Theory of Computing Systems Manuscript-Nr.
(will be inserted by hand later)

Periodic merging networks*

Mirosław Kutyłowski[1] and Krzysztof Loryś[2] and Brigitte Oesterdiekhoff[1]**

[1] Heinz Nixdorf Institute and
Department of Mathematics and Computer Science,
University of Paderborn,
D-33095 Paderborn, Germany,
{mirekk,brigitte}@uni-paderborn.de

[2] Dept. of Computer Science,
University of Trier,
D-54286 Trier, Germany,
lorys@TI.Uni-Trier.DE

Abstract. We consider the problem of merging two sorted sequences on constant degree networks performing compare-exchange operations only. The classical solution to this problem is given by the networks based on Batcher's Odd-Even Merge and Bitonic Merge running in $\log(2n)$ time. Due to the obvious $\log n$ lower bound for the runtime, this is time-optimal.

We present a new family of merging networks working in a completely different way than the previously known algorithms. They have a novel property of being periodic: this means that for some (small) constant k, each processing unit of the network performs the same operations at steps t and $t+k$ (as long as $t+k$ does not exceed the runtime.) The only operations executed are compare-exchange operations, just like in the case of Batcher's networks. The architecture of the networks is very simple, easy to be laid out.

We show that even for period 3 there is a network in our family merging two n-element sorted sequences in time $O(\log n)$. Since each network of period 2 requires $\Theta(n)$ steps to merge such sequences, 3 is the smallest period for which we may achieve a fast runtime.

In order to improve constants standing in front of $\log n$ we increment the period and tune the construction using additional techniques. We achieve the runtime $9 \cdot \log_3 n \approx 5.7 \cdot \log n$ for a network of period 4, and $2.25 \cdot (1+\frac{k+3}{k-1+\log 3}) \log n \approx 2.25 \cdot \log n$ for a network of period $k+3$, for $k \geq 1$.

Due to the periodic design, our networks have small area complexity. For instance, if each processing unit requires $O(1)$ area and a comparator uses a single wire of width $O(1)$ connecting the processing elements, then our networks

* A short version of this paper will be presented at *7th Int. Symposium on Algorithms and Computation*, Osaka'96. This research was partially supported by KBN grant 8 S503 002 07, DFG-Sonderforschungsbereich 376 "Massive Parallelität", DFG Leibniz Grant Me872/6-1 and EU ESPRIT Long Term Research Project 20244 (ALCOM-IT)

** This research has been done partially when the second author was with Institute of Computer Science, University of Wrocław, and while visiting University of Paderborn

require $\Theta((n/\log n)^2)$ area. This compares well with the Batcher's networks that require area $\Theta(n^2)$.

Keywords. merging, compare-exchange operation, comparator, periodic network

1. Introduction

Merging is the following problem: *given two sorted sequences A, B of n keys each, arrange all $2n$ elements of the sequences A and B in one sorted sequence.* Merging is one of the most fundamental problems in computer science and has been intensively studied from theoretical and practical point of view. It is used, as a subprocedure, by a large number of computer algorithms.

Due to its importance, there have been a lot of efforts to construct optimal merging algorithms. For sequential programs, one of the most important points for the theoretical analysis is to determine the number of comparisons necessary and sufficient to merge two sorted sequences. A lot of deep results have been published on this problem (for some starting reference points see for instance [10]).

In the parallel setting at least two very different situations have to be considered. For one of them we have a parallel machine performing non-oblivious computations, for instance a shared memory machine (PRAM). In this case the crucial points might be a low runtime and small total work done by the processors (which should be close to the runtime of the best sequential merging algorithms). Fascinating sublogarithmic time algorithms have been developed for this model, see for instance [4].

The second parallel setting considered are fixed interconnection networks performing oblivious computations. The algorithms developed in this setting might be better suited for practical computing than non-oblivious solutions. Moreover, if the primitives used by the algorithm are simple enough, the algorithm might be suitable for a VLSI implementation. The model that we consider in this paper contains several assumptions that make such an implementation relatively easy.

1.1. The Model

In this paper we construct *comparator network algorithms.* A comparator network algorithm is one of the simplest models for sorting and merging problems. It has an *underlying graph* $G = (V, E)$, where V is a set of processing units (serving as registers as well) and E is a set of the links connecting the processing units. We assume that G is a constant degree graph. The elements of E will be later called *comparators* due to the function they perform. During an execution of the algorithm every processing unit stores exactly one element at each moment. The set V is ordered and the goal of the merging algorithm is to relocate the input elements from two sorted sequences, so that they form a nondecreasing sequence according to the ordering of V. The convention for placing two sorted sequences of input elements might be arbitrary provided that it is simple and independent of the input sequences. The only basic operation that may be performed is a *compare-exchange operation.* It uses a comparator $(P_i, P_j) \in E$, where $P_i < P_j$

in the ordering of V, and is executed as follows: the elements held in P_i and P_j are compared, then the smaller element is placed in P_i and the bigger one is placed in P_j. Simply speaking, the elements stored in P_i and P_j are put in the right order. The processing unit P_i (P_j) is said to be on the minimum side (on the maximum side) of the comparator (P_i, P_j).

Different compare-exchange operations may be executed in parallel as long as they do not involve common processing units. Hence we may define a *parallel compare-exchange step* as a collection of compare-exchange operations executed simultaneously, where the comparators used define a matching in G (not necessarily a perfect matching).

We say that a comparator network algorithm is periodic with period k if for every t, the parallel compare-exchange steps t and $t+k$ consist of the same parallel compare-exchange operations. The steps $(i-1)\cdot k+1$ through $i \cdot k$ are called the ith round of the algorithm.

Quite often, comparator network algorithms are described in a slightly different way (see e.g. [7]). The processing units are replaced by input lines. Then step t of a compare-exchange algorithm corresponds to the level t of the comparators in the model with lines. If a compare-exchange algorithm is not periodic, such an interpretation is necessary for an implementation on a constant degree network.

1.2. Previous work

Merging by comparator networks algorithms has been studied for a long time. The most famous merging networks are the Odd-Even Merger and the Bitonic Merger of Batcher [1] and the Balanced Merger of Dowd et al. [5]. They run in time $\log(2n)$ and use $n \cdot \log n + O(n)$ comparators. There has been a lot of research on these networks and they have been modified in many ways (see [16], [14], [3], [2], [12]). Miltersen et al. [11] prove that the minimal number of comparators needed for merging two n-element sequences is at least $n \cdot \log n - O(n)$, improving the previous results (see Knuth [7]). It follows that the minimal number of steps for merging two n-element sequences performed by a comparator network algorithm is at least $\log n - O(1)$. Thus the networks of Batcher are asymptotically optimal.

Recently, *periodic* comparator network algorithms for sorting have been intensively studied. After a number of algorithms with a non-constant period ([5], [2], [12] and [3]) finally some fast sorting networks having a constant period have been found: [15], [8], [6] and [9]. The main result of the last paper mentioned is so called "periodification scheme". It shows that any sorting comparator network algorithm may be rebuilt so that we get a periodic comparator network algorithm. This modification causes a slowdown of a factor $\log n$ in the case of fast algorithms.

For merging we use some ideas of [9]. However, the periodification scheme cannot yield a time-optimal merging algorithm due to the slowdown inherent in the periodification scheme. Therefore we develop a different direct solution.

1.3. The new results

One could believe that constant period merging cannot be done much faster than in $O(n)$ parallel time. This intuition was proved to be wrong for sorting.

A comparator network algorithm of period 3 and runtime $O(\log^2 n)$ has been developed ([13], see [9] for a solution with period 5). We show that already with period 3 the optimal runtime $O(\log n)$ can be achieved for merging.

Theorem 1. *There is a periodic comparator network of period* 3 *that merges two sorted sequences of* n *numbers in time* $12 \log n$.

Let us note that with respect to the period the above result is optimal:

Proposition 1. *Any comparator network algorithm of period* 2 *merging two sorted sequences of length* n *has runtime* $\Omega(n)$.

In order to get more practical solutions than this given by Theorem 1 we further develop the method used for proving this theorem. Our goal is to decrease the constant 12. By increasing the period to 4 and through additional fine tuning of the algorithm we get the following result:

Theorem 2. *There is a periodic comparator network algorithm of period* 4 *that merges two sorted sequences of* n *numbers in time* $9 \cdot \log_3 n \approx 5.67 \cdot \log n$.

By increasing the period (but still keeping it constant) the running time can be further decreased as follows:

Theorem 3. *For each* $k \geq 1$ *there is a periodic comparator network algorithm of period* $k+3$ *that merges two sorted sequences of* n *numbers in time* $2.25 \cdot (\frac{k+3}{k-1+\log 3}) \cdot \log n \approx 2.25 \log n$.

Note that the results of Theorem 3 differ only by a small constant factor from the lower bound $\log n$, although the restrictions imposed on our network are very strong.

All comparator network algorithms mentioned above have quite simple structure. However, the analysis of their performance is rather technical. For this reason we split our construction into Theorems 1, 2 and 3 and gradually develop techniques leading finally to the best runtime.

An easy inspection of our network shows that it yields the following VLSI-circuit:

Corollary 1. *Assume that each input number can be stored by a processing unit requiring area* $\Theta(1)$ *and that such a number may be sent through a wire of width* $\Theta(1)$ *in* $O(1)$ *time. Then there is a VLSI merging circuit running in time* $\Theta(\log n)$ *and having area* $\Theta(n^2/\log^2 n)$.

