Finding the αn-th largest element

Dorit Dor * *Uri Zwick* *

February 27, 1994
Revised: December 19, 1994

Abstract

We describe an algorithm for selecting the αn-th largest element (where $0 < \alpha < 1$), from a totally ordered set of n elements, using at most $(1 + (1 + o(1))H(\alpha))\cdot n$ comparisons, where $H(\alpha)$ is the binary entropy function and the $o(1)$ stands for a function that tends to 0 as α tends to 0. For small values of α this is almost the best possible as there is a lower bound of about $(1 + H(\alpha))\cdot n$ comparisons. The algorithm obtained beats the global $3n$ upper bound of Schönhage, Paterson and Pippenger for $\alpha < 1/3$.

1 Introduction

The selection problem is one of the most fundamental problems of computer science. Given a set X containing n distinct elements, and given a number $1 \le i \le n$, we would like to find the element whose rank in X is exactly i, 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 element of rank i is simply the i-th smallest element. By symmetry, the problems of finding the i-th largest element and the i-th smallest element are equivalent.

Only in the early 70's, was it shown, by Blum, Floyd, Pratt, Rivest and Tarjan [BFP$^+$73], that the selection problem can be solved in $O(n)$ time. Blum et al. obtained a global $5.43n$ bound on the number of comparisons needed to select any element from a set of n elements. Their bound was improved a few years later by Schönhage, Paterson and Pippenger [SPP76]. Schönhage et al. describe a beautiful algorithm for finding the median (the $n/2$-th largest element) using at most $3n + o(n)$ comparisons. It is not difficult to see that any element, not just the median, can be found using their algorithm within the same comparison bound. In a recent work [DZ95], we have been able to improve the long standing upper bound of Schönhage, Paterson and Pippenger to about $2.95n$.

Blum et al. [BFP$^+$73], also describe a variant of their $5.43n$ algorithm that is particularly suited for finding (relatively) low ranked elements. They show that the αn-th element (or more precisely, the $\lceil \alpha n \rceil$-th element), where $0 < \alpha < 1$, can be found using, for small α's, about $(1 + 10.86\cdot\alpha \log \frac{1}{\alpha})\cdot n$ comparisons (all logarithms in this paper are taken to base two). This alternative is better than their original version when $\alpha < 0.203$. Schönhage et al. [SPP76] make no attempt to find lower ranked elements faster.

In this work we extend the results of both Schönhage et al. [SPP76] and Blum et al. [BFP$^+$73], and obtain an algorithm for the selection of the αn-th element using at most

$$(1 + \alpha \log \frac{1}{\alpha} + O(\alpha \log\log \frac{1}{\alpha}))\cdot n$$

*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 of authors: {ddorit,zwick}@math.tau.ac.il.

comparisons. This comes very close to a $(1+H(\alpha))\cdot n - o(n)$ lower bound obtained by Bent and John [BJ85] (see also [Joh88]). As $H(\alpha) = \alpha\log\frac{1}{\alpha} + (1-\alpha)\log\frac{1}{1-\alpha} = \alpha\log\frac{1}{\alpha} + O(\alpha)$, the coefficient of $\alpha\log\frac{1}{\alpha}$ in our upper bound cannot be improved.

Our bound beats the $3n$ bound of Schönhage et al. as soon as $\alpha < 1/3$. Our algorithm is efficient therefore not only for very small α's but also for α's not so far away from $1/2$. We hope that some of the new ideas presented here could be used to obtain a better algorithm for finding the median.

A previous attempt to combine the algorithms of Blum et al. [BFP$^+$73] and of Schönhage et al. [SPP76] was made by Motoki [Mot82]. His algorithm finds the αn-th element using, for small α's, about $(1+5\alpha\log\frac{1}{\alpha})\cdot n$ comparisons. In his approach, the median algorithm of Schönhage et al. is used as a 'black-box'. We obtain our near optimal upper bound by adapting the algorithm of Schönhage et al. for various values of α.

Many researchers have worked on the problem of finding the i-th largest element when i is a small fixed integer. The case $i=2$ was already considered by Lewis Carroll [Car83]. The exact number of comparisons required in this case, $n + \lceil\log n\rceil - 2$, was found by Schreier [Sch32] and Kislitsyn [Kis64]. Hadian and Sobel [HS69] (see also [Knu73], Section 5.3.3) use a variant of tree selection to find the i-th element using at most $n - i + (i-1)\lceil\log(n+2-i)\rceil$ comparisons. Their algorithm is improved slightly by Yap [Yap76] and by Ramanan and Hyafil [RH84]. All these algorithms have however non-linear complexities when $i = \alpha n$.

