Chapter 1
Selecting the median

Dorit Dor [*] *Uri Zwick* [*]

Abstract

Improving a long standing result of Schönhage, Paterson and Pippenger we show that the *median* of a set containing n elements can be found using at most $2.95n$ comparisons.

1 Introduction

The *selection problem* is defined as follows: given a set X containing n distinct elements drawn from a totally ordered domain, and given a number $1 \leq i \leq n$, find the i-th *order statistic* of X, i.e., the element of X larger than exactly $i-1$ elements of X and smaller than the other $n-i$ elements of X. The *median* of X is the $\lceil n/2 \rceil$-th order statistic of X.

The selection problem is one of the most fundamental problems of computer science and it has been extensively studied. Selection is used as a building block in the solution of other fundamental problems such as sorting and finding convex hulls. It is somewhat surprising therefore that only in the early 70's it was shown, by Blum, Floyd, Pratt, Rivest and Tarjan [BFP$^+$73], that the selection problem can be solved in $O(n)$ time. As $\Omega(n)$ time is clearly needed to solve the selection problem, the work of Blum *et al.* completely solves the problem. Or does it?

A very natural setting for the selection problem is the *comparison model*. An algorithm in this model can access the input elements only by performing pairwise comparisons between them. The algorithm is only charged for these comparisons. The comparison model is one of the few models in which *exact* complexity results may be obtained. What is then the exact comparison complexity of finding the median?

The comparison complexity of many comparison problems is exactly known. It is clear, for example, that exactly $n-1$ comparisons are needed, in the worst case, to find the maximum or minimum of n elements. Exactly $n + \lceil \log n \rceil - 2$ comparisons are needed to find the second largest (or second smallest) element (Schreier [Sch32], Kislitsyn [Kis64]). Exactly $\lceil 3n/2 \rceil - 2$ comparisons are needed to find both the maximum and the minimum of n elements (Pohl [Poh72]). Exactly $2n-1$ comparisons are needed to merge two sorted lists each of length n (Stockmeyer and Yao [SY80]). Finally, $n \log n + O(n)$ comparisons are needed to sort n elements (e.g., Ford and Johnson [FJ59]).

A relatively large gap, considering the fundamental nature of the problem, still remains however between the known lower and upper bounds on the exact complexity of finding the median. After presenting a basic scheme by which an $O(n)$ selection algorithm can be obtained, Blum *et al.* [BFP$^+$73] try to optimize their algorithm and present a selection algorithm that performs at most $5.43n$ comparisons. They also obtain the first non-trivial lower bound and show that $1.5n$ comparisons are required, in the worst case, to find the median. The result of Blum *et al.* is subsequently improved by Schönhage, Paterson and Pippenger [SPP76] who present a beautiful algorithm for the selection of the median, or any other element, using at most $3n + o(n)$ comparisons. In this work we improve the long standing result of Schönhage *et al.* and present a selection algorithm that uses at most $2.95n$ comparisons.

Bent and John [BJ85] (see also John [Joh88]), improving previous results of Kirkpatrick [Kir81], Munro and Poblete [MP82] and Fussenegger and Gabow [FG78], obtained a $(1 + H(\alpha)) \cdot n - o(n)$ lower bound on the number of comparisons needed to select the αn-th element of a set of n elements, where $H(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \frac{1}{1-\alpha}$ is the binary entropy function (all logarithms in this paper are taken to base 2). We have shown recently [DZ95] (using somewhat different methods from the ones used here) that the αn-th element can be selected using at most $(1 + \alpha \log \frac{1}{\alpha} + O(\alpha \log \log \frac{1}{\alpha})) \cdot n$ comparisons. This for small values of α is almost optimal. The bound of Bent and John gives in particular a $2n - o(n)$ lower bound on the number of comparisons needed to find the median.

Our work slightly narrows the gap between the best known lower and upper bounds on the comparison complexity of the median problem. Though our improvement is quite modest, many new ideas were required to obtain it. These new ideas shed some more light on the intricacy of the median finding problem.

[*]Department of Computer Science, School of Mathematical Sciences, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, ISRAEL. E-mail addresses {ddorit,zwick}@math.tau.ac.il.

Algorithms for selecting the i-th element for small values of i were obtained by Hadian and Sobel [HS69], Hyafil [Hya76], Yap [Yap76], Ramanan and Hyafil [RH84], Aigner [Aig82] and Eusterbrock [Eus93].

All the results mentioned so far deal with the number of comparisons needed in the *worst case*. Floyd and Rivest [FR75] showed that the i-th element can be found using an *expected* number of $n + i + o(n)$ comparisons. Cunto and Munro [CM89] had shown that the bound of Floyd and Rivest is tight.

The central idea used by Schönhage *et al.* in their $3n + o(n)$ median algorithm is the idea of *factories*. Schönhage *et al.* use factories for the mass production of certain partial orders at a much reduced cost. To obtain our results we extend the notion of factories. We introduce *green* factories and perform an *amortized* analysis of their production costs. We obtain improved green factories using which we can improve the $3n+o(n)$ result of Schönhage, Paterson and Pippenger.

The performance of a green factory is mainly characterized by two parameters A_0 and A_1 (the *upper* and *lower* element costs). Using a green factory with parameters A_0 and A_1 we obtain an algorithm for the selection of the αn-th element using at most $(A_0\alpha + A_1(1-\alpha))\cdot n + o(n)$ comparisons. To select the median, we use a factory with $A_0, A_1 \approx 2.95$. Actually, there is a tradeoff between the lower and upper costs of a factory. For every $0 < \alpha \leq 1/2$ we may choose a factory that minimizes $A_0\alpha + A_1(1-\alpha)$. We can select the $n/4$-th element, for example, using at most $2.69n$ comparisons, by using a factory with $A_0 \approx 4$ and $A_1 \approx 2.25$. In this paper, we concentrate on factories for median selection. It is easy to verify that the algorithm described here, as the median finding algorithms of both Blum *et al.* and Schönhage *et al.*, can be implemented in linear time in the RAM model.

In the next section we describe in more detail the concept of factory production and introduce our notion of *green* factories. We also state the properties of the improved factories that we obtain. In Section 3 we explain the way in which green factories are used to obtain efficient selection algorithms. The selection algorithm we describe is a generalization of the median algorithm of Schönhage *et al.* [SPP76] and is similar to the selection algorithm we describe in [DZ95]. In the subsequent sections we try to demonstrate the main ideas used in the construction of our new green factories. Due to lack of space, many of the details are omitted.

