Complexity of sorting and selection. Materials for lectures

Igor S. Sergeev[1]

preliminary version 1.1e — March 5, 2025

[1]e-mail: isserg@gmail.com

Contents

Theme 1

Minimizing the number of comparisons in the sorting problem

1.1 Introduction

Consider a classical and traditional (especially for programming) sorting problem. Usually it is required to sort a numeric array, but in general the elements of the array can belong to an arbitrary set on which an order relation is defined.

Partial order on a set $\mathcal{M}$ is a binary relation "$\leqslant$" satisfying the properties: 1) reflexivity: $a \leqslant a$; 2) transitivity: from $b \leqslant a$ and $c \leqslant b$ it follows $c \leqslant a$; 3) antisymmetry: from $b \leqslant a$ and $a \leqslant b$ it follows $a = b$ for any $a, b, c \in \mathcal{M}$. Instead of $b \leqslant a$ we will also use the notation $a \geqslant b$.

A set with a partial order defined on it, $(\mathcal{M}, \leqslant)$, is called a *partially ordered set*, abbreviated as *poset*. Elements $a, b \in \mathcal{M}$ for which either $a \leqslant b$ or $b \leqslant a$ are called *comparable*. A partial order in which any two elements of the set are comparable is called *linear*. It is easy to verify that any partial order can be complemented to a linear order.

The sorting problem is well defined for a set of elements from a linearly ordered ground set. This linear order is assumed to be unknown in advance, and can be determined using pairwise comparisons. The result of a comparison operation of two elements e_1, e_2 is an ordered pair (a, b) such that $\{a, b\} = \{e_1, e_2\}$ and $a \leqslant b$.

Algorithms consisting of comparisons can be divided into two classes: those in which the choice of two subsequent elements for comparison depends on the results of previous comparisons, and those in which the sequence of comparisons is predetermined. There are other classifications, but we will not consider them.

An algorithm from the first class can be conveniently represented as a comparison tree. Recall that a *tree* is a connected graph without cycles. A *rooted binary tree* (oriented toward the root) is a directed tree with a single vertex adjacent only to outgoing edges (the root), and with any internal vertex having exactly two outgoing

edges, directed away from the root. A *comparison tree* is a rooted binary tree in which each internal vertex is assigned a pair of elements from the input set, and the two edges outgoing from an internal vertex are labeled by the symbols "$\geqslant$" and "$\leqslant$".

An algorithm is constructed according to a comparison tree as follows: a comparison is performed between a pair of elements corresponding to the root; depending on the result of the comparison, a transition is made to the next vertex along one of the edges coming out of the root; the same procedure is performed for this vertex, and so on until we reach a leaf of the tree.

An example of a comparison tree for sorting a 3-element set is shown in Fig. 1.1. The leaves of the tree are assigned ordered permutations of the elements of the input set.

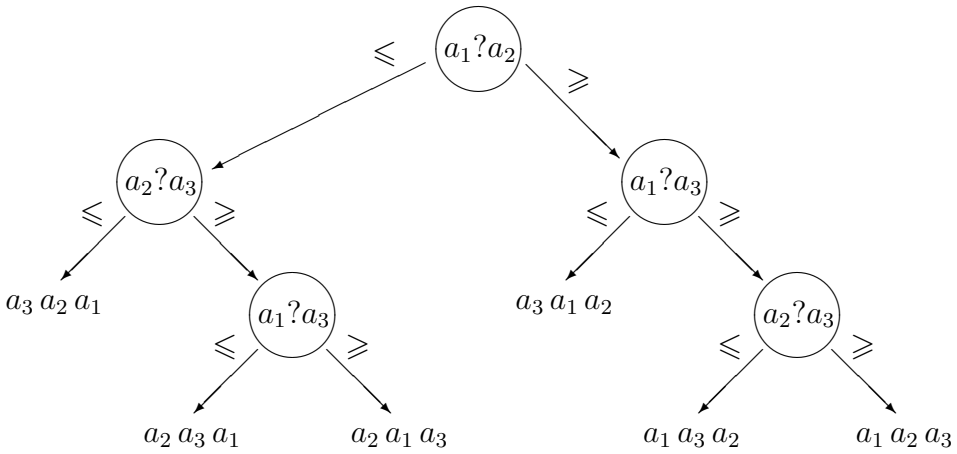

Figure 1.1: Example of a comparison tree

It is clear that the representation of an algorithm by a comparison tree is difficult from the point of view of clarity. Even the description of the algorithm for sorting five elements will require the depiction of about a hundred vertices. Therefore, the comparison tree can be kept in mind — it will be useful later when deriving lower bounds — but for the construction and analysis of specific algorithms it is more convenient to use other tools, for example, Hasse diagrams.

Hasse diagrams are used to graphically represent partially ordered sets. A *diagram* (or the *Hasse diagram*) of a *poset* is a directed graph in which the vertices correspond to elements of the poset, and an edge connects vertices v_x and v_y corresponding to elements x and y, and is directed toward v_y iff $x \leqslant y$ and there is no other element z such that $x \leqslant z \leqslant y$ (i.e. x immediately precedes y in the poset). When drawing a poset diagram, the orientation of the edges is often omitted, and it is assumed that of two elements connected by an edge, the one that is higher is greater.

In Fig. 1.2 Hasse diagrams are used to describe a 4-element sorting algorithm. At each step, we depict a diagram of a set with a partial order, which is determined by

the comparisons performed. Pairs of elements that are compared at the next step are marked.

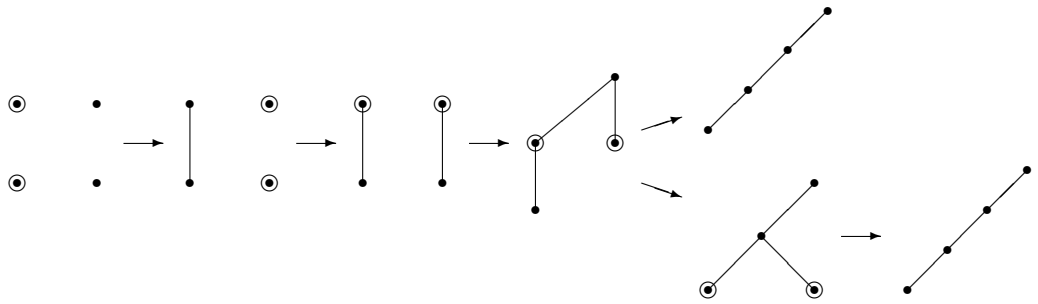

Figure 1.2: Hasse diagrams for a sorting procedure

1.2 Sorting. Lower bound on the number of comparisons

Let $(\mathcal{P}, \leqslant)$ be a finite poset[1] consisting of unequal elements. Denote by $e(\mathcal{P})$ the number of possible extensions of the partial order $\leqslant$ to a linear order on the set $\mathcal{P}$. In other words, $e(\mathcal{P})$ is the number of permutations of elements of $\mathcal{P}$ that are consistent with the partial order (i.e., permutations in which the elements are listed in "non-decreasing" order).

Let us state a *sorting* problem in the following general form. We are given a poset $\mathcal{P}$ of unequal elements subject to a linear order $\leqslant$, which is unknown. It is required to order the elements of $\mathcal{P}$ according to the order $\leqslant$, using pairwise comparisons of the elements.

Without loss of generality, we may assume that the set to be sorted is a subset of the set of real numbers $\mathbb{R}$, on which the usual linear order $\leqslant$ is defined.

The complexity of a sorting algorithm is the maximum number of comparisons performed by the algorithm over all possible orderings of the elements of the input set. The complexity coincides with the depth of the comparison tree representing the algorithm, i.e. the number of edges in the longest directed chain from the root to a leaf of the tree. By $\mathsf{S}(\mathcal{P})$ we denote the minimum complexity of algorithms sorting a poset $\mathcal{P}$, and by $\mathsf{S}(n)$ — the complexity of sorting n elements that are entirely not ordered (i.e., a poset with an empty partial order).

Theorem 1.1. $\mathsf{S}(\mathcal{P}) \geqslant \log_2 e(\mathcal{P})$.

▶ To prove this, we need a trivial fact: a rooted binary tree of depth h can have at most 2^h leaves. Therefore, the depth of a tree with N leaves is at least $\log_2 N$.

[1]Further, we will omit the symbol of relation in the notation of posets.

A leaf of a comparison tree in a sorting algorithm corresponds to some permutation of the elements of the input set; conversely, each possible permutation corresponds to some leaf (possibly more than one). Therefore, the tree has at least $e(\mathcal{P})$ leaves, which implies the stated lower bound. ■

Corollary 1.1. $\mathsf{S}(n) \geqslant \log_2(n!)$.

Recall that according to the well-known Stirling formula

$$\log_2(n!) = n\log_2 n - n/\ln 2 + 0.5\log_2 n + O(1).$$

We will see further that the lower bounds of Theorem 1.1 and Corollary 1.1, based on the most general considerations, turn out to be tight not just in an asymptotic sense, but principally they differ very little from the exact values of complexity.

1.3 Merging sorted sets. Merge sort

Consider another frequently encountered problem: merging two ordered arrays. Let the first set have m elements, and the second set have n. The task is to sort the given poset $\mathcal{L}_{m,n}$ of $n+m$ elements. It is easy to see that $e(\mathcal{L}_{m,n}) = C_{n+m}^m$, since there are C_{n+m}^m ways to merge these two ordered sets into one.

Theorem 1.1 immediately implies $\mathsf{S}(\mathcal{L}_{m,n}) \geqslant \log_2 C_{n+m}^m$. In the case $m = 1$ we obtain the bound $\mathsf{S}(\mathcal{L}_{n,1}) \geqslant \log_2(n+1)$, which is in fact tight.

Lemma 1.2. $\mathsf{S}(\mathcal{L}_{n,1}) = \lceil \log_2(n+1) \rceil$.

▷ It remains to prove the upper bound. The proof is by induction. In the case $n = 1$ the statement is obviously true. Consider the induction step from n to $n+1$. Let us compare the only element of one of the arrays with the middle element of the other array (with any of the two middle elements if n is even). Depending on the result of the comparison, we then insert the specified element into one of the two halves of the latter array. So we obtain

$$\mathsf{S}(\mathcal{L}_{n,1}) \leqslant \mathsf{S}(\mathcal{L}_{\lceil (n-1)/2 \rceil,1}) + 1 \leqslant \log_2 \lceil (n+1)/2 \rceil + 1 = \lceil \log_2(n+1) \rceil.$$

□

The procedure described in the proof of the lemma is called binary insertions.

Also interesting is the case of similar-size sets $m = n + O(1)$, for which, according to Stirling's formula, $\mathsf{S}(\mathcal{L}_{m,n}) \geqslant m + n - O(\log n)$ holds. In fact, we have

Lemma 1.3. $\mathsf{S}(\mathcal{L}_{n,n}) = 2n - 1,\ \mathsf{S}(\mathcal{L}_{n+1,n}) = 2n$.

▷ By induction we prove a more general upper bound $\mathsf{S}(\mathcal{L}_{m,n}) \leqslant m+n-1$. It is straightforward for $m+n \leqslant 2$. In the case $m+n > 2$ we compare the maximal elements of the two arrays — the larger of them is the maximal element of the entire set. After removing it, it remains to merge the two new arrays with a total of $m+n-1$ elements. Apply the inductive hypothesis and obtain the required bound.

Let us prove the lower bound for the case $m = n$. Denote the elements of the first array by a_i in ascending order, and the elements of the second set — by b_i. In a comparison tree, consider the path corresponding to the ordering

$$b_1 \leqslant a_1 \leqslant b_2 \leqslant a_2 \leqslant \ldots \leqslant b_n \leqslant a_n.$$

On this path, comparisons of any adjacent elements in a given permutation must be performed, since information about the relationship between adjacent elements is initially unavailable and cannot be obtained even if all other pairwise comparisons are done. Thus, at least $2n-1$ comparisons must be performed. The case $m = n+1$ is considered similarly. □

Based on the merge algorithm, it is possible to construct a simple and at the same time asymptotically optimal sorting algorithm in terms of complexity.

Theorem 1.2. $\mathsf{S}(n) \leqslant \log_2(n!) + O(n)$.

► The method essentially exploits the popular idea of "dividing in half". The sorted set is divided into two parts, these parts are ordered separately, and the two ordered sets are merged. Using the proven lemma, for the complexity $s(n)$ of the algorithm, we can write out the recurrence relations:

$$s(2n) = 2s(n) + 2n - 1, \qquad s(2n+1) = s(n) + s(n+1) + 2n \tag{1.1}$$

with the initial condition $s(1) = 0$.

Claim 1.4.

$$s(n) = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1. \tag{1.2}$$

▷ Introduce the function $q(n) = s(n) - s(n-1)$. It is handy to rewrite (1.1) as

$$q(2n) = q(2n-1) = q(n) + 1$$

setting $q(1) = 0$. The new relations are easily resolved as $q(n) = \lceil \log_2 n \rceil$. It remains to employ the formula $s(n) = \sum_{i=1}^{n} q(n) = \sum_{i=1}^{n} \lceil \log_2 n \rceil$.

Now it is easy to verify the validity of (1.2) by induction. If (1.2) is valid for $s(n)$, and $q(n+1) = q(n)$, then (1.2) is obviously valid for $s(n+1)$. Otherwise, if $q(n+1) = q(n) + 1$, i.e. $n = 2^k$, then

$$s(n) + q(n+1) = nk - n + 1 + (k+1) = (n+1)(k+1) - 2n + 1,$$

which is what was required. □

Therefore, $\mathsf{S}(n) \leqslant s(n) = n\log_2 n - O(n)$. ■

The complexity of the method is closest to the lower bound for the complexity of sorting at $n = 2^k$, when $s(n) = n\log_2 n - n + 1$.

1.4 Fast sorting of a partially ordered set

Let us move on to a more general problem of sorting an arbitrary poset $\mathcal{P}$. The proof of the following theorem is based on the binary insertion method.

Theorem 1.3 (M. L. Fredman)**.** $\mathsf{S}(\mathcal{P}) \leqslant \log_2 e(\mathcal{P}) + 2|\mathcal{P}|$.

▶ In fact, we will prove a slightly more general statement, from which the theorem immediately follows.

Lemma 1.5. *Let M be a finite subset of $\mathbb{Z}^n$. Then a given unknown vector $b = (b_1, \ldots, b_n) \in M$ can be determined in no more than $\log_2 |M| + 2n$ queries of the form $b_i \overset{?}{\leqslant} x$, executed sequentially in the order of increasing i.*

▷ We will determine the coordinates of b sequentially. Denote $M_k = \{c \in M \mid c_1 = k\}$ — the set of vectors from M with the first coordinate k. Let $k_1 < k_2 < \ldots < k_t$ be the indices of all nonempty sets M_k. Without loss of generality, we can assume $k_i = i$ for all $i = 1, \ldots, t$. Denote $m_k = |M_k|/|M|$. By construction, $m_1 + \ldots + m_t = 1$.

Divide the segment [0, 1] successively into intervals of length $m_1, \ldots, m_t$. Let $f(x)$ denote the number of interval centers to the left of a point x.

Via the binary search method, we can determine the number s of the interval for which $b \in M_s$ by sequentially executing the queries: $b_1 \overset{?}{\leqslant} f(1/2)$, then, depending on the result, $b_1 \overset{?}{\leqslant} f(1/4)$ or $b_1 \overset{?}{\leqslant} f(3/4)$, etc. Since it is obvious that $b_1 > f(0) = 0$ and $b_1 \leqslant f(1) = t$, after executing j queries we have the relation

$$\lambda_j = f(r_j/2^j) < b_1 \leqslant f((r_j + 1)/2^j) = \rho_j$$

for some $r_j \in \mathbb{Z}$. The search ends under the condition $\lambda_j = \rho_j - 1$. In this case the desired interval is found: $b_1 = \rho_j$.

Let $j = 1 - \lfloor \log_2 m_s \rfloor$, hence, $m_s \geqslant 2^{1-j}$. Then both points $r_j/2^j$ and $(r_j + 1)/2^j$ belong to the s-th subsegment, so the search termination condition is certainly satisfied after j comparisons.

The search continues inside the set M_s with $n - 1$ unknown coordinates. The estimate of the lemma follows by induction.

For $n = 1$, due to the above argument, in view of $m_s = 1/|M|$, it is sufficient to implement $1 + \lceil \log_2 |M| \rceil$ queries. In the general case (step from $n-1$ to n), the number of queries does not exceed

$$\log_2 |M_s| + 2(n-1) + 1 + \lceil \log_2 |M|/|M_s| \rceil \leqslant \log_2 |M| + 2n.$$

□

The theorem is proved by a procedure of successive insertions of $n = |\mathcal{P}|$ elements $X_1, \ldots, X_n \in \mathcal{P}$ into an ordered array. Initially, the array consists of a single element X_1, then the second element X_2 is added to it, then the third one is inserted, and so on. Such a procedure is equivalent to determining the vector $b = (b_1, \ldots, b_n)$, where b_i is the number of position for inserting the i-th element. The query $b_i \overset{?}{\leqslant} k$ corresponds to the comparison $X_i ? Y_k$, where Y_k is an element of rank k in the already constructed array. It is easy to see that any linear extension of the poset $\mathcal{P}$ is uniquely characterized by the vector $b \in \mathbb{Z}^n$. Finally, we apply Lemma 1.5. ■

The bound of Theorem 1.3 is good for "sparse" posets, with a large number of extensions. But for small $e(\mathcal{P})$ it is not quite efficient. We will show below that $O(\log e(\mathcal{P}))$ comparisons are always sufficient to sort a poset.

