Correction Networks

Marcin Kik † Miroslaw Kutyłowski ‡ Marek Piotrów †

†Institute of Computer Science, Wrocław University, Poland
‡Department of Mathematics and Computer Science, Poznań University, Poland
{kik,mirekk,mpi}@tcs.uni.wroc.pl

Abstract

We consider the problem of sorting sequences obtained from a sorted sequence of \( n \) keys by changing the values of at most \( k \) keys at some unknown positions. Since even for \( k = 1 \) a lower bound \( \Omega(\log n) \) on the number of parallel comparison steps applies, any comparator network solving this problem cannot be asymptotically faster than the AKS sorting network. We design a comparator network which sorts the sequences considered for a large range of \( k \)’s, has a simple architecture and achieves a runtime \( c \cdot \log n \), for a small constant \( c \). We present such networks of depth \( 4 \log n + O(\log^2 n \log \log n) \) with a small constant hidden behind the big “Oh”. In particular, for \( k = o(2^{\sqrt{\log n / \log \log n}}) \) the networks are of depth \( 4 \log n + o(\log n) \).

1. Introduction

Sorting is one of the most fundamental problems considered in computer science. Despite research lasting over decades, substantial improvements are still reported. Due to new application areas, such as packet handling in communication networks, it is necessary to design methods designed for specific practical settings. Particularly interesting are methods falling into category of parallel processing, especially those to be performed by hardware.

We consider the problem of sorting strings of \( n \) keys obtained from sorted strings by changing at most \( k \) keys at unknown positions – we call such sequences \( k \)-disturbed. Thus we would like to reorder some dynamic data which is kept in a sorted string, but may be changed at random. This is a typical situation in many application areas. Since there is no satisfactory dynamic solution, a typical approach is to reorder the data completely after some period of time.

If the changes are made explicitly and the keys are stored in a data structure such as dictionary, there are many methods to cope with this problem, even in a parallel setting. We may delete a key that has been changed and reinsert it into the dictionary (for a starting point for an overview see [3]). These methods, however, fall into a category of “software” solutions.

In this paper we consider algorithms working on comparator networks. This is a simple setting that can be easily implemented in VLSI circuits. Since the lower bound \( \log n \) applies even for \( k = 1 \), we are interested in solutions that are of depth \( O(\log n) \) for small constants hidden by the big “Oh”. Theoretically, we can sort \( n \) keys in time \( O(\log n) \) [1], however the constants hidden behind the big “Oh” are so large that the method of choice for a practical use are Batcher networks [2] of depth \( O(\log^2 n) \). For 1-disturbed strings, Schimmerl and Starke [11] present an elegant network of depth \( 2 \log n - 1 \). For higher values of \( k \), we may pipeline their networks, but this yields solutions of depth higher than the depth of the networks we propose.

Some widely known sorting networks have certain features that makes them relatively efficient in sorting \( k \)-disturbed strings. For instance, the Shearsort algorithm [10] designed for mesh architecture can be adopted to handle this case: the sorting time is reduced to \( O((\sqrt{n} + k) \log k) \) from \( O(\sqrt{n} \log n) \). Other promising direction of research in this area is to make use of periodic networks, especially of a small period (see [5] for fast sorting networks of a constant period). Since the steps of the algorithm repeat periodically, there is a chance that these networks may perform well in a dynamic systems, where some number of keys are changed at every moment.

Some related issues are considered for sorting on faulty comparator networks. A common strategy in this case is to consider a sorting network combined with some correcting network that puts misplaced items into the right places (these misplacements result from the faults in the network). However, the sorting and correcting parts cannot be separated, since we do not know where the faults occur – in particular, they may take place in the “correcting part” of the network. In this context, Piotrów [8] presented a depth-
optimal, k-fault tolerant correction network that corrects k
items in time \(O(\log n + k)\) (see also [7]).

In this paper we improve the result of Schimmler and
Starke [11]. For different ranges of \(k\), we give two con-
structions, each of depth \(O(\log n + \log^2 k \log \log n)\):

**Theorem 1.1** Let \(n\) and \(k\) be arbitrary positive integers
such that \(k \leq n\). Then there is an explicit construction of
a comparator network of depth \(4\log n + O(\log^3 k \log \log n)\)
that sorts any \(k\)-disturbed input sequence.

Note for \(k = o\left(\frac{\sqrt{\log n}}{\log \log n}\right)\), the depth of the network
is \(4\log n + o(\log n)\).

