ARTICLE

An 0(n log n) sorting network

MIKLÓS AJTAI

JÁNOS KOMLÓS

ENDRE SZEMERÉDI

Published: 01 December 1983

Citation in BibTeX format

Conference Sponsors:
SIGACT

An O(n log n) Sorting Network

M. Ajtai, J. Komlós, E. Szemerédi

Mathematical Institute of the Hungarian Academy of Science, Budapest
University of California, San Diego
University of South Carolina, Columbia

Abstract

The purpose of this paper is to describe a sorting network of size $O(n \log n)$ and depth $O(\log n)$.

A natural way of sorting is through consecutive halvings: determine the upper and lower halves of the set, proceed similarly within the halves, and so on. Unfortunately, while one can halve a set using only $O(n)$ comparisons, this cannot be done in less than $\log n$ (parallel) time, and it is known that a halving network needs $(1/2)n \log n$ comparisons.

It is possible, however, to construct a network of $O(n)$ comparisons which halves in constant time with high accuracy. This procedure (ε-halving) and a derived procedure (ε-nearsort) are described below, and our sorting network will be centered around these elementary steps.

Introduction

The network described in the abstract is a comparator network, i.e. a deterministic sequence of $O(n \log n)$ switches, where a switch is a pair (i,j), $1 \leq i < j \leq n$, and operates as follows: it compares the actual contents of the ith and jth registers, and (by switching if necessary) puts the larger one into the jth register and the smaller one into the ith register. At the end of the algorithm, the ith smallest element is in the ith register for all $i = 1,2,\dots,n$; the elements are sorted.

The algorithm works in $O(\log n)$ delay time (parallel time). We should mention, however, that the constant involved in $O(.)$ is very large and is determined by the size of certain expander graphs. This makes the algorithm unsuitable for actual implementation and much slower than, for example, Batcher's algorithm for "small" values of n. (The constants can drastically be reduced by generating random expander graphs, rather than using the known constructions.) No attempt is made to get best possible constants.

An earlier version of this algorithm has been submitted to Combinatorica. Although the main ideas are all contained in that paper, the two formal descriptions are so much different that we think it is worth presenting this second version.

We start with a short, informal description of our algorithm. This was provided by Joel Spencer, and we are extremely grateful to him for it. It prompted us to write this paper without further delay.

Spencer's outline will be followed by a brief review of expander graphs. The remainder of the paper will fill in the details of the informal description.

Sections:

Supported in part by the National Science Foundation under grant MCS-7906228.

1. Spencer's description

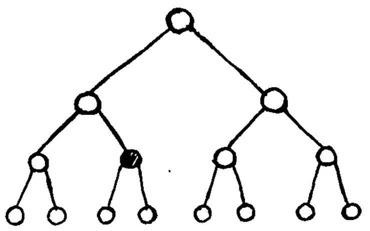
There are three parts: ε-halving, ε-nearsort, and The Algorithm. Fortunately for each part we need only the results of the previous parts.

ε-halving. Here ε is fixed, say $\varepsilon = 10^{-9}$. In fixed finite time (one time equals n/2 comparisons) an ε-halver on m registers will put the lower half of the numbers in the lower half of the registers with at most εm errors and for all k, $1 \leq k \leq m/2$ the first k numbers are in the lower half of the registers with at most εk errors (and similarly for the top k numbers). These ε-halvers are intimately connected to Expander Graphs, etc. As I think of it, each time unit takes a random 1-factor between the LH and the UH of the registers and makes a probabilistic computation.

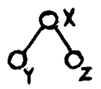
ε-nearsort. Here ε is fixed, say $\varepsilon = 10^{-6}$. In fixed finite time an ε-nearsort will put contents 1,...,m into registers 1,...,m so that for any interval I of registers the proper numbers are in I (though not necessarily in order) with at most εm mistakes and for all k, $1 \leq k \leq \varepsilon m$ the first k numbers are in the lower εm registers with at most εk mistakes. To construct a nearsort apply an ε-halver ($\varepsilon = 10^{-15}$) to the whole set of registers, then apply ε-halvers to the top and bottom half of registers, then to each quarter, eighth, sixteenth, until the pieces have size $m10^{-9}$. One can show quickly that each piece of size $w = m10^{-9}$ has at most εw errors. The fringes are small since $10^{-6} \gg 10^{-9}$ and this part of the argument is not too hard...

