right neighbour. Phase 2 lasts until the last number of the sorted sequence has left the array. In this phase the direction of the links are opposite to the direction in phase 1.

Each node is able to store two numbers, to compare them and to send and receive one number across different links in one cycle.

In phase 1 each node works as follows. If a node contains less than two numbers, it stores an incoming number from the left neighbour in a free register. If a node contains two numbers, it compares them and sends the larger one to the right neighbour. An incoming number is stored in the register that gets free because its content is sent to the right.

In phase 2 each node works as follows. If a node contains two numbers, it compares them and sends the smaller one to the left. If it contains one number it sends it to the left. Incoming numbers from the right are stored in a free (or getting free) register.

We illustrate the algorithm with two examples for $n = 6$. Figure 1 shows the array after the end of each step. Numbers that have been moved are placed in a box. In the first example the input sequence is in unsorted order, after phase 1 the numbers are already sorted. In the second example the input is almost sorted, the numbers are not sorted after phase 1.

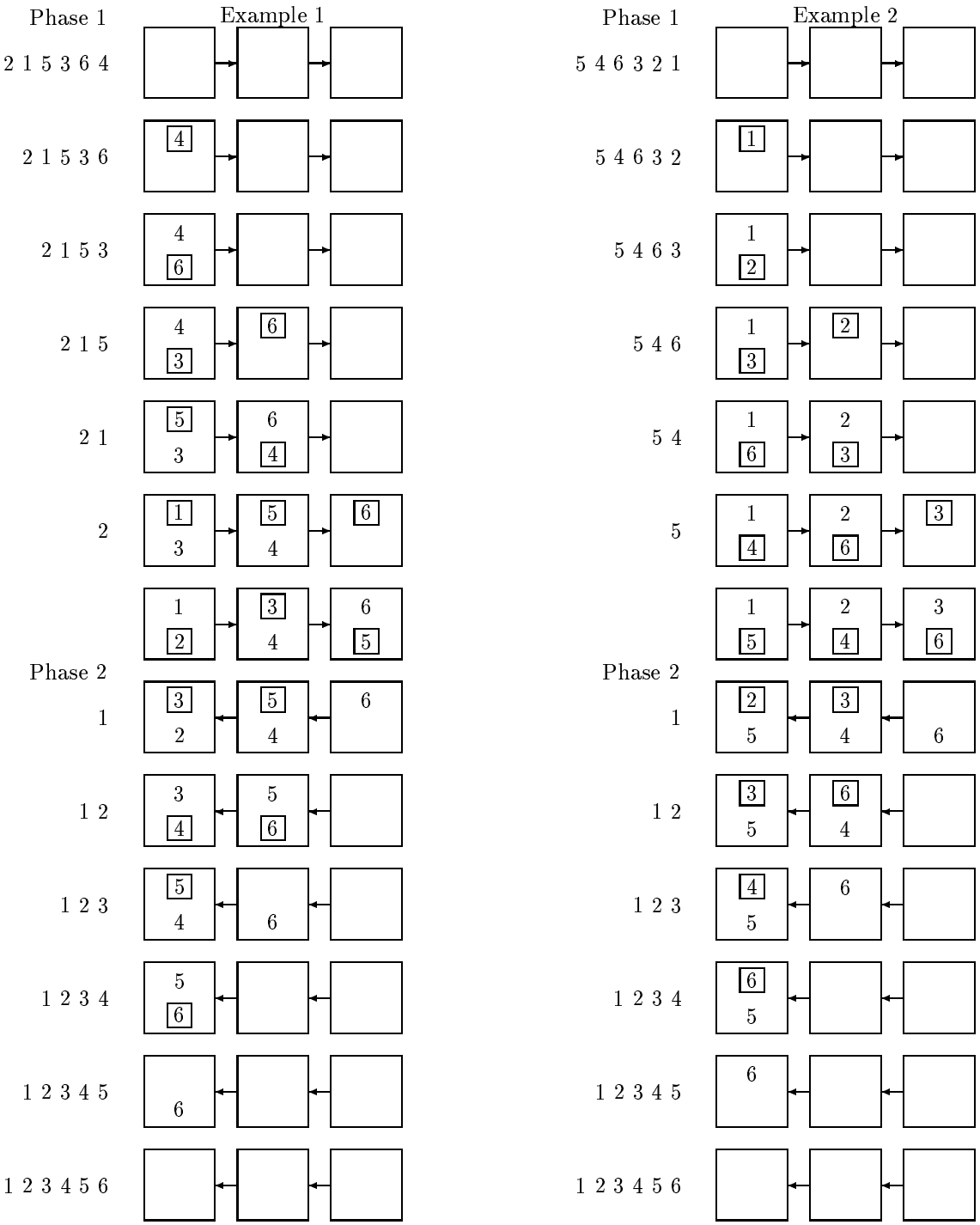

FIGURE 1. Sorting on a linear array for 6 numbers

3 Correctness and Optimality

3.1 Runtime

We show that the algorithm runs in time exactly $2n$ by proving the following claim.