### 2. Preliminaries

A comparator network may be described as a set of reg-
isters \(V = \{R_1, \ldots, R_n\}\), where each register may hold ex-
actly one key. The input is stored in the registers, the
input key in register \(R_1\). The output is the sequence of keys
stored in the registers \(R_1, \ldots, R_n\). The registers are con-
ected by comparators. A comparator \((R_i, R_j)\) between reg-
isters \(R_i\) and \(R_j\), \(i < j\), compares the keys stored currently in
\(R_i\) and \(R_j\), moves the bigger key to \(R_j\) and the smaller one
into \(R_i\) (thus, we consider only standard networks, another
variant would be to move the bigger key into the register
with the smaller index). In a comparator network, the com-
parators are grouped into layers so that within one layer no
register is an endpoint of more than one comparator. The
layers of the network are applied consecutively; the num-
ber of layers, called depth of the network, describes paral-
lel sorting time. Usually, we represent a comparator net-
work by listing its layers: \((L_1, L_2, \ldots, L_d)\). Often we pre-
cede this list by the number of registers and the network
depth, that is, an \(n\)-register network of depth \(d\) is denoted by
\((n, d, (L_1, L_2, \ldots, L_d))\). If \(s\) denotes a sequence stored in reg-
isters \(R_1, \ldots, R_n\) immediately before applying layer \(L\), then
\(L(s)\) denotes the string stored in the registers after applying
\(L\).

By a 0-1-sequence we mean a sequence consisting of ze-
roes and ones. We may confine ourselves to 0-1-sequences,
since the well known 0-1 Principle [6] can be easily re-
proved for \(k\)-disturbed sequences:

**Lemma 2.1** A comparator network \(N\) sorts all \(k\)-disturbed
sequences if and only if \(N\) sorts all \(k\)-disturbed 0-1-sequence.

Let us note another simple property:

**Lemma 2.2** For any \(k\)-disturbed 0-1-sequence \(s\) of length \(n\)
consisting of \(x\) zeroes and \(n-x\) ones:

- there are at most \(k\) ones on positions 1 through \(x\), and
- there are at most \(k\) zeroes on positions \(x+1\) through \(n\).

Let \(s\) be a 0-1-sequence of length \(n\) and \(x\) zeroes given
as the input of the comparator network. We call the first \(x\)
registers a zero area, and the remaining registers a one area.
We call all the ones that are out of the one area and all the
zeroes that out of the zero area displaced elements. Note
that Lemma 2.2 says that any \(k\)-disturbed 0-1-sequence has
at most \(k\) displaced ones and at most \(k\) displaced zeroes.

### 3. Auxiliary networks

In this section we present some basic constructions used
in the rest of the paper.

**Networks \(I_1^n\) and \(I_0^n\)** For any \(n = 2^m\), it is easy to construct
a network \(I_1^n\) of depth \(2m-1\) that sorts any 0-1 sequence
obtained from some sorted 0-1 sequence by replacing one 0
by a 1. Its definition is recursive: \(I_1^1 = (2, 1, (\{(R_1, R_2)\})\).
For \(n = 2^m, m \geq 1\), if \(I_1^m = (n, 2m-1, (L_1, L_2, \ldots, L_{2m-1}))\),
then we \(I_1^{2n} = (2n, 2m+1, (L_1', L_2', \ldots, L_{2m+1}'))\) where:

- \(L_1' = \{(R_{2i}, R_{2i+1}) \mid 1 \leq i \leq n-1\}\),
- \(L_{2i+1}' = \{(R_{2i-1}, R_{2i}) \mid 1 \leq i \leq n\}\),
- \(L_t' = \{(R_{2i-1}, R_{2j-1}) \mid (R_i, R_j) \in L_{t-1}\} \text{ for } t \in [2, 2m]\)

So for \(2 \leq j \leq 2m\), all comparators in the layers \(L_j\) are
incident to odd numbered registers only and the network
\(I_1^{2n}\) without the first and the last layers and even registers is
isomorphic to \(I_1^n\). So, by a simple induction we get:

**Lemma 3.1** Let \(k \leq n\). Let \(w\) be a 0-1 input sequence such
that on the first \(k\) positions there is exactly one 1. Then
network \(I_1^n\) applied to \(w\) outputs a sequence with no 1 on
positions 1 through \(k-1\).

Lemma 3.1 shows that \(I_1^n\) sorts an input obtained by
changing a zero into a one in a sorted 0-1 sequence. Let \(I_0^n\)
be a network dual to \(I_1^n\), i.e. \(I_0^n = (n, 2m-1, (L_1', L_2', \ldots, L_{2m-1}'))\) with \(L_t' = \{(R_i, R_j) \mid (R_{n-j+1}, R_{n-i+1}) \in L_t\}\), where \(L_t\) denotes layer \(t\) of \(I_0^n\).

The above simple construction is a basis of a (quite
tricky) network \(S_n\) of Schimmler and Starke [11]. The
network \(S_n\) is a standard comparator network of depth
\(2\lceil\log n\rceil - 1\) that sorts any 0-1 sequence containing one
displaced one and one displaced zero (so it replaces \(I_0^n\) and \(I_1^n\)
but needs no more layers than any of them).