It is easy to see that in this setting the circuits derived from Batcher's networks require area $\Theta(n^2)$. The advantage of our solution is that we use the same wires over and over again. This is not the case for the previous solutions, where each step requires a specific set of comparators, and therefore some extra space in the VLSI circuit, unless it can be hidden by pipelining techniques.

2. Basic construction methods

In this section we describe the techniques used by our merging algorithms. We describe the underlying ideas before we go into details of the particular networks.

The first point is that we may confine ourselves to inputs consisting of 0's and 1's. It is well known that comparator network algorithms that sort all inputs consisting of 0's and 1's sort correctly arbitrary input sequences (the so called 0-1-Principle [7]). In a similar way we may prove that this phenomenon holds for merging:

Proposition 2. *If a comparator network algorithm merges two sorted sequences consisting solely of* 0*'s and* 1*'s, then it correctly merges any two sorted input sequences.*

Backed by Proposition 2, for the rest of the paper we consider only the input sequences consisting of 0's and 1's.

Before we go into details leading to our construction, let us sketch the proof of Proposition 1. Consider a comparator network algorithm of period 2 that merges sorted sequences of n elements each. Let $G = (V, E)$ be the underlying graph of the algorithm. Let $P_1^A, \ldots, P_n^A$ denote the nodes where we load the first input sequence A and $P_1^B, \ldots, P_n^B$ denote the nodes where we load the second input sequence B.

Claim. G is a connected graph.

Proof of the claim. Let I_k be the input for which the first $k-1$ nodes of B contain zeroes, P_k^B contains a $\frac{1}{2}$ and the remaining nodes (of A and B) contain ones. Let J_k be equal to I_k except from the first node of A that contains a $\frac{1}{2}$ and P_k^B that contains a one (hence the outputs produced on I_k and J_k are the same).

Since the final destination of $\frac{1}{2}$ on input I_k is the kth output node, the node P_k^B must be in the same connected component of G as the kth output node. Consider now the input J_k. In this case, the output is the same as for I_k. Hence there must be a path from P_1^A to the kth output cell. We conclude that P_1^A and P_k^B are in the same connected component of G. Since k was arbitrary, P_1^A and $P_1^B, \ldots, P_n^B$ are in the same connected component of G. Similarly we show that P_1^B and $P_1^A, \ldots, P_n^A$ are in the same connected component of G. So the claim follows easily.

Since G is a connected graph of degree 2, its edges form a Hamiltonian path or a Hamiltonian circle. Therefore, we may find a node Z in G with distance n from the nth output node. Without loss of generality we assume that $Z = P_i^A$ for some $i \leq n$. We construct an input such that Z contains $\frac{1}{2}$, $P_1^A, \ldots, P_{i-1}^A$ and $P_1^B, \ldots, P_{n-i}^B$ contain zeroes, and the remaining nodes contain ones. Since this input contains together $n-1$ zeroes, the destination of $\frac{1}{2}$ is the nth output node. Hence $\frac{1}{2}$ must be moved n positions, which requires at least n steps. This concludes the proof of Proposition 1.

2.1. The network

Our merging algorithms use a network M the processing units of which are arranged in a $p \times q$-rectangle, where q is even. Let $P_{i,j}$ denote the processing unit of M in the row i and the column j, for $i \in \{1, 2, \ldots, p\}$ and $j \in \{1, 2, \ldots, q\}$. Let $C_j = \{P_{i,j} \mid i \leq p\}$ denote the jth column of M. We order the nodes of M by the snake-like ordering (see Fig. 1):

$$\begin{aligned} P_{i,j} \prec P_{i',j'} \quad \Leftrightarrow \quad & (i > i') \\ \vee \quad & (i = i' \wedge j < j' \wedge i \text{ odd}) \\ \vee \quad & (i = i' \wedge j > j' \wedge i \text{ even}). \end{aligned}$$

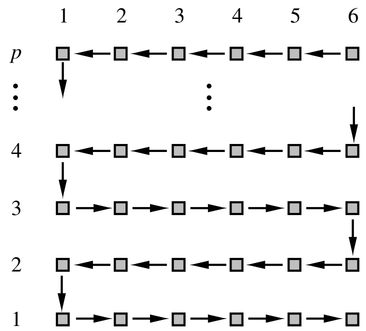

Fig. 1. Snake-like ordering of a $p \times 6$-rectangle for an even p

Each of the steps performed by our algorithms either acts separately within each column or affects pairs of neighboring columns. One of the key features of our construction is that in the last case two neighboring columns either exchange or keep their contents (this holds under the assumption that the input sequences consist of 0's and 1's, and that the comparators may switch the equal elements.) So we may interpret exchanging the contents of columns as moving columns through the network. Therefore we may talk about *moving columns* that travel through M. The purpose of these movements is that at different positions different sets of comparators are applied within the column. In this way a moving column may get sorted fast on a periodic network.

The input allocation for this network has the property that the number of 1's at each column is initially the same up to one. Since the columns move separately, this property will be preserved all the time. So after sorting each column separately, the contents of the network is sorted except for at most one row. Some additional rounds suffice to sort this row.

2.2. Input allocation.

Definition 1. *We say that a column C_i is of the form $1^d(01)^e0^*$ if d bottom positions in C_i are occupied by 1's, the $p-2e-d$ top positions in C_i are occupied by 0's, and in the remaining $2e$ positions there are alternately 0's and 1's, starting with a zero at position $d+1$.*

For our algorithms it is important that the input allocation fulfills the following properties (the first of them has already been mentioned):

Property 1. After loading the input sequences to M
- the number of 1's in the columns of M may differ by at most 1,
- each column has the form $1^d(01)^e0^*$ for some e and d (depending on the column).

The above properties are fulfilled for instance by the following *line-by-line allocation scheme*: The elements of A are loaded into the odd rows, and the elements of B into the even rows; elements of A (B) are placed according to the snake-like ordering. (This method seems to be practical, since we put blocks of input elements into contiguous areas of the network.) Only the two rows where the sorted sequences A and B change from 0 to 1 may contribute to a difference between the number of 1's in two different columns. Since these rows are ordered into opposite directions, the number of 1's in the columns may differ by at most 1. It is obvious that each column is of the form $1^d(01)^e0^*$ for some e and d.

An alternative *interleaved allocation scheme* would be to put the elements of A into the odd positions and the elements of B into the even positions according to the snake-like ordering of M. Since q is even, in each column any pair of two consecutive elements consists of one item from A and one from B. It can easily be shown that if A and B are sorted, then Property 1 holds for this allocation scheme, too.

Our algorithms will be designed in such a way that Property 1 will hold during the whole computation.

2.3. The horizontal steps

The algorithms we consider in this paper use so called *horizontal* steps H_1 and H_2. They correspond to the steps of Odd-Even-Transposition Sort. Each horizontal step consists of comparators connecting processing units in the same row lying in two neighboring columns (see Fig. 2). (An exception are the compara-

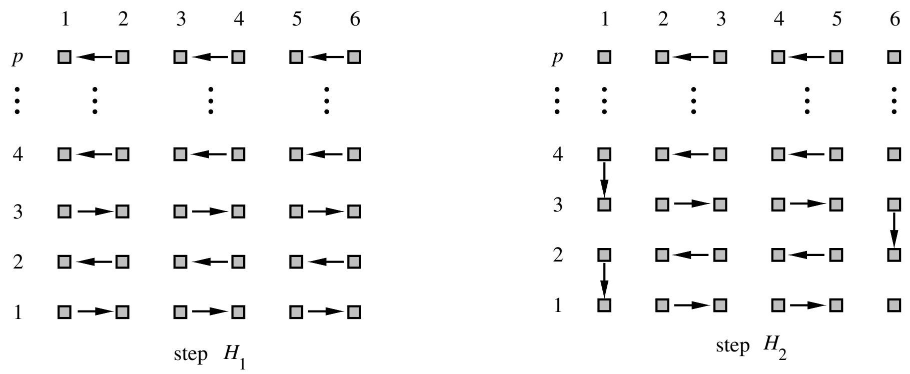

Fig. 2. The parallel steps H_1 and H_2 even p

tors in C_1 and C_q at step H_2 – they play an auxiliary role.) More precisely, step H_1 uses comparators $(P_{i,j}, P_{i,j'})$, $i \le p$, $j \le q$, where

$$\begin{aligned} j' &= j+1 && \text{if} \quad i, j \text{ are odd,} \\ j' &= j-1 && \text{if} \quad i, j \text{ are even.} \end{aligned}$$

step H_2 uses comparators $(P_{i,j}, P_{i,j'})$, $i \le p$, $j \le q$, where

$$\begin{aligned} j' &= j+1, && \text{if} \quad i \text{ is odd, } j \text{ is even, } j < q \\ j' &= j-1, && \text{if} \quad i \text{ is even, } j \text{ is odd, } j > 1. \end{aligned}$$

Additionally, at column C_1 there are comparators $(P_{i,1}, P_{i-1,1})$ for each even $i \le p$, and at C_q there are comparators $(P_{i,q}, P_{i-1,q})$ for each odd $i \le p$, $i \ne 1$.