2 Factory production

Denote by S_k^m a partial order composed of a *centre* element, m elements larger than the centre and k elements smaller than the centre (see Fig. 1). An S_k^m

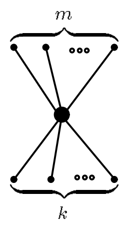

Figure 1: The partial order S_k^m.

is sometimes referred to as a *spider*. Schönhage *et al.* [SPP76] show that producing l disjoint copies of S_k^m usually requires fewer comparisons than l times the number of comparisons required to produce a single S_k^m. The best way, prior to this work, of producing a single S_k^k, for example, requires about $6k$ comparisons (find the median of $2k+1$ elements using the $3n+o(n)$ median algorithm). The cost per copy can be cut by almost a half if the S_k^k's are mass produced using factories.

A *factory* for a partial order P is a comparison algorithm with continual input and output streams. The input stream of a simple factory consists of single elements. When enough elements are fed into the factory, a new disjoint copy of P is produced. A factory is characterized by the following quantities: the *initial cost* I, which is the number of comparisons needed to initialize the factory; the *unit cost* U, which is the number of comparisons needed to generate each copy of P; and finally the *production residue* R, which is the maximal number of elements that can remain in the factory when lack of inputs stops production. For every $l \geq 0$, the cost of generating l disjoint copies of P is at most $I + l\cdot U$. Schönhage *et al.* [SPP76] construct factories with the following characteristics:

Theorem 2.1. *There is a factory F_k for S_k^k with initial cost I_k, unit cost U_k and production residue R_k satisfying: $U_k \sim 3.5k$, $I_k = O(k^2)$, $R_k = O(k^2)$.*

The notation $U_k \sim 3.5k$ here means that $U_k = 3.5k + o(k)$. Schönhage *et al.* also show that if there exist factories F_k, for S_k^k's, satisfying $U_k \sim Ak$, for some $A > 0$, and $I_k, R_k = O(k^2)$, then the median of n elements can be found using at most $An + o(n)$ comparisons. The above theorem immediately implies therefore the existence of a $3.5n + o(n)$ median algorithm.

The way factories are used by selection algorithms is described in the next section. For now we just mention that most S_k^m's generated by a factory employed by a selection algorithm are eventually broken, with either their upper elements eliminated and their lower elements returned to the factory or vice versa. While constructing an S_k^m, a factory may have compared elements that turned out to be on the same side of the centre. If such elements are ever returned to the factory, the known relations among them may save the factory some of the comparisons it has to perform. To capture this, we extend the definition of factories and define *green* factories (factories that support the recycling of known relations). This extension is implicit in the work of Schönhage *et al.* [SPP76]. Making this notion explicit simplifies the analysis of our factories. The $3n + o(n)$ median algorithm of Schönhage *et al.* is in fact obtained by replacing the factory F_k of Theorem 2.1 by a simple green factory.

A green factory for S_k^m's is mainly characterized by the following two quantities: the *lower element cost* u_0 and the *upper element cost* u_1. Using these quantities, the *amortized* production costs of the factory can be calculated as follows: The amortized production cost of an S_k^m whose upper m elements are eventually returned (together) to the factory is $k\cdot u_0$. The amortized production cost of an S_k^m whose lower k elements are eventually returned (together) to the factory is $m\cdot u_1$. The amortized production cost of an S_k^m such that none of its elements is returned to the factory is $k\cdot u_0 + m\cdot u_1$. Note that in this accounting scheme we attribute all the production cost to elements that are not returned to the factory. The initial cost I and the production residue R of a green factory are defined as before. A somewhat different definition of green factories was given by us in [DZ95]. The new definition uses amortized costs per *element* whereas our old definition used amortized costs per partial order. A green factory does not know in advance whether the lower or upper part of a generated S_k^m will be recycled. This is set by an adversary. Though not stated explicitly, the following result is implicit in [SPP76].

THEOREM 2.2. *There is a green factory G_k for S_k^k with lower and upper element costs $u_0, u_1 \sim 3$, initial cost $I_k = O(k^2)$ and production residue $R_k = O(k^2)$.*

The notation $u_0, u_1 \sim 3$ here means that $u_0, u_1 = 3 + o(1)$ where the $o(1)$ is with respect to k.

We shall see in the next section that a green factory for S_k^k with lower and upper element costs u_0 and u_1 yields a $(u_0 + u_1)/2 \cdot n + o(1)$ median algorithm. To improve the algorithm of Schönhage *et al.* it is enough therefore to construct an S_k^k factory with $(u_0 + u_1)/2 < 3$. Unfortunately, we are not able to construct such a factory.

However, we are able to reduce the upper and lower element costs if we allow variation among the partial orders generated by the factory. Let $\tilde{\mathcal{S}}_k^k = \{S_{k'}^{k''} : k \le k' \le 2k\ ,\ k \le k'' \le 2k\}$. We construct improved green factories that generate partial orders that are members of $\tilde{\mathcal{S}}_k^k$. These factories can be easily incorporated into the selection algorithm described in the next section. To obtain our $2.95n$ median algorithm we use green $\tilde{\mathcal{S}}_k^k$ factories $\mathcal{G}_k$ with the following characteristics:

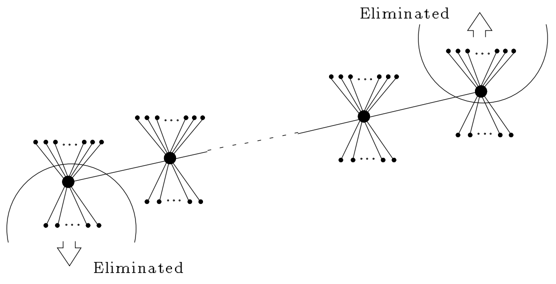

Figure 2: The ordered list of $\tilde{\mathcal{S}}_k^k$'s.

THEOREM 2.3. *There is a green factory $\mathcal{G}_k$ for $\tilde{\mathcal{S}}_k^k$ with $u_0, u_1 \sim 2.942$, $I_k = O(k^2)$, $R_k = O(k^2)$.*

The main ideas used to construct the factories $\mathcal{G}_k$ are described in Section 5.

3 Selection algorithms

In this section we describe our selection algorithm. This algorithm uses an $\tilde{\mathcal{S}}_k^k$ factory. The complexity of the algorithm is completely determined by the characteristics of the factory used. This algorithm is a generalization of the median algorithm of Schönhage *et al.* and a variation of the selection algorithm we describe in [DZ95].