First, we note that the proof of Theorem 1.3 implies

Corollary 1.6. *Let a poset $\mathcal{P}$ contain a linearly ordered subset of cardinality l. Then $\mathsf{S}(\mathcal{P}) \leqslant \log_2 e(\mathcal{P}) + 2(n-l)$.*

▷ The insertion procedure starts from an ordered array of length l. We can assume that the coordinates $b_1, \ldots, b_l$ in the condition of Lemma 1.5 are fixed. □

The following lemma shows that the number of linear extensions of a poset $\mathcal{P}$ is small exactly in the case when $\mathcal{P}$ contains a large linearly ordered subset (this is precisely the situation when the bound of Theorem 1.3 is inefficient).

Lemma 1.7. *Let the maximal linearly ordered subset of a poset $\mathcal{P}$ have cardinality l. Then $e(\mathcal{P}) \geqslant 2^{|P|-l}$.*

▷ The elements of the poset $\mathcal{P}$ can be divided into l layers: $V_1, \ldots, V_l$. The set V_i contains elements for which the length of the maximal ascending chain in $\mathcal{P}$ ending with this element is equal to i.

By construction, the elements of any layer are pairwise incomparable and can be ordered arbitrarily. Therefore, due to the simple inequality $k! \geqslant 2^{k-1}$,

$$e(\mathcal{P}) \geqslant \prod_{i=1}^{l} |V_i|! \geqslant \prod_{i=1}^{l} 2^{|V_i|-1} = 2^{|\mathcal{P}|-l}.$$

□

Theorem 1.4 (J. Kahn, M. Saks)**.** $\mathsf{S}(\mathcal{P}) = O(\log e(\mathcal{P}))$.

► Let $|\mathcal{P}| = n$. In the graph (Hasse diagram) of the poset $\mathcal{P}$, select a maximal chain Z; we denote its length by l. For each element v outside Z, we define the minimal interval (a_v, b_v) in Z that v can fall into. Here a_v is the maximal element of Z such that $a_v \leqslant v$, and b_v is the minimal element of Z such that $v \leqslant b_v$, provided that such elements exist. The interval may be unbounded on one or both sides.

Consider the set $\mathcal{P}'$ consisting of elements outside Z and of those elements in Z that are ends of minimal intervals. Clearly, $|\mathcal{P}'| \leqslant 3(n-l)$. The first step of the algorithm consists of sorting $\mathcal{P}'$ by the method of Theorem 1.3 and involves at most $\log_2 e(\mathcal{P}') + 6(n-l)$ comparisons. Since $\mathcal{P}' \subset \mathcal{P}$ and all relations between elements of $\mathcal{P}'$ are exactly the same as in $\mathcal{P}$, we have $e(\mathcal{P}') \leqslant e(\mathcal{P})$. In other words, any linear extension of $\mathcal{P}'$ is contained in some extension of $\mathcal{P}$.

After sorting $\mathcal{P}'$ the obtained poset consists of two ordered and, generally speaking, intersecting chains, as shown in Fig. 1.3 (the chains of elements are painted in bold lines). Sorting the poset reduces to independent merging of pairs of fragments of the chains. In this case, the second fragments in the pairs contain only elements outside Z, and their total size does not exceed $n-l$.

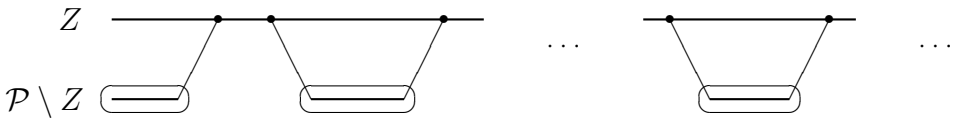

Figure 1.3: Form of a poset after the first stage

Denote by T_i the number of possible outcomes of merging in an i-th pair. Then $\prod_i T_i \leqslant e(\mathcal{P})$, since any combination of merge results in different fragments is a legal extension of $\mathcal{P}$ (this follows from the minimality of intervals for vertices outside Z — there are no additional relations between vertices in Z and outside Z in $\mathcal{P}$ besides those shown in Fig. 1.3).

Applying Corollary 1.6, we find that $\sum_i \log_2 T_i + 2(n-l)$ comparisons are sufficient to perform merges. Combining the complexity estimates of both stages, we obtain that the method for sorting the poset $\mathcal{P}$ utilizes

$$\log_2 e(\mathcal{P}') + 6(n-l) + \sum_i \log_2 T_i + 2(n-l) \leqslant 10 \log_2 e(\mathcal{P})$$

comparisons, taking into account Lemma 1.7. ■

Exercises

Ex. 1.1. Show that the binary insertion sort method also has complexity $\log_2(n!) + O(n)$. The method consists of partitioning the set of elements into pairs, performing comparisons in the pairs, then sorting the higher elements of the pairs, and then inserting the lower elements of the pairs into the sorted array.

Ex. 1.2. Determine the value of $\mathsf{S}(\mathcal{L}_{2,n})$ (the solution is in [6]). The result shows that it is often more efficient to insert two elements into an ordered array together than separately. [*R. Graham, F. Hwang, S. Lin*]

Ex. 1.3. Show that even $1.44 \log_2 e(\mathcal{P})$ comparisons may not be enough to sort a poset $\mathcal{P}$ in the general case. To do this, consider a set $x_1, \ldots, x_n$ with the given partial order:

$$x_i \leqslant x_{i+2}, \quad i = 1, \ldots, n-2, \qquad x_i \leqslant x_{i+3}, \quad i = 1, \ldots, n-3.$$

(See the proof of Lemma 1.3.) [*N. Linial*]

Comments. The material in sections 1.1–1.3 is standard and is discussed in great detail in [6]. Theorem 1.3 was proved by M. Fredman in [2]. The proof of Theorem 1.4 basically follows the scheme proposed in [3].

Several other sorting methods besides merge sort achieve an asymptotically tight $\log_2(n!)$ complexity bound with an excess of $O(n)$. The closest to it is the Ford—Johnson version of binary insertion sort, see [6]. The author proved [7] that actually $\mathsf{S}(n) = \log_2(n!) + o(n)$ (the group insertion method).

The result of Theorem 1.4 on the complexity of sorting an arbitrary poset was first obtained in [5] as a consequence of the presence in any not completely ordered poset $\mathcal{P}$ of a balanced pair of elements x, y, i.e. such that ordering this pair reduces the number of extensions by a factor of $c \leqslant \frac{e(\mathcal{P}, x \leqslant y)}{e(\mathcal{P})} \leqslant 1 - c$, where c is a universal constant. According to a well-known conjecture, $c = 1/3$ should hold, but so far it has only been proven that $c \geqslant \frac{5-\sqrt{5}}{10} \approx 0.28$ [1]. Methods for proving bounds on the balance constant [5, 1] use a rather nontrivial apparatus of convex geometry. Another approach [4] to constructing a sorting algorithm is related to the entropy analysis of poset graphs and is also quiet sophisticated. The method proposed in [3] is elementary, although it leads to a larger multiplicative constant in the complexity bound.

Bibliography

[1] Brightwell G. R., Felsner S., Trotter W. T. *Balancing pairs and the cross product conjecture.* Order. 1995. **12**, 327–349.

[2] Fredman M. L. *How good is the information theory bound in sorting?* Theor. Comput. Sci. 1976. **1**, 355–361.

[3] Haeupler B., Hladík R., Iacono J., Rozhoň V., Tarjan R. E., Tětek J. *Fast and simple sorting using partial information.* In: Proc. SODA (New Orleans, 2025). SIAM, 2025, 3953–3973.

[4] Kahn J., Kim J. H. *Entropy and sorting.* Proc. STOC (Victoria, Canada, 1992). NY: ACM, 1992, 178–187.

[5] Kahn J., Saks M. *Balancing poset extensions.* Order. 1984. **1**, 113–126.

[6] Knuth D. E. *The art of computer programming. Vol.* 3. *Sorting and searching.* Reading: Addison-Wesley, 1998.

[7] Sergeev I. S. *On the upper bound of the complexity of sorting.* Computational Mathematics and Mathematical Physics. 2021. **61**(2), 329–346.

Theme 2

Complexity of the selection problem

2.1 Introduction

The problem of *selection* of elements of rank $r_1, \ldots, r_k$ in a set of cardinality n, where $r_i \leqslant n$, with an unknown linear order $\leqslant$ is posed similarly to the sorting problem. This problem is a special case of the poset generation problem, in which it is required to obtain a given poset or its extension by a series of comparisons. We denote by $\mathsf{C}(\mathcal{P})$ the complexity of generating a poset $\mathcal{P}$. Theorem 1.1 implies

Corollary 2.1. $\mathsf{C}(\mathcal{P}) \geqslant \log_2(|\mathcal{P}|!/e(\mathcal{P}))$.

Selection of elements of rank $r_1, \ldots, r_k$ in an n-element set consists of constructing a poset with the structure

$$V_1 \leqslant x_1 \leqslant V_2 \leqslant x_2 \leqslant \ldots \leqslant V_k \leqslant x_k \leqslant V_{k+1}, \tag{2.1}$$

where x_i is an element of rank r_i, and V_i is a set of $r_i - r_{i-1} - 1$ elements (here we set $r_0 = 0$ and $r_{k+1} = n + 1$). Let us verify that this is indeed the case.

Lemma 2.2. *If an element of rank m is known in a poset, then the set of $m - 1$ elements greater than it is also known.*

$\triangleright$ Assume the contrary. Let e be the element in question. Denote by M_G, M_L, and M_N the set of elements greater than e, less than e, and the set of elements not comparable to e, respectively. By assumption, $M_N \neq \varnothing$.

By construction, an element of M_N cannot be greater than any element of M_G and cannot be less than any element of M_L. Therefore, there exists a linear extension of the poset in which the set $M_N \cup \{e\}$ lies strictly between the sets M_G and M_L. An element e can occupy any position in the set $M_N \cup \{e\}$, and therefore its order in the entire set is not uniquely determined. $\square$

The complexity of the selection problem in the general formulation is denoted by $\mathsf{C}_{r_1,\ldots,r_k}(n)$.

2.2 Selection of extreme elements

It is easy to show that the complexity of selecting the maximum (minimum) element of a set is $n-1$.

Theorem 2.1. $\mathsf{C}_1(n) = n-1$.

► The upper bound $\mathsf{C}_1(n) \leqslant n-1$ follows from an obvious algorithm, in which at each step the current maximum (in the set of already considered elements) is compared with a new element, the maximum is selected, etc.

The lower bound follows from the trivial observation: each element, except for the maximum, must lose in at least one comparison. ■

Less trivial is the question of how many comparisons are required to determine the maximum and minimum elements of a set. An elegant solution to this problem was found by Pohl in 1972.

Theorem 2.2 (I. Pohl)**.** $\mathsf{C}_{1,n}(n) = \lceil 3n/2 \rceil - 2$.

► The upper bound is proved by the following simple algorithm. Divide all elements into pairs, perform comparisons in pairs. Then, in the set of maximal elements of pairs, including in the odd case an unpaired element, determine the maximum. Similarly, determine the minimum among the minimal elements of pairs and, if necessary, an unpaired element. A total of

$$\lfloor n/2 \rfloor + 2(\lceil n/2 \rceil - 1) = \lceil 3n/2 \rceil - 2$$

comparisons will be performed.

Let us prove the lower bound. At each step of an algorithm, we will denote by the quadruple (a, b, c, d) the number of elements that did not participate in the comparisons, the number of elements that only won, the number of elements that only lost, the number of elements that both won and lost, respectively.

Before the start of the algorithm $(a,\ b,\ c,\ d) = (n,\ 0,\ 0,\ 0)$, at the end of the algorithm $(a,\ b,\ c,\ d) = (0,\ 1,\ 1,\ n-2)$. In the comparison tree, we consider a chain in which at each vertex a branch is chosen such that if both elements in a pair are from the same subset (of the four mentioned above), then the outcome of the comparison is chosen arbitrarily; otherwise, the element that has only won before (if any) wins, and the element that has only lost (if any) loses; otherwise, any outcome is allowed.

Then, during the algorithm, no comparison changes both a and d. Thus, to "empty" the first subset, at least $\lceil n/2 \rceil$ comparisons are required, and at least $n-2$ comparisons are required to "fill" the fourth subset. ■

The question of the complexity of selecting the rank-2 element is not so obvious: incorrect proofs (of the lower bound) were published twice, until in 1962 Kislitsyn proposed a correct, although rather sophisticated, proof. It was subsequently significantly simplified.

Theorem 2.3 (S. S. Kislitsyn)**.** $\mathsf{C}_2(n) = n + \lceil \log_2 n \rceil - 2$.

▶ To prove the upper bound, we construct a knockout tournament using the cup system to determine the largest element. In such a tournament, the winner participates in no more than $\lceil \log_2 n \rceil$ comparisons. Any element that was not compared with the largest element during the algorithm lost to some other element, and therefore cannot be second. It remains to determine the largest element among those dropped out in direct comparisons with the winner. Thus, to select the second element, it is sufficient to perform no more than $(n-1) + (\lceil \log_2 n \rceil - 1)$ comparisons.

In particular, it follows from Lemma 2.2 that any algorithm for selecting the second element also finds the first element. Now we show that there exists an ordering of the input set elements such that the maximal element participates in at least $\lceil \log_2 n \rceil$ comparisons.

At any moment of the algorithm, an element that has never lost will be called a "leader". We introduce a "subordination" relationship, according to which each element at any moment is subordinate to some leader, and only one; a leader is subordinate to itself. Before the algorithm starts, every element is a leader subordinate to itself, and at the end there remains only one leader (the maximal element), to which all elements are subordinate.

Now let us consider a path in the comparison tree such that in any comparison (not counting comparisons with a predetermined outcome) involving a leader and a non-leader, the leader is recognized as the winner. Among two leaders, the leader who has participated in more comparisons wins; otherwise, the outcome is arbitrary. If a leader loses the next comparison, then it and its subordinate elements become subordinate to the winner of the comparison.

By induction we check that a leader who participated in r comparisons has at most 2^r subordinate elements (including itself). For $r = 0$ this is obvious. Otherwise, if a leader wins its r-th comparison, then in addition to at most 2^{r-1} subordinate elements (by the induction hypothesis) it also receives at most 2^{r-1}, because its opponent could not be a leader who won more than $r-1$ comparisons.

Thus, the largest of n elements will participate in at least $\lceil \log_2 n \rceil$ comparisons. Any element in the set of elements directly compared to the winner, except that of the second rank in the entire set, must lose in at least one more comparison. Hence, $n-1$ elements lose in at least one comparison and at least $\lceil \log_2 n \rceil - 1$ elements lose once more. ■

2.3 Median selection

Let us introduce the standard notation $H(x) = -x\log_2 x - (1-x)\log_2(1-x)$ for the binary entropy function defined on the interval [0, 1]. At the ends of the interval we set $H(0) = H(1) = 0$ by continuity.

Theorem 2.4 (S. W. Bent, J. W. John)**.** *Let $t = \alpha n$. Then*

$$\mathsf{C}_t(n) \geqslant (1 + H(\alpha))n - O(\sqrt{n}).$$

▶ Consider an arbitrary algorithm that solves the problem of selection of the rank-t element in a set M of size n. Fix some subset $T \subset M$ of size t. Let it contain $t-1$ maximal elements of M. Denote $U = M \setminus T$ and set $U_0 = U$, $T_0 = T$. Further, $T_0 \cup U_0$ plays the role of the set of candidates for the position of the element of rank t. Let $r > 1$ and $q \geqslant r$ be parameters that will be defined later.

a) In the comparison tree corresponding to the algorithm, we will proceed along the following path. While $|T_0| > r$, the outcome of the comparison of elements x and y is defined as $x \leqslant y$ if $x \in U$ and $y \in T$ and arbitrarily if $x, y \in T$ or $x, y \in U$. By the way, in the penultimate case, the larger element is removed from T_0, and in the latter case, the smaller element is removed from U_0.

We will stop this process at the moment when $|T_0| = r$. This will definitely happen, because based on the comparisons performed, none of the elements of the set T_0 can be excluded from the list of candidates. At the end of the algorithm, if it were not stopped, $|T_0| = 1$ would be.

b) For $y \in T_0$, denote by W_y the set of elements from U that have been directly compared with y. Let $y^* \in T_0$ be an element such that $|W_{y^*}|$ is minimal. Set $Q = (U_0 \setminus W_{y^*}) \cup \{y^*\}$.