**Bucketing \(S_n\)** Given a network \(M = (n, d, L)\), we define
the \(k\)-merge version of \(M\) as the network \(M_k^n = (kn, c_kd, L')\),
where \(c_k\) is the depth of the Batcher merging network [2, 6]
for two sequences of length \(k\), each register \(R_i\) from \(M\) is
replaced in $M_k'$ by a group of $k$ registers $R_{(i-1)k+1}, \ldots, R_{ik}$, called bucket $i$ of $M_k'$, and each comparator $(i,j)$ from the layer $L_i$ of $M$ is replaced by the Batcher merging network for buckets $i$ and $j$.

For $n,k \in \mathbb{N}$, we define the network $S_{n,k} = (nk, d_k + c_k(2\lceil \log n \rceil - 1), L)$ (where $d_k$ is depth of the Batcher sorting network for $k$ elements) as follows:

- The first $d_k$ layers of $S_{n,k}$ sort each of its buckets of size $k$ using Batcher’s algorithm.
- The remaining layers form the $k$-merge version of $S_n$.

**Theorem 3.1** $S_{n,k}$ sorts any $k$-disturbed sequence of length $nk$.

Theorem 3.1 seems to follow directly from the correctness properties of $S_n$. The idea is that each bucket contains enough place to store all displaced ones (zeroes), so they cannot block themselves while going to the final destinations. This seems to be a correct argument, but there is a gap in it. The problem is that in $S_{n,k}$ the displaced elements need not to follow the same routes between the buckets as between the registers in $S_n$. May be, they make some shortcuts. This seems to make no harm (and even accelerate sorting). In general this intuition is wrong. Perhaps the most spectacular known case of this phenomenon is two-dimensional Bubble Sort on a mesh. Its runtime reduces dramatically, if we remove certain comparators $(9, 5)$.

**Proof of Theorem 3.1.** Let $a = (a_1, \ldots, a_{nk})$ be a $k$-disturbed 0-1 sequence with $x$ zeroes. Let $a' = (a'_1, \ldots, a'_{nk})$ be a sequence obtained after sorting the buckets within the first $d_k$ layers of $S_{n,k}$. Let $x' = \lceil x/k \rceil$. Thus bucket $x'$ is the last one that intersects the zero area.

For $1 \leq v \leq y < w \leq n$, let $\gamma_{v,w,y}$ denote a sequence obtained from the sorted 0-1 sequence with exactly $y$ zeroes by changing a zero on position $v$ into a one and a one on position $w$ into a zero. Let $d_{v,w,y}$ denote the minimal $d$ such that after applying the first $d$ layers of $S_n$ on input $\gamma_{v,w,y}$ we get a sorted sequence. Note that layer $d$ is the only layer within which the displaced zero is compared with the displaced one.

We consider all displaced zeroes in $a'$ in buckets $x' + 1, \ldots, n$. We show that $S_{n,k}$ gets rid of displaced zeroes in these buckets. In the same way, we may show that $S_{n,k}$ gets rid of displaced ones in buckets $1, \ldots, x' - 1$. Since $S_{n,k}$ outputs bucket $x'$ in a sorted state, it follows that the whole output is sorted.

Let $m$ be the number of (displaced) zeroes in buckets $x' + 1, \ldots, n$ in $a'$ and let $W$ denote the set of their positions. Let $l$ denote the number of ones in buckets $1$ through $x'$ in $a'$ and let $V$ be the set of their positions (some of these ones are displaced, those from bucket $x'$ are not necessarily displaced). Obviously, $m \leq l$ and $m \leq k$. For each $j \in W$, we choose an $i \in V$ using the following inductive procedure:

- $V_0 = V$ and $W_0 = W$.
- For each $t, 1 \leq t \leq m$, we choose as $(i_t, j_t)$ a pair $(v, w) \in V_t \times W_{t-1}$ that minimizes $d_{v/w}$.
- $V_t = V_{t-1} \setminus \{i_t\}$ and $W_t = W_{t-1} \setminus \{j_t\}$.

The idea is the following. A displaced zero terminates to be displaced at the moment when the bucket containing it is merged with a bucket with an index at most $x'$ and containing a one. In fact, if the second bucket contains less ones than there are zeroes in the first bucket, then some of the zeroes remain displaced in the first bucket. Our definition fixes for each displaced zero a one that may cause the zero to finish its status of an displaced element.

Let $\gamma_{v,w,y,t}$ denote the sequence obtained after applying the first $t$ layers of $S_n$ on input $\gamma_{v,w,y}$. For $1 \leq i \leq n$ and $0 \leq t \leq 2\lceil \log n \rceil - 1$, let $p_{i,t}$ denote the number of sequences among $\gamma_{v,w,y,t}$ that contain ones at position $i$. Let $\gamma_{v,w,y,t}$ denote the number of ones in bucket $i$ after applying the first $d_k + t \cdot c_k$ layers of $S_{n,k}$ to input $a$. We prove the following technical lemma (the reader may skip the proof at the first reading, since its ideas are not used in the main construction):

**Lemma 3.2** (a) If $1 \leq i \leq x'$, then $p_{i,t} \leq p'_{i,t}$.

That is, the number of ones in the bucket $i$ at moment $t$ is at least $p_{i,t}$.