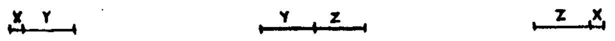
The Algorithm. In one sense the algorithm is simple to describe. The n registers are bolted down and numbered 1 through n. For each i, $0 \leq i \leq 3\log n$, a partition of the registers is given. These partitions are explicitly given and are completely independent of the contents of the registers. For a given i we perform an ε-nearsort separately (and simultaneously) on each set in the i^{th} partition.

To describe the partitions a binary tree format is used. Consider a binary planar tree. At time t we distribute the registers into the nodes as follows. No registers are at a level deeper than t. To each node there corresponds a natural interval of registers — for the root all of them, for the marked node (n/4+1, 2n/4). (This interval is interpreted as an estimate for the contents of registers assigned to the node.)

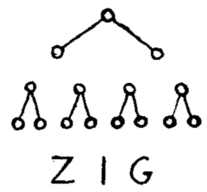


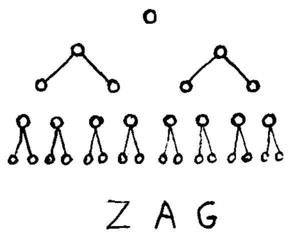
A node at level t first appears at time t and then it has the whole natural interval except that already taken by nodes above it. (In one sense the higher nodes get higher priority.) A node at level t at time t+i will have two intervals of registers — on each edge A^{-i} of its natural interval. (It will be as near to the edge as possible but the higher nodes have priority.) (A = 100 is an absolute constant. A << 1/ε.) Here is a triangle of nodes and the registers they represent. Each Y and Z interval is A = 100 times as long as the X intervals (and the gap between the intervals is 100 times as long as the intervals).





At time t we have placed (mentally) the registers in this binary tree. Partition the tree into triangles with the apexes of the nodes at even level. This is a partition of the registers, apply an ε-nearsort to every triangle of registers. That is the Zig step. Now partition the tree into triangles with the apexes of the nodes at odd level. Apply the ε-nearsort to each triangle. That is the Zag step. Go ZigZagZig. (We call the entire ZigZagZig step one time unit.)





Then time changes to t+1. We (mentally) move the registers about in the binary tree. (I think of that as the Bookkeeping step, but note that it takes no time and involves no comparisons.) Then we apply the next ZigZagZig step. We do this $\log n$ times. (We may think of all of the registers lying in the root at time 0. As time continues they filter down to the bottom of the tree, but if a register's position is near $\alpha n 2^{-s}$ it will get held up at level s as an extreme register.) Now it is claimed that after this procedure the numbers have been sorted.

To prove that this algorithm works an appropriate induction hypothesis is required. At a given time a register R with content x lies on a node N. We define the wrongness w(R). For any node N there corresponds a natural interval of registers and hence of numbers I(N). If $x \in I(N)$ then $w(R) = 0$, the register R has an appropriate number. Otherwise look at the node N_1 directly above N with interval $I(N_1)$ double that of I(N). If $x \in I(N_1)$ then $w(R) = 1$. Otherwise, w(R) is the least w so that $x \in I(N_w)$ where N_w is the node directly w steps above N. Since the root node N has I(N) equal all numbers this is well defined. For example, let R be a register in the node marked in the diagram in the previous page. Let x be its content . (We may assume the contents are also 1,...,n.) If $0 < x < n/4$ then $w(R) = 1$, if $n/4 < x < n/2$ then $w(R) = 0$, and if $n/2 < x < n$ then $w(R) = 2$ since one must go to the root before having x correctly placed.

Now for the induction hypothesis. Look at any node at any time and let r be any positive integer. The fraction of registers on that node with wrongness $\geq r$ is at most cA^{-3r}.

At the end the only nodes with registers are at the bottom and contain only one register, and so none of the registers have any wrongness and so we would be done.