We assume that in the linear ordering the set Q is located strictly between the sets $(U \setminus U_0) \cup W_{y^*}$ (of small elements) and $T \setminus \{y^*\}$ (of large elements). Then the element of rank t in M is maximal in Q. Note also that no two elements of Q have yet been compared with each other.

c) If $|W_{y^*}| > q - r$, then the algorithm already performed:
– at least $(r-1)(q-r+1)$ comparisons of elements from T_0 and U;
– $|T \setminus T_0| + |U \setminus U_0| = n - |U_0| - r$ comparisons within the sets T and U;
– $|W_{y^*}|$ comparisons of the element y^* with elements from U.
In addition, at least $|U_0| - |W_{y^*}|$ comparisons will need to be performed to determine the maximum element in Q. Therefore, the overall complexity of the algorithm is no less than

$$(r-1)(q-r+1) + (n - |U_0| - r) + |W_{y^*}| + (|U_0| - |W_{y^*}|) = n - 1 + (r-1)(q-r).$$

d) In the case $|W_{y^*}| \leqslant q - r$, we prove that any choice of the set T in the comparison tree corresponds to at least 2^{n-q} leaves. To do this, we show that in total at least $n - q$ comparisons are performed within the sets T, U, and then Q.

Indeed, at least $n - |U_0| - r$ comparisons are performed within the sets T and U and at least $|U_0| - |W_{y^*}| \geqslant |U_0| + r - q$ comparisons within the set Q.

e) The set T can be selected in C_n^t ways. Any outcome of the algorithm (a leaf in the comparison tree) corresponds to $n + 1 - t$ sets T ($t - 1$ maximal elements of the set T are determined, the t-th element can be chosen arbitrarily).

f) Thus, the depth of the comparison tree and the complexity of the algorithm can be estimated as $n - 1 + (r - 1)(q - r)$ (case of item c) or the binary logarithm of the number of leaves (case of item d), of which there are no fewer than $2^{n-q}C_n^t/(n+1-t)$ in the tree. Hence,

$$\mathsf{C}_t(n) \geqslant \min\left\{n - 1 + (r-1)(q-r),\, n - q + \log_2 \frac{C_n^t}{n+1-t}\right\}.$$

When choosing parameters $r \sim \sqrt{n}$ and $q \sim 2\sqrt{n}$ and taking into account

$$\log_2 C_n^t = \log_2 C_n^{\alpha n} = nH(\alpha) - O(\log n)$$

(a consequence of Stirling's formula), we arrive at the statement of the theorem. ■

Note that for $\alpha = 1/2$, i.e. for the problem of selecting the median, the bound of the theorem takes its maximum which is asymptotically $2n$.

The most impressive upper bound for the complexity of the selection problem, which has only been slightly improved to date, appeared in 1976. For simplicity of presentation, we restrict ourselves to selecting the middle element (median); the general case is not principally different from the one under consideration.

Theorem 2.5 (A. Schönhage, M. Paterson, N. Pippenger)**.** $\mathsf{C}_{\lfloor n/2\rfloor}(n) \leqslant 3.5n + o(n)$.

► The method is based on the idea of mass production (of partially ordered sets). A large number of posets $\mathcal{X}_k$ of size $2k + 1$ with a central element (median) is constructed. In the course of the algorithm, the central elements of the sets are ordered. The general form of the obtained poset is shown in Fig. 2.1.

Note that if the number of sets $\mathcal{X}_k$ in the chain Z is large enough — so that the total number of elements in them is close to n —, then the larger half of the set with the maximum central element m_H and the smaller half of the set with the minimum central element m_L (they are highlighted in Fig. 2.1) can be excluded from further consideration as obviously not containing the median.

Indeed, let s denote the number of sets $\mathcal{X}_k$ in the chain Z, and r be the number of elements not included in the poset of Fig. 2.1. It is clear that $s = (n - r)/(2k + 1)$. According to the diagram in Fig. 2.1, the element m_H is greater than another $s(k+1)-1$ elements.

Solving the inequality $s(k + 1) - 1 \geqslant n/2 + 1$, we obtain the condition

$$r \leqslant \frac{n+4}{2k+2} - 4, \tag{2.2}$$

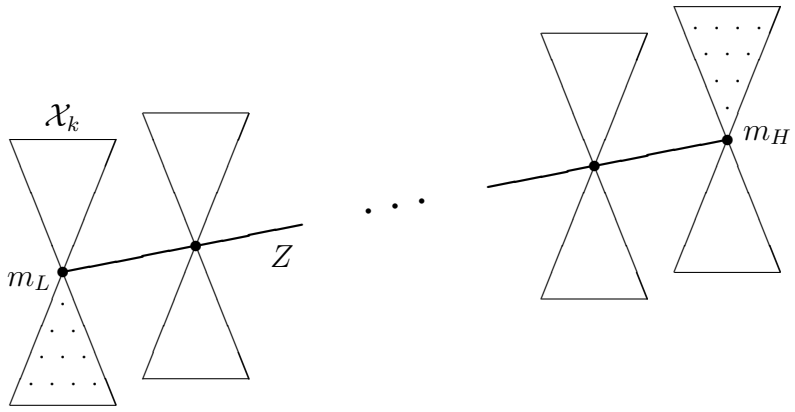

Figure 2.1: General form of the poset in the process of computations

under which m_H and the elements above it do not pretend to be the median. The same goes with m_L and the elements below it. Then, we seek the median among the remaining $n - 2k - 2$ elements.

Those elements that do not belong to the prepared posets $\mathcal{X}_k$ and are not excluded from consideration are considered to be in the factory (producing new posets $\mathcal{X}_k$; is not shown in Fig. 2.1).

The factory is characterized by the following parameters: J_k — the minimum number of elements sufficient for producing a poset, I_k — the number of comparisons necessary to start mass production, C_k — the number of comparisons sufficient to produce one poset $\mathcal{X}_k$.

Let S_n denote the total number of posets produced at the factory, and R_n — the number of remaining median candidates at the end of the algorithm. By (2.2), the latter can be found from the inequality $\frac{R_n+4}{2k+2} - 4 < J_k$, and the former can be easily determined as $S_n = (n - R_n)/(k + 1)$.

The complexity $T(n)$ of the algorithm can now be estimated as

$$T(n) = I_k + C_k \cdot S_n + S_n \log_2 n + O(R_n \log R_n), \tag{2.3}$$

where the third term corresponds to the complexity of the insertion of the center of a poset $\mathcal{X}_k$ into an ordered list, and the last term corresponds to selecting the median of the residual set of R_n elements, which can be accomplished by ordinary sorting.

The parameters are chosen as $k \asymp \sqrt[4]{n}$ and $R_n \asymp n^{3/4}$. To complete the proof of the theorem, it remains to construct a factory with parameters $I_k = O(k^2)$, $J_k = O(k^2)$ and $C_k \sim 3.5k$.

We define inductively a poset $\mathcal{H}_p(v)$ — a *hyperpair* of order p with center v. A hyperpair $\mathcal{H}_0(v)$ consists of a single element v. A hyperpair $\mathcal{H}_p(v)$ is obtained from two hyperpairs $\mathcal{H}_{p-1}(v_1)$ and $\mathcal{H}_{p-1}(v_2)$ by comparing their centers v_1 and v_2. As v we choose the winner of the comparison if $p = 2$ or $p \geqslant 3$ is odd, and the loser if $p = 1$ or $p \geqslant 4$ is even. Younger representatives of the family $\mathcal{H}_p$ are shown in Fig. 2.2.

The following properties of hyperpairs can be easily verified by induction.

Lemma 2.3. (*i*) *A hyperpair* $\mathcal{H}_p(v)$ *consists of* 2^p *elements.*

(*ii*) *For* $p \geqslant 2$, *a hyperpair* $\mathcal{H}_p(v)$ *contains* $2^{\lfloor p/2 \rfloor} - 1$ *elements greater than* v *and* $2^{\lceil p/2 \rceil} - 1$ *elements smaller than* v.

Thus, from a hyperpair of order $2p$ one can extract a poset $\mathcal{X}_{2^p-1}$.

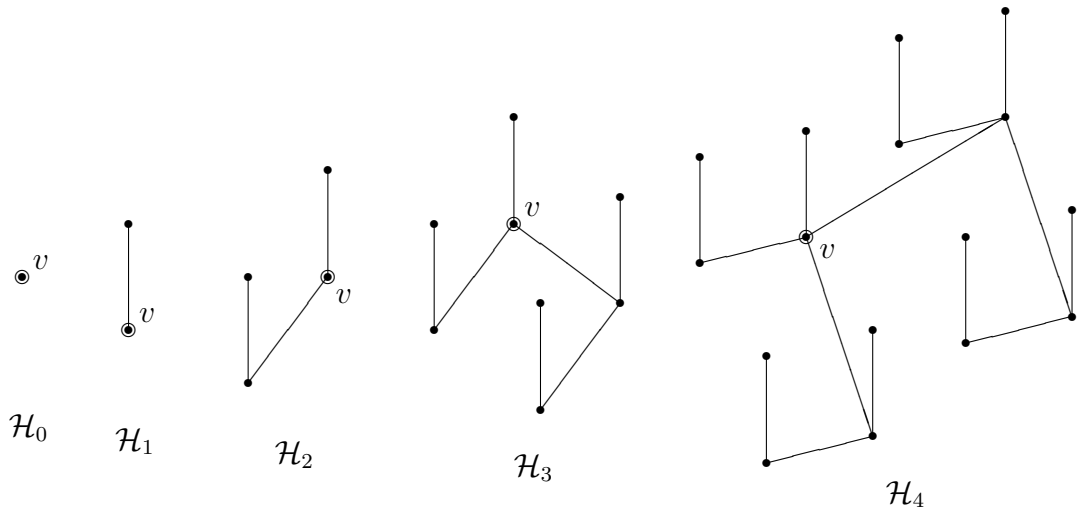

Figure 2.2: Family of hyperpairs

Note that there is a one-to-one correspondence between the edges in the hyperpair diagram and the comparisons performed when constructing a hyperpair. Also note that when the center v of a hyperpair $\mathcal{H}_p(v)$ is removed, it splits into a family of isolated hyperpairs $\mathcal{H}_0 \cup \{\mathcal{H}_{2i+1} \mid 1 \leqslant i < (p-1)/2\}$ whose centers are higher than v, and a family of hyperpairs $\mathcal{H}_1 \cup \{\mathcal{H}_{2i} \mid 1 \leqslant i < p/2\}$ whose centers are lower than v.

Further, it is easy to see that the set of elements greater (smaller) than v in the hyperpair $\mathcal{H}_p(v)$ is the union of sets of elements greater (smaller) than the center in the first (second) family of hyperpairs.

What is important in this case is that the removal of the center of a hyperpair leads to the formation of hyperpairs of lower order, which can be used to "reconstruct" a hyperpair. Moreover, the procedure for extracting a poset consisting of the center of a hyperpair and a number of elements smaller and larger than the center can be viewed as a sequence of removals of the centers of certain hyperpairs, so that the result will also be a set of hyperpairs of lower orders.

In view of the mentioned correspondence between comparisons and edges of the hyperpair diagram, the number of comparisons needed to restore a hyperpair after extracting the poset is equal to the number of edges removed from the diagram.

Let $V(p)$ and $N(p)$ denote the number of edges eliminated when removing the center of a hyperpair of order p and all elements greater than the center and, correspond-

ingly, smaller than the center. It can be verified directly that $V(0) = N(0) = 0, V(1) = N(1) = 1, V(2) = 2, N(2) = 3$.

Lemma 2.4. *For $p \geqslant 2$,*

$$N(2p) = N(2p-1) + 1 = 5 \cdot 2^{p-1} - 2, \quad V(2p) = V(2p+1) - 1 = 5 \cdot 2^{p-1} - 3.$$

▷ The center of a hyperpair $\mathcal{H}_p$ in the diagram has degree p, so we obtain the following recurrence relations:

$$\begin{gathered} N(2p) = 2p + N(1) + N(2) + \ldots + N(2p-2), \\ V(2p+1) = 2p + 1 + V(0) + V(3) + \ldots + V(2p-1), \\ N(2p-1) = N(2p) - 1, \quad V(2p) = V(2p+1) - 1. \end{gathered}$$

By substituting the initial values of $N(1), N(2), V(0)$, the first two (main) relations are solved by induction as $N(2p) = V(2p+1) = 5 \cdot 2^{p-1} - 2$. □

As a consequence, we can extract from a hyperpair $\mathcal{H}_{2p}$ a poset $\mathcal{X}_{2^p-1}$ with the recovery complexity no greater than $5 \cdot 2^p$, which allows us to construct a factory with the characteristic $C_k \sim 5k$, and this, due to (2.3) and the choice of parameters, leads to an algorithm complexity estimate of $5n + o(n)$.

In order to obtain a stronger bound, we note (this follows from the above reasoning) that from a hyperpair $\mathcal{H}_{2p}$ we can also extract a poset consisting of the center and $l \leqslant 2^p - 1$ elements smaller (larger) than it, by removing no more than $2p + 2.5l$ edges.

Now, before extracting a poset from a hyperpair, we will perform comparisons of the center with elements that do not belong to the hyperpair until we have $k = 2^p - 1$ such elements, larger or smaller than the center.

Into the poset $\mathcal{X}_k$ obtained from a hyperpair, we include all elements added in the last way, the rest we select directly from the hyperpair. Then the cost of the production of a single poset (in the number of comparisons) may be estimated as $k + l$ (the number of elements added in the second way) plus $2p + 2.5(k - l)$ (the number of comparisons required to restore a hyperpair), i.e. no more than $3.5k + o(k)$ in total. (Formally, the latter summand relate to the production of the next poset, but when summed up, this is insignificant.)

Taking into account $I_k = (k+1)^2 - 1$, $J_k = (k+1)^2 + 2k$ and $C_k \sim 3.5k$, we establish the validity of the theorem. ■

2.4 Fast multiselection algorithm

Let us return to the general problem of selecting elements of rank $r_1, \ldots, r_k$ in an n-element set, formulated in the introduction (that is, the *multiselection* problem). We assume $1 \leqslant r_1 < r_2 < \ldots < r_k \leqslant n$.

Denote $\Delta_i = r_i - r_{i-1} - 1$. The number of linear extensions of the poset $\mathcal{P}$ corresponding to the multiselection problem is $e(\mathcal{P}) = \prod_{i=1}^{k+1} \Delta_i!$, see (2.1). Therefore,

$$\mathsf{C}_{r_1,\ldots,r_k}(n) \geqslant B_{r_1,\ldots,r_k}(n) = \log_2(n!) - \sum_{i=1}^{k+1} \log_2(\Delta_i!) = n\log_2 n - \sum_{i=1}^{k+1} \Delta_i \log_2 \Delta_i - O(n) \tag{2.4}$$

by Corollary 2.1.

Obviously, $\mathsf{C}_{r_1,\ldots,r_k}(n) = \mathsf{C}_{n+1-r_1,\ldots,n+1-r_k}(n)$, because any comparison tree that selects elements of rank r_i can be associated with a dual tree that selects elements of rank $n+1-r_i$ (it selects elements of rank r_i from the point of view of the order relation $\geqslant$).

The example of selecting median shows that the information-theoretic lower bound, in this case equal to $B_{n/2}(n) \sim n$, does not necessarily provide the correct asymptotics for the complexity of the (multi)selection problem. However, for large values of B, the bound (2.4) is achievable in the asymptotic sense. The following holds:

Theorem 2.6 (K. Kaligosi, K. Mehlhorn, J. I. Munro, P. Sanders)**.**

$$\mathsf{C}_{r_1,\ldots,r_k}(n) \leqslant (1+o(1))B_{r_1,\ldots,r_k}(n) + O(n).$$

A weakened version of the bound of the theorem, $O(B+n)$, is proposed to be proved in Exercise 2.3.

▶ Set $l = \max\{2, \lceil B/n \rceil\}$. Consider the following recursive procedure.

Input: a family C of ordered chains of length $< 2l$ each and a set of ranks $R = \{r_1, \ldots, r_k\}$.

Output: elements of rank $r_1, \ldots, r_k$.

Algorithm.

(i) If the total number of elements in C does not exceed $8l^2$, then a complete sorting is performed, hence, all the desired elements are found.

(ii) Call an ordered chain *short* if its length is less than l. If there are two short chains of the same length in C, then merge them. Perform such merges until the lengths of all short chains become different. Denote the obtained family by D.

(iii) Find the median (middle element) of the medians of chains from D. Denote it by m.

(iv) Split each chain from D into two by the "point" m. Into one part (the younger one) we assign elements not exceeding m, and in the other (the older one) — the remaining elements. Denote by r the rank of m — it becomes known after performing the partitions.

(v) Recursively call the procedure for the family C' of younger chains and the set of ranks $R' = R \cap [1, r]$ (if R' is not empty), and also for the family C'' of older chains and the set of ranks[1] $R'' = (R \setminus R') - r$ (if R'' is not empty).