THEOREM 3.1. *Let $0 < \alpha \le 1/2$. Let $\mathcal{F}_k$ be an $\tilde{\mathcal{S}}_k^k$ factory with lower element cost $u_0 \sim A_0$, upper element cost $u_1 \sim A_1$, initial cost $I_k = O(k^2)$ and production residue $R_k = O(k^2)$. Then, the αn-th smallest element, among n elements, can be selected using at most $(\alpha\cdot A_0 + (1-\alpha)\cdot A_1)n + o(n)$ comparisons.*

Proof. We refer to the αn-th smallest element among the n input elements as the percentile element. The algorithm uses the factory $\mathcal{F}_k$ where $k = \lfloor n^{1/4} \rfloor$. The n input elements are fed into this factory, as singletons, and the production of partial orders $S \in \tilde{\mathcal{S}}_k^k$ commences. The centres of the generated S's are inserted, using binary insertion, into an ordered list L, as shown in Fig. 2. When the list L is long enough we either know, as we shall soon show, that the centre of the upper (i.e., last) S in L and the elements above it are too large to be the percentile element, or that the centre of the lower (i.e., first) S and the elements below it are too small to be the percentile element. Elements too large or too small to be the percentile element are eliminated. The lower elements of the upper S, and the upper elements of the lower S are returned to the factory for recycling.

Let t be the current length of the list L and let r be the number of elements currently in the factory. The number of elements that have not yet been eliminated is therefore $N = \Theta(k)\cdot t + r$. Let i be the rank of the percentile element among the non-eliminated elements.

Initially $N = n$ and $i = \lceil \alpha n \rceil$.

The number of elements in the list known to be smaller or equal to the centre of the upper S of the list is $N_0 = \Theta(k) \cdot t$. The number of elements known to be greater or equal to the centre of the lowest S of the list is $N_1 = \Theta(k) \cdot t$. Note that $N_0 + N_1 = N + t - r$ as the centres of all the S's in the list satisfy both these criteria, the r elements are currently in the factory satisfy neither, and all the other non-eliminated elements satisfy exactly one of these criteria.

The algorithm consists of the following interconnected processes:

(i) Whenever sufficiently many elements are supplied to the factory $\mathcal{F}_k$, a new partial order $S \in \tilde{\mathcal{S}}_k^k$ is produced and its centre is inserted into the list L using binary insertion.

(ii) Whenever $N_0 > i$, the centre of the upper partial order $S \in \tilde{\mathcal{S}}_k^k$ in the list and the elements above it are eliminated, as they are too big to be the percentile element. The lower elements of S are recycled.

(iii) Whenever $N_1 > N - i + 1$, the centre of the lowest partial order $S \in \tilde{\mathcal{S}}_k^k$ in the list and the elements below it are eliminated, as they are too small to be the percentile element. The upper elements of S are recycled. The value of i is updated accordingly, i.e., i is decremented by the number of elements in the lower part of S (including the centre).

If (ii) and (iii) are not applicable then $N_0 \le i$ and $N_1 \le N - i + 1$. Thus $N + t - r = N_0 + N_1 \le N + 1$ and $t - 1 \le r$. If (i) is not applicable then by the factory definition we have $r \le R_k$. When no one of (i),(ii) and (iii) can be applied we get that $t - 1 \le r \le R_k = O(k^2)$. At this stage $N = O(k^3)$, which is $O(n^{3/4})$, and the i-th element among the surviving elements is found using any linear selection algorithm.

We now analyze the comparison complexity of the algorithm. Whenever (ii) is performed, the upper partial order $S \in \tilde{\mathcal{S}}_k^k$ of the list is broken. Its centre and upper elements are eliminated and its lower elements are returned to the factory. The amortized production cost of the partial order S is at most A_1 comparisons per each element above the centre.

Whenever (iii) is performed, the lowest partial order $S \in \tilde{\mathcal{S}}_k^k$ of the list is broken. Its centre and lower elements are eliminated and its upper elements are returned to the factory. The amortized production cost of the partial order S is at most A_0 comparisons per each element below the centre.

The algorithm can eliminate at most $(1 - \alpha)n$ elements larger than the percentile element and at most αn elements smaller than the percentile element. The total production cost of all partial orders $S \in \tilde{\mathcal{S}}_k^k$ that are eventually broken is therefore at most $(\alpha A_0 + (1 - \alpha)A_1) \cdot n + o(n)$. At most $O(k^2)$ generated partial orders $S \in \tilde{\mathcal{S}}_k^k$ are not broken. Their total production cost is $O(k^3)$. The initial production cost is $O(k^2)$. The total number of comparisons performed by the factory is therefore $(\alpha A_0 + (1 - \alpha)A_1) \cdot n + o(n)$.

Let t^* be the final length of the list L (when none of (i),(ii) and (iii) is applicable). The total number of partial orders generated by $\mathcal{F}_k$ is at most $n/k + t^*$, as at least k elements are eliminated whenever a partial order is removed from L. The total cost of the binary insertions into the list L is at most $O((n/k + t^*) \cdot \log n) = O((n/k + k^2) \log n)$ which is $o(n)$. The total number of comparisons performed by the algorithm is therefore at most $(\alpha A_0 + (1 - \alpha)A_1) \cdot n + o(n)$, as required. $\square$

Using the factories of Theorem 2.3, we obtain our main result:

Theorem 3.2. *Any element, among n elements, can be selected using at most $2.942n + o(n)$ comparisons.*

4 Basic principles of factory design

In this section we give some of the basic principles used to construct efficient factories. The section is divided into three subsections. In the first subsection we remind the reader what *hyperpairs* are and what their *pruning cost* is. In the second subsection we describe the notion of *grafting*. In the third subsection we sketch the construction of the S_k^k factories of Schönhage *et al.* [SPP76]. These factories are described as an example for a simple factory design.

Before going into details, we describe a clever accounting principle introduced by Schönhage *et al.* to simplify the complexity analysis. The information we care to remember on the elements that pass through the factory can always be described using a Hasse diagram. Each comparison made by the algorithm adds an edge to the diagram and possibly deletes some edges. At some stages we may decide to 'forget' the result of some comparisons and the edges that correspond to them are removed from the diagram. Schönhage *et al.* noticed that instead of counting the number of comparisons made, we can count the number of edges cut! To this we should add the number of edges in the eliminated parts of the partial orders as well as the edges that remain in the factory when the production stops. The second number, in our factories, is at most a constant times the production residue of the factory and it can be attributed to the initial cost.