(b) If $x' < i \leq n$, then $m - p_{i,t} \geq k - p'_{i,t}$.

That is, the number of zeroes in the bucket $i$ at moment $t$ is at most $m - p_{i,t}$.

Note that by Lemma 3.2(b), for $x' < i \leq n$, the output of $S_{n,k}$ contains at most $m - p_{i,t} \geq k - p'_{i,t}$ zeroes, i.e. no zero, in bucket $i$. As already noticed, this implies Theorem 3.1.

**Proof of Lemma 3.2.** The proof is by induction on $t$. The case $t = 0$ follows from the definitions. Let $t > 0$. For each register $R_i$ of $S_n$, $1 \leq i \leq n$, there are three possibilities:

1. there is no comparator incident to $R_i$ in layer $t$ of $S_n$.
2. there is a comparator $(R_j, R_i)$ in layer $t$ of $S_n$.
3. there is a comparator $(R_i, R_j)$ in layer $t$ of $S_n$.

In the first case, we have $p_{i,t} = p_{i,t-1}$ and $p'_{i,t} = p'_{i,t-1}$, so (a) and (b) follow from the induction hypothesis.

**Part (a):** $1 \leq i \leq x'$

In the second case, $p_{i,t} = p_{i,t-1} + p_{j,t-1}$ and, as always, $p_{i,t} \leq m \leq k$. In $S_{n,k}$ there is a network merging buckets $j$ and $i$ in the corresponding layers. Thus $p'_{i,t} = \min\{k, p'_{i,t-1} + p'_{j,t-1}\}$. Combining this with the induction hypothesis we get $p_{i,t} \leq p'_{i,t}$.
In the third case, there are two sub-cases: either \( j \leq x' \) or \( j > x' \). If \( j \leq x' \), then \( p_{ij} = 0 \) and hence \( p_{ij} \leq p'_{ij} \). The reason is that a one in each of the sequences \( \gamma_{ij/k}, \gamma_{j/k}, x', -1 \) can freely move to any position \( j, i < j \leq x' \).

The sub-case \( j > x' \) is more tedious. We claim that \( p_{ij} \leq \max \{0, p_{ij-1} - (m - p_{ij-1})\} \). Indeed, if \( \gamma_{ij/k}, \gamma_{j/k}, x', -1 \) contains a displaced one at position \( i \) and a displaced zero at position \( j \), then \( \gamma_{ij/k}, \gamma_{j/k}, x', -1 \) contains a zero at position \( i \). Therefore it contributes to decrease of \( p_{ij} \). So if \( p_{ij} > \max \{0, p_{ij-1} - (m - p_{ij-1})\} \), then there are two different pairs \((i, j), (i', j')\) such that \( \gamma_{ij/k}, \gamma_{j/k}, x', -1 \) contains a displaced one at position \( i \) and a displaced zero at position \( j \). Then, of course, \( d_{ij/k}, \gamma_{j/k}, x' > t \) and \( d_{ij/k}, \gamma_{j/k}, x' > t \), since we have detected displaced elements after step \( t \). On the other hand, \( d_{ij/k}, \gamma_{j/k}, x' < t \), since in the worst case the displaced zero and displaced one meet at layer \( t \). So we should have chosen a pair \((i, j)\) instead of the first of \((i, j), (i', j')\). Contradiction, so we have proved our claim. On the other hand, \( p_{ij} = \max \{0, p_{ij-1} - (k - p_{ij-1})\} \). By the induction hypothesis, \( p_{ij-1} \leq p_{ij-1} \) and \( (m - p_{ij-1}) \geq (k - p_{ij-1}) \). Combining this all we get \( p_{ij} \leq p'_{ij} \).

**Part (b):** \( x' + 1 \leq i \leq n \)

In the second case, we distinguish two sub-cases: either \( j > x' \) or \( j \leq x' \). In the first sub-case \( k - p'_{ij} = 0 \), since the total number of zeroes in buckets \( j \) and \( i \) is not greater than \( m, m \leq k \), and the corresponding merging sub-network moves all the zeroes to bucket \( j \). Hence (b) holds.