Claim 1 *A node can always send a number to its appropriate neighbour (right in phase 1, left in phase 2).*

It follows that both phases are not delayed and take time n each.

Proof of Claim 1: A node x could be unable to send a number to its neighbour only in case that this neighbour contains two numbers and itself is unable to send one of them because of the same reason. We will call this situation A.

In phase 1 situation A would by induction include that all nodes $y > x$ would contain two numbers and were unable to send. It follows that there must be a node $z \leq x$ that contains less than two numbers, as otherwise phase 1 would have ended, because $n/2$ nodes contain two numbers each. Node z must have contained two numbers some time before because otherwise it never would have sent anything to nodes right of it. It is an easy induction that a node that contains two numbers at one time in phase 1 will keep to contain two numbers until phase 1 ends[1]. So situation A cannot happen.

In phase 1 situation A would include that all nodes $y < x$ would contain two numbers and would be unable to send. This cannot happen as node 0 always is able to output. ∎

3.2 Correctness

We show the correctness of the algorithm by proving the following two claims. We assume that the numbers we sort are integers that can be ordered by the usual $<$ relation such that we can refer to the i-th smallest number ($1 \leq i \leq n$).

Claim 2 *After the end of phase 1, the i-th smallest number ($1 \leq i \leq n$) resides in node $j < i$.*

Claim 3 *After t steps of phase 2, the i-th smallest number ($i > t$) resides in node $j < i - t$.*

Claim 2 is necessary to prove claim 3. Claim 3 leads to the following corollary which states the correctness.

Corollary 4 *After t steps of phase 2, the $(t+1)$-th smallest number resides in node 0 and therefore leaves the array in the next step.*

Proof of Claim 2: We prove the claim by an induction on i. The claim holds for $i = 1$ because the smallest number will never be moved from node 0. An element is only sent to the right because it is the larger one in a comparison.

Assume the claim holds for $i < k$. If the k-th smallest number resides in node $j \geq k$, then it has been sent there because of a comparison with a number $k' < k$ in node $j - 1 \geq k - 1$. Then number $k' < k$ would reside in node $j - 1 \geq k'$ which contradicts the assumption. Therefore the claim holds for k as well. ∎

Proof of Claim 3: We prove the claim by an induction on t. The claim holds for $t = 0$ because of claim 2.

Assume the claim holds for $t < k$. If after k steps of phase 2, the i-th smallest number resides in node $j \geq i - k$, then there are only two possibilities because of claim 1. Either this number was sent to node j in step k and therefore it resided in node $j + 1 \geq i - (k - 1)$ after step $k - 1$, which contradicts the assumption, or this number resided in node j after step $k - 1$ as well. This means that during step k a number $i' < i$ was sent from node j to $j - 1$. Element i' must have resided in node $j \geq i' - (k - 1)$ after step $k - 1$ which contradicts the assumption. Therefore the claim holds for k as well. ∎

[1] It could contain one number only if it sends one of two numbers but does not receive another.

3.3 Number of Nodes

We want to show that at most $n' = \lceil n/2 \rceil$ nodes are necessary. We need to prove this only for phase 1, because in phase 2, nodes send numbers only to their left neighbour. The bound n' follows directly from claim 5.

CLAIM 5 *After $t \geq 2$ steps of phase 1 exactly the nodes $0, \ldots, \lfloor t/2 \rfloor - 1$ contain two numbers.*

Proof of Claim 5: We prove the claim by induction on t. The claim holds certainly for $t = 2$. We assume the claim holds for $t' < t$. If t is even, then from step $t-2$ on, node $x = \lfloor t/2 \rfloor - 2$ contains two numbers. In the following two steps each of the nodes $0, \ldots, x$ sends one number per step. As they also receive one number per step, they still contain two numbers each. These are the first steps in which node x sends numbers. Then node $x+1$ contains two numbers for the first time at the end of step t. A similar argument holds for an odd t. ∎

4 EXTENSIONS

4.1 Elimination of Duplicates

Until now we assumed that all n input numbers are distinct. If the same number can appear multiple times, we want to be able to eliminate duplicates which means that the sorted sequence consists of distinct numbers. If two equal numbers are compared in a node, we send one of them and keep the other one. The strategy to select one of the numbers does not matter here. However if we use the elimination strategy to support more complex operations, we might need to fix it (see subsection 4.2).