4.1 Hyperpairs

A factory usually starts the production of a partial order from $\tilde{\mathcal{S}}_k^k$ by producing a large partial order, *a hyperpair*, that contains a partial order from $\tilde{\mathcal{S}}_k^k$.

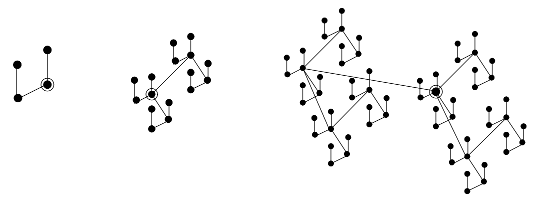

Figure 3: Some small H_r's ($H_2 = P_{01}$, $H_4 = P_{0110}$ and $H_6 = P_{011010}$).

DEFINITION 4.1. *An hyperpair P_w, where w is a binary string, is a finite partial order with a distinguished element, the* centre*, defined recursively by (i) P_λ is a single element (λ here stands for the empty string). (ii) P_{w1} is obtained from two disjoint P_w's by comparing their centres and taking the higher as the new centre. P_{w0} is obtained in the same way but taking the lower of the two centres as the new centre.*

The Hasse diagrams of some small hyperpairs are shown in Fig. 3. Some basic properties of hyperpairs are given in the following Lemma.

LEMMA 4.1. *Let c be the centre of a hyperpair P_w. Let w_j be the prefix of w of length j. Let h_0 be the number of 0's in w and h_1 be the number of 1's in w. Then: (i) The centre c together with the elements greater than it form a $P_{0^{h_0}}$ with centre c. The elements greater than c form a disjoint set of hyperpairs P_λ, P_0, ... , $P_{0^{h_0-1}}$. The centre c together with the elements smaller than it form a $P_{1^{h_1}}$ with centre c. The elements smaller than c form a disjoint set of hyperpairs P_λ, P_1, ... , $P_{1^{h_1-1}}$. (ii) The hyperpair P_w can be parsed into its centre c and a disjoint set $\{P_{w_j} : 0 \le j < |w|\}$ of smaller hyperpairs. Moreover, the centre of P_{w_j} is above c if w_{j+1} ends with 0, and below c if w_{j+1} ends with 1.*

The Lemma can be easily proved by induction. Note, in particular, that if $m < 2^{h_0}$ and $k < 2^{h_1}$ then P_w contains an S_k^m. No edges are cut during the construction of hyperpairs. But, before outputting an S_k^m contained in a hyperpair, all the edges connecting the elements of this S_k^m with elements not contained in this S_k^m have to be cut. This rather costly operation is referred to as *pruning*.

The *downward pruning cost* $PR_0(w)$ of a hyperpair P_w with centre c is the number of edges that connect elements of P_w that are below the centre c with the other elements of P_w (excluding c). The *upward pruning cost* $PR_1(w)$ of a hyperpair P_w is defined analogously.

Usually, especially if a grafting process is applied, we do not want to prune all the elements above or below the centre c of a hyperpair P_w. It is then more convenient to consider the *amortized* per element pruning costs. Let h_0 and h_1 be the number of 0's and 1's in w and let $h = h_0 + h_1$. We define $pr_0(w) = PR_0(w)/2^{h_1}$ and $pr_1(w) = PR_1(w)/2^{h_0}$ to be the *lower element pruning cost* and the *upper element pruning cost* of w. It can be easily shown that the cost of pruning k_0 elements below c is at most $k_0 \cdot pr_0(w) + h$ and the cost of pruning k_1 elements above c is at most $k_1 \cdot pr_1(w) + h$. The h terms are usually negligible. Note that h is the number of edges connected to the centre c of P_w. When an edge connected to c is cut, a hyperpair $P_{w'}$, where w' is a prefix of w, is obtained. This hyperpair can then be used in the construction of the next P_w. The following Lemma is easily proved.

LEMMA 4.2.

(i)	$pr_0(\lambda) = 0$	,	$pr_1(\lambda) = 0$
(ii)	$pr_0(0w) = pr_0(w) + 1$	,	$pr_0(1w) = \frac{1}{2} \cdot pr_0(w)$
(iii)	$pr_1(0w) = \frac{1}{2} \cdot pr_1(w)$	,	$pr_1(1w) = pr_1(w) + 1$

To produce partial orders from $\tilde{\mathcal{S}}_k^k$ for larger and larger values of k, we have to construct larger and larger hyperpairs. When we design a family $\{\mathcal{F}_k\}_{k=1}^\infty$ of factories, we usually choose an infinite binary string $\mathcal{W}$. In each member $\mathcal{F}_k$ of this family we construct a hyperpair whose sequence is a long enough prefix of $\mathcal{W}$. Let w_i be the finite prefix of $\mathcal{W}$ of length i. The lower and upper element pruning costs of an infinite sequence $\mathcal{W}$ are defined as the limits $pr_0(\mathcal{W}) = \lim_{i\to\infty} pr_0(w_i)$ and $pr_1(\mathcal{W}) = \lim_{i\to\infty} pr_1(w_i)$. These limits do exist for the chosen infinite strings.

Schönhage *et al.* base their factories on the infinite string $\mathcal{W} = 01(10)^\omega$ for which, as can be easily verified, $pr_0(\mathcal{W}) = pr_1(\mathcal{W}) = 1.5$. In our factories, we also need hyperpairs with cheaper lower element pruning cost and, alas, more expensive upper element pruning cost, or vice versa. The following Theorem presents a tradeoff between the upper and lower element pruning costs. Its proof is omitted due to lack of space.

THEOREM 4.1. *For any two numbers $1 \le a, b \le 2$ such that $a + b = 3$, there exists a binary sequence $\mathcal{W} \in \{01, 10\}^\omega$ for which $pr_0(\mathcal{W}) = a$ and $pr_1(\mathcal{W}) = b$.*

We are already in a position to describe a simple but complete S_k^k factory. Select a string $\mathcal{W}$. Construct a hyperpair P_w that contains the partial order S_k^k, where w is a long enough prefix of $\mathcal{W}$. Prune k elements above and k elements below the centre of this P_w. These $2k+1$ elements form a copy of S_k^k. By Lemma 4.1(ii), the remaining elements of P_w form a disjoint collection of partial orders of the form P_{w_i}, where w_i is a prefix of w. These partial orders are used to construct a new copy of P_w that will be used to construct the next S_k^k. Before we output an S_k^k, we cut the $2k$ edges it contains. When some part of an S_k^k generated by the factory is recycled, the elements returned to the factory (as singletons) are used again for the construction of hyperpairs. It is easy to check that the lower and upper element costs of this simple factory are both $u_0, u_1 \sim pr_0(\mathcal{W}) + pr_1(\mathcal{W}) + 2$. For any $\mathcal{W} \in \{01, 10\}^\omega$ we get that the lower and upper element costs are $u_0, u_1 \sim 5$.