Now let \( j \leq x' \). Note that \( m - p_{ij} \geq (m - p_{ij-1}) - p_{ij-1} \), since in at most \( p_{ij-1} \) cases \( \gamma_{ij/k}, \gamma_{j/k}, x', -1 \) contains a one on position \( j \). Thus, for at most \( p_{ij-1} \) cases a zero at position \( i \) is exchanged with a one at layer \( t \). On the other hand, \( k - p'_{ij} = \max \{0, (k - p_{ij-1}) - p_{ij-1} \} \). By the induction hypothesis, \( k - p_{ij-1} \leq m - p_{ij-1} \) and \( p_{ij-1} \leq p'_{ij-1} \). Hence \( k - p'_{ij} \leq m - p_{ij} \).

In the third case \( m - p_{ij} = (m - p_{ij-1}) + (m - p_{ij-1}) \) and \( k - p'_{ij} = \min \{k, (k - p_{ij-1}) + (k - p_{ij-1})\} \). So the claim follows by the induction hypothesis. \( \blacksquare \) (Lemma 3.2 and Theorem 3.1)

### 4. Construction of network \( N_{n,k} \)

In this section we describe a construction of correction network \( N_{n,k} = (n, D, (L_1, \ldots, L_p)) \) sorting \( k \)-disturbed sequences of length \( n \) where \( 3 \leq k \leq \lceil \sqrt{k} \rceil \).

We assume that the input for \( N_{n,k} \) is a \( k \)-disturbed 0-1-sequence. Let \( x \) be the number of zeroes in this sequence.

First we arrange \( n \) registers in the matrix \( n_1 \times n_2 \) (i.e. with \( n_1 \) rows and \( n_2 \) columns) in the row-major order. The values \( n_1, n_2 \) will be specified later. The rows are numbered 1 through \( n_1 \) starting at the top of the matrix and the columns are numbered 1 through \( n_2 \) starting at the leftmost column. Note that the rows \( 1, \ldots, x' - 1 \), where \( x' = \lceil x/n_2 \rceil \), are contained in the zero area and the rows \( x' + 1, \ldots, n_1 \) are contained in the one area. The row \( x' \) may intersect both areas.

The network consists of five groups of layers performing computations called below phrases. The crucial part of the computation is Phase 4.

#### 4.1. Phase 1

For each row we apply network \( S_{n_2/k,k} \). For the case of a \( k \)-disturbed sequence, this suffices to sort each row. Note that after applying Phase 1 we have the following configuration (see Fig. 1): In the rows \( 1, \ldots, x' - 1 \) all the (displaced) ones are in the \( k \) rightmost columns, and in the rows \( x' + 1, \ldots, n_1 \) all the (displaced) zeroes are in the \( k \) leftmost columns, and each row of the matrix is sorted.

#### 4.2. Phase 2

The aim of Phase 2 is to move the displaced zeroes that are below the row \( x' + 1 \) to the leftmost column and the displaced ones that are above the row \( x' - 1 \) to the rightmost column.

We partition the sub-matrix of \( k \) rightmost (respectively, leftmost) columns into the squares of size \( k \times k \). For each right (respectively, left) square, except the lowest one, the subset consisting of the first \( k \) columns of the square and the last column of next lower square (respectively first column of the the square and the \( k \) last columns of the next lower square) is called a cluster (see Fig. 2).

During Phase 2 each cluster is sorted by a Batcher’s sorting network for input size \( k^2 \). Additionally, we sort the
parts of the squares that do not belong to any cluster: the last \( k - 1 \) columns of the uppermost left square and the first \( k - 1 \) columns of the lowest right square.

Since the total number of displaced ones is at most \( k \), each of the right clusters lying above the row \( x' \) will have all its ones in the rightmost column. There are at most two clusters that intersect the row \( x' \) (see Fig. 3), call them upper and lower cluster for a moment. Consider the first \( k - 1 \) columns of these two clusters after sorting them. This area contains at most \( k - 1 \) displaced ones. (It follows from the fact that at least one displaced one must be in the rightmost column after the first two phases.) All those \( k - 1 \) ones are placed in the the last row of the left \( k - 1 \) columns of the upper cluster and in the last row of the lower cluster that is in the zero area.

4.3. Phase 3

This phase ensures that all displaced ones above row \( x' \) are in the rightmost column, except may be some displaced ones in row \( x' - 1 \). Analogous actions are taken for zeroes. As we have seen, after Phase 2 this property might not be fulfilled in the clusters crossing the row \( x' \), which we have called lower and upper cluster.

For each (except the lowest one) of the right squares, for \( 1 \leq l \leq k - 1 \), we apply network \( I_{1l} \) on the sequence of the \( l \)th register from the last row of the square followed by the \( k - 1 \) uppermost registers from column \( k - l \) column of the next (lower) square (see Fig. 4). It is easy to see that in this way all misplaced ones from the upper cluster will be moved to row \( x' - 1 \) or lower.