Definition 2. *A column C with the contents of the form $1^d(01)^e0^*$ is classified as*

$$\begin{aligned} &\text{an E-column} && \text{if} \quad d \text{ is even and } e > 0, \\ &\text{an O-column} && \text{if} \quad d \text{ is odd and } e > 0, \\ &\text{an S-column} && \text{if} \quad e = 0. \end{aligned}$$

("S" stands for "sorted", "O" for "odd", and "E" for "even".)
The lowest d positions in C form the foot *of C, the next $2e$ positions form the* 01-region *of* size $2e$.
We say that a column is an E/S-column if it is an E- or S-column, and we call it an O/S-column if it is an O- or S-column.

By the definition, no column is simultaneously an S-column and an E-column or an O-column.

The following lemma shows that under certain conditions the columns merely exchange their contents during horizontal steps.

Lemma 1. *Consider columns C_i and C_{i+1} of M and assume that the number of 1's in them differ by at most 1. Consider a horizontal step (H_1 or H_2) during which comparators connect the columns C_i and C_{i+1}. Then the following holds for this step:*
(a) If C_i is an O/S-column, C_{i+1} is an E/S-column but they are not both S-columns, then C_i and C_{i+1} exchange their contents.
(b) If either both of the columns C_i and C_{i+1} are E/S-columns or both are O/S-columns, then C_i and C_{i+1} either keep or exchange their contents.
(c) If C_i and C_{i+1} are both S-columns, then they sort together according to the snake-like ordering of M.

Proof. (c) is obvious. For (a), assume that C_i is an O-column and C_{i+1} is an E-column. Let C_i be of the form $1^d(01)^e0^*$ and C_{i+1} be of the form $1^{d'}(01)^{e'}0^*$. Suppose that $d < d'$. Then the d bottom positions in C_i and C_{i+1} are occupied by 1's and this does not change by applying the horizontal step. So we may say that within the first d rows the 1's of C_i move to C_{i+1} and those of C_{i+1} move to C_i. In the next $2e$ rows the 1's of C_i are at the minimum sides of the comparators, whereas the 0's are at the maximum sides of the comparators. Thus the comparators move these 0's and 1's to column C_{i+1} no matter what are the contents of C_{i+1} (equal items are supposed to be switched.) In that way the contents of C_{i+1} inside the considered region move to C_i. Now let us consider the positions above the row $d + 2e$ in both columns. C_i contains only 0's there, so we have to show that all 1's from C_{i+1} in this part are moved to C_i. Column C_i contains $d+e$ ones, hence C_{i+1} contains at most $d+e+1$ ones. Assume that C_{i+1} contains at least one 1 above the row $d+2e$. Since d ones are contained in the first d rows and the next $2e$ rows contain at least e ones, there is exactly one 1 above the row $d+2e$ in C_{i+1}. Since C_{i+1} is an E-column, its last one is at an even row and therefore must be moved into C_i during this step. So we see that also above the row $d+2e$, the columns C_i and C_{i+1} exchange their 1's.

The case $d' < d$ and the cases when one of the columns is an S-column, can be checked in a similar way.

Now we prove (b). We consider the case when both columns considered are O-columns. The other cases can be checked in a similar way. Assume that the moving columns that are at C_i and C_{i+1}, say N_1 and N_2, have the forms $1^d(01)^e0^*$ and $1^{d'}(01)^{e'}0^*$, where $d > d'$ or ($d = d'$ and $e' \geq e$). We do not assume anything about the location of N_1 and N_2 before the execution of the horizontal step. Nevertheless, we show that immediately after the horizontal step, C_i contains N_1 and C_{i+1} contains N_2, regardless of the initial allocation of N_1 and N_2. Thereby we may conclude that either N_1 and N_2 do not move or they switch their positions at this step.

First we consider the case $d > d'$. It suffices to check that all 1's of N_1 go to C_i and all 1's of N_2 go to C_{i+1}. At the bottom d' rows both columns contain only 1's, so we may assume that in the bottom d' positions the 1's of N_1 are in C_i and the 1's of N_2 are in C_{i+1} after the horizontal step. Now let us consider the rows $d'+1, \ldots, d$. By the definition, each of the rows $d'+1, d'+3, \ldots, d-1$ contains a 1 inside N_1 and a 0 inside N_2. The mentioned rows are even, so after the horizontal step, the 1's of N_1 are in C_i. Now we consider the even rows $d'+2$, $d'+4$, $\ldots$, d. Since N_1 and N_2 contain almost the same number of 1's, it follows $d'+e' \geq d-1$. Since $d \geq d'+2$, we know that $e' \geq 1$. So $d'+2e' \geq d$. It follows that the 01-region in N_2 reaches at least the top of the foot in N_1. So the rows $d'+2, d'+4, \ldots, d$ contain 1's in both columns. Hence again we may assume that the 1's of N_1 are in C_i, and the 1's of N_2 are in C_{i+1} after the horizontal step. Above the row d, columns N_1 and N_2 may contain 1's only in the odd rows. Since $d \geq d'+2$ and $d'+e' \geq d+e-1$, it follows that $e' > e$. Consequently $d'+2e' \geq d+2e$. It means that the top of the 01-region in N_2 cannot lie below the top of the 01-region of N_1. According to the ordering of the even rows the 1's of the higher 01-region lying above row d will be in C_{i+1} after performing the horizontal step. Thus we may assume that the 1's of N_2 lying above row d are in C_{i+1} and the 1's of N_1 are in C_i after the horizontal step.

The case $d = d'$ and $e' \geq e$ it can be checked similarly. □

Lemma 1 essentially says that the 0's and 1's being in the same column at the beginning of the computation move together and are within the same column all the time. This justifies the name *moving column*.

Corollary 2. *Suppose each column of M is an E-, O- or S-column and the number of 1's in two different columns may differ by at most 1.*

(a) Let $i > 1$, i be even (odd), and C_i be an E/S-column. Then immediately after executing step H_1 (H_2) column C_{i-1} is an E/S-column.

(b) Let $i < q$, i be odd (even), and C_i be an O/S-column. Then immediately after executing step H_1 (H_2) column C_{i+1} is an O/S-column.

(c) At step H_2 the leftmost column becomes an O/S-column and the rightmost column becomes an E/S-column.

In particular, if each even column is an E/S-column and each odd column is an O/S-column, then after performing the steps H_1 and H_2 even columns are again E/S-columns and odd columns are again O/S-columns.

2.4. The vertical steps

Beside the horizontal steps our algorithms use *vertical steps*. The vertical steps affect each column separately: they use only comparators connecting two processing units from the same column. The compare-exchange operations executed within one column will be called a *column step*. So a vertical step consists of many column steps executed simultaneously in different columns.

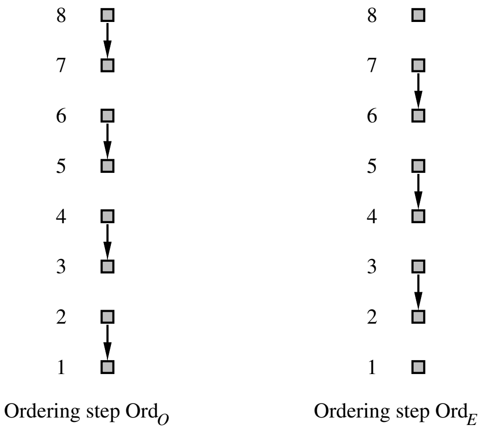

Fig. 3. The ordering steps Ord_O and Ord_E

We use two different types of column steps called the *ordering steps* and the *jump steps*. At certain moments of the computation we have to guarantee that some columns are O/S-columns and some are E/S-columns. For this purpose we execute ordering steps Ord_O (for making an O/S-column) and Ord_E (for making an E/S-column) on E-, O-, and S-columns.

Definition 3. *We define the comparators of* Ord_O *in column C_j as $(P_{i,j}, P_{i-1,j})$ for all even $i \leq p$ (see Fig. 3); the comparators of* Ord_E *are $(P_{i,j}, P_{i-1,j})$ over all odd $i \leq p$, $i \neq 1$.*

By the definition we get immediately:

Property 2. Suppose C_i is an E-, O-, or S-column. By performing Ord_O on C_i, column C_i becomes an O/S-column; by performing Ord_E, C_i becomes an E/S-column.

Now we explain the purpose of the jump steps, the second kind of the column steps we use. Our algorithms relocate the moving columns by performing horizontal steps. During these steps each moving column preserves not only the number of 1's but also their allocation within the column (the only exception are C_1 and C_q where step H_2 makes slight changes). Therefore the moving columns have to become sorted by operations performed by some specially designed steps, namely the jump steps.

Definition 4. *Let $\ell < p$ be odd. Then an* E-jump step of size ℓ *at column* C_i *uses all comparators* $(P_{j,i}, P_{j-\ell,i})$ *where* $j - \ell > 0$ *and* j *is even (see Fig. 4.) An* O-jump step *is defined in the same way except the comparators originate in the processing units* $P_{j,i}$*, where* j *is odd.*
We denote an O-jump step (E-jump step) of size ℓ *by* J^O_ℓ (J^E_ℓ)*. The comparators used by a jump step are called* jump comparators.

Note that the size of a jump step is always odd, so for the case of an O-jump step the comparators originating in odd rows point to processing units in even rows. Therefore no processing unit is used twice and the definition is sound. The situation for E-jump steps is similar.