4.2 Grafting The costs of the simple factories described above can be significantly improved using *grafting*. We can cheaply find elements that are smaller than the centre, or elements that are larger than the centre (but not both usually). The process of finding such elements is called *grafting*. Pruning is then used to obtain elements on the opposite side.

We demonstrate this notion using a simple example, the grafting of singletons. Take an element x, not contained in the hyperpair, and compare it to the centre c of the hyperpair. Continue is this way, comparing new elements to the centre, until either k elements above the centre, or k elements below the centre are found. Note that no edges are cut in this process. All the grafted elements are put in the output partial order. The pruning process is then used to complete the partial order into an S_k^k. Adding this process to our simple factory for S_k^k, the upper and lower element costs are reduced to: $u_1, u_0 \sim \max\{pr_0(\mathcal{W}), pr_1(\mathcal{W})\} + 2$ (note that now we have to prune elements from at most one side). Thus $u_0, u_1 \sim 3.5$ if we take $\mathcal{W} = 01(10)^\omega$ or $\mathcal{W} = 10(01)^\omega$. This supplies a proof to Theorem 2.1. Note that at least one side of each generated S_k^k is composed of singletons, and if this side is recycled, no comparisons can be reused.

4.3 The factories of Schönhage, Paterson and Pippenger We now sketch the operation of the green factories G_k obtained by Schönhage *et al.* [SPP76]. These factories improve upon the simple factories described above by grafting and recycling pairs. The factory G_k starts by producing hyperpairs corresponding to prefixes of the string $\mathcal{W} = 01(10)^\omega$ (the string $\mathcal{W} = 10(01)^\omega$ could be used instead). Let w_i be the prefix of $\mathcal{W}$ of length i. For brevity we let $H_i = P_{w_i}$. Some small H_i's were shown in Fig. 3. By Lemma 4.1, an H_{2r}, where $r = \lceil \log(k+1) \rceil$ contains an S_k^k. After constructing an H_{2r}, the factory initiates the following pair grafting process:

Let $x < y$ be a pair of elements and let p_1 and p_0 be two counters initially set to zero. Let c denote the centre of the hyperpair. If $p_0 > p_1$ compare y and c. If $y > c$ then compare also x and c. If $y > x > c$ then increase p_1 by one. On the other hand, if $p_0 \le p_1$ then compare x and c. If $x < c$ then compare also y and c. Finally, if $x < y < c$ then increase p_0 by one.

As in the simple factory described in the previous subsection, the grafting continues until k elements are found above or below the centre and then a pruning process is used to complete the generation of an S_k^k. The elements above the centre of the generated S_k^k form a collection of disjoint P_{0^i}'s and the elements below the centre form a collection of disjoint P_{1^i}'s. When the lower or upper part of an S_k^k is returned to the factory, some of the existing relations among the elements returned are utilized. The amortized analysis of the green factory G_k encompasses a trade-off between the cost of generating an S_k^k and the utility obtained from its lower or upper parts when these parts are recycled. Although the S_k^k's generated by the factory of Schönhage *et al.* may contain P_{0^i}'s and P_{1^i}'s, where $i > 1$, their factory is only capable of utilizing pairwise disjoint relations among the elements returned to it (as the grafting process uses pairs). If a P_{0^i} or a P_{1^i}, with $i > 1$, is returned to the factory, it is immediately broken into 2^{i-1} P_0's or P_1's. Note that both P_0 and P_1 simply stand for a pair of elements. It can be checked, see [SPP76], that the upper and lower element costs of this factory are $u_1, u_0 \sim 3$. This is Schönhage *et al.*'s best result.

5 Advanced principles of factory design

In this section, we outline the principles used to construct our improved factories that yield the $2.95n$ median algorithm. The first of these principles was already mentioned.

- Allowing variations in the produced partial orders.

  Our factories construct partial orders from $\tilde{\mathcal{S}}_k^k$. The exact proportion between the number of elements below and above the centre of a generated partial order is not fixed in advance.

- Recycling larger relations.

  The factories of Schönhage *et al.* are only capable of recycling pairs (i.e., P_0's and P_1's). Our factories recycle larger constructs such as quartets (P_{00}'s and P_{11}'s), octets (P_{000}'s and P_{111}'s), 16-tuples (P_{0000}'s and P_{1111}'s) as well as pairs, singletons and other structures which are not hyperpairs. The non-hyperpair constructs are obtained by the more sophisticated grafting processes used.

- Constructing hyper-products.

  As mentioned, our factories may receive partial orders that could not be used for the construction of hyperpairs. These partial orders are used instead for the construction of *hyper-products*. A hyper-product $P_w \circ I$, where I is some partial order with a distinguished element which is again called a centre, is a hyperpair P_w that each of its elements is also the centre of a disjoint I. Hyperpairs are of course special cases of hyper-products as $P_w \circ P_0 = P_{0w}$ and $P_w \circ P_1 = P_{1w}$.

- Grafting larger relations and mass-grafting.

  The factories of Schönhage *et al.* use a simple pair grafting process. We use more complicated grafting processes, even if only pairs are involved. For each input construct we have different grafting processes. Some of our grafting processes use the technique of mass production.

- Using sub-factories.

  The factories of Schönhage *et al.* generate only a single family of hyperpairs (corresponding to $\mathcal{W} = 01(10)^\omega$). Our factories generate several types of hyperpairs and hyper-products, as mentioned above. The construction of each one of these hyper-products is carried out in a separate sub-production unit that we refer to as a *sub-factory*. Different sub-factories also differ in the 'raw-materials' that they can process.

- Using credits in the amortized complexity analysis.

  The last principle is an accounting principle. The different constructs recycled by our factories are of different 'quality'. Some of them can be used very efficiently for the construction of partial orders from $\tilde{\mathcal{S}}_k^k$. Others are not so appropriate for this process and using them as raw materials for the construction of partial orders from $\tilde{\mathcal{S}}_k^k$ results in a much higher production cost. To equalize these costs, each construct used by our factories is assigned a credit (or debit if negative).