4.4. Phase 4

This phase is the core of the construction. Its purpose is to move all displaced ones from the rightmost column to rows \( x' \), \( x' - 1 \) and \( x' - 2 \) (and simultaneously, the displaced zeroes from the leftmost column to rows \( x', x' + 1, x' + 2 \)). The first idea could be to apply network \( S_{n_1/k,k} \) inside the rightmost (and the leftmost) column. However, this would yield runtime approximately \( \log k \cdot (\log n_1 + \log n_2) = \log k \log n \) for Phases 1 and 4. We may perform much better using the fact that in the middle of the matrix of registers are no displaced ones in rows \( x' - 2 \) and higher. This leaves an empty room for constructing “channels” through which displaced ones may fall down.

The basic element of the construction is the network \( I_{1n_1} \). However, we apply this network for an input containing many displaced ones, and these ones may block each other on their way to the proper positions. We allude this problem by providing “second opportunities” for the blocked ones. Each time when blocking may occur, we split \( I_{1n_1} \) into two copies. The additional copy works on a column that has obtained no displaced one so far. At the moment of splitting, the displaced ones blocked by another displaced ones from the “old copy” are allowed to fall into the “new copy” into the same position that they aimed for in the “old copy”. Since there are still no displaced ones in the “new copy”, no blocking occurs. Thus for an individual displaced one, its route down is just the same as in \( I_{1n_1} \). The only difference is that from time to time a falling one has to change the column.

Now we describe in detail how to implement this idea.
For $d = 1, 2, \ldots$ we define a tree $T_d$ with edges labeled by positive integers. $T_0$ contains only the root and no edges. For $d \geq 0$, tree $T_{d+1}$ consists of two trees $T'_d$ where $T'_d$ denotes $T_d$ with all edge labels increased by one. The root of the second $T'_d$ is the root of $T_{d+1}$. The root of the first $T'_d$ is connected through an edge labeled 1 with the root of $T_d$ and is the leftmost child of the root.

Note that a node of $T_d$ may be incident to many edges with different labels. For instance, the root is incident to edges with labels 1, 2, \ldots, $d$. The following property is easy to check by induction.

**Lemma 4.1** If a node $x$ of $T_d$ is connected with its parent with an edge with label $r$, then the edges connecting $x$ with its children have labels $r+1, r+2, \ldots, d$.

Let $T'_d$ denote the subtree of $T_d$ consisting of the vertices at distance at most $r$ from the root. Let $\alpha'_d$ denote the number of vertices in $T'_d$ and $\alpha = \alpha'_d^{[\log k]}$. Now we fix the parameters $n_1, n_2$ so that $n_2 = 2 \cdot \alpha$ (and of course $n = n_1 \cdot n_2$).

In order to describe Phase 4, we label the columns of the matrix of registers by the nodes of a graph $G$ that consists of two disjoint copies of tree $T_{d+[\log n]}$, called respectively the right and the left tree. The right tree is used to move down the displaced ones while the left tree takes care of the displaced zeroes. The $n_2/2$ columns from the left half of the matrix are labeled by the nodes of the left tree. The leftmost column corresponds to the root of the left tree. Symmetrically, the $n_2/2$ columns from the right half of the matrix are labeled by the nodes of the right tree and the rightmost column is labeled by the root of the right tree.

Now we describe the next $4[\log n] = 2$ layers of the network. Let $d$ denote the number of layers of the first three phases. For $1 \leq l \leq 2 \cdot [\log n] - 1$, the layers $L_{d+2l-1}$ and $L_{d+2l}$ in the right half of the matrix of registers are defined as follows:

- Let $C$ be a column from the right tree labeled either by the root of the tree or by a node connected with its parent in $G$ through an edge with a label less than $l$, we insert comparators corresponding to the $l$th layer of $L'_d$.
  If this level contains a comparator between registers $r_1$ and $r_2$, then there is a comparator between the registers $r_1$ and $r_2$ of $C$ in layer $L_{d+2l-1}$.

- Let $C'$ be a child of $C$ in $G$ connected to $C$ by an edge of label $l$. Layer $L_{d+2l}$ contains comparators connecting columns $C$ and $C'$; if layer $l$ of $L'_d$ contains a comparator between registers $r_1$ and $r_2$, then there is a comparator between register $r_1$ of $C$ and register $r_2$ of $C'$.

In the left half, the construction is symmetric. The main property is that if layer $l$ of $L''_n$ contains comparator between registers $r_1$ and $r_2$, then for each column $C$ from the right tree with a child $C'$ connected by an edge of label $l$, there are two comparators incident to register $r_1$ of $C$ in the layers $L_{d+2l-1}$ and $L_{d+2l}$. The first of these comparators has its lower endpoint at register $r_2$ of column $C$. The second one has its lower endpoint at register $r_2$ of column $C'$. The idea
is that if a displaced one from register $r_1$ of $C$ is blocked by another displaced one from register $r_2$ of $C$ in the layer $L_{d+2l-1}$, then in the layer $L_{d+2l}$ the blocked one has a second chance of reaching row $r_2$. The second chance is given by switching to the column $C'$. Note that the column $C'$ does not contain any displaced ones (above the row $x'-1$) until the layer $L_{d+2l}$ is applied. This follows from the fact that $T$ is a tree. Therefore the one from the register $r_1$ of $C$ cannot be blocked by a displaced one (above the row $x'-1$) in the layer $L_{d+2l}$.