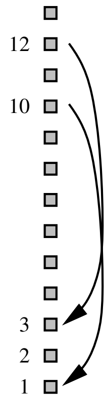

Fig. 4. Comparators of the jump step J^E_9

If an O-jump step is applied to an O-column, then the comparators originate at the odd rows. These are the places, where the 1's of the 01-region in this column are standing. The idea is that with some luck, some of these 1's jump into the places previously occupied by 0's of the 01-region and thereby reduce the size of the 01-region. This phenomenon is discussed in detail by the next lemma (some examples are given by Fig. 5 and 6).

Lemma 2. *Suppose that a column* C_i *has the form* $1^d(01)^e0^*$*. Assume further that* C_i *is an O-column (E-column). Then after performing jump step* J^O_ℓ (J^E_ℓ)*:*

(a) The contents of C_i *remains unchanged, if* $e < \ell/2$.
(b) For $\ell/2 \leq e < \ell$*, column* C_i *gets the form* $1^{d+2e-\ell+1}(01)^{\ell-1-e}0^*$.
(c) For $e \geq \ell$*, column* C_i *gets the form* $1^{d+\ell}(01)^{e-\ell}0^*$.

Moreover, in the cases (a) and (b) the status of C_i *as an O/S- or E/S-column does not change. In the case (c) the status of* C_i *changes from an O/S- to E/S-column and vice versa.*

Proof. Of course, performing the jump step leaves the old foot intact, also a 1 cannot be moved upwards above the 01-region. So the only part that may change is the 01-region.

(a) If $\ell > 2e$, then each jump comparator has either the maximum side in the foot, or the minimum side above the 01-region. Such a comparator does not change the contents of its endpoints, hence performing the jump step has no affect in this case.

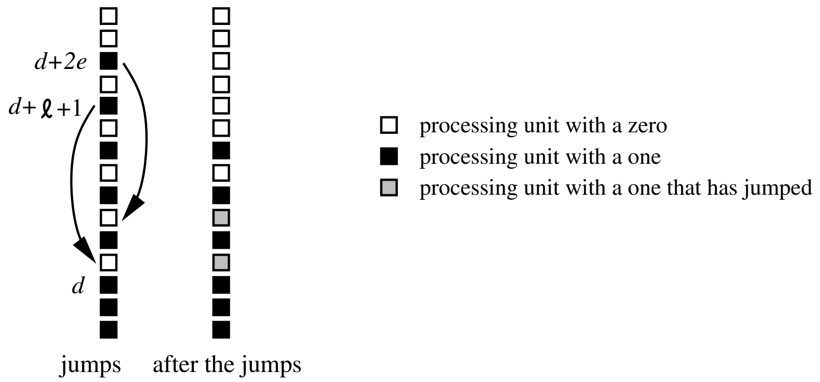

Fig. 5. Case (b) of Lemma 2

(b) This case is depicted by Fig. 5. As already seen, the jump comparators originating below the row $d + \ell + 1$ cannot have 0 on the maximum side and thereby do not change the contents of the column. The jump comparators originating in the rows $d+\ell+1$ up to $d+2e$ have 1's on the minimum side and 0's on the maximum side. Therefore their 1's jump down. There are $e-(\ell-1)/2$ such comparators, so these 1's move into rows $d+1, d+3, \ldots, d+2(e-(\ell-1)/2)-1$. Since $2(e-(\ell-1)/2)-1 = 2e-\ell < \ell$, all these 1's are moved into the part of the 01-region inside which the 1's remain on their places. Thereby, the foot grows up to row $d+2(e-(\ell-1)/2) = d+2e-\ell+1$, and there is a new 01-region containing the remaining 1's that have been not moved. There are $e-2(e-(\ell-1)/2) = \ell-1-e$ such 1's. So the column gets the form $1^{d+2e-\ell+1}(01)^{\ell-1-e}0^*$.

Since the 1's of the new 01-region have not moved during this step, the column preserves its old classification as an E-column or O-column or becomes an S-column.

(c) This case is depicted by Fig. 6. The difference between case (c) and the previous one is that not all 1's that jump down fit into the ℓ lowest rows of the old 01-region – there are too many jumping 1's. The 1's from the 01-region that remain on their places and the 1's jumping into the lowest ℓ rows of the old 01-region cause the foot to grow by ℓ. Above the row $d+\ell$ there are only 1's that have been moved, they form a new 01-region with $e-\ell$ ones. So the column gets the form $1^{d+\ell}(01)^{e-\ell}0^*$.

Since the 1's of the emerging 01-region are the 1's that have jumped, they change their position from odd or even rows into, respectively, even or odd rows. Thereby the status of the column changes from E/S- to O/S-column or vice versa. □

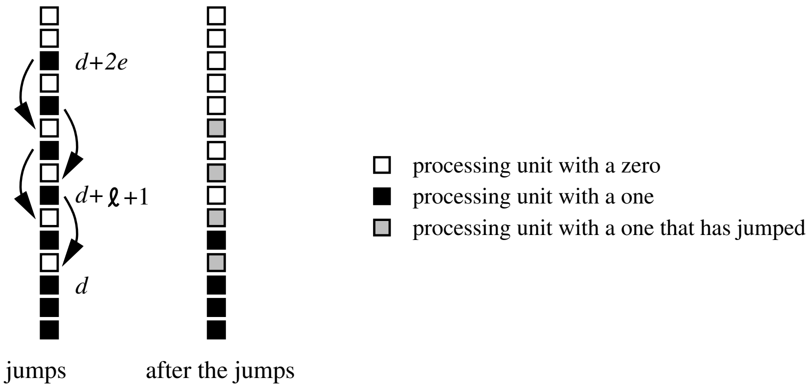

Fig. 6. Case (c) of Lemma 2

Finally, let us notice that jump steps have no effect, if the types of the jump step and of the column disagree:

Property 3. Performing J_ℓ^O (J_ℓ^E) on an E-column (O-column) does not change the contents of the column.

Property 3 follows from the fact that in the situation described, each jump comparator either originate in the foot (and thereby cannot change anything) or contains a zero on its minimum side.

Property 3 shows that before performing jump steps the columns must have a right status in order to achieve any progress. The trouble may be caused by the jump steps themselves. According to Lemma 2, the status of a column may change during a jump step depending on the size of the 01-region. Thereby, if the right status is not reestablished between the consecutive jump steps, the next jump steps might have no effect.

col.	C_1	C_2	C_3	C_4	C_5	C_6	C_7		C_{q-1}	C_q
col. step	Ord_O	Ord_E	$J^O_{\frac{p}{2}-1}$	Ord_E	$J^O_{\frac{p}{4}-1}$	Ord_E	$J^O_{\frac{p}{8}-1}$	...	J^O_1	Ord_E

Table 1. Column steps executed by vertical step V.

2.5. General outline of the algorithms

Each of the algorithms presented in this paper follows a simple strategy: Conceptually the computation consists of two phases: during the first phase all columns become sorted, but it may be they are still in a wrong order causing one row to be unsorted. During the second phase the order of the columns is corrected.

The second phase is simple. Since column steps do not change the contents of the sorted columns, only the horizontal steps make any progress. Except from the only unsorted row the horizontal steps make no change, either. What the horizontal steps do is performing Odd-Even Transposition Sort of the unsorted row. It is known that q steps of Odd-Even Transposition Sort suffice to sort q items [7]. Therefore for relocating the sorted columns we require at most q horizontal steps.

The first phase is more complicated. It is a crucial problem how to arrange the column steps in order to guarantee fast sorting of every column. This problem will be discussed in detail for each of the algorithms presented separately.

3. Merging with the minimal period

In this section we prove Theorem 1 by describing a comparator network algorithm MERGE3 of period 3 that merges two input sequences of length n in $12 \log n$ steps.

MERGE3 uses processing units organized in a $p \times q$-rectangle, where p is a power of 2 and $q = 2 \log p$. It executes periodically 3 different steps H_1, H_2 and V. The horizontal steps H_1 and H_2 have already been described in Subsection 2.3. The vertical step V has to sort each moving column by using jump steps and simultaneously to guarantee the proper type of the columns by executing ordering steps.

3.1. Design of vertical step V

Step V uses only ordering and jump steps described in Subsection 2.4:

- in the even columns V applies the ordering step Ord_E,
- inside C_1, step V applies the ordering step Ord_O,
- in all odd columns except C_1, step V applies different O-jump steps described below.

In order to sort a moving column it suffices to reduce the size of the 01-region to zero. This can be reached by applying jump steps repeatedly. The size of the jumps is essential: if they are too long, then the size of the 01-region is unchanged (see Lemma 2(a)); if they are too short, then the decrease of the size is not substantial (see Lemma 2(c)). By Lemma 2 we get immediately the following corollary:

Corollary 3. *If the size of the 01-region in a O/S-column (E/S-column) is $2e \leq 2\ell$, then performing J^O_ℓ (J^E_ℓ) on this column reduces the size of the 01-region to at most $\ell - 1$ and the column preserves its status as an O/S-column (E/S-column) or becomes an S-column.*

In order to sort an O-column with a 01-region of size $2e \leq p - 2$ we could do the following: First we apply $J^O_{p/2-1}$. Thereby the size of the 01-region is reduced to at most $p/2 - 2$. Then we apply $J^O_{p/4-1}$ in order to reduce the size of the 01-region to at most $p/4 - 2$. We continue in this way and after $\log p - 1$ steps get a 01-region of size 0, i.e. a sorted column.

In order to perform a computation like this on a periodic comparator network, V uses the jump steps $J^O_{p/2-1}$, $J^O_{p/4-1}$, $\ldots$, J^O_3, J^O_1 in the columns, respectively, $C_3, C_5, \ldots, C_{2\log p-1}$. The column steps performed by V are depicted by Table 1.