Unfortunately, we do not have enough space in this extended abstract for a full description of our factories. In the next section, we describe a factory that can be used to obtain a $2.97n$ median algorithm. This is a greatly simplified version of our best factory that yields the $2.95n$ median algorithm.

6 Factories for median selection

The construction of the factory $\mathcal{G}_k$ satisfying the conditions of Theorem 2.3 is extremely involved. To keep this section relatively short, we describe here a simplified version $\hat{\mathcal{G}}_k$ of the factory $\mathcal{G}_k$. This factory yields the following result which is only slightly weaker then Theorem 2.3:

THEOREM 6.1. *There is a green factory $\hat{\mathcal{G}}_k$ for $\tilde{\mathcal{S}}_k^k$ with $u_0, u_1 < 2.9677$.*

As was the case with all the other factories we considered, the unit cost of this factory is $O(1)$ and the initial cost and production residues are $O(k^2)$.

The main differences between $\hat{\mathcal{G}}_k$ and $\mathcal{G}_k$ are the following: $\hat{\mathcal{G}}_k$ utilizes only singletons, pairs and quartets for grafting. Therefore, $\hat{\mathcal{G}}_k$ does not use credits or unbalanced hyperpairs. Moreover, $\hat{\mathcal{G}}_k$ is able to recycle only a fraction of at most $\gamma \approx 0.638$ of the elements in quartets. If the proportion of elements in quartets in the recycled side is larger than γ, then some of these quartets have to be broken into pairs.

This section is divided into four subsections. In the first subsection we give a preliminary description of the factory. In the second and the third subsections we describe the pairs and quartets grafting processes. Finally, in the last subsection we give a full description of the factory.

6.1 Preliminaries The factory $\hat{\mathcal{G}}_k$ recycles, and therefore receives as inputs, singletons, pairs (P_0's and P_1's) and quartets (P_{01}'s and P_{10}'s). Singletons are immediately joined into pairs.

The factory $\hat{\mathcal{G}}_k$ employs four processes: hyperpair generation, pair grafting, quartet grafting, and pruning. The factory $\hat{\mathcal{G}}_k$ employs two sub-factories that generate balanced hyperpairs. The first constructs hyperpairs according to the sequence $\mathcal{W} = 01(10)^\omega$. The second constructs hyperpairs according to the sequence $\mathcal{W} = 10(01)^\omega$. Input P_{01}'s are passed to the first sub-factory (as P_{01}'s can be used for the construction of hyperpairs that correspond to $\mathcal{W} = 01(10)^\omega$) while input P_{10}'s are passed to the second factory (as P_{10}'s can be used for the construction of hyperpairs that correspond to $\mathcal{W} = 10(01)^\omega$). Input pairs are spread between the two sub-factories according to demand.

We use the accounting scheme described in Section 4 to simplify the complexity analysis. Hence, the cost of an operation is the number of edges it cuts. When no ambiguity occurs, we let the *upper cost* (*lower cost*) of an operation be the cost of the operation when the upper part (or lower part) of its result is eliminated. Note that upper and lower costs are calculated for whole structures, whereas the upper and lower *element* costs are calculated per eliminated element.

The factory $\hat{\mathcal{G}}_k$ is not capable of recycling elements in structures larger than quartets. Any P_{0^i} (or P_{1^i}), where $i > 2$, has to be cut therefore into a collection of disjoint P_{00}'s (P_{11}'s). The price of this operation is $1/4$ edge per element.

The factory $\hat{\mathcal{G}}_k$ requires some of the elements it receives to be organized in pairs (to be used for pair grafting). Therefore, some of the quartets that are to be recycled may have to be cut. The exact proportion of quartets that would have to be cut is not known in advance. Which partial order $S \in \tilde{\mathcal{S}}_k^k$ should be charged for the cutting of these edges? The one being recycled or the one being constructed? The answer is that the cost should be split between these two. The optimal charging scheme, in the case of $\hat{\mathcal{G}}_k$, turns out to be the following: When an $S \in \tilde{\mathcal{S}}_k^k$ is recycled, we make sure that at most a fraction $\gamma \approx 0.638$ of the recycled elements are organized in quartets. If more elements are organized in quartets then some of the quartets are cut and this is charged to the partial order being recycled. If during of the construction of a partial order $S \in \tilde{\mathcal{S}}_k^k$ more quartets have to be cut, the cost of these additional cuts is charged to the partial order being constructed. In some cases the factory $\hat{\mathcal{G}}_k$ runs out of quartets. It then takes pairs and turns them into quartets. No cost is associated with this operation as no edges are cut.

The general approach taken by the factory $\hat{\mathcal{G}}_k$ is the following. If enough elements are supplied to the factory then in at least one the two sub-factories, a large enough hyperpair can be built. Additional relations arriving at the factory are then either used for grafting in the first sub-factory or used for the construction of a large enough hyperpair also in the second sub-factory. Whenever a large enough hyperpair is formed, a quartet grafting process is applied on it, then a pair grafting process is applied on it. Each one of the these grafting processes has a collection of possible outcomes. In some outcomes elements with low upper element cost but high lower element cost are obtained. In other outcomes elements with high upper element cost but low lower element cost are obtained. Some of these outcomes can be combined with some pruned elements into a tuple with low enough upper *and* lower element costs. We show that if there are no such outcomes (which can be combined with pruned elements) we can always combine outcomes from the two preceding cases so that tuples with low enough upper *and* lower element costs are obtained.

In general, the upper (or lower) element cost of each case is the sum of the upper (or lower) costs of the two grafting processes, the pruning cost, the cost of cutting quartets into pairs (if necessary) and the cost of the elimination itself. The elimination cost of each element is always a single edge as the output of the grafting precesses is always a partial order which does not contain undirected cycles (undirected cycles, if obtained, are broken).

The last remark regards our optimization scheme. At a certain point in the algorithm, we decide upon the optimal number of elements, from each category, that are to be added to the output partial order. The optimal number of elements from each category may be non-integral. The sum of the optimal numbers, of each category, is rounded to the nearest integer value (which will be the actual number of elements, from this category, in the output partial order). The factory maintains a counter for each category and makes sure that the number of grafted elements will not differ from this counter by more than a constant value.

6.2 Grafting pairs In this subsection we describe our pair grafting process, which is considerably more complicated than the pair grafting process used by Schönhage *et al.* [SPP76]. Our process uses a mass production scheme to construct a sequence of *dominated hyperpairs.*

The process receives two parameters: a direction bit d and the centre c of the output partial order S. These parameters are set by the factory when initiating this process. The grafting recursively builds hyperpairs which are *dominated* by the centre c of the output partial order. A dominated hyperpair P of direction d and level i is a hyperpair $P = P_{d^i}$ with centre c' such that each element of P, except for c', is known to be larger (if $d = 0$) or smaller (if $d = 1$) than c. The relation between c' and c is usually not determined.