Note that if we had labeled the columns by the nodes of $T_2[\log n]^{-1}$ instead of $T = T_2[\log k]$, then it would follow that every displaced one falls down as in $I_{n_1}^2$ without being blocked by another displaced one. Indeed, each time a displaced one is blocked by another displaced one in the same column, on the next level the blocked one is sent into another column to the row it aimed for in the original column. The depth of the tree would be sufficient to perform all levels of $I_{n_1}^2$, if a displaced one arrives at a column with a label being a leaf of the tree, then the execution of $I_{n_1}^2$ is over. The problem is that the number of columns of $T_2[\log n]^{-1}$ is too large.

In our construction, we label the columns only by the nodes of $T_2[\log n]^{-1}$ of depth at most $[\log k]$. The reason is that in fact we do not need “second chances” for the displaced ones arriving at the columns labeled by the nodes of depth $[\log k]$. We shall see that in such columns at most one displaced one may arrive during Phase 4. Note that if columns $C$ and $C'$ are connected by an edge with label $l$, then only at layer $L_{d+2l}$ some displaced ones may enter $C'$. For each such one there is a one left in $C$ that caused blocking. By a simple induction we get the following lemma:

**Lemma 4.2** For $0 \leq m \leq [\log k]$, if column $C$ is labeled by a node of $T$ at level $m$, then there are at most $k/2^m$ (displaced) ones above the row $x'-1$ in column $C$ during Phase 4. In particular, any column labeled by a node of $T$ at level $[\log k]$ may contain at most one above row $x'-1$.

We see that after Phase 4 no displaced one is above row $x'-1$. So they are all in the rows $x', x'-1$ and $x'-2$. A similar property is obtained for displaced zeroes: they are in the rows $x', x'+1, x'+2$.

### 4.5 Phase 5

After Phase 4, above row $x'-2$ there are only zeroes, below row $x'+2$ there are only ones, and perhaps there is a dirty area with zeroes and ones in rows $x'-2$ through $x'+2$. The rows $x'-2, \ldots, x'+2$ form a $k'$-disturbed sequence, $k' \leq k$.

First we partition the rows into groups of 4 consecutive rows and in each group apply network $S_{k/4n_2/k,k}$. Since for each such a group the input of $S_{d/4n_2/k,k}$ is at most $k$-disturbed, the result will be a sorted sequence. However, since the “dirty” rows $x'-2, \ldots, x'+2$ may be inside two groups, we need to merge sorted sequences. First we merge sequences in groups $2i-1$ and $2i$, for $i = 1, 2, \ldots$, and then merge sequences in groups $2i$ and $2i+1$ for $i = 1, 2, \ldots$.

### 5. Runtime analysis

We start with an estimation of the depth of $S_{n,k}$. This network uses $nk$ registers. One can easily observe that we can apply the same construction for a number of registers which is not divisible by $k$. Indeed, let $SS_{n,k}$ denote the network $S_{m/k,k}$ with deleted registers $R_{m+1}, R_{m+2}, \ldots, R_{k+1}$ and comparators pointing to them. Obviously, $SS_{n,k}$ sorts any $k$-disturbed input sequence of $m$ items, because it emulates $S_{m/k,k}$ provided that we input $+\infty$ to registers $R_{m+1}, \ldots, R_{k+1}$.

Recall that depth($S_{n,k}$) = $d_3 + c_k \cdot (2[\log n] - 1)$, where $c_k \cdot (2[\log n] - 1)$ is the depth of Batcher’s merging network applied to two sorted sequences of length $k$ and $d_3 = c_k + 1$ is the depth of Batcher’s sorting network for $k$ inputs.

Putting these things together, we get

$$\text{depth}(SS_{n,k}) = c_k \left( \frac{1}{2}[\log k] + 2\log\left(\frac{n}{k}\right) + 1 \right) \leq 2\log n + 2\log k + 2[\log k] + 7$$

Now we are ready to estimate the depth of $N_{n,k}$. Let $D_i$, $1 \leq i \leq 5$, denotes the depth of the network constructed in Phase $i$. Then

$$D_1 = \text{depth}(SS_{n,2k}) = c_k \left( \frac{1}{2}[\log k] + 2\log\left(\frac{2k}{n}\right) + 1 \right),$$

$$D_2 = d_2^2 = c_k \cdot \log(k^2) \leq [\log k] \cdot (1 + 2[\log k]),$$

$$D_3 = \text{depth}(I_{n_1}^2) = 2[\log k] - 1,$$

$$D_4 = 2 \cdot \text{depth}(S_{n_1,k}) = 4[\log(\frac{n_1}{n})] - 2 \leq 4\log n - 4\log n_2 + 2,$$

$$D_5 = \text{depth}(SS_{4n_2,n_2}) = 2c_2d_2_\text{nr_2} = c_k \left( \frac{1}{2}[\log k] + 2\log\left(\frac{4n_2}{k}\right) + 1 \right) + 2[\log n_2] + 6.$$