Schönhage, Paterson and Pippenger [SPP76] also show that a conjecture of Yao [Yao74] (the statement of this conjecture is given in Section 5) implies the existence of a median finding algorithm that uses at most $2.5n + o(n)$ comparisons. We extend this implication and show that Yao's conjecture also implies that the αn-th element can be found using at most

$$(1 + \alpha\log\frac{1}{\alpha} + O(\alpha))\cdot n$$

comparisons. This is even closer, but still above, the $(1+H(\alpha))\cdot n$ lower bound. We show, in particular, that if Yao's conjecture is true then the αn-th element, where $1/3 \le \alpha \le 1/2$ can be found using at most $(2+\alpha)\cdot n + o(n)$ comparisons.

All the results mentioned so far deal with the number of comparisons needed to select the i-th largest element in the *worst case*. Floyd and Rivest [FR75] had shown that the i-th largest element can be found using an *expected* number of $n + i + o(n)$ comparisons. Cunto and Munro [CM89] had shown the bound of Floyd and Rivest is tight. For $i \le n/2$, at least $n + i - o(n)$ comparisons are necessary, on the average, to find the i-th largest element.

In the next section we explain the notion of *factory production* that lies at the heart of the algorithm of Schönhage et al. In Section 3 we describe our new selection algorithm. Our selection algorithm uses the new factories we describe in Section 4. In Section 5 we describe the further implications of Yao's conjecture. We end, in Section 6 with some concluding remarks.

2 Factory production

Denote by S_k^m a partial order composed of a *center* element, m elements larger than the center and k elements smaller than the center (see Fig. 1). 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 currently best way of producing a single S_k^k, for example, requires about $6k$ comparisons (find the median of $2k+1$ elements using the $3n$ 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,

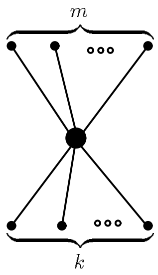

Figure 1: The partial order S_k^m.

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 input 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 *For any $k \geq 1$, 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*

$$U_k \sim 3.5k, \quad I_k = O(k^2), \quad R_k = O(k^2).$$

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 implies therefore the existence of a $3.5n + o(1)$ median algorithm to which Schönhage et al. refer as the basic median algorithm.

The basic median algorithm of Schönhage et al. sets up a factory F_k, where $k \sim n^{1/4}$, and starts generating S_k^k's. The centers of these S_k^k's are inserted into an ordered list using binary insertion. When the list is long enough, the center of the upper S_k^k and the elements above it are too large to be the median, and the center of the of lower S_k^k and the elements below it are too small to be the median. These elements are eliminated. The lower half of the upper S_k^k and the upper half of the lower S_k^k are returned to the factory.

The $3n + o(n)$ median algorithm is obtained by replacing the factory F_k by a factory G_k designed to utilize known relations among elements returned to the factory, when there are such relations. The factory G_k is described by Schönhage et al. only as a part of their final $3n + o(n)$ algorithm. For our purposes it is more convenient to describe these factories separately. To facilitate the analysis of algorithms that use *green* factories (i.e., factories that support recycling) for S_k^m, we introduce the following two additional characteristics: the *lower unit cost* U^0, which is the *amortized* production cost of an S_k^m whose upper m elements will eventually be returned (together) to the factory; and the *upper unit cost* U^1, which is the *amortized* production cost of an S_k^m whose lower k elements will eventually be returned (together) to the factory. A green factory does not know in advance whether the lower or upper part of a generated S_k^m will be recycled and if so which part. This is set by an adversary.

If a green factory generated l_0 S_k^m's whose upper part was recycled, l_1 S_k^m's whose lower part was recycled and l S_k^m's of which no part was recycled, and if production has stopped due to lack of input, then the total number of comparisons performed by the factory is at most $I + l_0 \cdot U^0 + l_1 \cdot U^1 + l \cdot U$. Note that this is now an amortized bound that holds only when the production of the factory has stopped.

Though not stated explicitly, the following result is implicit in [SPP76].

Theorem 2.2 *For any $k \geq 1$, there is a green factory G_k for S_k^k with initial cost I_k, upper and lower unit costs U_k^0, U_k^1, unit cost U_k and production residue R_k satisfying*

$$U_k^0, U_k^1 \sim 3k, \quad U_k \sim 3.5k, \quad I_k, R_k = O(k^2).$$

To reduce the number of comparisons required to find the αn-th element, we need factories that generate unbalanced S_k^m's. In Section 4 we construct a new family $G_k^{k,l}$ of factories that generate $S_k^{k,l}$'s, where $S_k^{k,l} = S_k^{(k+1)2^l-1}$. These factories have the following characteristics:

Theorem 2.3 *For any $k \geq 1$ and $l \geq 0$, there is a green factory $G_{k,l}$ for $S_k^{k,l} = S_k^{(k+1)2^l-1}$ with initial cost $I_{k,l}$, upper and lower unit costs $U_{k,l}^0, U_{k,l}^1$, unit cost $U_{k,l}$, and production residue $R_{k,l}$ satisfying*

$$U_{k,l}^0 \sim (l+3)k, \quad U_{k,l}^1 \sim (2^l + l + 2)k, \quad U_{k,l} = O(k), \quad I_{k,l}, R_{k,l} = O(k^2).$$

The construction of these new factories is the more difficult part of our work and we defer their description till after we describe, in the next section, the relatively simple selection algorithm that uses them.