[1]For brevity, $S - r = \{s - r \mid s \in S\}$.

When solving a general problem, the initial call to the algorithm is made with a set of n singletons (individual elements). The correctness of the algorithm is obvious: with each recursive call, the number of elements is reduced.

By $I(C) = \sum_{c \in C} |c| \log_2 |c|$ we define the information capacity of the family C, where $|c|$ denotes the length of a chain c. Let T be the recursion tree of the algorithm, and v be its arbitrary node. Let us introduce additional notations associated with a node v:

$I_v = I(C)$ — information capacity of the input family;
$p_v = |C|$ — number of input chains;
n_v — total number of elements in C;
$J_v = I(D)$ — information capacity after merges;
$q_v = |D|$ — number of chains in family D;
$I'_v = I(C')$, $I''_v = I(C'')$ — information capacity of output families;
$p'_v = |C'|$, $p''_v = |C''|$ — number of output chains;
$\alpha_v n_v$ — total number of elements in family C'.

Let us estimate the complexity of the algorithm by steps. Step (i) is performed at the end nodes (leaves) of the tree for disjoint sets of elements, so its overall complexity for all calls is $O(n \log l)$.

Claim 2.5. *The number of comparisons performed during the merge stage at node v does not exceed $J_v - I_v + q_v - p_v$.*

▷ One merging of length-s chains requires at most

$$2s - 1 = 2s \log_2(2s) - 2s \log_2 s + 1 - 2$$

comparisons according to Lemma 1.3. □

Recall that $H(x) = -x \log_2 x - (1 - x) \log_2(1 - x)$ is the binary entropy function defined on the interval [0, 1]. It is easy to see that the function $H(x)$ is symmetric with respect to the point $x = 1/2$, where it takes the maximum value 1. In addition, the function is convex (upward): $H''(x) < 0$.

Claim 2.6. $J_v \leqslant I'_v + I''_v + n_v H(\alpha_v)$.

▷ Let a chain d_i of a family $D = \{d_1, \ldots, d_q\}$ split into chains $d'_i \in C'$ and $d''_i \in C''$. Denote $\alpha_i = |d'_i|/|d_i|$. Then we obtain[2]

$$\begin{aligned} J_v - I'_v - I''_v &= \sum_{i=1}^{q} (|d_i| \log_2 |d_i| - |d'_i| \log_2 |d'_i| - |d''_i| \log_2 |d''_i|) \\ &= \sum |d_i| \left(\log_2 |d_i| - \alpha_i \log_2(\alpha_i |d_i|) - (1 - \alpha_i) \log_2((1 - \alpha_i)|d_i|)\right) \\ &= \sum |d_i| H(\alpha_i) \leqslant n_v H\left(\sum \alpha_i |d_i|/n_v\right) = n_v H(\alpha_v). \end{aligned}$$

[2] As usual, assuming $0 \log 0 = 0$.

The latter inequality is valid due to the convexity of the entropy function. □

Let's check that the recursive partitioning into two subproblems is sufficiently balanced.

Claim 2.7. *For $n_v \geqslant 8l^2$, we have $\frac{1}{16} \leqslant \alpha_v \leqslant \frac{15}{16}$.*

▷ Let us estimate from below the number of elements that are certainly higher than the median m. By construction, $q_v > n_v/(2l) \geqslant 4l$. In the worst case, the medians of the shortest chains are higher than m. Among those chains may be (at most) one chain of every length $1, \ldots, l-1$. The remaining chains contain at least

$$(q_v/2 - l) \cdot l/2 > (n_v - 4l^2)/8 > n_v/16$$

elements greater than m. The number of elements smaller than m is estimated similarly. □

Claim 2.8. *For a nonempty set of ranks $R = \{r_1, \ldots, r_k\}$,*

$$\tau(n, R) = \sum_{v \in T;\ \deg v > 1} n_v H(\alpha_v) \leqslant n \log_2 n - \sum_{i=1}^{k+1} \Delta_i \log_2 \Delta_i + r_k - r_1 + 16n$$

(*summation on the left side is performed over the internal nodes of the recursion tree T*).

▷ The proof of the inequality in by induction on the tree depth. For $n < 8l^2$ we have $\tau(n, R) = 0 \leqslant 16n$, hence, there is nothing to prove.

Now we prove the induction step. Let a problem with parameters n, k, R be split into subproblems with parameters n', k', R' and n'', k'', R'' according to the algorithm under consideration, where

$$n' = \alpha n, \quad n'' = (1-\alpha)n, \quad k = k' + k'', \quad R' = R \cap [1,\, n'], \quad R'' = (R \setminus R') - n'.$$

Since we always have $k > 0$, without loss of generality assume that $k' > 0$.

1. Consider the case $k'' > 0$. Denote $\Delta' = n' - r_{k'}$ and $\Delta'' = r_{k'+1} - n' - 1$ — these are the lengths of the parts into which the interval $(r_{k'},\, r_{k'+1})$ is divided. By the induction hypothesis, we obtain

$$\begin{aligned}
\tau(n, R) &= nH(\alpha) + \tau(n', R') + \tau(n'', R'') \\
&\leqslant nH(\alpha) + n' \log_2 n' + n'' \log_2 n'' - \sum_{i=1}^{k+1} \Delta_i \log_2 \Delta_i \\
&\quad + \Delta_{k'+1} \log_2 \Delta_{k'+1} - \Delta' \log_2 \Delta' - \Delta'' \log_2 \Delta'' + r_k - r_{k'+1} + r_{k'} - r_1 + 16n \\
&\leqslant n \log_2 n - \sum_{i=1}^{k+1} \Delta_i \log_2 \Delta_i + r_k - r_1 + 16n,
\end{aligned}$$

since

$$H(\alpha) + \alpha \log_2(\alpha n) + (1-\alpha)\log_2((1-\alpha)n) = \log_2 n, \tag{2.5}$$

and due to the inequality (a consequence of the downward convexity of the function $x \log_2 x$)

$$(x+y)\log_2(x+y) - x\log_2 x - y \log_2 y \leqslant x + y, \tag{2.6}$$

applied with $x = \Delta'$, $y = \Delta''$, $x + y = \Delta_{k'+1}$.

2. In the case $k'' = 0$, keeping the notation $\Delta' = n' - r_k$, we have

$$\begin{aligned}\tau(n, R) &= nH(\alpha) + \tau(n', R') \\ &\leqslant nH(\alpha) + n' \log_2 n' + n'' \log_2 n'' - \sum_{i=1}^{k+1} \Delta_i \log_2 \Delta_i \\ &+ \Delta_{k+1} \log_2 \Delta_{k+1} - \Delta' \log_2 \Delta' - n'' \log_2 n'' + r_k - r_1 + 16n' \\ &\leqslant n \log_2 n - \sum_{i=1}^{k+1} \Delta_i \log_2 \Delta_i + r_k - r_1 + 16n' + \Delta_{k+1}\end{aligned}$$

again in view of (2.5) and inequality (2.6) for $x = \Delta'$, $y = n''$, $x+y = \Delta_{k+1}$. It remains to note that $\Delta_{k+1} < n \leqslant 16n''$ by Lemma 2.7. □

Claim 2.9.

$$\sum_{v \in T} (I'_v + I''_v - I_v + q_v - p_v) \leqslant n \log_2(2l). \tag{2.7}$$

▷ Given $q_v \leqslant p'_v + p''_v$, the sum on the left-hand side (2.7) is bounded from above as

$$-I_u - p_u + \sum_{v \in T;\ \deg(v)=1} (I_v + p_v).$$

Here, the summation is performed over the leaves of the tree T, and u is the root node. By assumption, $I_u = 0$ and $p_u = n$. Families associated with different leaves of the tree are pairwise disjoint, so $\sum p_v = n$. Finally, since all chains have length less than $2l$, we obtain $\sum I_v \leqslant n \log_2(2l)$. □

Together, Claims 2.8 and 2.9 yield an estimate for the complexity of merges — step (ii) of the algorithm. Let us proceed to steps (iii) and (iv).

After the merge stage, at node v there are at most l short and n_v/l long chains. Let the selection of the median of k elements be performed in βk comparisons. Insertion of an element (median) into a chain requires at most $\log_2 l$ comparisons, since the half-length of any chain is less than l. Therefore, given $n_v \geqslant 8l^2$, the complexity of steps $(iii), (iv)$ over all internal vertices of the tree is estimated as

$$\sum_{v \in T;\ \deg v > 1} (n_v/l + l)(\log_2 l + \beta) \asymp \frac{\log l}{l} \sum_{v \in T;\ \deg v > 1} n_v.$$

The sum on the right-hand side can be estimated using Claim 2.8, since $n_v \leqslant 3n_v H(\alpha_v)$ due to $H(\alpha_v) \geqslant H(1/16) > 1/3$ according to Claim 2.7.

Summing up the complexity of all steps, we finally obtain

$$\mathsf{C}_{r_1,\ldots,r_k}(n) \leqslant \left(1 + O\left(\frac{\log l}{l}\right)\right) B_{r_1,\ldots,r_k}(n) + O(n \log l).$$

It remains to note that $n \log l = O(n + (B/l) \log l)$. ■

Exercises

Ex. 2.1. For selecting an element of rank $n/4$ from an n-element set, construct an algorithm that has complexity asymptotically less than $3n$. [*D. Dor, U. Zwick*]

Ex. 2.2. Show that if Yao's conjecture is true that the complexity of selecting a subset with a partial order $\mathcal{X}_{k,l}$ (k elements are greater than, and l elements are less than some element) in an m-element set is the same for any $m \geqslant k + l + 1$, then $\mathsf{C}_{n/2}(n) \leqslant 2.5n + o(n)$. [*A. Schönhage, M. Paterson, N. Pippenger*]

Ex. 2.3. Prove a weakened version of Theorem 2.6. Estimate the complexity of an algorithm constructed recursively by selecting the median and splitting the problem in half. [*D. Dobkin, J. Munro*]

Comments. The material of sections 2.1–2.2 is presented in [6], partly in the form of exercises (including a simpler proof of Kislitsyn's Theorem 2.3 than in the original paper [5]). Theorem 2.2 is proved in [7]. Theorem 2.4 is proved in [1]. Theorem 2.5 is proved in [8] with a better estimate of $3n + o(n)$. The result of Theorem 2.6 is obtained in [4].

To improve the bound of Theorem 2.5 to $3n + o(n)$, at the second stage of the construction of a poset $\mathcal{X}_k$, instead of comparing the center with singletons, one should perform comparisons with elements of ordered pairs, and also take into account comparisons between elements of poset fragments that are returned to the factory. Developing a very sophisticated modification of the method, D. Dor and U. Zwick [3] improved the bound further to $2.95n + o(n)$. In [2] they slightly improved the lower bound of Theorem 2.4 asymptotically to $(2 + \varepsilon)n$, where $\varepsilon \approx 2^{-70}$.

Bibliography

[1] Bent S. W., John J. W. *Finding the median requires* $2n$ *comparisons.* Proc. STOC (Providence, USA, 1985). NY: ACM, 1985, 213–216.

[2] Dor D., Zwick U. *Median selection requires* $(2+\epsilon)n$ *comparisons.* Proc. FOCS (Burlington, USA, 1996). Los-Alamitos: IEEE, 1996, 125–134.

[3] Dor D., Zwick U. *Selecting the median.* SIAM J. Comput. 1999. **28**(5), 1722–1758.

[4] Kaligosi K., Mehlhorn K., Munro J. I., Sanders P. *Towards optimal multiple selection.* Proc. ICALP (Lisbon, 2005). Berlin, NY: Springer, 2005, 103–114.

[5] Kislitsyn S. S. *On the selection of the k-th element of an ordered set by pairwise comparisons.* Sibir. Matem. Zhurn. 1964. **5**(3), 557–564. (in Russian)

[6] Knuth D. E. *The art of computer programming. Vol.* 3. *Sorting and searching.* Reading: Addison-Wesley, 1998.

[7] Pohl I. *A sorting problem and its complexity.* Communic. ACM. 1972. **15**(6), 462–464.

[8] Schönhage A., Paterson M., Pippenger N. *Finding the median.* J. Comp. Sys. Sci. 1976. **13**, 184–199.

Theme 3

Comparator circuits

3.1 Introduction

This chapter discusses a popular model of *comparator circuits*. A comparator is a circuit with two inputs x, y and two outputs, at which the functions $\min(x, y)$ and $\max(x, y)$ are computed. A comparator circuit is a special case of a circuit of functional elements. Its elements are comparators, and branching of the circuit inputs and comparator outputs is prohibited (each circuit input or comparator output is used only once). Due to this restriction, the number of circuit outputs coincides with the number of inputs. As a result of the computation, some reordering of the input set is implemented at the circuit output. The most important property of comparator circuits: the choice of elements for the next comparison does not depend on the results of previous comparisons. A comparator circuit corresponds to a special case of a comparison tree, but unlike a general comparison tree, it is suitable for implementation in electronic circuits. Fig. 3.1a shows a standard representation of a 4-element sorting circuit "a la" a circuit of functional elements (bold lines denote the outputs of the comparator maximums).

However, an alternative way of representing a comparator circuit is often more convenient, see Fig. 3.1b. In it, a circuit is depicted as horizontal lines, with inputs coming in from the left and output values appearing on the right. Comparators are shown as jumpers connecting a pair of lines. A jumper sends the smaller of the elements coming in to the comparator input upwards, and the larger one downwards.

The complexity of a circuit Σ is the number of comparators in it; denoted by $\mathsf{S}^*(\Sigma)$. By $\mathsf{S}^*(n)$ we denote the minimum complexity of a circuit sorting n elements. By definition, $\mathsf{S}^*(n) \geqslant \mathsf{S}(n)$.

Another important indicator of the circuit efficiency is its *depth*. This is the number of layers of parallel (i.e. independent) comparators. Alternatively, as in the case of ordinary circuits of functional elements, the depth can be defined as the maximum number of comparators in an input-output chain in the circuit. To denote the circuit

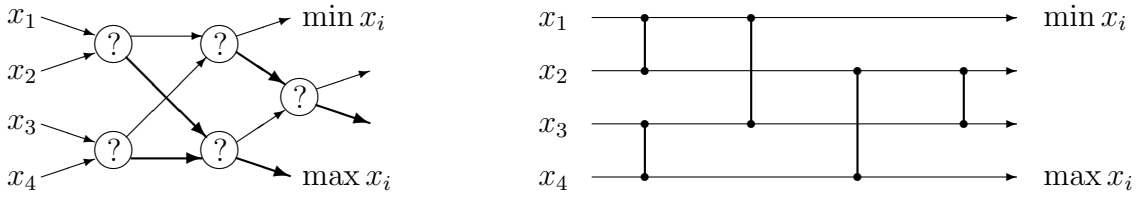

Figure 3.1: Comparator circuit sorting 4 elements: standard (a) and special (b) representations

depth, we use the symbol D. The circuit in Fig. 3.1 has complexity 5 and depth 3.

For the analysis of sorting circuits (in general, circuits sorting posets) the well-known *principle of zeros and ones* is useful.

Lemma 3.1 (D. Knuth)**.** *If a comparator circuit sorts a poset on any input of zeros and ones, then it does it correctly on an arbitrary input.*

▷ Since a comparator maps an input vector (x, y) to the vector $(\min\{x, y\}, \max\{x, y\})$, then for any monotone function f it maps a vector $(f(x), f(y))$ to $(f(\min\{x, y\}), f(\max\{x, y\}))$. Therefore, if a comparator circuit maps an input vector $(x_1, \ldots, x_n)$ to $(y_1, \ldots, y_n)$, then it maps the vector $(f(x_1), \ldots, f(x_n))$ to $(f(y_1), \ldots, f(y_n))$.

Let a comparator circuit map some input $(x_1, \ldots, x_n)$ into an output $(y_1, \ldots, y_n)$, in which the order is violated due to, say, $y_i > y_{i+1}$. Define the function $f : \mathbb{R} \to \{0, 1\}$ as $f(x) = (x \geqslant y_i)$. Then we obtain that the circuit does not order the poset on the boolean input $(f(x_1), \ldots, f(x_n))$. □

3.2 Merging circuits

The limitations of the comparator circuit model become apparent when attempting to construct efficient sorting circuits. Simple approaches, on which fast sorting algorithms are based, do not lead to optimal solutions for comparator circuits. This can be clearly demonstrated on the example of the problem of merging ordered sets. Recall that in the unconstrained model, merging has linear complexity (see Lemma 1.3).

Theorem 3.1 (P. B. Miltersen, M. Paterson, Y. Tarui)**.** $\mathsf{S}^*(\mathcal{L}_{n,n}) > n \log_2 n - O(n)$.

► Let the inputs of the circuit be ordered arrays $x_1 \leqslant \ldots \leqslant x_n$ and $y_1 \leqslant \ldots \leqslant y_n$, and the outputs be $z_1 \leqslant \ldots \leqslant z_{2n}$.