The pair grafting process is composed of rounds. The i-th round receives two dominated hyperpairs P^1 and P^2 (with centres c_1 and c_2, respectively) of level i and attempts to construct a dominated hyperpair of level $i+1$. At first a hyperpair $P = P_{d^{i+1}}$ is constructed by comparing c_1 and c_2. Assume, without loss of generality, that c_2 is the centre of the new hyperpair P. Then, compare c_1 with c. The two possible outcomes, when $d = 1$, are:

(1) $c_1 < c$ and P is a dominated hyperpair of level $i+1$.

(2) $c_1 > c$ and P is not dominated by c (as $c_2 > c_1 > c$).

Let $(1)'$ and $(2)'$ be the corresponding cases for $d = 0$, i.e., $c_1 > c$ and $c_1 < c$, respectively. If (2) or $(2)'$ occur, the process is stopped. For the purposes of $\hat{\mathcal{G}}_k$ (and also for those of $\mathcal{G}_k$), we also stop the process when a dominated hyperpair P_{000} or P_{111} of level 3 is generated.

When $i = 0$, the hyperpairs P^1 and P^2 are just singletons c_1 and c_2. The 'centres' c_1 and c_2 are compared and a pair is obtained. The pair grafting process receives its elements as pairs. It therefore starts with the second comparison of the 0-th round.

The flow of the pair grafting process, for $d = 1$, is shown in Fig. 4. The four possible outcomes of the pair grafting process with $d = 1$ are denoted by U_0^2, $U_{1,1}^2$, U_6^2, and U_4^0. The four possible outcomes of this process with $d = 0$, which we denote by L_2^0,$L_2^{1,1}$,L_2^6, and L_0^4, are symmetric. It is therefore enough to consider the upper and lower costs of U_0^2, $U_{1,1}^2$, U_6^2, and U_4^0. Due to lack of space, we omit the detailed cost analysis. The costs incurred are summerized in Table 1. All these costs exclude the elimination cost which is 1 for each eliminated element.

6.3 Grafting quartets A P_{01} that contains the four elements u, v, w, z, where $u < v$ and $u < w < z$, is grafted using the following simple algorithm:

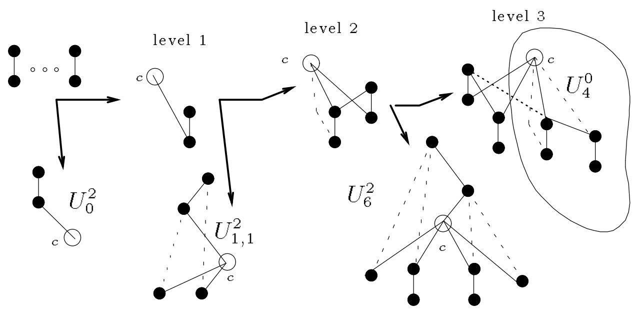

Figure 4: Flow of pair grafting when $d = 1$.

Class	Upper part eliminated		Lower part eliminated	
	Cost	Number of elements	Cost	Number of elements
U_0^2	0	2	1	0
$U_{1,1}^2$	4	2	3	2
U_6^2	10	2	7	6
U_4^0	$6-\gamma$	0	4	4

Table 1: Costs of pair grafting

1. Compare w and c.
2. If $w > c$ then remove the edge (u, w) and return the pair (u, v) to the input queue.
3. If $w < c$ then compare c with each of v and z.

The five possible outcomes of this process are shown in Fig. 5. Note that the fourth partial order obtained (denoted by $\overline{Q}_3^1$) is a special case of the third partial order obtained (denoted by Q_3^1). It is not necessary therefore to consider the fourth outcome and we are left with four cases that we denote by Q_0^2,$Q_2^{1,1}$,Q_3^1 and Q_4^0. The grafting process employed for P_{10}'s is symmetric. The quartet grafting process continues until three quartets from the same category are obtained. Oncemore, due to lack of space, we omit the detailed costs analysis. The costs, for $\gamma \le 2/3$, are summerized in Table 2.

6.4 The factory algorithm As mentioned before, the factory $\hat{\mathcal{G}}_k$ is composed of two sub-factories. The first uses the string $\mathcal{W} = 01(10)^\omega$ while the second one uses the string $\mathcal{W} = 10(01)^\omega$. We describe the operation of the first sub-factory (whose inputs are P_{01}'s, pairs and singletons). The other sub-factory works in a symmetric way. The operation of the first sub-factory is composed of the following steps:

(1) Generate a hyperpair P_w, where $w = 01(10)^h$ and $h = \lceil \log_2 n^{1/8} \rceil$, and let c be its centre. The centre c will be the centre of the generated partial order.

(2) The following steps are applied until k elements above c, or k elements below c, are placed in the output partial order.

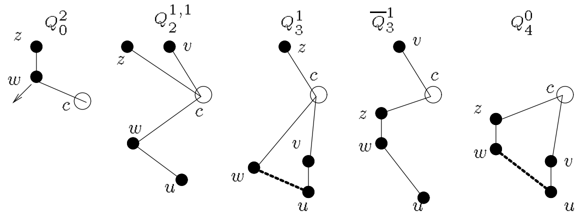

Figure 5: Possible outcomes of P_{01} grafting.

Class	Upper part eliminated		Lower part eliminated	
	Cost	Number of elements	Cost	Number of elements
Q_0^2	1	2	2	0
$Q_2^{1,1}$	3	2	4	2
Q_3^1	$4-3\gamma/4$	1	3	3
Q_4^0	$4-\gamma$	0	2	4

Table 2: Costs of P_{01} grafting.

(2a) Apply the quartet grafting process until three tuples from one of the categories Q_3^1, Q_4^0 or Q_0^2 are available. Elements from category $Q_2^{1,1}$ are immediately placed in the output partial order and the grafting continues.

(2b) If three tuples from Q_3^1 or from Q_4^0 are available, apply the pair grafting process with $d = 0$, until either $L_2^{1,1}$, L_0^4, or L_2^6 is obtained. Elements found in category L_2^0 are immediately placed in the final partial order.

(2c) If three tuples from Q_0^2 are available, apply the pair grafting process with $d = 1$ until either $U_{1,1}^2$, U_4^0, or U_6^2 is obtained. Elements found in category U_0^2 are immediately placed in the final partial order.