If the sorted sequence leaves the array, duplicates follow each other immediately. We extend node 0 by an additional register and a comparator. Instead of leaving the array immediately, the output first enters the additional register. In the next step the content of the additional register is compared with the next output. If both are different, the content of the register leaves the array and the next output enters the register. If both are equal, the register content is thrown away and the next output enters the register.

The time to sort n numbers is increased by one step, the array is extended by one register and one comparator, which is less than one node.

It would also be possible to eliminate duplicates during sorting. But this method allows to support more complex operations as the next section shows.

4.2 Prefix Computations

The extension to eliminate duplicates can also be used to perform prefix computations. We assume that we want to sort packets instead of integers. A packet consists of a key and a piece of data. We will denote packets by $\langle key, data \rangle$. We sort packets by their keys. In case that a number of packets $\langle key_i, data_i \rangle$, $0 \leq i < I$, have identical keys (i.e. $\forall 0 \leq i, j < I : key_i = key_j$), we want to perform a prefix operation with an assoziative operator $\oplus$ on their data pieces:

In the sorted sequence, the i-th packet shall carry a new data piece $data'_i := \oplus_{j \in I, j \leq i} data_j$.

In order to do this we first have to specify the strategy which of two packets with identical keys should be sent if they meet in a node. In phase 1, we will send the packet that arrived later (i.e. it has a higher position in the input stream). If in phase 2 two packets with identical keys are compared we send the one that arrived earlier which means that it has a lower position in the input stream.

Packets with identical keys now follow each other immediately in the sorted sequence and they are sorted by their positions in the input sequence. In order to implement the prefix operation we have to extend node 0 by an additional register and a comparator (just as for duplicate elimination) and a unit to perform $a \oplus b$.

If now a packet $\langle key, data'_i \rangle$ waits in the additional register and its successor $\langle key, data_{i+1} \rangle$ is ready to leave the sorting array, we send the first packet, replace $data_{i+1}$ by $data'_{i+1} = data'_i \oplus data_{i+1}$ and put the packet $\langle key, data'_{i+1} \rangle$ in the additional register. It follows by an easy induction that this

implements the prefix operation. The runtime of the algorithm is increased by one step, the array is extended by one register, one comparator and one $\oplus$–unit.

5 Conclusions

We presented an algorithm to sort n numbers in a linear array. The two–phase algorithm uses a global wire to inform all nodes about the end of phase 1. This reduces the lower bound on runtime from $3n$ to $2n$ steps. The runtime of the algorithm matches the lower bound. The algorithm needs an array of minimum length $\lceil n/2 \rceil$. The addition of one global wire thus leads to a speedup of 1.5. Therefore this seems to be worthwhile to be implemented. Furthermore $2n$ is a lower bound for sequential input and output. Therefore, more than one global wire cannot lead to any further advantage.

The algorithm can be extended to eliminate duplicate numbers and to support more complex operations like prefix computations. The runtime is only increased by one step, only node 0 of the array is extended.

The algorithm has applications in packet routing networks. Some routing algorithms for butterfly networks assume that the packets that enter the network by one input are sorted by their destination addresses [4, 6]. As normally a stream of packets is not sorted by destination, it has to be sorted before the packets enter the network. As the packets arrive sequentially, a linear sorting array can be used. As network latency always is a critical parameter in any architecture, the presorting should be fast. The abililty to eliminate duplicates can be used in combining networks. The ability to perform prefix computations allows for support of parallel prefix computations during routing [6]. In this surrounding the algorithm is used to implement a sorting chip for a parallel machine architecture [1].

Acknowledgements

The authors want to thank Werner Massonne for helpful discussions.

References

[1] Ferri Abolhassan, Jörg Keller, and Wolfgang J. Paul. On the cost–effectiveness of PRAMs. In *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing*, pages 2–9. IEEE, December 1991.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th ACM Annual Symposium on Theory of Computing*, pages 1–9, New York, 1983. ACM.

[3] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, 1992.

[4] F.T. Leighton, Bruce Maggs, and Satish Rao. Universal packet routing algorithms. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 256–269, 1988.

[5] Joseph Y.-T. Leung and Sunil M. Shende. Packet routing on square meshes with row and column buses. In *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing*, pages 834–837. IEEE, December 1991.

[6] Abhiram G. Ranade. How to emulate shared memory. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 185–194, 1987.