On the set $\Pi_{n,n}$ of permutations $\pi : \{x_1, \ldots, x_n, y_1, \ldots, y_n\} \to \{z_1, \ldots, z_{2n}\}$ consistent with the partial order, we consider some probability distribution. Then $\sigma_k = \pi^{-1}(z_k)$ is a random variable taking values in the set of input variables.

First, recall a well-known fact from information theory. Let σ be a random variable taking values in a finite set $A = \{a_i\}$, where $\mathbf{P}[\sigma = a_i] = p_i$. *Entropy* (information entropy, Shannon entropy) of σ is defined as $H(\sigma) = -\sum p_i \log_2 p_i$. A *prefix code* is a mapping of a set A into a set of words of some alphabet such that no coding word is the beginning of another.

Lemma 3.2 (C. Shannon)**.** *For any prefix encoding of the elements of an alphabet A by binary words, $a_i \to c(a_i)$, we have $\mathbf{E}[|c(\sigma)|] \geqslant H(\sigma)$, where $|b|$ is the length of a word b.*

$\triangleright$ Consider an optimal encoding c that provides the minimum average symbol length $\mathbf{E}[|c(\sigma)|]$. We apply induction on the size of the set A.

In the case $|A| = 2$, a one-bit code is always optimal, for which the inequality $p_1 + p_2 = 1 \geqslant H(\sigma) = H(p_1)$ obviously holds: here the entropy of a random variable coincides with the binary entropy function of one variable.

Now note that with optimal encoding there are two symbols whose codes differ only in the last position. Otherwise, the last digit could be removed from the longest code word.

For $|A| = n > 2$, assume that the codes of the symbols a_{n-1} and a_n differ exactly in the last position. Identifying these symbols ($a_n := a_{n-1}$), we move to the alphabet A' of size $n - 1$. Consider the random variable σ' on A', which differs from σ in that it takes the value a_{n-1} with probability $p_{n-1} + p_n$, and the encoding c', in which the common part of the code words of the two identified symbols is used to encode the symbol a_{n-1}. Then, by the induction assumption,

$$\mathbf{E}[|c(\sigma)|] = p_{n-1} + p_n + \mathbf{E}[|c'(\sigma')|] \geqslant p_{n-1} + p_n + H(\sigma') =$$
$$H(\sigma) + p_{n-1} + p_n + (p_{n-1} + p_n)\log_2(p_{n-1} + p_n) - p_{n-1}\log_2(p_{n-1}) - p_n \log_2(p_n).$$

Due to inequality (2.6), the expression on the right-hand side is not less than $H(\sigma)$. $\square$

Our next goal is to estimate the number of segments R into which horizontal lines are divided by comparators in the minimal merging circuit S. Each segment conducts some input (or output, depending on which side you look at it) variable. Let P_i^π denote the path (set of segments) that an input variable passes to an output z_i under a permutation π. Each such path can be specified as a binary code: scanning the path from right to left, each time a comparator is reached by zero, determine the continuation through the upper input of the comparator, and by one – through the lower input.

Then, to each possible value x of a random variable σ_k, we can associate a code $C_k(x)$ of the shortest path passing through the variable x to the output z_k for some suitable permutation from $\Pi_{n,n}$. For any k, the set of words $C_k(x)$ will be a prefix code, since the paths have different starting (final from the code's point of view) points.

Now, applying Lemma 3.2, for a random permutation π we obtain

$$R = \sum_{k=1}^{2n} \mathbf{E}[|P_k^\pi|] \geqslant 2n + \sum_{k=1}^{2n} \mathbf{E}[|C_k(\sigma_k)|] \geqslant 2n + \sum_{k=1}^{2n} H(\sigma_k). \tag{3.1}$$

It remains to construct a probability distribution on the set $\Pi_{n,n}$ that provides a high lower bound.

Note that the feasible permutations in the merge problem are in one-to-one correspondence with monotone paths in an integer lattice on vertices with coordinates from 0 to n, see Fig. 3.2. The paths start in the lower left corner and end in the upper right. If the next element in order belongs to the group $\{x_i\}$, then we move to the right, if to the group $\{y_j\}$, then up[1]. Conversely, it is easy to reconstruct a permutation from a path.

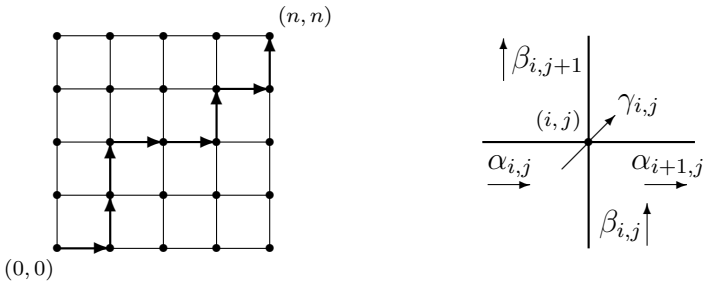

Figure 3.2: Permutations and monotone paths in a lattice

Next, we denote by $\alpha_{i,j}$ and $\beta_{i,j}$ the probabilities that a path passes through the edge $((i-1,\, j),\, (i,\, j))$ and, respectively, through $((i,\, j-1),\, (i,\, j))$, see Fig. 3.2. Set $\gamma_{i,j} = \alpha_{i,j} + \beta_{i,j}$ — the probability that a path visits the vertex $(i,\, j)$.

By construction, $\alpha_{i,j} = \mathbf{P}[z_{i+j} = x_i]$ and $\beta_{i,j} = \mathbf{P}[z_{i+j} = y_j]$. At all vertices $(i,\, j)$, except the starting and final ones, we have

$$\alpha_{i,j} + \beta_{i,j} = \alpha_{i+1,j} + \beta_{i,j+1} = \gamma_{i,j}. \tag{3.2}$$

For the extreme vertices,

$$1 = \gamma_{0,0} = \alpha_{1,0} + \beta_{0,1} = \alpha_{n,n} + \beta_{n,n} = \gamma_{n,n}. \tag{3.3}$$

Conditions (3.2), (3.3) define a *unit flow* (through the lattice). Thus, the probability distribution on $\Pi_{n,n}$ determines a flow. We will show that the correspondence between flows and distributions is one-to-one.

Claim 3.3. *Any unit flow, i.e. a system of nonnegative quantities $\alpha_{i,j}$, $\beta_{i,j}$, satisfying conditions* (3.2), (3.3)*, corresponds to some probability distribution on the set $\Pi_{n,n}$.*

[1]The path shown in Fig. 3.2 corresponds to the permutation $(x_1, y_1, y_2, x_2, x_3, y_3, x_4, y_4)$.

▷ A random path is constructed according to the rule: from a vertex $(i,\, j)$, a move to the right is performed with probability $\alpha_{i+1,j}/\gamma_{i,j}$, and upwards — with probability $\beta_{i,j+1}/\gamma_{i,j}$. Now, by induction on $i+j$, it is easy to verify that the probability of visiting a vertex $(i,\, j)$ is $\gamma_{i,j}$, from which it follows that the proposed probability distribution determines the given flow. □

Let us define the flow by the following rules: for $0 < i + j \leqslant n$ set

$$\alpha_{i,j} = \beta_{j,i} = \alpha_{n+1-i,\,n-j} = \beta_{n-j,\,n+1-i} = \frac{i}{(i+j)(i+j+1)}$$

(the coefficients are symmetric with respect to the diagonals $x = y$ and $x + y = n$). It is easy to verify that the conditions (3.2), (3.3) are satisfied[2].

Noting that $H(\sigma_k) = -\sum_{i=1}^{k} \alpha_{i,k-i} \log_2 \alpha_{i,k-i} - \sum_{i=1}^{k} \beta_{k-i,i} \log_2 \beta_{k-i,i}$, by taking into account the symmetry of the coefficients, we obtain

$$\begin{aligned}
\sum_{k=1}^{2n} H(\sigma_k) &= -4 \sum_{k=1}^{n} \sum_{i=1}^{k} \alpha_{i,k-i} \log_2 \alpha_{i,k-i} = -4 \sum_{k=1}^{n} \frac{1}{k(k+1)} \sum_{i=1}^{k} i \log_2 \frac{i}{k(k+1)} \\
&= 4 \sum_{k=1}^{n} \frac{\log_2(k(k+1))}{k(k+1)} \sum_{i=1}^{k} i - 4 \sum_{i=1}^{n} i \log_2 i \cdot \sum_{k=i}^{n} \frac{1}{k(k+1)} \\
&= 2 \sum_{k=1}^{n} \log_2(k(k+1)) - 4 \sum_{i=1}^{n} \left(1 - \frac{i}{n+1}\right) \log_2 i \\
&= 4 \log_2(n!) + 2 \log_2(n+1) - 4 \log_2(n!) + \frac{4}{n+1} \sum_{i=1}^{n} i \log_2 i \\
&\geqslant 2 \log_2(n+1) + \frac{4}{n+1} \int_1^n x \log_2 x \, dx \\
&= 2 \log_2(n+1) + \frac{1}{n+1} \cdot x^2 (2 \log_2 x - \log_2 e) \Big|_{x=1}^{n} \\
&= 2 \log_2(n+1) + 2 \frac{n^2}{n+1} \log_2 n - (n-1) \log_2 e > 2n \log_2 n - n \log_2 e.
\end{aligned}$$

Since each comparator adds two segments to the circuit, from (3.1) we finally derive

$$\mathsf{S}^*(\mathcal{L}_{n,n}) = (R - 2n)/2 > n \log_2 n - 0.5n \log_2 e.$$

■

The asymptotic accuracy of the obtained estimate is demonstrated by the simple Batcher method.

Theorem 3.2 (K. E. Batcher)**.** $\mathsf{S}^*(\mathcal{L}_{n,n}) \leqslant n\lceil \log_2 n \rceil + n$.

[2]With a uniform distribution, a random path will almost certainly pass close to the diagonal $x = y$. The proposed distribution increases the probability of a path moving away from the diagonal and uniformly distributes paths along the diagonals $x + y = k$. The probability of visiting any point of the lattice on the diagonal $x + y = k$ is the same and equals $1/(\min\{k, 2n - k\} + 1)$.

► Let us construct recursively circuits M_n of merging of size-n ordered sets. The inputs of the circuit M_n are $x_1 \leqslant x_2 \leqslant \ldots \leqslant x_n$ and $y_1 \leqslant y_2 \leqslant \ldots \leqslant y_n$. The circuit is structured as follows. The subcircuits $M_{\lceil n/2 \rceil}$ and $M_{\lfloor n/2 \rfloor}$ sort the groups of elements with odd and even indices, respectively, i.e. $x_1, y_1, x_3, y_3, \ldots$ and $x_2, y_2, x_4, y_4, \ldots$ The result of their work are ordered sets $u_1 \leqslant u_2 \leqslant \ldots \leqslant u_{2\lceil n/2 \rceil}$ and, respectively, $v_1 \leqslant v_2 \leqslant \ldots \leqslant v_{2\lfloor n/2 \rfloor}$. Note that $u_i \leqslant v_i$, since each element with an even index can be associated with a preceding element with an odd index. Thus,

$$u_1 \leqslant v_1, u_2 \leqslant v_2, u_3 \leqslant \ldots$$

Therefore, to complete the sorting, it is sufficient to perform comparisons in pairs v_i, u_{i+1}. So we obtain the relation

$$\mathsf{S}^*(\mathcal{L}_{n,n}) \leqslant \mathsf{S}^*(\mathcal{L}_{\lceil n/2 \rceil, \lceil n/2 \rceil}) + \mathsf{S}^*(\mathcal{L}_{\lfloor n/2 \rfloor, \lfloor n/2 \rfloor}) + n - 1,$$

which, taking into account $\mathsf{S}^*(\mathcal{L}_{1,1}) = 1$, is resolved as stated in the assertion the theorem. ■

An attempt to construct a sorting circuit via merging circuits, as in Theorem 1.2, leads only to circuits of complexity $O(n \log^2 n)$, although they are quite practical. Constructing circuits of complexity $O(n \log n)$ required a nontrivial mathematical apparatus (see in Section 3.5 below).

3.3 Lower bounds on the complexity and depth of sorting

The information-theoretic lower bound $\mathsf{S}(n) \geqslant \log_2 n! \sim n \log_2 n$ is, of course, also valid for comparator circuits. But given the limitations of the computational model of the circuits, it can be strengthened.

A *good potential function* is a continuous function $f(x) : [0,\, 1] \to \mathbb{R}$, for which the following conditions are satisfied:

(1) $f(0) = f(1) = 0$;

(2) For any p, q, r such that $0 \leqslant pq \leqslant r \leqslant p \leqslant q \leqslant 1$,

$$f(p) + f(q) - f(r) - f(p + q - r) \leqslant p + q - 2r. \tag{3.4}$$

Theorem 3.3 (N. Kahale, T. Leighton, Y. Ma, C. G. Plaxton, T. Suel, E. Szemerédi)**.** *If $f(x)$ is a good potential function, then $\mathsf{S}^*(n) \geqslant (f(1/2) - o(1))n \log_2 n$.*

► Consider an arbitrary circuit Σ from the set $\mathbf{\Sigma_n}$ of n-input sorting circuits and some input vector $\alpha \in \{0,\, 1\}^n$. Denote by $a(\Sigma, \alpha)$, $b(\Sigma, \alpha)$ and $c(\Sigma, \alpha)$ the number of comparators in the circuit whose input values are both equal to 1, both equal to 0 and different, respectively. Such comparators will be called 11-comparators, 00-comparators and 01-comparators. Further $|\alpha|$ denotes the weight of a boolean vector α.

Lemma 3.4. *Let* $\alpha \in \{0, 1\}^n$. *If* $\Sigma \in \mathbf{\Sigma_n}$, *then* $a(\Sigma, \alpha) \geqslant \mathsf{S}^*(|\alpha|)$ *and* $b(\Sigma, \alpha) \geqslant \mathsf{S}^*(n - |\alpha|)$.

$\triangleright$ Let us prove the first inequality (the second is proved similarly). To do this, we will show that from the set of 11-comparators of Σ, it is possible to compose a k-input sorting circuit.

Remove the circuit inputs corresponding to the zeros of the vector α. Next, sequentially, moving from the inputs to the outputs, remove the 00-comparators together with the edges coming out of them, as well as the 01-comparators, connecting their 1-valued inputs to the outputs on which maximums are obtained and removing the edges outgoing from the minimum outputs. As a result, we will construct a compatator circuit[3] Σ_1 with k inputs and complexity exactly $a(\Sigma, \alpha)$.

In this case, a natural one-to-one correspondence is obtained between:

– inputs of the circuit Σ_1 and a subset of inputs (corresponding to 1s of the vector α) of the circuit Σ;

- comparators of the circuit Σ_1 and 11-comparators of the circuit Σ;

- outputs of Σ_1 and the subset O_k of outputs that produce the maximum k values in the circuit Σ.

Now consider an arbitrary size-n input vector β in which the maximum k elements are at the positions of 1s of the vector α. By feeding the corresponding components of β to the corresponding inputs of the circuits Σ and Σ_1, we obtain that the inputs and outputs of the corresponding comparators in both circuits take the same values, and therefore the corresponding outputs of both circuits also take the same values. Since the outputs O_k of the circuit Σ contain an ordering of the maximal k elements of the vector β, the same is true for the outputs of the circuit Σ_1. Therefore, due to the arbitrariness of the choice of β, we conclude that Σ_1 is a sorting circuit. $\square$

Denote

$$C(n) = \min_{\Sigma \in \mathbf{\Sigma_n}} \max_{\alpha \in \{0,1\}^n} c(\Sigma, \alpha).$$

Lemma 3.5.

$$\mathsf{S}^*(n) \geqslant \min_{0<i<n} (\mathsf{S}^*(i) + \mathsf{S}^*(n-i)) + C(n).$$

$\triangleright$ Let us choose as Σ a circuit that delivers the minimum of $\mathsf{S}^*(n)$. By the definition of $C(n)$, there exists a vector $\alpha \in \{0, 1\}^n$ such that $c(\Sigma, \alpha) \geqslant C(n)$. Let $k = |\alpha|$. Then by Lemma 3.4,

$$\begin{aligned}\mathsf{S}^*(n) &= a(\Sigma, \alpha) + b(\Sigma, \alpha) + c(\Sigma, \alpha) \\ &\geqslant \mathsf{S}^*(k) + \mathsf{S}^*(n-k) + C(n) \geqslant \min_{0<i<n} (\mathsf{S}^*(i) + \mathsf{S}^*(n-i)) + C(n).\end{aligned}$$

$\square$

[3]If we depict the circuits as in Fig. 3.1, it is easy to see that the reduced circuit is placed on k lines that can change a level in the positions of the 01-comparators of the original circuit.

Claim 3.6. *If* $C(n) \geqslant (\gamma - o(1))n$, *then* $\mathsf{S}^*(n) \geqslant (\gamma - o(1))n \log_2 n$.