3.2. Runtime analysis

The crucial point for the runtime analysis of MERGE3 is to check how many steps are required for sorting the columns. For this purpose we have to analyze in more detail the behavior of the algorithm.

The use of the steps Ord_O and Ord_E at step V guarantees the following properties:

Property 4. Consider the situation when each column is an E-,O- or S-column immediately before step V. Consider the next round of the algorithm. Then:
(i) after performing step V the even columns become E/S-columns and C_1 becomes an O/S-column,
(ii) after performing step H_1 the odd columns become E/S-columns,
(iii) after performing step H_2 the even columns become E/S-columns.

Proof. Part (i) follows immediately by the construction of step V. Additionally, it follows from Lemma 2 that after performing step V the odd columns can still be classified as O-, E- or S-columns. Then (ii) follows from Corollary 2(a). Additionally, we know by Lemma 1 that the other columns remain O-, E- or S-columns. Similarly, after step H_2 every even column except for C_q is an E/S-column by Corollary 2(a). By Corollary 2(c) it follows that C_q becomes an E/S-column after the step H_2. □

Property 4 has many consequences. One of them is the following:

Property 5. If at some moment some odd column C_i is an O-column, then performing the steps H_1 and H_2 moves the moving column allocated at C_i into C_{i+2} while preserving its contents.

Property 5 follows immediately from Property 4 and Lemma 1(a).

In the next lemma we will examine what happens to the moving column originally located at C_1 during execution of MERGE3. First it becomes an O/S-column (by performing V). Then it moves through the network from the left to the right. While this moving column reaches C_3, C_5, ..., then step V is performed. Each application of step V means for this moving column execution of a jump step. These jump steps are, by the construction, $J^O_{p/2-1}$, $J^O_{p/4-1}$, ..., J^O_3, J^O_1. We will show how the jump steps reduce the size of the 01-region of this moving column to zero.

Lemma 3. *After performing i rounds of MERGE3, each odd column C_j for $j < \min\{2i, q\}$ is an O/S-column. The size of the 01-region of C_j is bounded by $p/2^{(j-1)/2} - 2$.*

Proof. The lemma can be easily proved by induction on i. For $i = 1$ it follows from the construction of V that C_1 is an O/S-column. Note also that either this column contains no 1's, or after performing Ord_O there is a one in the lowest position of the column. It means that the foot has hight at least 1. Analogously, there is a zero in the top position of C_1. It follows that the size of the 01-region in this column is at most $p-2$. This is what Lemma 3 claims for $i = 1$.

Now assume that the lemma holds for i. At round $i+1$, for each odd $j < \min\{2i, q-1\}$ a moving column staying in C_j (which is an O/S-column) is moved to C_{j+2} without changing its contents. Indeed, it follows from Property 5. Then this column is affected by the jump step in C_{j+2}. By the induction hypothesis and Corollary 3, the size of the 01-region in C_{j+2} is now bounded by $p/2^{(j+1)/2} - 2$ and preserves its status of an O/S-column. Thereby we have proved the claim of the lemma for the odd columns $C_3, \ldots, C_{\min\{2i+1,q-1\}}$. The claim for the column C_1 after step $i+1$ can be derived as for $i = 1$. □

Lemma 3 says also that once a moving column arrives at C_1 it takes not much time to sort this moving column. Now we check how much time has to elapse until the last moving column starts the process of sorting at C_1. By Lemma 3, after $q/2$ rounds of MERGE3 the status of each column is fixed. Moreover, all odd columns (at the beginning of a round) are on the way of sorting. Once a column reaches C_{q-1} it becomes sorted. Let us consider the columns that are still not sorted. We do not have to worry about the moving columns being at the odd columns since each of them arrives at C_{q-1} in at most $q/2$ rounds. More time is required for the moving columns staying at even columns immediately after round $q/2$. By Lemma 1 and Lemma 3 each of these columns goes to the left until it reaches C_1. At most $q/2$ rounds are necessary for this purpose. After this column arrived in C_1, the next $q/2$ rounds are necessary to sort it on the way to the leftmost column. We see that in at most q additional rounds of MERGE3 every such column becomes sorted. Together we need $1.5q$ rounds to sort all columns, that is $4.5q$ steps.

For the second phase of the algorithm (relocating the sorted columns) q horizontal steps are sufficient. It makes $1.5q$ steps of MERGE3. Concluding, the algorithm needs together at most $6q$ steps. Since the sequences that we merge have size $n = p \cdot q/2 = 2^{q/2} \cdot q/2$, we have $q \leq 2\log n$ and therefore the runtime of MERGE3 is bounded by $12 \log n$ completing the proof of Theorem 1.

3.3. Bad Inputs

The following example shows that the upper bound for the runtime of MERGE3 is exact:

Example 1. We consider an input consisting of two sorted sequences such that the first sequence has $3q$ ones and the second sequence has one 1. Then after loading these sequences to M the column C_q has the form $11(01)^2 0^*$ and the remaining columns have the form $(01)^3 0^*$. Note that C_q has an additional 1. Then MERGE3 requires $6q - \Theta(1)$ steps.

Indeed, at the beginning each column is an E-column. By performing MERGE3 O-columns will be generated in C_1. These O-columns move to the right until they reach C_{q-1}. Once the first O-column reaches C_{q-1}, i.e. after q horizontal steps, the column originating in C_q starts to move to the left border. While moving to the left, the 01-region of the moving column does not decrease. In C_1, this column will become an O-column and its 01-region decreases to the size of 2. Now, this column moves again through the network to the right border. The jump comparators it meets do not decrease the size of the 01-region till the moving column reaches C_{q-1} (see Lemma 2). In C_{q-1}, the column becomes sorted. Till this moment total of $3q - \Theta(1)$ horizontal steps are performed. Now the additional 1 in C_{q-1} must go to the left border, because this is in an even row. It takes the next $q - \Theta(1)$ horizontal steps. Together $4q - \Theta(1)$ horizontal steps, i.e. $6q - \Theta(1)$ steps are performed. □

4. Faster merging networks of period 4

In this section we present an algorithm MERGE4 establishing Theorem 2.

4.1. Basic ideas

In order to construct an algorithm faster than MERGE3 we make three significant modifications. First we sketch these changes.

SIZE OF THE JUMPS. The jump steps of MERGE3 are applied as follows: if we know that the size of an 01-region is upper bounded by $p/2^i - 2$, then applying a jump step of size $p/2^{i+1} - 1$ reduces the 01-region so that its size becomes upper bounded by $p/2^{i+1} - 2$. That is, the upper bound on the size of the 01-region is halved. However, we may improve this:

Lemma 4. *Suppose that an O-column (E-column) contains a 01-region of size $2e < 3\ell$ (ℓ odd). Then after applying jump step J^O_ℓ (J^E_ℓ) the size of the 01-region is less than ℓ.*

Proof. If already $2e < \ell$, then there is nothing to prove, since a 01-region cannot get bigger. If $\ell \leq 2e < 2\ell$, then by Lemma 2(b) the new 01-region has size $2 \cdot (\ell - 1 - e) < \ell$. For $2e \geq 2\ell$ we use Lemma 2(c): the size of the 01-region becomes $2e - 2\ell < 3\ell - 2\ell = \ell$. □

Lemma 4 shows that the bound on the size of the 01-region may be reduced through one jump step to **one-third** of the original value. The reader might wonder why we have not used such jumps for MERGE3. The point is that the jump steps of Lemma 4 do not guarantee that the columns preserve their type as O- or E-columns. Preserving the type of columns was crucial for correctness of MERGE3. Of course, the type of the columns might be adjusted by ordering steps, but it requires an extra step at each round.

WRAPPING AROUND. During execution of MERGE3 the moving columns are sorted while moving from C_1 to C_q. Since $\log p - 1$ jump steps are to be executed and for one vertical step we make 2 horizontal steps, we require $2 \log p$ columns. This gives a lower bound on q. On the other hand, q occurs at many places in the estimation of the runtime (for instance, how many horizontal steps we need to relocate the sorted columns). So it would be desirable to reduce q as much as possible.

The idea for reducing q is that we perform jump steps in **all** columns. We shall arrange a round so that it contains: (1) one vertical step performing jumps, (2) one vertical step fixing the types of the columns, (3) two horizontal steps for moving the columns. Therefore, a moving column that goes to the right is affected by jump steps at every second column C_i. Say, these are *odd* columns. After reaching C_q the moving column starts going to the left. From now on, the jump steps will be applied to this moving column at the *even* columns, until the moving column reaches the left border.

With the above scheme, we can distribute all k jump steps that are to be executed among k columns (and not $2k$ columns, as before.) This reduces q by half.

STARTING POSITION. After the columns are sorted we have to relocate the columns. This phase mimics Odd-Even Transposition Sort. Assume that the last moving column that becomes sorted contains one more 1 than the other columns. If this moving column gets sorted, say, at column C_1 and the final destination of the extra one is in C_q, then at least $q-1$ horizontal steps are necessary to move this one to the final position.

In order to speed up relocating the columns we arrange the jump steps so that the sorted columns are produced at the **middle** column and not at C_1 or C_q. The example considered above shows that this makes a difference: at most $q/2$ horizontal steps are needed to move the column with the extra one into C_1 or C_q. In general the argumentation is more complicated. We shall show that while the last moving columns are on their way to get sorted, the moving columns that are already sorted get ordered in some sense. Thereby, when relocating of the columns begins, we do not start Odd-Even Transposition Sort from the very beginning.