The induction is on time t. A time unit consists of two parts. In the bookkeeping step wrongness is increased. The key step at time t is when at level t a node splits and sends its left and right sides to level t+1, saving the extremes for itself. The node had already had an ε-nearsort applied to it but now a fraction ε of registers that were correct (i.e. wrongness = 0) have wrongness 1. Also, all registers that had wrongness $r \geq 1$ now have wrongness r+1. This is natural since in the bookkeeping step we are claiming further refinement of the numbers and so wrongness will creep in. In the ZigZagZig step wrongness is decreased. (Actually — lets make it ZigZagZigZagZigZag since we are not worried about the constant.) Look at the XYZ triangle and consider the registers in Y with wrongness. If wrongness = 1, they should be in Z and those numbers will move into registers in X or Z and decrease wrongness. If wrongness > 1, say they should be less than I(X), then those elements form an initial segment, and thus ε-nearsort will put almost all of them into X (being on the extreme left of the X-Y-Z picture), decreasing wrongness by 1. If, say, wrongness is large then the number will be pushed up the tree by the alternate ZigZagZig step — X will be part of a bigger triangle A-X-V and it will go to apex A, etc. Note that in the X-Y-Z picture the leftmost X is only .01 of the picture but that is plenty of room since the errors are all on the order of one in a million.

2. Expander graphs

Notation. Given a graph $G = (V,E)$ and a set $A \subset V$ of vertices, we write $\Gamma(A)$ for the set of neighbors, i.e.

$$\Gamma(A) = \{v \in V \mid (v,a) \in E \quad \text{for some} \quad a \in A\}$$

Given two sets $V_1, V_2, |V_1| = |V_2| = n$, a bipartite graph G with vertex-sets V_1, V_2 is called an *expander* graph with parameters (λ,α,c) if for any set

$A \subset V_1$, $|A| \leq \alpha n$ we have $|\Gamma(A)| \geq \lambda A$

and the same holds for subsets of V_2, and yet the maximum valency in G does not exceed c (and hence G has only a linear number of edges).

The above notion is slightly different from the usual one.

The following theorem is the result of works by several authors:

Theorem. For any λ and α, $\lambda > 1$, $\alpha > 0$, $\lambda\alpha < 1$, there is a $c = c(\lambda,\alpha)$ such that there are expander graphs with parameters (λ,α,c) for all n.

Remark. The condition $|A| = |B|$ in the definition was not essential, we will also use expanders with $|A| = |B| + 1$.

Comments. The first result on the existence of expanders is due to Pinsker (1973).

The first explicit constructions were made by Margoulis (1973); and though the graphs themselves had a very simple structure, the proof of their expanding property involved group-representation theory.

After this breakthrough, Gabber and Galil (1979) contributed by giving a more direct proof (using Fourier analysis), which led to effective estimates on the extent (λ) of expanding.

(The graph involved in the Gabber-Galil result — a modification of that of Margoulis — is worth mentioning for its simplicity:

$$V_1 = V_2 = \{(i,j),\ 1 \leq i,j \leq m\}$$

and there are (at most) five edges going out from each vertex(i,j) (in V_1, say), namely to (i,j), (i,i+j), (i+j,j), (i,i+j+1), (i+j+1,j) (in V_2), where addition is mod m.

To understand why Margoulis used linear transformations on the torus rather than on the real line, read Klawe (1981). She proves that one-dimensional linear transformations (used for generating pseudo-random numbers) cannot work.)

For more on expanders and superconcentrators read Valiant (1975), Paul-Tarjan-Celeni (1976), Pippinger (1977), Angluin (1979).

The known constructions lead to expanders with arbitrary large λ by forming high powers of the graph. The price, however, is high too. The above mentioned construction leads to an expander with $\lambda = 2$ (and $\alpha = 1/3$, say) only with c of the order of magnitude 2^{100}. Counting arguments show, however, that there is an expander with parameters (2,1/3,8). The astronomical gap $8\text{-}2^{100}$ makes it questionable whether it is worth working with explicit constructions. (Simple counting shows that the graph consisting of c randomly chosen one-factors is, with high probability, an expander with parameters $(\lambda,1/(\lambda+1),c)$ if only

$$c > \frac{2[(\lambda+1)\log(\lambda+1)-\lambda\log\lambda]}{(\lambda-1)\log(\lambda-1)+(\lambda+1)\log(\lambda+1)-2\lambda\log\lambda}$$

$$= 2[\lambda\ln\lambda+\lambda+1/2+o(1)]$$

while the above constructions give something like λ^{100}.

Note that generating a random one-factor simply needs generating a random permutation.)