▷ Let us show that for any $\varepsilon > 0$, we have $\mathsf{S}^*(n) \geqslant (\gamma - \varepsilon)n \log_2 n$ for all $n \geqslant n_0(\varepsilon)$.

Assume that $C(n) \geqslant (\gamma - \varepsilon/2)n$ holds for $n \geqslant n_1$, and let the constant c_ε be chosen so that for $n \leqslant n_1$,

$$\mathsf{S}^*(n) \geqslant (\gamma - \varepsilon/2)n \log_2 n - c_\varepsilon n. \tag{3.5}$$

We prove (3.5) by induction for any $n > n_1$.

Let k provide the minimum of $\mathsf{S}^*(k) + \mathsf{S}^*(n-k)$. Denoting $\gamma' = \gamma - \varepsilon/2$, according to Lemma 3.5 and the induction hypothesis, we obtain

$$\begin{aligned}\mathsf{S}^*(n) &\geqslant \mathsf{S}^*(k) + \mathsf{S}^*(n-k) + \gamma' n \\ &\geqslant \gamma' k \log_2 k + \gamma'(n-k)\log_2(n-k) + \gamma' n - c_\varepsilon k - c_\varepsilon(n-k) \geqslant \gamma' n \log_2 n - c_\varepsilon n,\end{aligned}$$

where in the last step we applied inequality (2.6).

Now we can choose n_0 from the condition $(\varepsilon/2)n_0 \log_2 n_0 \geqslant c_\varepsilon n_0$. □

In view of what has been proved, it remains to obtain a good lower bound for $C(n)$. Instead of $C(n)$, it is more convenient to consider an average quantity

$$\bar{C}(n) = \min_{\Sigma \in \mathbf{\Sigma_n}} \left(\frac{1}{2^n} \sum_{\alpha \in \{0,1\}^n} c(\Sigma, \alpha) \right).$$

Obviously, $C(n) \geqslant \bar{C}(n)$.

Let $h(x_1, \ldots, x_n)$ be a boolean function. Denote by $|h|$ the number of inputs on which the function is 1. Recall the following well-known fact from the field of boolean cube combinatorics:

Lemma 3.7. *Let* u, v *be monotone boolean functions of* n *variables. Then* $2^n|uv| \geqslant |u||v|$.

This statement follows from a more general fact. For the function $\varphi(x) : \{0,1\}^n \to \mathbb{R}_{\geqslant 0}$ we introduce the notation $|\varphi| = \sum_{x \in \{0,1\}^n} \varphi(x)$. The following holds:

Theorem 3.4 (R. Ahlswede, D. E. Daykin)**.** *Let* α, β, γ *be nonnegative functions such that for any* $x, y \in \{0,1\}^n$,

$$\alpha(x)\beta(y) \leqslant \gamma(x \vee y),$$

where $\vee$ *denotes the bitwise disjunction of binary vectors. Then* $|\alpha||\beta| \leqslant 2^n|\gamma|$.

Note that Lemma 3.7 follows from this theorem under the substitution $\alpha = u$, $\beta = v$, $\gamma = uv$, since by monotonicity, $u(x \vee y)v(x \vee y) \geqslant u(x)v(y)$, and the two definitions for $|\varphi|$ coincide in the case of a monotone boolean function.

► Proof is by induction on n. Consider the case $n = 1$. For any function $\varphi \in \{\alpha, \beta, \gamma\}$, denote $\varphi_x = \varphi(x)$, $x \in \{0, 1\}$. By the induction hypothesis,

$$\alpha_0\beta_0 \leqslant \gamma_0, \qquad \alpha_0\beta_1, \alpha_1\beta_0, \alpha_1\beta_1 \leqslant \gamma_1.$$

The relation to be proved in this case has the form

$$(\alpha_0 + \alpha_1)(\beta_0 + \beta_1) \leqslant 2(\gamma_0 + \gamma_1).$$

If $\gamma_1 = 0$, then it is obvious. Otherwise, from

$$(\gamma_1 - \alpha_0\beta_1)(\gamma_1 - \alpha_1\beta_0) \geqslant 0$$

it follows

$$(\alpha_0 + \alpha_1)(\beta_0 + \beta_1)\gamma_1 \leqslant (\gamma_1 + \alpha_0\beta_0)(\gamma_1 + \alpha_1\beta_1).$$

Dividing both sides by $\gamma_1 > 0$, we obtain

$$(\alpha_0 + \alpha_1)(\beta_0 + \beta_1) \leqslant (\gamma_1 + \alpha_0\beta_0)\left(1 + \frac{\alpha_1\beta_1}{\gamma_1}\right) \leqslant (\gamma_0 + \gamma_1)(1 + 1),$$

which is what we needed.

Now, consider the induction step from $n-1$ to n. We will write a vector $x \in \{0, 1\}^n$ as $\bar{x}, x^*$, where $\bar{x} \in \{0, 1\}^{n-1}$, and $x^* \in \{0, 1\}$, i.e., selecting a single component x^* of the vector x. From the function $\varphi \in \{\alpha, \beta, \gamma\}$ derive the function $\varphi'(y) = \varphi(y, 0) + \varphi(y, 1)$ for any $y \in \{0, 1\}^{n-1}$.

1. Let us show that $\alpha'(y)\beta'(z) \leqslant 2\gamma'(y \vee z)$ for any $y, z \in \{0, 1\}^{n-1}$. To do this, having fixed y and z, we consider the functions $\bar{\alpha}(x) = \alpha(y, x)$, $\bar{\beta}(x) = \beta(z, x)$, $\bar{\gamma}(x) = \gamma(y \vee z, x)$, where $x \in \{0, 1\}$. For $\bar{\alpha}, \bar{\beta}, \bar{\gamma}$ the conditions of the theorem are satisfied in the case $n = 1$, for which it has already been proved, thus,

$$\alpha'(y)\beta'(z) = (\bar{\alpha}(0) + \bar{\alpha}(1))(\bar{\beta}(0) + \bar{\beta}(1)) \leqslant 2(\bar{\gamma}(0) + \bar{\gamma}(1)) = 2\gamma'(y \vee z).$$

2. Now we can apply the induction hypothesis to the set of functions $\alpha', \beta', 2\gamma'$ and obtain $|\alpha'||\beta'| \leqslant 2^{n-1}|2\gamma'|$, and hence, since $|\varphi'| = |\varphi|$, the desired inequality $|\alpha||\beta| \leqslant 2^n|\gamma|$. ■

Recall that the output of any comparator circuit implements a monotone function of the input variables, in particular, a monotone boolean function if the inputs take values from the set $\{0, 1\}$. This follows from the fact that the function implemented at the output is a composition of the monotone functions min and max, implemented by individual comparators.

Let x_i and y_i denote the probability that an i-th input (respectively, output) of a comparator circuit Σ takes the value 1 under the assumption that random variables uniformly distributed on the set $\{0, 1\}$ are fed to the inputs of the circuit. In this case, the probability of event X coincides with the ratio of the number of input vectors for which X is satisfied to the number of all vectors, that is, 2^n for n dimensions. Denote $f_I(\Sigma) = \sum_{i=1}^n f(x_i)$ and $f_O(\Sigma) = \sum_{i=1}^n f(y_i)$, where f is a good potential function.

Lemma 3.8. *Let Σ' be the comparator circuit obtained by connecting a comparator e to the outputs of Σ. If f is a good potential function, then the probability that e is a* 01*-comparator is no less than $f_O(\Sigma) - f_O(\Sigma')$.*

$\triangleright$ Let p and q be the probabilities that the first and, respectively, the second input of comparator e is 1, and let r be the probability that both inputs are equal to 1. Then the probability that the smaller output of e is equal to 1 is r, and that the larger output is equal to 1 is $p + q - r$. Since the outputs implement monotone functions, $r \geqslant pq$ by Lemma 3.7. On the other hand, $r \leqslant \min\{p, q\}$. Therefore, we obtain

$$f_O(\Sigma) - f_O(\Sigma') = f(p) + f(q) - f(r) - f(p + q - r) \leqslant p + q - 2r,$$

where $p + q - 2r$ is precisely the probability that e is a 01-comparator. $\square$

Corollary 3.9. *In an arbitrary circuit Σ, the average number of* 01*-comparators is at least $f_I(\Sigma) - f_O(\Sigma) = nf(1/2) - f_O(\Sigma)$.*

Lemma 3.10. *If f is a good potential function, then $\bar{C}(n) \geqslant (f(1/2) - o(1))n$.*

In order to obtain also a good depth estimate, we will prove this lemma in a stronger formulation, as applied to the approximate sorting problem. Let $\mathbf{\Sigma_{n,r}}$ consist of circuits obtained from sorting circuits by removing the last r layers of comparators, and let $\bar{C}_r(n)$ denote the minimum average number of 01-comparators in such circuits. First, we prove an auxiliary lemma.

Lemma 3.11. *If elements of ranks r and s can appear at some output of a comparator circuit, then (on suitable input vectors) elements of any intermediate rank can appear at the same output.*

$\triangleright$ Assuming that a set of numbers from 1 to n is fed to the inputs of the circuit, with each input set v we associate a permutation $\pi([n]) = (\pi(1), \pi(2), \ldots, \pi(n))$, where $\pi(k)$ is the number of line which receives the number k.

Let π_1 and π_2 be the permutations corresponding to the input vectors for which the numbers r and s appear at the output of interest to us. It is well known that one permutation can be transformed into another via transpositions of the form $\pi(i) \leftrightarrow \pi(i + 1)$.

As a result of applying a transposition to the input set, the values at the comparators' outputs change by no more than 1: in fact, all possible changes are reduced to transformations of the numbers i and $i + 1$ into one another. Consequently, at the output of the circuit under consideration, in the process of transforming π_1 into π_2 by means of transpositions, every number in the interval between r and s will appear. $\square$

Lemma 3.12. *If f is a good potential function, and $r \leqslant \log_2 n - \omega(n)$, then $\bar{C}_r(n) \geqslant (f(1/2) - o(1))n$.*

▷ As follows from Corollary 3.9, it suffices to show that for an arbitrary circuit $\Sigma \in \mathbf{\Sigma_{n,r}}$ and a good potential function f, we have $f_O(\Sigma) = o(n)$. Let the sorting circuit $\Sigma' \in \mathbf{\Sigma_n}$ be obtained from Σ by adding r layers of comparators.

Let $t = \max\{2^r, \sqrt{n \log_2 n}\}$. In the set of outputs of Σ', we distinguish three groups: group G_1 of outputs producing $n/2 - 2t$ maximal elements, group G_2 of outputs producing $n/2 - 2t$ minimal elements, and group G_3 of outputs producing $2t$ middle elements.

Since the output of the circuit Σ is connected by directed paths with at most 2^r outputs of the circuit Σ', then by Lemma 3.11 no output of Σ can be connected with outputs from two different groups G_1, G_2, G_3. By H_1 and H_2 we denote the sets of outputs of Σ connected with the groups of outputs G_1 and, respectively, G_2. By construction, $|H_i| \geqslant n/2 - 2t$.

Recall the standard estimate of the sum of binomial coefficients (a variant of Chernoff's inequality):

$$\sum_{k=0}^{n/2-t} C_n^k \leqslant 2^n e^{-2t^2/n}.$$

It follows from this that the proportion of length-n vectors in which there are less than $n/2 - t$ ones or less than $n/2 - t$ zeros is $o(1)$. Therefore, for any output from H_1, the probability that it takes the value 1 is $1 - o(1)$, and for an output from H_2, the similar probability is $o(1)$.

Thus, $f_O(\Sigma)$ can be estimated as

$$f_O(\Sigma) \leqslant (n/2 - 2t)\,(f(o(1)) + f(1 - o(1))) + 4t \max_{x \in [0,\,1]} f(x) = o(n),$$

since $f(o(1)), f(1-o(1)) \in o(1)$ due to the continuity of f, and $\max_{x\in[0,\,1]} f(x) = 1$. □

Now Theorem 3.3 follows from the proved lemma and Claim 3.6. ■

In order to obtain a nontrivial lower bound by Theorem 3.3, it remains to present a good potential function with a high value at 1/2. The simple examples $\min\{x,\, 1-x\}$ and $4(x - x^2)$ are not suitable for this purpose. Therefore, we turn to polynomials of higher degrees.

Corollary 3.13. $\mathsf{S}^*(n) \geqslant (C - o(1))n \log_2 n$, *where* $C = 25/22 \approx 1.136$.

▷ First, note that if $f(0) = f(1) = 0$ and $f''(x) \leqslant 0$, then f is a good potential function if condition (2) is satisfied for $r = pq$. Indeed, the difference between the right and left sides (3.4) as a function of the variable r is convex upward: its second derivative is $f''(r) + f''(p+q-r) \leqslant 0$, and a convex upward function can take minimal values on a segment only at the ends of the segment. Since inequality (3.4) is certainly satisfied for $r = p$, it remains to check it for $r = pq$.

1. We will search for f in the form $\gamma \cdot \hat{f}(x - 1/2)$, where $\hat{f}(y) = \frac{1}{16} + \frac{a}{4} - ay^2 - y^4$. The form of the function is determined by natural requirements: $\hat{f}$ is symmetric with

respect to the point $y = 0$, where it has a local maximum, and is zero at the points $y = \pm\frac{1}{2}$. We will determine the parameter $a \geqslant 0$ later. The constant γ is chosen so that inequality (3.4) turns into equality at $p = q = 1/2$ and $r = pq = 1/4$ (we assume this case is extremal). Hence, $\gamma = \frac{64}{16a+1}$.

By construction, condition (1) is satisfied. It is also easy to see that $\hat{f}''(y) = -2a - 12y^2 \leqslant 0$ for all y. It remains to check condition (2) for $r = pq$.

2. In the notations $y = (p+q-1)/2$ and $x = (q-p)/2$, inequality (3.4) for $r = pq$ may be rewritten as

$$\hat{f}(y-x) + \hat{f}(y+x) - \hat{f}((y+1/2)^2 - x^2 - 1/2) - \hat{f}(2y - (y+1/2)^2 + x^2 + 1/2)$$
$$= D_y(x) - D_y(1/4 - y^2 + x^2) \leqslant \frac{16a+1}{32}(1/4 - y^2 + x^2), \quad (3.6)$$

where $D_y(x) = \hat{f}(y-x) + \hat{f}(y+x) = \frac{1}{8} + \frac{a}{2} - 2ay^2 - 2ax^2 - 2y^4 - 2x^4 - 12x^2y^2$.

3. First, we ensure that the difference between the left and right sides of (3.6) is maximal at $x = 0$ (i.e. when $p = q$). To do this, it suffices to show that

$$D_y(0) - D_y(1/4 - y^2) - D_y(x) + D_y(1/4 - y^2 + x^2) \geqslant \frac{16a+1}{32}x^2. \quad (3.7)$$

We have

$$D_y(0) - D_y(x) = (2a + 12y^2)x^2 + 2x^4.$$

Denote $Y = 1/4 - y^2$. Due to $Y \leqslant 1/4$ and the convexity of the functions x^2 and x^4, we obtain

$$D_y(Y) - D_y(Y + x^2) = (2a + 12y^2)((Y + x^2)^2 - Y^2) + 2((Y + x^2)^4 - Y^4)$$
$$\leqslant (2a + 12y^2)(x^4 + x^2/2) + 2(x^8 + x^6 + 3x^4/8 + x^2/16)$$
$$= (a + 1/8 + 6y^2)x^2 + (2a + 3/4 + 12y^2)x^4 + 2x^6 + 2x^8.$$

Now the left-hand side in (3.7) is estimated from below as

$$(a - 1/8 + 6y^2)x^2 - (2a - 5/4 + 12y^2)x^4 - 2x^6 - 2x^8.$$

Since $x \leqslant 1/2$, we have $y^2x^2 \geqslant 4y^2x^4$, so the above expression is minimal at $y = 0$. Now the validity of (3.7) is ensured by the inequality

$$\left(\frac{a}{2} - \frac{5}{32}\right)X - \left(2a - \frac{5}{4}\right)X^2 - 2X^3 - 2X^4 = X\left(\frac{1}{4} - X\right)\left(2X^2 + \frac{5}{2}X + 2a - \frac{5}{8}\right) \geqslant 0$$

for $X = x^2 \in [0,\, 1/4]$ and under the additional restriction $a \geqslant 5/16$.

4. It remains to prove (3.6) for $x = 0$. It holds

$$D_y(0) - D_y(Y) - \frac{16a+1}{32}Y = (2a + 12y^2)Y^2 + 2Y^4 - \left(\frac{a}{2} + \frac{1}{32}\right)Y$$
$$= (2a+3-12Y)Y^2 + 2Y^4 - \left(\frac{a}{2} + \frac{1}{32}\right)Y = Y\left(Y - \frac{1}{4}\right)\left(2Y^2 - \frac{23}{2}Y + 2a + \frac{1}{8}\right) \leqslant 0$$

for $Y \in [0,\, 1/4]$, if only the last quadratic factor $P(Y)$ is nonnegative. Since it takes its minimum value at the right end of the segment, for $Y = 1/4$, it is sufficient to require $P(1/4) \geqslant 0$. So we obtain $a \geqslant 21/16$. Then, condition (2) is satisfied.