4.2. Definition of MERGE4

MERGE4 uses processing units arranged in a $p \times q$-rectangle, where $p = 3^q$, q is even. Note that p is odd. A round of MERGE4 consists of the steps A, H_1, H_2, J, where H_1 and H_2 are the horizontal steps previously defined, and A and J are vertical steps.

Step A performs Ord_O in the odd columns and Ord_E in the even columns (see Fig. 7). By the definition we get immediately the following property:

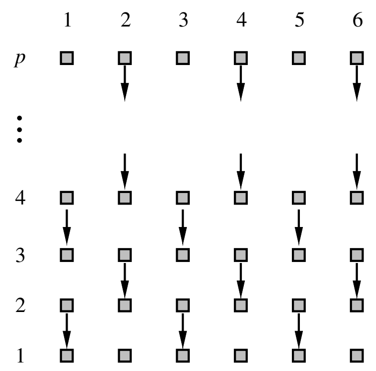

Fig. 7. step A on a $p \times 6$-rectangle for odd p

Property 6. Suppose each column is an E-, O- or S-column. Then by step A, each odd column of M becomes an O/S-column, and each even column becomes an E/S-column.

By Property 6, immediately before step H_1 the unsorted columns have the right status in order to be moved (see Corollary 2). By applying Corollary 2 once again for the situation after step H_1 we see that after step H_2 again each odd column is an O/S-column and each even column is an E/S-column.

Step J performs jump steps in all columns and is responsible for sorting the moving columns. Before we discuss how to choose the sizes of the jumps let us summarize our observations:

Property 7. – During steps H_1 and H_2 the moving E-columns are moved to the left, the moving O-columns are moved to the right. The only exception are the moving columns at C_1 and C_q during step H_2 – they cannot move any further. Instead they change their status: the leftmost column becomes an O/S-column (previously being an E/S-column) and the rightmost column becomes an E/S-column (previously an O/S-column).
- During step J some changes within unsorted columns are made. Thereby an E-column (O-column) may become an O-column (E-column), but at the next step A the old status is recovered.
- Every unsorted moving column is moved back and forth through M, the changes of the direction occur only at C_1 and C_q.

column C_i	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}
$\mathrm{mid}^{-1}(i)$	9	8	10	7	11	6	12	5	1	4	2	3
jump steps executed	$J^O_.$	$J^E_.$	$J^O_.$	$J^E_.$	$J^O_.$	$J^E_.$	$J^O_.$	$J^E_.$	$J^O_.$	$J^E_.$	$J^O_.$	$J^E_.$
size of the jumps	$\frac{p}{3^9}$	$\frac{p}{3^8}$	$\frac{p}{3^{10}}$	$\frac{p}{3^7}$	$\frac{p}{3^{11}}$	$\frac{p}{3^6}$	$\frac{p}{3^{12}}$	$\frac{p}{3^5}$	$\frac{p}{3^1}$	$\frac{p}{3^4}$	$\frac{p}{3^2}$	$\frac{p}{3^3}$

Table 2. Construction of step J for $q = \log_3 p = 12$.

We define step J so that the essential part of sorting of the moving columns starts and ends in the middle of the network. In order to define the jump steps at step J, we list the columns of the network in the following way. Let $r = q/2$. Without loss of generality we assume that $q/2$ is even. First we take all odd columns starting with C_{r+3}:

$$C_{r+3}, C_{r+5}, \ldots, C_{q-1};$$

then we take all even columns in the reverse order starting with C_q:

$$C_q, C_{q-2}, \ldots, C_2;$$

finally we take all remaining odd columns starting with C_1:

$$C_1, C_3, \ldots, C_{r+1}.$$

Let $\mathrm{mid}(i)$ denote the index of the ith column of this list. More formally, for $i \leq q$ let $\mathrm{mid}(i)$ denote

$$\begin{array}{rll} r+2i+1 & \text{if} & 1 \le i < \frac{1}{2}r, \\ q-2(i-\frac{1}{2}r) & \text{if} & \frac{1}{2}r \le i < \frac{3}{2}r, \\ 2i-3r+1 & \text{if} & \frac{3}{2}r \le i \le 2r. \end{array}$$

Now we can determine the sizes of the jump steps executed at step J: in column $C_{\mathrm{mid}(i)}$ the column step of size $p/3^i$ is executed. The jump steps are O-jumps in the odd columns and E-jumps in the even columns. (For $q = \log_3 p = 12$ the jump steps of step J are depicted by Table 2).

4.3. Runtime analysis of MERGE4

4.3.1. Getting sorted columns in the middle of M

Property 8. Assume that a moving column X is at $C_{r+3} = C_{\mathrm{mid}(1)}$ immediately before step J. Then for $i \le q$ after the consecutive i steps J column X resides in $C_{\mathrm{mid}(i)}$ and the 01-region of X has size less than $p/3^i$.

Proof. The proof is by induction on i. Immediately before step J performed at $C_{\mathrm{mid}(1)}$ the moving column X has a 01-region smaller than p. Indeed, the size of the 01-region is even and the height of the columns is odd, hence the size of the 01-region is at most $p-1$. Since the jump step $J^O_{p/3}$ is performed in $C_{\mathrm{mid}(1)}$, the size of the 01-region of X becomes less than $p/3$ (see Lemma 4). Hence the lemma holds for $i = 1$. Now assume that the lemma holds for i. Let us consider the next round. At step A column X gets the proper status as an E/S- or O/S-column. The size of the 01-region may only be reduced at this moment. By Property 7, at steps H_1 and H_2 column X moves to $C_{\mathrm{mid}(i+1)}$. By the induction hypothesis, the 01-region in $C_{\mathrm{mid}(i+1)}$ immediately before step J has size less than $p/3^i$. In $C_{\mathrm{mid}(i+1)}$ the jump step of size $p/3^{i+1}$ is performed and by Lemma 4 the size of the 01-region of X becomes less than $p/3^{i+1}$. □

If we apply Property 8 to $i = q$, then we get that immediately after q steps J moving column X becomes sorted and resides in $C_{\mathrm{mid}(q)} = C_{r+1}$.

From now on it will be convenient to count the horizontal steps instead of all steps. Let h_t denote the tth horizontal step. Due to its special importance, let C_{r+1} be called *midcolumn*. Now our last observation may be formulated as follows:

Corollary 4. *Let $t \ge 2q$, t even. Then immediately before step h_{t+1}, the moving column stored in the midcolumn is sorted.*

4.3.2. Filling the right half of M with sorted columns

We trace what happens with the sorted moving columns mentioned by Corollary 4 during the next steps:

Property 9. For $t \ge 1$, immediately before the horizontal step h_{2q+t}, the following columns are sorted:

$$\begin{array}{rll} C_{r+1}, C_{r+3}, \ldots, C_{\min\{r+t,q-1\}}, & \text{if} & t \text{ is odd,} \\ C_{r+2}, C_{r+4}, \ldots, C_{\min\{r+t,q\}}, & \text{if} & t \text{ is even.} \end{array}$$

	C_7 $\Vert$ C_{r+1}	C_8	C_9	C_{10}	C_{11}	C_{12} $\Vert$ C_q
$t = 1$	S					
$t = 2$		S				
$t = 3$	S		S			
$t = 4$		S		S		
$t = 5$	S		S		S	
$t = 6$		S		S		S

Table 3. The sorted columns before the steps h_{2q+t} for $q = \log_3 p = 12$ are indicated by an S.

Proof. The proof is by induction on t (see Table 3). The case $t = 1$ holds by Corollary 4. Now we prove our claim for $t > 1$. Let t be odd. We show that for each odd j, $r + 1 \le j \le \min\{r + t, q - 1\}$, column C_{j+1} becomes sorted during step h_{2q+t}. Note that C_j has been a sorted column. At step h_{2q+t} column C_j is compared with C_{j+1} which is an E/S-column. By Lemma 1, if C_{j+1} is an E-column, then the columns C_j and C_{j+1} exchange their places. Thereby C_{j+1} becomes sorted. If C_{j+1} is already sorted, then both C_j and C_{j+1} remain sorted (maybe they exchange their places). This proves the lemma for $t + 1$, if t is odd. For an even t, it can be shown in the same way that the columns C_j become sorted for each odd j, $r + 3 < j \le \min\{r + t + 1, q - 1\}$. Additionally, by Property 4 column C_{r+1} becomes sorted before step h_{2q+t}. □

In particular, Property 9 yields:

Corollary 5. *For $t \ge 0$, immediately before step h_{2q+r+t} the following columns are sorted:*

$$\begin{array}{rll} C_{r+1}, C_{r+3}, \ldots, C_{q-1}, & \textit{if} & t \text{ is odd,} \\ C_{r+2}, C_{r+4}, \ldots, C_q, & \textit{if} & t \text{ is even.} \end{array}$$

Now let us consider the situation at step h_{2q+r}. At this moment the moving column X that became sorted at the midcolumn immediately after step h_{2q} reaches C_q. Later, the moving columns that come from the left to the right border of M are sorted, too. We show that they start to build a region of sorted columns at the right side of the network. This region, called the *sorted sector*, contains initially only the moving column X, but grows by one column at each horizontal step until the sorted sector reaches the midcolumn. More formally, we may formulate the following property:

Property 10. For $t \geq 0$, immediately before step h_{2q+r+t} the columns C_q, $C_{q-1}, \ldots, C_{\max\{r+2, q-t\}}$ are sorted.