3. ε-halving

A permutation $\pi = (\pi1,\pi2,\ldots,\pi m)$ of $(1,2,\ldots,m)$ is said to be *ε-halved* if for any *initial segment* $S = (1,\ldots,k)$, $k \leq \lfloor m/2 \rfloor$,

$\pi i > \lfloor m/2 \rfloor$ occurs for at most $\varepsilon|S|$ numbers $i \in S$,

and for any *endsegment* $S = (k,\ldots,m)$, $k > \lfloor m/2 \rfloor$,

$\pi i \leq \lfloor m/2 \rfloor$ occurs for at most $\varepsilon|S|$ numbers $i \in S$.

(In other words, the two halves have very few errors and they are mostly concentrated "in the middle.")

A procedure is called an *ε-halver* (of m elements) if it accepts permutations of $1,\ldots,m$ as inputs, and produces ε-halved permutations of $1,\ldots,m$ as outputs.

We construct now a *bounded depth ε-halver network*. Fit an even (bipartite) expander graph with parameters $((1-\varepsilon)/\varepsilon,\varepsilon,c)$ to the register sets $A = (1,\ldots,\lfloor m/2 \rfloor)$, $B = (\lfloor m/2 \rfloor+1,\ldots,m)$. Split the graph into c one-factors $F_1,\ldots,F_c$. Interpreting the edges of a one-factor as switches (performing comparisons along the edges and always putting the smaller element to A), we obtain a network of depth c. We claim it is an ε-halver.

Note first that a comparator network defined by *any* bipartite graph has the property that if $R_1 \in A$ and $R_2 \in B$ are connected by an edge then the content of R_1 is less than that of R_2. Indeed, there was a stage (when we have just compared R_1 and R_2) when this was true, and the content of R_1 decreases that of R_2 increases in time.

Assume now that for some $k \le \lfloor m/2 \rfloor$, more than εk of the elements of $S = (1,\ldots,k)$ end up in registers of B. Since more than εk vertices in B are connected to at least $(1-\varepsilon)k$ vertices in A, we must have a pair of registers $R_1 \in A$, $R_2 \in B$, such that R_2 holds an element of S, R_1 holds an element outside S, and R_1 and R_2 are connected by an edge. This, however, contradicts the above mentioned property, since any element of S is less than any element outside S (initial segment).

4. ε-nearsort

Notation. Given a permutation $\pi = (\pi 1, \pi 2, \ldots, \pi m)$ of $(1,2,\ldots,m)$ and a subset S of $[m] = \{1,2,\ldots,m\}$, we write

$$\pi S = \{\pi i \mid i \in S\}$$

Given $\varepsilon > 0, S^\varepsilon$ denotes the blown-up set

$$S^\varepsilon = \{j \in [m] \mid |j-i| \le \varepsilon m \quad \text{for some } i \in S\}$$

In particular, if $S \subset M$ is a segment $(a,a+1,\ldots,b)$ then S^ε is the segment $\{a',a'+1,\ldots,b'\}$, where

$$a' = \max(1, \lceil a-\varepsilon m \rceil), \qquad b' = \min(m, \lfloor b+\varepsilon m \rfloor)$$

We say that a permutation $\pi = (\pi 1,\ldots,\pi m)$ of $(1,\ldots,m)$ is ε-*nearsorted* if

(i) $$|S - \pi S^\varepsilon| < \varepsilon |S|$$

holds for all *initial segments* $S = (1,\ldots,k)$ and *endsegments* $S = (k,\ldots,m)$, $(1 \le k \le m)$.

A procedure is called an ε-*nearsort* (of m elements) if it accepts permutations of $1,\ldots,m$ as inputs, and produces ε-nearsorted permutations of $1,\ldots,m$ as outputs.

Remarks. (i) clearly implies that

(ii) $$|S - \pi S^\varepsilon| < 3\varepsilon m$$

holds for *all* segments $S = (a,\ldots,b)$.

Note that for "middle" segments we only have an absolute bound $3\varepsilon m$, while for initial (and end) segments (i) provides a relative bound, meaningful even for very short segments. This will enable us to obtain, by repeated applications of ε-nearsort, an exponential decay of errors.

ε-nearsort and ε-halving will usually be applied to a sequence $a_1,a_2,\ldots,a_m$, where $a_1,\ldots,a_m$ is a permutation of an ordered string $e_1 < e_2 < \ldots < e_m$. In this case we think of the elements $e_1,\ldots,e_m$ as identified by $1,\ldots,m$.