Finally, to maximize the value $f(1/2) = \gamma \hat{f}(0) = 1 + \frac{3}{16a+1}$, we choose $a = 21/16$. □

Lemma 3.12 and Corollary 3.13 imply a lower bound for the depth of sorting circuits (note that at most $n/2$ comparators can be placed on one layer).

Theorem 3.5. $\mathsf{D}(n) \geqslant (36/11 - o(1)) \log_2 n$.

3.4 Lower bounds on the complexity and depth of selection

The following method provides decent bounds on the complexity and depth of selection. Let the layers of comparators divide the horizontal lines of the circuit into segments. Associate *weights* with the line segments: let $w_{l,j}$ refer to the l-th segment of line j, see Fig. 3.3. By $[i : j]$ we further denote the comparison of elements on the i-th and j-th lines.

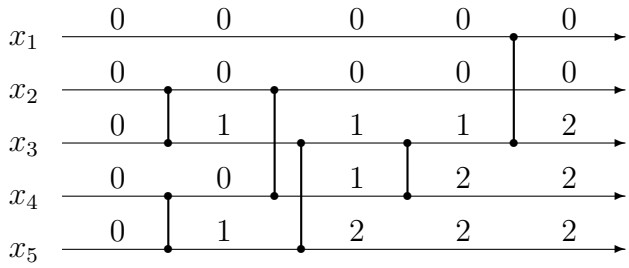

Figure 3.3: Comparator circuit with weights assigned to segments

1) Set $w_{0,j} = 0$ for all j.

2) If there are no comparators in the l-th layer connected to line j, set $w_{l+1,j} = w_{l,j}$.

3) If there is a comparator $[i : j]$ in the l-th layer, where $i < j$, set $w_{l+1,i} = \min\{w_{l,i},\, w_{l,j}\}$ and $w_{l+1,j} = \max\{w_{l,i},\, w_{l,j}\} + 1$.

Lemma 3.14. *To ensure that the j-th output of a comparator circuit always returns an element of rank $\geqslant r$, it is necessary that the weight of the output segment of line j be at least* $\log_2 r$.

▷ Let $v_{l,j}$ denote the minimal possible rank of an element on the l-th segment of line j. Obviously, $v_{0,j} = 1$. Let the comparator $[i : j]$, where $i < j$, be in the l-th layer of the circuit. Then

$$v_{l+1,i} = \min\{v_{l,i},\, v_{l,j}\}, \qquad v_{l+1,j} \leqslant v_{l,i} + v_{l,j}.$$

The first relation is obvious, the second follows from the estimate of the cardinality of the union of the sets of elements subordinate to the two compared ones.

It remains to note that by the definition of weight, $2^{w_{l,j}} \geqslant v_{l,j}$. □

Further, the complexity and depth of selecting an element of rank t are denoted by $\mathsf{C}^*_t(n)$ and $\mathsf{D}_t(n)$. Since the sum of weights of output segments is equal to the complexity of the circuit, Lemma 3.14 immediately implies

Theorem 3.6 (V. E. Alekseev)**.** $\mathsf{C}^*_t(n) \geqslant (n-t)\log_2 t$.

The maximum estimate, asymptotically $n\log_2 n$, is obtained for selecting an element of rank $t \asymp n/\log n$.

A good depth lower bound requires a little more careful argument. We show that the weights of the segments cannot, all together, grow too quickly from layer to layer.

Consider an arbitrary n-input comparator circuit. Let $x_{l,k} = |\{j \mid w_{l,j} = k\}|$ be the number of lines with l-th segment weights equal to k. We also introduce an auxiliary quantity

$$y_{l,k} = \sum_{i=0}^{k}(k+1-i)x_{l,i}.$$

It has the meaning of the total "shortage" of the l-th segment weights to the threshold value $k+1$. By construction, $y_{0,k} \geqslant y_{1,k} \geqslant y_{2,k} \geqslant \ldots$ The proof strategy requires that $y_{l,k}$ be estimated from below.

Claim 3.15.

$$y_{l,k} \geqslant \frac{n}{2^l}\sum_{j=0}^{k}(k+1-j)C_l^j.$$

▷ First, note that

$$y_{l,k} - y_{l+1,k} \leqslant (x_{l,0} + x_{l,1} + \ldots + x_{l,k})/2. \tag{3.8}$$

This is true because there are at most $s/2$ comparators attached to s lines on a single layer.

Next, we note that by definition of $y_{l,k}$,

$$y_{l,0} = x_{l,0}, \qquad y_{l,k} - y_{l,k-1} = x_{l,0} + x_{l,1} + \ldots + x_{l,k}. \tag{3.9}$$

From here we find that $y_{l+1,0} \geqslant y_{l,0}$ and in view of (3.8), (3.9),

$$y_{l+1,k} \geqslant y_{l,k} - (y_{l,k} - y_{l,k-1})/2 = (y_{l,k} + y_{l,k-1})/2.$$

Thus, for the normalized quantities $y'_{l,k} = 2^l y_{l,k}/n$, we have

$$y'_{0,k} = k+1, \qquad y'_{l+1,k} \geqslant y'_{l,k} + y'_{l,k-1}.$$

Here the first relation provides the base of induction $l = 0$, and the second — the induction step from l to $l + 1$:

$$y'_{l+1,k} \geqslant \sum_{j=0}^{k}(k+1-j)C_l^j + \sum_{j=1}^{k}(k+1-j)C_l^{j-1} = \sum_{j=0}^{k}(k+1-j)C_{l+1}^j.$$

□

Corollary 3.16. *For a depth-d circuit selecting an element of rank t in an n-element set,*

$$t(\lfloor \log_2 t \rfloor + 1) \geqslant n2^{-d}\left(C_d^0 + C_d^1 + \ldots + C_d^{\lfloor \log_2 t \rfloor}\right). \tag{3.10}$$

▷ Estimate $y_{d,\lfloor \log_2 t \rfloor + 1}$ with the use of Lemma 3.14 on the one hand, and by Claim 3.15 on the other. □

For fixed n, t, the right-hand side of inequality (3.10) is a decreasing function with respect to d. Therefore, the maximum d for which the inequality does not hold serves as a lower bound for the depth. We restrict ourselves to the extreme case of selecting the median.

Let $\mathsf{D}(n, t)$ denote the depth of selecting an element of rank t in an n-element set.

Theorem 3.7 (A. Yao)**.** $\mathsf{D}(n, n/2) \geqslant (a - o(1)) \log_2 n$, *where* $a = \log_{4/3} 2 \approx 2.41$.

► To estimate the depth of a selection circuit, we choose $t = n^\alpha$. Let $d = \beta \log_2 n$. Taking the base-n logarithm of (3.10), we obtain the condition under which d is a lower bound for the depth of the selection of a rank-t element for sufficiently large n:[4]

$$\alpha < 1 - \beta + \beta H(\alpha/\beta).$$

When substituting $\beta = 3\alpha$, the specified condition becomes $\alpha < 1/(4 - 3H(1/3)) = a/3$. Therefore, for some $\alpha = a/3 - o(1)$, inequality (3.10) is violated.

Thus, the desired bound is proved for the selection of an element of rank $t = n^\alpha$. It remains to note that $\mathsf{D}(n, t) \leqslant \mathsf{D}(2n - 2t, n - t)$. Indeed, if in a circuit for selecting the median of $2n - 2t$ elements we fix the lines of the smallest $n - 2t$ elements and remove the comparators connected to them, then the rest of the circuit will perform the selection of an element of rank t. ■

3.5 Fast sorting (AKS-circuits)

This section presents the design of comparator circuits, now known as AKS-circuits[5]. The method exploits several ideas at once, the central one being the idea of approximate computations. It turned out to be advantageous to construct sorting circuits from subcircuits that perform sorting approximately, with a controlled error probability.

[4]Recall that $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function. In what follows, we use the well-known fact: $C_n^{\alpha n} = 2^{n(H(\alpha)-o(1))}$ for $\alpha > 0$.

[5]By the names of the authors of the method: M. Ajtai, J. Komlós, E. Szemerédi.

Let us introduce the concept of approximate sorting. Assume $0 < \varepsilon \leqslant 1$ and $0 < \lambda \leqslant 1/2$. A comparator circuit on n inputs is called a (λ, ε)-*separator* if for any $m \leqslant \lambda n$ the circuit places at least $(1-\varepsilon)m$ of the m largest elements among the λn right outputs and at least $(1-\varepsilon)m$ of the m smallest elements — among the λn left outputs. The following two lemmas establish the existence of separators of constant depth (and hence of linear complexity).

Lemma 3.17. *For any $\varepsilon > 0$ and any n, there exists a $(1/2, \varepsilon)$-separator*[6] *on $2n$ inputs of depth $O(1/\varepsilon^3)$ as $\varepsilon \to 0$.*

$\triangleright$ If n is small, say, $n < 64/\varepsilon^3$, then apply any sorting circuit. Therefore, we further assume that $n \geqslant 64/\varepsilon^3$.

We will show that the required circuit can be composed of several layers of comparators, where at each layer the elements from the junior (left) and senior (right) halves are compared according to a randomly selected matching. An example of a circuit is shown in Fig. 3.4a.

Let us associate such a circuit with a bipartite (n, n)-graph: the vertices of one part correspond to the positions of elements from the junior half, and the other – from the senior half. Two vertices of the graph are connected by an edge if a comparison of the corresponding elements was performed at some layer of the circuit, see Fig. 3.4b. In other words, the graph is a composition of matchings that define the layers of the circuit.

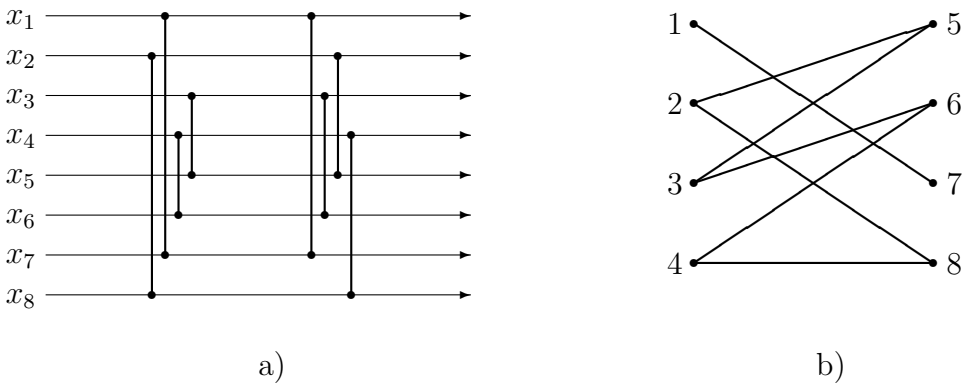

Figure 3.4: Comparator circuit (a) and its graph (b)

I. First, we prove that a circuit is a $(1/2, \varepsilon)$-separator if its graph is an (ε, α)-*expander*, $\alpha = 1/\varepsilon - 1/2$, which means: for every $m \leqslant \varepsilon n$, any set of m vertices in one part is connected by edges to at least αm vertices in the other part.

[6] $(1/2, \varepsilon)$-separators are also called ε-halvers.

Note that if two nodes are connected by an edge in the graph, then the functions computed at the corresponding outputs of the circuit satisfy the relation $\leqslant$ (one part collects only the minima of ordered pairs, and the other only the maxima).

Now suppose that the circuit is not a separator, i.e., for some input set, say, among $k \leqslant n$ largest elements $p > \varepsilon k$ are placed into the junior half. Consider the set of nodes in the graph corresponding to the latter elements. In this set $m = \min\{p, \lfloor \varepsilon n \rfloor\}$ nodes are connected by edges with at least αm nodes in the other part. This means that the minimum of p elements is inferior to at least $p - 1 + \alpha m$ other elements. But for $m = p$ we have

$$p - 1 + \alpha m = p - 1 + m/\varepsilon - m/2 > p/\varepsilon - 1 > k - 1,$$

and for $m = \lfloor \varepsilon n \rfloor < p$,

$$p-1+m/\varepsilon-m/2 > (p-1)/2+m/\varepsilon > (\varepsilon n-1)(1/\varepsilon+1/2) > (1+\varepsilon/2)n-1/\varepsilon-1 \geqslant n-1.$$

This contradicts the fact that the chosen element is among the k largest.

II. It remains to prove that a bipartite graph composed of a suitable number $r = r(\varepsilon)$ of random matchings[7] is an (ε, α)-expander with a positive probability.

Note that a graph is an (ε, α)-expander if it does not contain empty (i.e. edgeless) $(k, n - \alpha k)$-subgraphs for any $k \leqslant \varepsilon n$. The probability that a random matching does not intersect (by edges) a given $(k, n - \alpha k)$-subgraph is $C^k_{\alpha k}/C^k_n$. Then the probability P_k that r random matchings do not intersect at least one of the $(k, n-\alpha k)$-subgraphs is estimated with the help of simple relations $\frac{1}{4\sqrt{k}}(\frac{en}{k})^k \leqslant C^k_n \leqslant (\frac{en}{k})^k$ as

$$P_k \leqslant 2C^k_n C^{\alpha k}_n \left(\frac{C^k_{\alpha k}}{C^k_n}\right)^r \leqslant 2(4\sqrt{k})^r \left(\frac{e^{1+\alpha} n^{1+\alpha} (\alpha k)^r}{k^{1+\alpha} \alpha^\alpha n^r}\right)^k =$$
$$2(4\sqrt{k})^r \left(\alpha e^{1+\alpha} \left(\frac{\alpha k}{n}\right)^{r-\alpha-1}\right)^k \leqslant \left(c_2 \left(\frac{c_1 \alpha k}{n}\right)^{r-\alpha-1}\right)^k,$$

where $c_1 = (4\sqrt{k})^{1/k} \leqslant 4$ and $c_2 = 2^{1/k}\alpha(c_1 e)^{1+\alpha} \leqslant (8e)^{1+\alpha}$.

For $k \leqslant \varepsilon n/4$, we have $c_1 \alpha k \leqslant (1 - \varepsilon/2)n$. Otherwise $k \geqslant \varepsilon n/4 \geqslant 16/\varepsilon^2$, so $c_1 = e^{\ln(16k)/2k} < 1 + \ln(16k)/k \leqslant 1 + \frac{\varepsilon^2}{8} \ln \frac{16}{\varepsilon} \leqslant 1 + \varepsilon/2$. Therefore, $c_1 \alpha k < (1 - \varepsilon^2/4)n$. In either case, if r is chosen somewhat larger than $\alpha + 4\ln(2c_2)/\varepsilon^2$, we obtain $P_k < 2^{-k}$. Then the probability that the graph under consideration is not an (ε, α)-expander does not exceed $\sum_k P_k < 1$. □

It is easy to check that any r-regular (all vertices have degree r) bipartite graph is a union of r matchings, so a circuit can be constructed from a graph.

Lemma 3.18. *For any constant $\varepsilon > 0$, any $\lambda < 1/2$ and n, there exists a (λ, ε)-separator on $2n$ inputs of depth $O(\log^4(1/\lambda))$ as $\lambda \to 0$.*

[7]The distribution is uniform: all matchings are equally probable.

▷ Let $s = \lfloor 2\lambda n \rfloor$ and $k = \lceil \log_2(1/\lambda) \rceil$. The circuit is composed of $k+1$ layers of $(1/2, \varepsilon_0)$-separators (the parameter ε_0 will be chosen later): on the first layer — a separator on $2n$ inputs, on the next two layers — two separators on the left and right for $2^{k-1}s$ marginal elements[8], on the next layer — separators for $2^{k-2}s$ elements from each end, and so on up to the last layer of two separators on $2s$ marginal elements, see Fig. 3.5 (in the figure the circuit is oriented from top to bottom).

Of the $m \leqslant s$ largest (smallest) elements, the separator of the first layer allocates in the "wrong" half no more than $\varepsilon_0 m$ elements. In the next pair of separators, regardless of their order: the separator of the right (left) $2^{k-1}s$ elements leaves a maximum of $\varepsilon_0 m$ elements beyond the outermost interval, and the separator on the other side definitely does not worsen the characteristics of cutting off the largest (smallest) elements[9]. In each of the subsequent layers, the separator on the corresponding side erroneously throws out no more than $\varepsilon_0 m$ elements from the outermost interval.