(2d) The tuple obtained using the pair grafting, and r elements from tuples obtained using quartet grafting, are placed in the output partial order. There are nine different cases here: $\{Q_3^1, Q_4^0\} \times \{L_2^{1,1}, L_0^4, L_2^6\} \cup \{Q_0^2\} \times \{U_{1,1}^2, U_4^0, U_6^2\}$. For each one of these cases we choose an optimal value of r.

(3) Finally, prune elements from P_w in order to achieve an optimal size (this is required only if $Q_2^{1,1}$, U_0^2 or L_2^0 were encountered) and output this $S \in \hat{\mathcal{S}}_k^k$.

The sub-factory maintains three counters q_3^1, q_4^0 and q_0^2 which are initially set to 0. Whenever a Q_3^1,Q_4^0 or a Q_0^2 is obtained, in step (2a), the corresponding counter is incremented. When a certain part of a Q_3^1,Q_4^0 or a Q_0^2 is 'consumed', in step (2d), the corresponding counter is decremented by the appropriate, not necessarily integral, amount. The quartet grafting process activated in step (2a) is carried out until one of these counters reaches a value of at least 3.

For each $Q_2^{1,1}$ obtained in step (2a), and each U_0^2 or L_2^0 obtained in steps (2b) and (2c), an appropriate number of elements is to be pruned in step (3). Two counters p_0 and p_1 maintain the number of elements that need to be pruned below and above c, respectively. Before outputting the partial order, $\lfloor p_1 \rfloor$ elements above c and $\lfloor p_0 \rfloor$ elements below c are pruned.

We depict the flavour of the cost analysis by considering one of the worst cases of the factory. In the following, we fix $\gamma \simeq 0.637985$ which is the optimal value. For each U_0^2 obtained in (2c), we prune $r \simeq 2.1382$ elements below c. The pruning of the r elements below c cuts $pr_0(\mathcal{W}) \cdot r$ edges. This pruning generates however r new elements in either singletons or pairs (because the pruning process separates r singletons, or $r/2$ pairs, from the centre). These elements can be returned to the factory as pairs and since $r > 2$, at least one pair is returned to the factory for every pair that was utilized. Hence, there is no need to break quartets into pairs.

Recycling the upper part cuts one edge for each pair and recycling the lower part cuts $(1/2 - \gamma/4)r$ edges (because of the recycling restrictions). Thus, the lower cost is $pr_0(\mathcal{W}) \cdot r + 1$, obtaining r eliminated elements and the upper cost is $(pr_0(\mathcal{W}) + 1/2 - \gamma/4)r$ obtaining one eliminated element. Recall also that the elimination cost is a single edge per eliminated element. Hence, the upper and lower element costs are:

$$1 + \frac{(1.5 + 1/2 - \gamma/4)r}{2} = 1 + \frac{1.5r + 1}{r} \simeq 2.96768 \,.$$

The cost analysis of all the other cases is omitted.

7 Concluding remarks

We have improved the results of Schönhage *et al.* [SPP76] and Blum *et al.* [BFP+73] and obtained a better algorithm for the selection of the median. Although the improvement, is quite modest, many new ideas were needed to obtain this improvement. The new ideas introduced may lead to further improvements. Our current constructions, however, are already quite involved and a considerable effort was devoted to their optimization. Obtaining further improvements is not likely to be an easy task. Further narrowing the gap between the known upper and lower bounds on the number of comparisons needed to select the median remains a challenging open problem.

References

[Aig82] M. Aigner. Selecting the top three elements. *Discrete Applied Mathematics*, 4:247–267, 1982.

[BFP+73] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[BJ85] S.W. Bent and J.W. John. Finding the median requires $2n$ comparisons. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, Providence, Rhode Island*, pages 213–216, 1985.

[CM89] W. Cunto and J.I. Munro. Average case selection. *Journal of the ACM*, 36(2):270–279, 1989.

[DZ95] D. Dor and U. Zwick. Finding percentile elements. In *Proceedings of the 3rd Israel Symposium on Theory and Computing systems*, 1995.

[Eus93] J. Eusterbrock. Errata to "Selecting the top three elements" by M. Aigner: A result of a computer-assisted proof search. *Discrete Applied Mathematics*, 41:131–137, 1993.

[FG78] F. Fussenegger and H.N. Gabow. A counting approach to lower bounds for selection problems. *Journal of the ACM*, 26(2):227–238, April 1978.

[FJ59] L.R. Ford and S.M. Johnson. A tournament problem. *American Mathematical Monthly*, 66:387–389, 1959.

[FR75] R.W. Floyd and R.L. Rivest. Expected time bounds for selection. *Communication of the ACM*, 18:165–173, 1975.

[HS69] A. Hadian and M. Sobel. Selecting the t-th largest using binary errorless comparisons. *Colloquia Mathematica Societatis János Bolyai*, 4:585–599, 1969.

[Hya76] L. Hyafil. Bounds for selection. *SIAM Journal on Computing*, 5:109–114, 1976.

[Joh88] J.W. John. A new lower bound for the set-partition problem. *SIAM Journal on Computing*, 17(4):640–647, August 1988.

[Kir81] D.G. Kirkpatrick. A unified lower bound for selection and set partitioning problems. *Journal of the ACM*, 28:150–165, 1981.

[Kis64] S.S. Kislitsyn. On the selection of the k-th element of an ordered set by pairwise comparisons. *Sibirsk. Mat. Zh.*, 5:557–564, 1964.

[MP82] I. Munro and P.V. Poblete. A lower bound for determining the median. Technical Report Research Report CS-82-21, University of Waterloo, 1982.

[Poh72] I. Pohl. A sorting problem and its complexity. *Communication of the ACM*, 15:462–464, 1972.

[RH84] P.V. Ramanan and L. Hyafil. New algorithms for selection. *Journal of Algorithms*, 5:557–578, 1984.

[Sch32] J. Schreier. On tournament elimination systems. *Mathesis Polska*, 7:154–160, 1932. (in Polish).

[SPP76] A. Schönhage, M. Paterson, and N. Pippenger. Finding the median. *Journal of Computer and System Sciences*, 13:184–199, 1976.

[SY80] P. Stockmeyer and F.F. Yao. On the optimality of linear merge. *SIAM Journal on Computing*, 9:85–90, 1980.

[Yap76] C.K. Yap. New upper bounds for selection. *Communication of the ACM*, 19(9):501–508, September 1976.