To construct an ε-nearsort network, apply an ε_1-halver (network), with $\varepsilon_1 < \frac{\varepsilon}{\log 1/\varepsilon}$, to the whole set of registers, then apply ε_1-halvers to the top and bottom half of registers, then to each quarter, eighth, sixteenth, until the pieces have size $w < \varepsilon m$. It is easy to see that the obtained network of *bounded depth* is an ε-nearsort.

5. Register assignment strategy

We are going to use a paramenter $A > 10$, whose value will be discussed in the proof section. All logarithms are to the base 2, and we assume that n and A are powers of 2.

Define the following numbers

$$X_t(i) = \lfloor cn2^{-t}A^{i-t} \rfloor, \quad Y_t(i) = \sum_{j=1}^{i} X_t(j),$$

$$t = 1,2,\ldots,\log n; \quad i = 1,2,\ldots,t,$$

where $c = 1/(2A)$, and we interpret $Y_t(i)$ as zero for $i \le 0$.

At time t, the registers assigned to the j^{th} node on the i^{th} level, $0 \le i \le t-1$, $1 \le j \le 2^i$, form two intervals J_1, J_2

$$J_1 = (j-1)n2^{-i} + [Y_t(i)+1,\ Y_t(i+1)] \quad \text{and}$$

$$J_2 = (j-1)n2^{-i} + [n2^{-i} - Y_t(i+1)+1, n2^{-i}]$$

if j is odd

$J_1 = (j-1)n2^{-i}+[1, Y_t(i+1)]$ and

$J_2 = (j-1)n2^{-i}+[n2^{-i}-Y_t(i+1)+1, n2^{-i}-Y_t(i)]$

if j is even.

The registers assigned to the j^{th} node on the t^{th} level, $1 \le j \le 2^t$, form a single interval J,

$J = (j-1)n2^{-t}+[Y_t(t)+1, n2^{-t}]$ if j is odd

$J = (j-1)n2^{-t}+[1, n2^{-t}-Y_t(t)]$ if j is even

Dynamics. As time increases from t to $t+1$, we reassign the registers to nodes. Analyzing the above formulas one sees that about $1/A$ proportion of some intervals remain unchanged, but most registers of an interval move down one or two levels (about half-half). No register moves up, and $A > 2$ ensures that no register moves down more than two levels ($Y_{t+1}(i+3) > Y_t(i+1)$). These facts will be used in the proof.

After $\log n/\log(2A)$ steps the tree splits into two trees, and then disintegrates to small trees very fast. Since there is no movement of elements between the disjoint trees, by that time the corresponding intervals of elements should be (and will be) properly ordered among each other.

6. The partitions

Although $3t+1$ partitions would suffice (three in each time-cycle — called ZigZagZig in Section 1), in order to simplify the proof we define the algorithm using $2at+1$ partitions (ZigZag is repeated a times), where $a \le 15$.

The 0^{th} partition consists of only one set $[n] = \{1,\dots,n\}$. For $t \ge 1$, define the partitions P_{ti}, $1 \le i \le 2a$, by using the register assignments at time t. The basic part — called cherry — is the set of registers assigned to a node and its two (possibly empty) children-nodes. P_{t1} consists of these parts for parent-nodes on even levels of the tree. P_{t2} has parts with parents on odd levels, P_{ti} is identical to P_{t1} or P_{t2} according to the parity of i. (When the parent-nodes are on odd levels, the nodes assigned to the root, which has no parent, form a part themselves.)

Since the registers assigned to a particular node form at most two intervals, the above parts consist of at most six intervals, but they actually melt together to at most three. Thus, this partition can be described explicitly, without referring to the tree structure, as it will be done in the next section. The tree structure shows, however, why the algorithm works:

The two intervals of a parent-node frame the register-sets of the two children nodes. When applying ε-nearsort to these registers (sitting on parent and two children), the contents that are foreign elements — too large or too small — are forced to the edges, i.e. up to the parent-node. Thus, in the bookkeeping steps the elements move down with the registers but in the ε-nearsort steps most of the misplaced elements move up on the tree, and thus they have a chance of finding the proper place in $O(\log n)$ time.