Proof. The proof is by induction on t (see Table 4). For $t = 0$ it follows immediately from Corollary 5. Now let us assume that the property holds for t and consider the next horizontal step. First we consider the case when $q-t > r+2$. At step h_{2q+r+t} the following columns are compared: C_{q-t} with C_{q-t+1} and C_{q-t+2} with $C_{q-t+3}, \ldots$. Thereby the columns $C_{q-t}, \ldots, C_q$ remain sorted. Additionally, by Corollary 5 column C_{q-t-1} is sorted after step h_{2q+r+t}. Note that the vertical steps (if any) executed between steps h_{2q+r+t} and $h_{2q+r+t+1}$ do not change the sorted columns. So before step $h_{2q+r+t+1}$ the columns C_q, $C_{q-1}, \ldots, C_{q-t-1}$ are sorted.

Now assume that $\max\{r+2, q-t\} = r+2$. If h_{2q+r+t} is a step H_1, then C_{r+2} is compared with C_{r+1}. But by Property 4 column C_{r+1} is sorted at this moment. Hence C_{r+2} remains sorted. During this step the columns $C_{r+3}, \ldots, C_q$ are compared between themselves, so they remain sorted. If h_{2q+r+t} is a step H_2, then $C_{r+2}, \ldots, C_q$ are compared between themselves and therefore remain sorted, too. □

	$C_7 = C_{r+1}$	$C_8 = C_{r+2}$	C_9	C_{10}	C_{11}	$C_{12} = C_q$
$t = 0$		S		S		S
$t = 1$	S		S		S	S
$t = 2$		S		S	S	S
$t = 3$	S		S	S	S	S
$t = 4$		S	S	S	S	S
$t = 5$	S	S	S	S	S	S
$t = 6$		S	S	S	S	S

Table 4. Forming the sorted sector for $q = \log_3 p = 12$.

By Property 10, before the step h_{3q-2} the sorted sector reaches C_{r+2}, and afterwards the columns C_{r+2}, $C_{r+3}, \ldots, C_q$ remain sorted. No further growth of the sorted region can be guaranteed using the same method, since some moving columns may be unsorted till they reach the midcolumn.

4.3.3. Positions of high and low sorted columns

The sorted columns may differ in the number of ones by at most 1. Let the sorted columns containing more ones be called *H-columns* and the sorted columns containing less ones be called *L-columns.* (“*H*” stands for *high* and “*L*” for *low.*)

For the rest of this section we assume that the number of 1’s in the H-columns is even. (The other case is analogous and we skip it.) By the assumption, the final destination of the H-columns is on the left part of M.

Property 11. If there are two adjacent L-columns in the sorted sector, then no column on the right side of them is an H-column.

Proof. The proof is by induction on the size of the sorted sector. The property obviously holds, if the sorted sector consists of only one column. Let the property hold for the sorted sector consisting of k columns $C_{q-k+1}, \dots, C_q$. Consider a horizontal step h_t during which a new moving column Y joins the sorted sector enlarging it to $k+1$ columns. Consider two columns C_g and C_{g+1} ($g \geq q-k$) that are L-columns inside the sorted sector immediately after step h_t. If C_g and C_{g+1} have been compared at step h_t, then they have been L-columns already before step h_t. Therefore, by the induction hypothesis, all columns C_j for $j > g+1$ have been L-columns. Of course, this remains true after step h_t. The second case is that C_g and C_{g+1} have not been compared at step h_t. Then either $g+1 = q$ (and in this case the claimed property obviously holds), or C_{g+1} and C_{g+2} have been compared at step h_t. Since after this comparison C_{g+1} is an L-column, C_{g+2} must be an L-column, too. Using the induction hypothesis for C_{g+1} and C_{g+2} we conclude that no column on the right side of C_{g+2} is an H-column. □

4.3.4. Filling the left half of M with sorted columns

The following property may be proved similarly as Property 9:

Property 12. For $t \geq 0$, immediately before horizontal step h_{3q-2+t}, the following columns are always sorted:

$$\begin{array}{rll} C_{\max\{1,r+2-t\}}, \dots, C_{r-1}, C_{r+1}, & \text{if} & t \text{ is odd,} \\ C_{\max\{2,r+2-t\}}, \dots, C_r, C_{r+2}, & \text{if} & t \text{ is even.} \end{array}$$

In particular, for $t \geq 1$, immediately before horizontal step $h_{3q-2+r+t}$, the following columns are always sorted:

$$\begin{array}{rll} C_1, \dots, C_{r-1}, C_{r+1}, & \text{if} & t \text{ is odd,} \\ C_2, \dots, C_r, C_{r+2}, & \text{if} & t \text{ is even.} \end{array}$$

Now we consider the situation after step h_{3q-2+r}. The following property may be derived analogously as Property 10:

Property 13. For $t \geq 1$ immediately before step $h_{3q-2+r+t}$, the columns C_1, C_2 $\dots$, C_t are sorted. In particular, after step h_{4q} all columns are sorted.

4.3.5. Relocating the sorted columns

Our aim in this subsection is to show that after step h_{4q} either the left half of M contains only L-columns or the right half of M contains only H-columns. Thereby, relocating the columns can be restricted to the region of $q/2$ columns. Therefore we need the following observation:

Property 14. Suppose that column C_{r+2} is an H-column before step $h_{3q-1+2t}$ for each $0 \le t \le r$. Then for each t, $0 \le t \le r/2$, the following columns are H-columns:

$$\begin{array}{rll} C_{r+2-2t}, \ldots, C_r, C_{r+2}, & \text{immediately before} & h_{3q-1+2t}, \\ C_{r+1-2t}, \ldots, C_{r-1}, C_{r+1}, & \text{immediately before} & h_{3q+2t}. \end{array}$$

For each t, $r/2 \le t \le r$ the following columns are H-columns:

$$\begin{array}{rll} C_2, \ldots, C_r, C_{r+2}, & \text{immediately before} & h_{3q-1+2t}, \\ C_1, \ldots, C_{r-1}, C_{r+1}, & \text{immediately before} & h_{3q+2t}. \end{array}$$

Proof. The proof is by induction on t. For $t = 0$ the first claim holds by the assumption on C_{r+2}. For the second part recall that by Corollary 4 column C_{r+1} is sorted before step h_{3q-1}. During step h_{3q-1} column C_{r+1} is compared with the H-column stored in C_{r+2} and by Lemma 1(c) the H-column moves to C_{r+1}. So the second claim holds for $t = 0$.

Now let $0 < t \le r/2$ and the property hold for t. So the columns C_{r+1-2t}, $\ldots$, C_{r-1}, C_{r+1}, are all H-columns before step h_{3q+2t}. At step h_{3q+2t}, which is a step H_2, these H-columns are compared with the columns preceding them. Each such H-column is compared with a O/S-column. If this is an O-column, then the H-column moves to the left, by Lemma 1(a). If in front of the H-column there is a sorted column, then either both columns are H-columns (and it remains so), or the H-column moves to the left (cf. Lemma 1(c).) Additionally, by our assumption, C_{r+2} is an H-column. Thus the first claim holds for $t+1$. The second claim can be checked in a similar way.

The above proof works for $r/2 \le t \le r$ after slight changes. □

Now we consider the situation after the step h_{3q+r}. Since the H-columns have the tendency to go to the left, they build a region of H-columns on the left side. More formally, we get:

Property 15. Suppose that column C_{r+2} is an H-column before step $h_{3q-1+2t}$ for each $0 \le t \le r$. Then for each $1 \le s \le r+1$ the columns C_1, C_2, $\ldots$, C_s are H-columns immediately before step $h_{3q+r-1+s}$.
In particular, all the columns C_1, $\ldots$, C_{r+1} are H-columns after step h_{4q}.

Proof. The proof is by induction on s. For $s = 1$ this follows from Property 14. Now let us assume that the induction hypothesis holds for s. At step $h_{3q+r-1+s}$ the H-columns C_1, C_2, $\ldots$, C_s are compared between themselves, so they remain on their places. C_{s+1} is compared with C_{s+2}, which is an H-column by Property 14. By Property 13, C_{s+1} is sorted. Thus, after comparing C_{s+1} with C_{s+2}, column C_{s+1} is an H-column. The vertical steps (if any) executed between steps $h_{3q+r-1+s}$ and h_{3q+r+s} do not change anything in the sorted columns. □

Now we are ready to prove our key lemma:

Lemma 5. *After step h_{4q} either the columns $C_{r+2}, \ldots, C_q$ are all L-columns, or the columns $C_1, \ldots, C_{r+1}$ are all H-columns.*

Proof. We consider two cases: The first case is when for each $0 \le t \le r$ immediately before step $h_{3q-1+2t}$ the column C_{r+2} is an H-column. Then Property 15 says that the columns $C_1, \ldots, C_{r+1}$ are H-columns in this case.