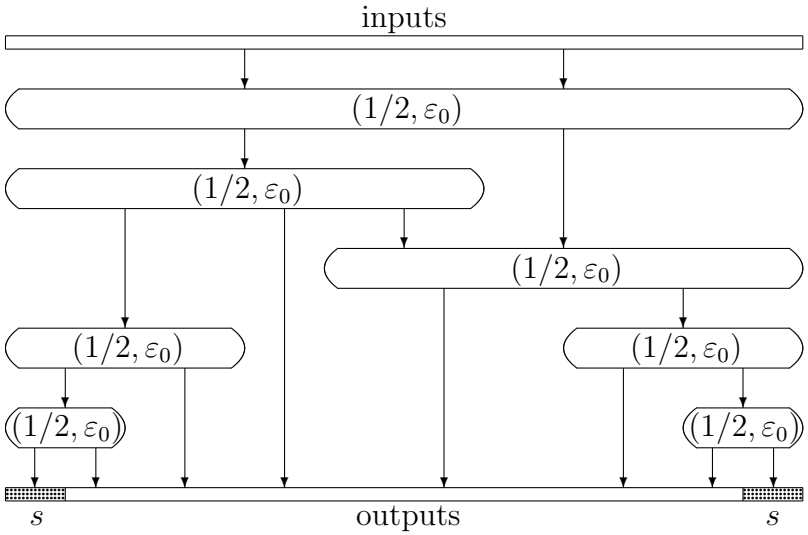

Figure 3.5: Construction of a (λ, ε)-separator

Thus, no more than $k\varepsilon_0 m$ largest (smallest) elements can miss the s extreme right (left) outputs of the circuit. Therefore, it is sufficient to choose $\varepsilon_0 = \varepsilon/k$. Now the circuit depth estimate follows from Lemma 3.17. □

Theorem 3.8 (M. Ajtai, J. Komlós, E. Szemerédi)**.** $\mathsf{D}(n) = O(\log n)$.

► For simplicity of reasoning we assume $n = 2^k$. Let $\mu, \varepsilon > 0$ be parameters to be chosen later. The sorting circuit is constructed from layers of parallel (λ, ε)-separators,

[8] If $2^{k-1}s = n$, then these two separators can be placed in parallel on the same layer.

[9] In any comparator circuit, the set of m largest (smallest) elements moves strictly to the right (left).

where λ is determined individually for each layer, and $\lambda \geqslant \mu$. It is convenient to imagine the circuit functioning in time: transformations of one layer are performed in one time unit. We consider the action of a separator layer as a rearrangement of elements in a structure associated with the complete binary tree of depth k. At each vertex of the tree there is a container for storing elements.

At the initial time $t = 0$ all n elements are in the root container. Then, at any time, the elements of each nonempty container are subjected to approximate sorting by a separator: elements from the outermost intervals are sent to the parent node, the remaining ones are distributed equally between the containers of child nodes, in the direction to the leaves.

We define the *capacity* of a container at depth d at time t as $B_{d,t} = na^d\nu^t$, where the constant parameters $a > 1$ and $\nu < 1$ will be specified later. If the container stores b elements, then in the case $\mu B_{d,t} < b/2$ the container's contents, except for a possible odd element, are ordered by a composition of $(1/2, \varepsilon)$- and (λ, ε)-separators, $\lambda = \mu B_{d,t}/b$, after which $\lfloor \mu B_{d,t} \rfloor$ elements from each end, as well as the odd element (if any), are sent to the parent node above. The remaining elements are moved down one level: the left half — to the container of the left child node, the right half — to the container of the right one. Otherwise, in the case $\mu B_{d,t} \geqslant b/2$, no separation is performed — all elements should be returned to the parent container. The scheme of migration of elements is shown in Fig. 3.6. The root container is an exception — its elements are rearranged by a $(1/2, \varepsilon)$-separator, divided equally into two parts (the number of elements in the container is necessarily even), which are sent to the corresponding containers of the child nodes.

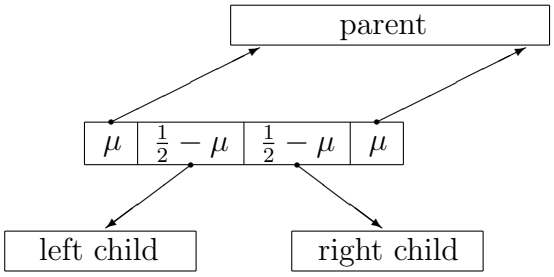

Figure 3.6: Scheme of migration of elements

The process continues until all elements are at the bottom $O(1)$ levels of the tree; the exact termination condition is stated as $B_{k,t} < 1/\mu$. After that, sorting circuits are applied in each subtree of depth $O(1)$. The constructed circuit contains $k \log_{1/\nu}(2a) + O(1) \asymp \log n$ layers of separators and therefore has the desired depth. It remains to check the correctness of the algorithm.

First of all, note that at any time moment, the containers of the same level are filled equally. In particular, this is why the root container always has an even number

of elements. In addition, a leaf container cannot keep more than one element. Hence, the described procedure does not throw elements outside the tree.

It is also clear from the construction that at any given time, all elements are concentrated either at even or at odd levels of the tree.

I. By induction on t we prove that under certain conditions on the parameters a, ε, μ, ν the number of elements in any container never exceeds its capacity. The statement is obvious for $t = 0$.

Consider an arbitrary container at depth d, empty at time $t-1$. The number of elements in it at the next time t in the case $B_{d,t} \geqslant a\nu$ does not exceed

$$2(2\mu B_{d+1,t-1} + 1) + B_{d-1,t-1}/2 = B_{d,t}(4\mu a + 1/(2a))/\nu + 2 < B_{d,t}(4\mu a + 3/a)/\nu,$$

which is less than $B_{d,t}$ subject to

$$\underline{\nu \geqslant 4\mu a + 3/a}. \tag{3.11}$$

In the remaining case $B_{d,t} < a\nu$ at time $t-1$ all containers at higher levels $d' < d$ are empty, because $B_{d',t-1} < 1$. This means that all elements are at levels $d+1$ and lower. In this case $B_{d+1,t-1} < a^2$, and under the additional assumption

$$\underline{a^2 = 1/\mu} \tag{3.12}$$

we conclude that $d + 1 \neq k$ (otherwise the process of constructing the circuit would have already been completed). This means that in each subtree rooted at a node of depth $d+1$ at time $t-1$ there are an even number of elements, therefore, an even number of elements are at the root of the subtree (this applies to the child nodes with respect to the one under consideration). Thus, at most $4\mu B_{d+1,t-1} = 4\mu a B_{d,t}/\nu < B_{d,t}$ elements are sent to a container of depth d at time t according to (3.11).

II. The proof of the correctness of the algorithm is based on the evaluation of the number of irrelevant elements in each container. We assume that when placed in the tree leaves, elements should obey to ascending order from left to right. For any element, the *native* vertices are the tree leaf in which the element should be located after ordering, as well as all vertices on the path from the tree root to this leaf. During the execution of the algorithm, an element of some container will be called an *r-stranger* if it is at a distance $\geqslant r$ along the tree edges from the nearest native vertex.

By induction on t we check that at time t the number of r-strangers ($r \geqslant 1$) in a container of depth d does not exceed $\varepsilon^{r-1}\mu B_{d,t}$ for an appropriate choice of parameters. The induction base is trivial, since at time $t = 0$ all elements are in the root container, native to them. We are going to prove the induction step.

III. First, let us consider the simpler case $r \geqslant 2$ (then we can assume $d \geqslant 2$). The r-strangers of a container of depth d can include $(r+1)$-strangers from containers of child nodes and $(r-1)$-strangers from the parent container at the previous moment of time. Taking into account filtering, their number is bounded from above as

$$2\varepsilon^r \mu B_{d+1,t-1} + \varepsilon(\varepsilon^{r-2}\mu B_{d-1,t-1}) = \varepsilon^{r-1}\mu B_{d,t}(2\varepsilon a + 1/a)/\nu,$$

i.e., does not exceed $\varepsilon^{r-1}\mu B_{d,t}$ provided

$$\underline{\nu \geqslant 2\varepsilon a + 1/a}. \tag{3.13}$$

IV. Now consider the case $r = 1$. In a vertex v at time t strangers can arrive from two sources: 2-strangers from child vertices, and from the parent vertex — both strangers and elements native to its other child vertex v' at the previous moment of time. The number of strangers from child vertices can be easily estimated as $2\varepsilon\mu B_{d+1,t-1} = 2\varepsilon\mu a B_{d,t}/\nu$. The parent container requires a more careful analysis.

Let the parent container at time $t-1$ contain q elements, of which q_0 are native elements for v, q_1 are native elements for v', and q_2 are strangers. If $q_0 \geqslant q/2$, then the number of strangers for the vertex v sent to its container is estimated as $\varepsilon(q_1 + q_2) \leqslant \varepsilon q/2$, i.e., as the sum of errors of a (λ, ε)-separator that did not send strangers upward, and a $(1/2, \varepsilon)$-separator that sent elements native to v' to a wrong half. Otherwise, if $q_0 < q/2$, then this number should be estimated as $\varepsilon q/2 + (q/2 - q_0)$, where the second term takes into account native elements of v', which end up in the container of v even after correct sorting. Let us estimate $q/2 - q_0$.

Consider a special hypothetical distribution of elements into containers at time $t-1$ with the same number of elements in each container as in the real distribution. Sort and distribute all elements uniformly among the nodes at level d (where the vertices v and v' reside), and then arbitrarily move elements up and down the tree to fill all containers correctly, but so that the parent node of v and v' receives $\lceil q/2 \rceil$ and $\lfloor q/2 \rfloor$ elements from these child nodes, respectively.

In the considered distribution, the subtree rooted at vertex v contains only elements native to it, and the container of the parent vertex contains $\geqslant q/2$ elements native to v. Let us estimate the maximum number of elements native to v that can be moved from this container to any other. This will yield an estimate for $q/2 - q_0$.

For containers of the same level $d-1$: for one container the specified elements will be 1-strangers, for two — 2-strangers, for four — 3-strangers, etc. The total number of positions available for placement at this level is estimated as

$$\mu\left(1 + 2\varepsilon + (2\varepsilon)^2 + \ldots\right) B_{d-1,t-1} < \mu B_{d-1,t-1}/(1 - 2\varepsilon). \tag{3.14}$$

At an arbitrary higher level $d-h$, for one container the elements in question will be native, for another — 1-strangers, for two more — 2-strangers, for four — 3-strangers, etc. The total number of available positions at these levels (for odd $h \geqslant 3$) is estimated as

$$\left(1 + \mu(1 + 2\varepsilon + (2\varepsilon)^2 + \ldots)\right) \sum_{i=1}^{d/2} B_{d-2i-1,t-1} < \left(1 + \frac{\mu}{1-2\varepsilon}\right) \sum_{i \geqslant 1} a^{-2i} B_{d-1,t-1} = \left(1 + \frac{\mu}{1-2\varepsilon}\right) \frac{B_{d-1,t-1}}{a^2 - 1}. \tag{3.15}$$

Since there are no more free positions to fill in the subtree of the vertex v, at any lower level $d+h$ there are available 2^h containers[10] for which the specified elements will be $(h+1)$-strangers, 2^{h+1} containers for which these elements will be $(h+2)$-strangers, etc. The total number of available positions does not exceed

$$\sum_{i=1}^{(k-d)/2} \mu\left((2\varepsilon)^{2i-1} + (2\varepsilon)^{2i} + \ldots\right) B_{d+2i-1,t-1} < \mu \sum_{i\geqslant 1} \frac{(2\varepsilon)^{2i-1}}{1-2\varepsilon}\, a^{2i-1} B_{d,t-1} = \frac{2\varepsilon a \mu B_{d,t-1}}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}. \quad (3.16)$$

Summing up (3.14), (3.15), (3.16), we obtain

$$q/2 - q_0 < \left(\frac{1}{a^2-1} + \frac{\mu a^2}{(1-2\varepsilon)(a^2-1)} + \frac{2\varepsilon a^2 \mu}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}\right) B_{d-1,t-1}.$$

As a consequence, the total number of strangers in the container of vertex v at time t, including those coming from child vertices, is estimated as

$$\left(2\varepsilon a\mu + \frac{1}{a^3-a} + \frac{\mu a}{(1-2\varepsilon)(a^2-1)} + \frac{2\varepsilon a\mu}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}\right) B_{d,t}/\nu,$$

which does not exceed $\mu B_{d,t}$ provided

$$\underline{\nu \geqslant 2\varepsilon a + \frac{1}{\mu(a^3-a)} + \frac{a}{(1-2\varepsilon)(a^2-1)} + \frac{2\varepsilon a}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}}. \quad (3.17)$$

V. Now it is easy to see that at the moment t of completion of the circuit construction procedure there are no strangers in any container (assuming that the parameters are chosen correctly). Indeed, in an arbitrary container of depth d, according to what was proved above, there are at most $\mu B_{d,t} \leqslant \mu B_{k,t} < 1$ strangers.

Moreover, by (3.12), $B_{d,t} = a^{d-k} B_{k,t} < a^{d-k}/\mu \leqslant 1$ for $d \leqslant k-2$, which means that all elements are in the lower two layers of the tree.

It remains to specify the choice of parameters that satisfies all the necessary conditions (3.11), (3.12), (3.13), (3.17). For example, $\varepsilon = \mu = 1/100$, $a = 10$, $\nu = 0.7$ will do. ■

Corollary 3.19. $\mathsf{S}^*(n) = O(n \log n)$.

[10] In the subtree rooted in v'.

Exercises

Ex. 3.1. Show that the result of Theorem 3.1 can be generalized to $\mathsf{S}^*(\mathcal{L}_{m,n}) \geqslant 0.5(m+n)\log_2 m - O(m)$ for $m \leqslant n$ [*P. Miltersen, M. Paterson, Y. Tarui*]

Ex. 3.2. Estimate the limits of the method of Theorem 3.3. Show that if a good potential function $f(x)$ monotonically increases on the interval $[0, 1/2]$ and decreases on the interval $[1/2, 1]$, then, say, $f(1/2) \leqslant 1.75$.

Ex. 3.3. Using Corollary 3.16, show that the depth of selecting an element of rank 2 is $\log_2 n + \log_2 \log n \pm O(1)$. [*A. Yao*]

Comments. Theorem 3.1 is proved in [7] with a slightly sharper bound $n\log_2 n - 0.66n$, and also in a more general form $\mathsf{S}^*(\mathcal{L}_{m,n}) \geqslant 0.5(m+n)\log_2 m - 0.73m$ for $m \leqslant n$. A somewhat simpler proof of an asymptotically similar bound, but weaker in the precision of the remainder term, is given in [6]; a simple and order-accurate lower bound is proved in [5]. An upper bound is obtained in [3], see also [5].

Theorem 3.3 and Corollary 3.13 with constant $C = 1.12$ were proved in [4]. Theorem 3.4 is a special case of the four-function theorem of R. Ahlswede and D. Daykin. The results of section 3.4 were obtained by V. E. Alekseev [2] (see also [5]) and A. Yao [9].

The proof of Theorem 3.8 follows the adaptation by J. Seiferas [8] of the sorting method of M. Ajtai, J. Komlós, and E. Szemerédi [1]. The multiplicative constant hidden in the depth estimate of Theorem 3.8 is insanely large. A number of papers have attempted to reduce it. The estimates published with the proofs (for various modifications of the algorithm) have an order of several thousands. Some rough estimations admit the existence of circuits with the depth of approximately $100\log_2 n$. In practice, better results are provided by variants of Batcher's circuits [3] with a theoretical depth of the order $\log^2 n$.

The choice of the polynomial $\hat{f}(y) = \frac{1}{64} + \frac{a}{4} - ay^2 - y^6$ with $a = 0.268$ allows to improve the constant in Corollary 3.13 to $C > 1.215$, and the bound of Theorem 3.5 to $(3.43 - o(1))\log_2 n$.

Bibliography

[1] Ajtai M., Komlós J., Szemerédi E. *Sorting in $c\log n$ parallel steps.* Combinatorica. 1983. **3**(1), 1–19.

[2] Alekseev V. E. *Sorting algorithms with minimum memory.* Cybernetics. 1969. (5), 642–648.

[3] Batcher K. E. *Sorting networks and their applications.* Proc. AFIPS spring joint comput. conf. (Atlantic City, 1968). **32**. NY: AFIPS, 1968, 307–314.

[4] Kahale N., Leighton T., Ma Y., Plaxton C. G., Suel T., Szemerédi E. *Lower bounds for sorting networks.* Proc. STOC (Las Vegas, 1995). NY: ACM, 1995, 437–446.

[5] Knuth D. E. *The art of computer programming. Vol.* 3. *Sorting and searching.* Reading: Addison-Wesley, 1998.

[6] Leighton T., Ma Y., Suel T. *On probabilistic networks for selection, merging, and sorting.* Proc. ACM Symp. Parall. Alg. and Architect. (Santa Barbara, 1995). NY: ACM, 1995, 106–118.

[7] Miltersen P. B., Paterson M., Tarui Y. *The asymptotic complexity of merging networks.* J. ACM. 1996. **43**(1), 147–165.

[8] Seiferas J. *Sorting networks of logarithmic depth, further simplified.* Algorithmica. 2009. **53**(3), 374–384. [see also: Seiferas J. *AKS sorting networks.* Manuscript, 2017. available at http://www.researchgate.net/profile/Joel-Seiferas]

[9] Yao A. *Bounds on selection networks.* SIAM J. Comput. 1980. **9**(3), 566–582.