In other words, we have a correction mechanism for the errors ε committed in each step that does not let the errors accumulate.

7. A too formal description

Having the ε-nearsort algorithm one can describe the sorting network directly, without introducing the above tree structure. (The price paid is that this form is too mystic to understand.)

For a given t, we define two partitions of the registers $1,2,\dots,n$.

Define the sets $T_t(i,j)$ as follows:

For $0 \le i \le t-2$, $1 \le j \le 2^i$, $T_t(i,j)$ consists of the following three intervals (for definition of $Y_t(i)$ see Section 5).

$(j-1)n2^{-i}+[Y_t(i)+1, Y_t(i+2)]$

$(j-1)n2^{-i}+[n2^{-i-1}-Y_t(i+2)+1, n2^{-i-1}+Y_t(i+2)]$

$(j-1)n2^{-i}+[n2^{-i}-Y_t(i+2)+1, n2^{-i}]$

if j is odd, and

$(j-1)n2^{-i}+[1,Y_t(i+2)]$
$(j-1)n2^{-i}+[n2^{-i-1}-Y_t(i+2)+1,n2^{-i-1}+Y_t(i+2)]$
$(j-1)n2^{-i}+[n2^{-i}-Y_t(i+2)+1,n2^{-i}-Y_t(i)]$
if j is even.

For $i=t-1$ or t, $1\le j\le 2^i$, $T_t(i,j)$ consists of one single interval

$(j-1)n2^{-i}+[Y_t(i)+1,n2^{-i}]$ if j is odd
$(j-1)n2^{-i}+[1,n2^{-i}-Y_t(i)]$ if j is even.

We also define $T_t(-1)$ as the union of the two intervals $[1,Y_t(0)]$ and $[n-Y_t(0)+1,n]$.

Now partition P_{t1} consists of the "even" sets $T_t(2i,j)$, $i=0,1,\dots,\lfloor(t-1)/2\rfloor$, $j=1,2,\dots,2^{2i}$; while partition P_{t2} consists of $T_t(-1)$ and the "odd" sets $T_t(2i-1,j)$, $i=1,2,\dots,\lfloor t/2\rfloor$, $j=1,2,\dots,2^{2i-1}$.

Given these two partitions, the t-cycle of the algorithm consists of the application of ε-nearsort simultaneously within the parts of P_{t1} (Zig step), then the same for P_{t2} (Zag step), then repeat this alternately for P_{t1} and P_{t2}, altogether a times each.

The whole algorithm starts with an ε-nearsort applied to the whole set of registers $1,2,\dots,n$, then the t-cycles are performed for $t=1,2,\dots,\log n$.

8. Proof

Notations. Both the registers and the contents are identified by the numbers $1,\dots,n$. We write $R(i)=j$ if the content of the register i is j, and $R(I)$ denotes $\bigcup_{i\in I} R(i)$. We will suppress time t in the notations and resolve the missing notational problems verbally.

The name "interval" and the letters J,K will stand for any one of the intervals defined in Section 5 (and called J,J_1,J_2 there). The intervals J form a partition of $(1,\dots,n)$; J' will denote the next interval on the right of J. The lower section $L(J)$ consists of all intervals not exceeding J. The word "lower section" always means $L(J)$ for some interval J. The definition of upper sections is similar.

For an interval J and a lower section $L=L(K)$, $K<J$, $K'\neq J$, we define the distance $d(J,L)$ as the distance of the two nodes holding J and K on the tree. (Similar definition for upper section U.) Note that d might be 0, namely if J and K are on the same node (siblings).

We define the numbers $\Delta_r(J)$ as the proportion of elements sitting in registers of J that are misplaced with r or more. More precisely, for a given J and $r\ge 0$, find $S_1=\sup|R(J)\cap L(K)|$ over all intervals K, $K<J$, $K'\neq J$, $d(J,L(K))\ge r$, and also $S_2=\sup|R(J)\cap U(K)|$ over all intervals K, $K>J$, $K\neq J'$, $d(J,U(K))\ge r$. Then $\Delta_r(J)$ is defined by

$$\Delta_r(J) = \max(S_1,S_2)\Big/|J|$$

(when sup is taken over an empty set, we define it as 0).