The other case is that for some t, $0 \le t \le r$, immediately before step $h_{3q-1+2t}$ the column C_{r+2} is an L-column. In this case C_{r+3} is also an L-column, since it has been compared with C_{r+2} at step $h_{3q-2+2t}$. The columns C_{r+2}, C_{r+3} belong at this moment to the sorted sector, so by Property 11 all columns on the right side of C_{r+2} are L-columns. This property is preserved until the end of the computation. Indeed, these columns are compared with the rest of M only by comparing C_{r+2} and C_{r+1}. Since C_{r+1} contains a sorted column when compared with C_{r+2}, an L-column must be left in C_{r+2}. Hence the L-columns remain in the right half of M. □

Since in each case either the left half of M contains only H-columns, or the right half of M contains only L-columns, after step h_{4q} the columns have to be relocated only in one half of M and r horizontal steps suffice for this purpose.

We conclude that MERGE4 requires $4.5q$ horizontal steps, that is $9q$ steps. Since $2n = p \cdot q$ and $q = \log_3 p$, it follows that $q \le \log_3 n$. Therefore MERGE4 requires at most $9 \log_3 n \approx 5.67 \log n$ steps. This concludes the proof of Theorem 2.

4.4. Bad Inputs

The following example shows that the upper bound for the runtime of MERGE4 is exact:

Example 2. Let $y = q - 3 + 3^{q-2} + \sum_{i=0}^{q-3} 3^i$. We consider an input consisting of two sorted sequences such that the first sequence has $(y-2)q + r + 5$ ones and the second sequence has $q - (r+4)$ ones. Then after loading this input to M the column C_{r+5} has the form $11(01)^{y-2}0^*$ and each of the remaining columns contains $y - 1$ ones. Then MERGE4 requires $9q - \Theta(1)$ steps.

In order to determine the runtime we follow the moving column that starts at C_{r+5}. Let x denote the size of the 01-region of this column. So initially $x = 2(y-2)$. First the moving column goes to the right border, then back to the left border and then again to the right till it reaches column C_{r+3}. Each step A reduces x by 2. Each step J decreases x so that one term of the sum $\sum_{i=0}^{q-3} 3^i \cdot 2$ disappears. Additionally x is reduced by 2 at the borders, so finally the moving column arrives at C_{r+3} with $x = 2 \cdot (3^{q-2} - 4)$ immediately before step J. At this moment the "main sorting phase" begins. In C_{r+5} the size of the 01-region is reduced to 6. Afterwards, while moving to the right no improvement is achieved inside this column. Indeed, all jumps have too large size and the moving column has always the proper status, so steps A have no effect as well. At the right border, due to the change of the status at step H_2, x is reduced by 2. On the way to the left border x remain unchanged for the same reason. At the left border, x is reduced by 2. While going to the right again nothing happens till finally in C_{r+1} the 01-region disappears and the column becomes sorted. Before the correct output is finally generated, the excessive 1 in C_{r+1} must go to a border position. Together $9q - \Theta(1)$ steps are performed. □

5. Networks with periods larger than 4 (sketch)

In this section we describe a network of width q, height $p = (2^{k-1} \cdot 3)^{q-1} \cdot 2^k$, and $2n = p \cdot q$ nodes. The idea to speed up the algorithm of Theorem 2 is to modify MERGE4 by performing more than one jump step at one location, i.e. before applying the next horizontal step. Now a round consists of the steps

$$A, H_1, H_2, J_1, J_2, \ldots, J_k.$$

Hence at one position a moving column is subject to k jump steps. As already seen, sometimes a jump may change the status of a column. Therefore we perform step A and thereby recover the proper status of each column immediately before executing horizontal steps. So the moving columns travel through the network as in the case of MERGE4.

Together we have to perform jumps that reduce the size of the 01-region to 0. For determining the sizes of the jumps we combine the ideas of algorithms MERGE3 and MERGE4. As for MERGE4, the crucial part of the computation for sorting a moving column, let call it $\mathcal{K}$, starts when it reaches the midcolumn. Then at $C_{\mathrm{mid}(1)}$ column $\mathcal{K}$ is subject to steps $J_1, \ldots, J_k$ performing (at this column) jumps of size

$$\frac{p}{2}+1, \quad \frac{p}{4}+1, \quad \ldots, \quad \frac{p}{2^{k-1}}+1, \quad \frac{p}{2^{k-1}\cdot 3}+1.$$

By Corollary 3, the size of 01-region of $\mathcal{K}$ after performing J_i, for $i \leq k-1$, is at most $p/2^i$ and it preserves its status as an O/S-column or an E/S-column. By Lemma 4, step J_k performs a jump that reduces the size of 01-region of $\mathcal{K}$ to at most $p/(2^{k-1} \cdot 3)$. Note that at this moment $\mathcal{K}$ may loose its previous status (see Lemma 2). However, this is not a problem since the next step is the ordering step A and the proper status of $\mathcal{K}$ will be recovered.

More generally for each $i \leq q$ the construction guarantees that $\mathcal{K}$ arrives at $C_{\mathrm{mid}(i)}$ with a 01-region of size less than p/a_{i-1}. At $C_{\mathrm{mid}(1)}$ the steps $J_1, \ldots, J_k$ perform jumps of sizes, respectively

$$\frac{p}{a_{i-1}\cdot 2}+1, \quad \frac{p}{a_{i-1}\cdot 2^2}+1, \quad \ldots, \quad \frac{p}{a_{i-1}\cdot 2^{k-1}}+1, \quad \frac{p}{a_{i-1}\cdot 2^{k-1}\cdot 3}+1.$$

After these steps the size of the 01-region of $\mathcal{K}$ is at most $p/(a_{i-1} \cdot 2^{k-1} \cdot 3)$. This yields $a_i = a_{i-1} \cdot 2^{k-1} \cdot 3$. The only exception are the column steps executed in $C_{\mathrm{mid}(q)}$. These are the jump steps with sizes, respectively,

$$\frac{p}{a_{q-1}\cdot 2}+1, \quad \frac{p}{a_{q-1}\cdot 2^2}+1, \quad \ldots, \quad \frac{p}{a_{q-1}\cdot 2^{k-1}}+1,$$

and one ordering step, namely Ord_E.

Recall that $p = (2^{k-1} \cdot 3)^{q-1} \cdot 2^k$. So when $\mathcal{K}$ finally arrives at $C_{\mathrm{mid}(q)}$ and $k-1$ jump steps are performed on $\mathcal{K}$, then the size of the 01-region is bounded by 2. Notice that $\mathcal{K}$ is still an O/S-column, so performing a single step Ord_E on this column makes it sorted.

Just like in the case of MERGE4 the present algorithm requires $4.5q$ horizontal steps so its runtime equals

$$4.5 \cdot q \cdot \frac{k+3}{2} \leq 2.25 \cdot (\frac{k+3}{k-1+\log 3}) \cdot \log n.$$

This concludes the proof of Theorem 3. □

Acknowledgment

Some elements of our construction (jump comparators) have been considered earlier by Grzegorz Stachowiak in the context of sorting networks. The second author would like to thank Friedhelm Meyer auf der Heide for inviting him to Paderborn, where this research has started.

10

1. Kenneth E. Batcher. Sorting networks and their applications. In *AFIPS Conf. Proc. 32*, pages 307–314, 1968.
2. Ronald I. Becker, David Nassimi, and Yehoshua Perl. The new class of g-chain periodic sorters. In *Proc. 5th ACM-SPAA*, pages 356–364, 1993.
3. Gianfranco Bilardi. Merging and sorting networks with the topology of the Omega network. *IEEE Transactions on Computers*, 38:1396–1403, 1989.
4. Allan Borodin and John E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *J. Comput. Syst. Sci.*, 30(1):130–145, 1985.
5. Martin Dowd, Yehoshua Perl, Larry Rudolph, and Michael Saks. The periodic balanced sorting network. *Journal of the ACM*, 36:738–757, 1989.
6. Marcin Kik, Mirosław Kutyłowski, and Grzegorz Stachowiak. Periodic constant depth sorting networks. In *Proc. 11th STACS*, pages 201–212, 1994.
7. Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.
8. Josef Krammer. *Lösung von Datentransportproblemen in integrierten Schaltungen.* Dissertation, TU München, 1991.
9. Mirosław Kutyłowski, Krzysztof Loryś, Brigitte Oesterdiekhoff, and Rolf Wanka. Fast and feasible periodic sorting networks of constant depth. In *Proc. 35th IEEE-FOCS*, pages 369–380, 1994.
10. G. K. Manacher, T. D. Tui, and T. Mai. Optimum combinations of sorting and merging. *Journal of the ACM*, 36(2):290–334, 1989.
11. Peter Bro Miltersen, Mike Paterson, and Jun Tarui. The asymptotic complexity of merging networks. In *Proc. 33rd IEEE-FOCS*, pages 236–246, 1992.
12. David Nassimi, Yehoshua Perl, and Ronald I. Becker. The generalized class of g-chain periodic sorting networks. In *Proc. 8th IPPS*, pages 424–432, 1994.
13. Brigitte Oesterdiekhoff. On the minimal period of fast periodic sorting networks. Technical Report TR-RI-95-167, Universität-GH Paderborn, Heinz-Nixdorf-Institut, 1995.
14. Yehoshua Perl. Better understanding of Batcher's merging networks. *Discrete Applied Mathematics*, 25:257–271, 1989.
15. Uwe Schwiegelshohn. A shortperiodic two-dimensional systolic sorting algorithm. In *International Conference on Systolic Arrays*, pages 257–264, 1988.
16. Robert Sedgewick. Data movement in odd-even merging. *SIAM J. Comput.*, 7:239–272, 1978.

This article was processed by the author using the LaTeX style file *cljour1* from Springer-Verlag.