After a few transformations, we get

$$D_1 + D_5 \leq (1 + [\log k])([\log k] + 4\log n_2 - 4\log k + 8) + 2[\log n_2] + 6,$$

$$D_2 + D_3 \leq 2[\log k]^2 + 3[\log k] - 1.$$

Finally,

$$\text{depth}(N_{n,k}) \leq 4\log n + 4\log n_2([\log k] + \frac{1}{2}) + O(1)$$

The only thing which is left to do is an estimation of $n_2$ — the number of columns in the register matrix of $N_{n,k}$. We
have defined \( n_2 \) as \( 2 \cdot \sum_{i=0}^{\lceil \log k \rceil} \binom{\lceil \log k \rceil}{i} \), the number of vertices of depth at most \( \lceil \log k \rceil \) in the labeled binomial tree \( T_2^{\lceil \log n \rceil - 1} \).

It is well known that there are exactly \( \binom{n}{m} \) nodes at depth \( i \) in a binomial tree \( T_m \). Thus

\[
n_2 = 2 \cdot \sum_{i=0}^{\lceil \log k \rceil} \binom{\lceil \log n \rceil}{i} - 1
\]

(3)

**Lemma 5.1** If \( n \geq 256 \) and \( 3 \leq k \leq \frac{n}{\log n}^{\frac{1}{2}} \), then \( \log n_2 \leq \log k \lceil \log n + 2 \rceil \) and \( n_2 \leq n/k \).

**Proof.** By an easy induction one can prove that \( \sum_{i=0}^{\lceil \log k \rceil} \binom{\lceil \log n \rceil}{i} \leq \binom{\lceil \log n \rceil}{m} \) for \( m \geq 2 \) and \( j \leq \frac{\pi}{2} \). In our case

\[
\log n_2 \leq \left( \frac{2}{3} - \frac{1}{3} \log \pi \lceil \log k \rceil \right)
\]

+ \( \lceil \log k \rceil \left( \log \log n + \log e(2/1 + \log n) \right) \)

Since \( \lceil \log k \rceil \geq 2 \) and \( \log n \geq 8 \), the expression in the first parentheses is bounded by 0.2 and \( \log e(2/1 + \log n) \leq 1.6 \). The first part of the lemma follows. The second one is a simple consequence of the first part and the upper bound on \( k \): \( \log k \leq 1 + \log k \leq \log n + 3 \) and \( \log n_2 \leq \lceil \log k \rceil (\log \log n + 3) - \log k \leq \log n_2 \).

Lemma 5.1 shows that the construction in Section 4 is correct: the required number of columns does not exceed the total number of registers and there are at least \( k \) rows.

**Proof Theorem 1.1.** If \( n \leq 256 \) or \( k \leq 2 \), the Batcher sorting network or pipelined networks \( S_n \) of Schimmels and Starke do the job and have depth \( 4 \log n + O(1) \). Thus we will use networks constructed in this paper only if \( n \geq 256 \) and \( k > 2 \). The network \( N_n \) is used if \( k \leq \frac{n}{\log n}^{\frac{1}{2}} \) and \( SS_{n,k} \) otherwise. In both cases, the depth of the selected network is bounded by \( 4 \log n + O(\log^2 k \log \log n) \). To prove this in the case of \( N_n \) one should apply the equation \( \log n_2 = O(\log k \log \log n) \) from Lemma 5.1 in (2):

\[
\text{depth}(N_{n,k}) = 4 \log n + O(\log n_2 \log k)
\]

\[
= 4 \log n + O(\log k \log \log n).
\]

In the case of \( SS_{n,k} \), \( k \geq \frac{n}{\log n}^{\frac{1}{2}} \) and then \( \log n \leq (\log k + 1)(\log n + 2) \) and \( \log n_2 = O(\log k \log \log n) \). So by (1):

\[
\text{depth}(SS_{n,k}) = O(\log n \log k) = O(\log^2 k \log \log n).
\]

### 6. Conclusions

Our methods work for moderately large values of \( k \). However, further research is necessary for applications where the value of \( n \) is small (in this case the small “oh” terms in the expression for the depth of the network play a substantial role and prevent immediate practical applications).

Another very interesting and important area of research would be to apply techniques of correcting disturbed sequences for the case of sorting sets of keys with dynamically changing values.

### References