We also set

$$\delta(J) = |R(J)-J|\Big/|J|$$

$$\delta = \sup_J \delta(J), \qquad \Delta_r = \sup_J \Delta_r(J), \quad r\ge 0.$$

Throughout the proof we will assume that α is small (A is large), and ε is sufficiently small in terms of α (actually, ε is a small power of α).

The idea behind the proof is that in every step of the algorithm (ε-nearsort within all parts of a partition) the elements (contents) not sitting on the right node of the tree are all moving to the right direction — except a negligible proportion.

The following theorem clearly implies that the algorithm sorts (at time $\log n$ there are no errors since $\delta<1$).

Theorem. After every completed cycle, we have

(1) $\Delta_r < \alpha^{3r+40}$, $r\ge 1$

and

(2) $\delta < \alpha^{30}$

Proof. We use induction on time t. For $t=0$, $\delta=\Delta_r=0$. Assume the theorem holds for some t and prove it for t+1. The cycle starts with changing the register assignments (pointers), thus redefining intervals, sections, Δ_r and δ. This will greatly deteriorate the achieved accurateness; the following steps of the algorithm are supposed to decrease the errors again.

Lemma 1. If Δ_r and δ are the errors before the register reassignment, then for the corresponding numbers Δ'_r and δ' after reassignment we have the estimates

$$\Delta'_r < 6A\Delta_{r-4}, \qquad r \geq 6$$

and $$\delta' < 6A(\delta+\varepsilon)$$

We show now that a Zig (or a Zag) step cannot increase the errors too much, while a combined ZigZag step must decrease them.

Lemma 2. Let Δ_r and δ be the errors before a Zig (or a Zag) step, and Δ'_r and δ' after it. If $\delta < \alpha^2$, then

$$\Delta'_r < 8A(\Delta_r+\varepsilon\Delta_{r-2}) \qquad r \geq 3$$
$$\Delta'_r < 8A(\Delta_r+\varepsilon) \qquad r=1,2$$
$$\delta' < 4A(\delta+\varepsilon)$$

Lemma 3. If $\delta < \alpha^4$, then a ZigZag step will change the errors as follows

$$\Delta'_r < 64A^2(\Delta_{r+1}+3\varepsilon\Delta_{r-4}) \qquad r \geq 5$$

$$\Delta'_r < 64A^2(\Delta_{r+1}+3\varepsilon) \qquad r=1,2,3,4$$

$$\delta' < 16A^2(\delta+2\varepsilon)$$

Starting from the inductive hypothesis, Lemma 1 and repeated applications of Lemma 3 show that after the new cycle the errors Δ_r are as small as required. However, the error δ seems to have increased through all steps. This is not so, as shown by the following lemma.

Lemma 4. After a completed cycle, we have

$$(3) \qquad \delta < 10(A\varepsilon+\sum_{r\geq 1}(4A)^r\Delta_r) < \alpha^{30}$$

It remains to prove the lemmas.

Proof of Lemma 1. A new interval J' is the union of at most three segments, each one contained in an old interval J, and $|J'| > |J|\alpha/2$ for any one of them. Such a segment moved at most 2 levels on the tree. Similarly, a new section L(K') is contained in an old section L(K), and K' is at most two levels away from K on the tree. Hence the lemma follows.

Proof of Lemma 2. We use the word near if referring to distances on the tree, and close for distances on the real line. Of the six (or less) intervals of a cherry, the closest one to but outside of a section L(K) is either the nearest one to L(K), or identical to K'. Most elements belonging to L will be forced to the left, and wander into L or to this closest interval. The number of exceptional elements cannot exceed ε proportion of the cherry, which is less than $8A\varepsilon$ proportion of any interval of the cherry. Since this closest interval is (one of the) nearest, too, the order of the error did not increase, except for at most the above $8A\varepsilon$ proportion, where it could increase at most two.

Some elements of L might have come out from L, but they simply changed place with elements already belonging to L, not changing their number.

Proof of Lemma 3. The only extra remark we need here is that if an interval J, $d(J,L) \geq 1$, was closest to L (outside L) in the Zig step, then it will not be a nearest one in the Zag step. Thus, not only errors will not increase (except a small proportion) but will strictly decrease.

Remark. We always assumed that $\delta < \alpha^2$ to ensure that the extreme interval of the cherry (which represents about $\alpha/4$ proportion of the whole cherry) can accommodate all foreign elements. But this extreme interval might be empty. Then however, the total number of registers of the cherry is less than 4A, and $4A\delta < 1$, i.e. all registers of the cherry contain their own elements ($R(i)=i$).

Proof of Lemma 4. We consider an interval J, and estimate the number $x=|R(J) \cap U(J')|$. It is certainly bounded by the number $y=|R(L(J)) \cap U(J')|$, which is of course equal to

$z = |R(U(J')) \cap L(J)|$. The equality $y = z$ implies for the numbers $x_1 = |R(J) \cap J'|$ and $y_1 = |R(J') \cap J|$ the identity

$$\begin{aligned} y_1 - x_1 &= |R(J) \cap (U(J')-J')| - |R(J') \cap (L(J)-J)| \\ &+ |R(L(J)-J) \cap U(J')| - |R(U(J')-J') \cap L(J)| \\ &= x_2 - y_2 + x_3 - y_3 \end{aligned}$$

Now we estimate the terms on the right-hand side; for reason of symmetry, it is enough to work with x_2 and x_3. Clearly, $x_2 \leq \Delta_1 |J|$, the hard part is estimating x_3.

Let us partition $L(J)-J$ into intervals. We may have an interval J_0 among them which is at a distance 0 from J'. For this J_0

$$|R(J_0) \cap U(J')| \leq |R(J_0) \cap U(J)| < \Delta_1 |J_0| < \Delta_1 |J|$$

The number of intervals of this partition that are at a distance $r \geq 1$ from J' is at most 2^{r+1}, and their size is at most $(2A)^r |J|$. Thus

$$(4) \qquad x_3 < |J| (\Delta_1 + \sum_{r \geq 1} 2^{2r+1} A^r \Delta_r)$$

We have shown now that $|y_1 - x_1|$ is small. Assume that the intervals J and J' are in the same cherry for a Zig step (if they are not in the same cherry neither in a Zig nor in a Zag step, then they belong to different components, and we have seen before that in this case $\delta(J) = 0$). If the bound on the right-hand side of (4) is less than $\alpha^{35}|J|$ after a Zag step, then the next Zig step will exchange all foreign elements of J and J' except for at most

$$|x_1 - y_1| + 8A(\Delta_1' + \varepsilon)|J| < (8A + 20A\Delta_1' + \sum_{r \geq 2} 2^{2r+1} A^r \Delta_r')|J| < \alpha^{30}|J|$$

We tacitly used the following trival

Lemma 5. For any k, $1 \leq k \leq n$, the numbers $|R(\{1,\ldots,k\}) \cap \{k+1,\ldots,n\}|$ and $|R(\{k+1,\ldots,n\}) \cap \{1,\ldots,k\}|$ change monotone decreasing throughout the algorithm.

(This is clearly true for all comparator networks in which switches place the larger element to the larger register.)

References

1. D. Angluin, A note on a construction of Margulis, *Information Processing Letters* 8(1), 1979, 17-19.

2. K. Batcher, Sorting networks and their applications, AFIPS Spring Joint Computer Conference 32(1968), 307-314.

3. O. Gabber and Z. Galil, Explicit constructions of linear size superconcentrators, *Proc. 20th Ann. Symp. Found. Comp. Sci.*, 1979, 364-370.

4. M. Klawe, Non-existence of one-dimensional expanding graphs, *IEEE*, 1981, 109-114.

5. D.E. Knuth, *The Art of Computer Programming*, Addison-Wesley, Vol. 3, pp. 220-246 (and Exercises 49 and 51).

6. G. Margulis, Explicit constructions of concentrators, *Problemy Peredachi Informatsii* 9(4), 1973, 71-80 (English translation in *Problems of Information Transmission*, Plenum, N.Y., 1975.

7. W. Paul, R. Tarjan and J. Celoni, Space bounds for a game on graphs, *Mathematical Systems Theory* 10, 1979, 239-251.

8. M. Pinsker, On the complexity of a concentrator, *7th International Teletraffic Conference*, Stockholm, June 1973, 318/1-318/4.

9. N. Pippenger, Superconcentrators, *SIAM J. Computing* 6(2), 1977, 298-304.

10. L.G. Valiant, On nonlinear lower bounds in computational complexity, *Proc. 7th Annual ACM Symp. on Theory of Computing,* Albuquerque, N.M., May 1975, 45-53.