3 New selection algorithm

In this section we describe our algorithm for the selection of the αn-th element. The algorithm presented is a generalization of the median algorithm of Schönhage et al. [SPP76]. Instead of using the factories of Theorem 2.2 to generate S_k^k's, we use the factories of Theorem 2.3 to generate $S_k^{k,l}$'s. Recall that $S_k^{k,l} = S_k^{(k+1)2^l-1}$. When k is large, a proportion of about $1/(2^l+1)$ of the elements in each $S_k^{k,l}$ are below the center. For every $0 < \alpha < 1$ we choose such a proportion that minimizes the number of comparisons. The simple framework of the algorithm is presented in the next Theorem.

Theorem 3.1 *Let $0 < \alpha < 1$ and let $l \geq 0$ be a fixed integer. If there exist factories $G_{k,l}$ for the generation of $S_k^{k,l}$'s with the following characteristics*

$$U_{k,l}^0 \sim A^0 \cdot k, \quad U_{k,l}^1 \sim A^1 \cdot k, \quad U_{k,l} = O(k), \quad I_{k,l}, R_{k,l} = O(k^2)$$

then there exists an algorithm for the selection of the αn-th element using at most

$$\left(\alpha \cdot A^0 + \frac{1-\alpha}{2^l} \cdot A^1\right) \cdot n + o(n)$$

comparisons.

Proof: The algorithm sets up a factory $G_{k,l}$ for the generation of $S_k^{k,l}$'s, where $k = \lfloor n^{1/4} \rfloor$. The n input elements are fed into this factory, as singletons, and the production of $S_k^{k,l}$'s commences. The centers of the generated $S_k^{k,l}$'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 that the center of the upper (i.e., last) $S_k^{k,l}$ in L and the elements above it are too large to be the αn-th element, or that the center of the lower (i.e., first) $S_k^{k,l}$ and the elements below it are too small to be the αn-th element (sometimes, both conditions hold). Elements too large or too small to be the αn-th element are eliminated. The lower elements of the upper $S_k^{k,l}$, and the upper elements of the lower $S_k^{k,l}$ are returned to the factory for recycling.

Let t be the 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 $N = ((k+1)2^l + k)t + r$.

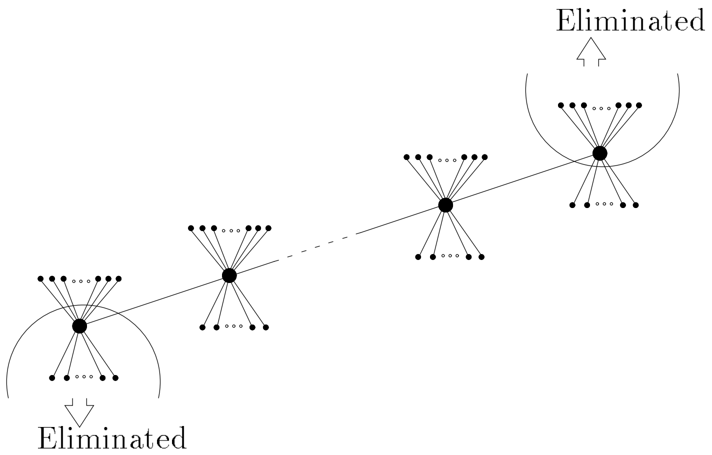

Figure 2: The ordered list of $S_k^{k,l}$'s.

We refer to the αn-th element of the original set, i.e., the element we are supposed to select, as the *percentile element*. 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 than or equal to the center of the upper $S_k^{k,l}$ of the list is $N_0 = (k+1)t$. The number of elements known to be greater or equal to the center of the lowest $S_k^{k,l}$ of the list is $N_1 = (k+1)2^l t$. Note that $N_0 + N_1 = N + t - r$ as the centers of all the $S_k^{k,l}$'s of the list satisfy both these criteria.

The algorithm consists of the following interconnected processes:

(i) Whenever sufficiently many elements are supplied to the factory $G_{k,l}$, a new copy of $S_k^{k,l}$ is produced and its center is inserted into the list L using binary insertion.

(ii) Whenever $N_0 > i$, the center of the upper $S_k^{k,l}$ and the elements above it are eliminated, as they are too big to be the percentile element. The lower elements of the upper $S_k^{k,l}$ are recycled.

(iii) Whenever $N_1 > N - i + 1$, the center of the lower $S_k^{k,l}$ and the elements below it are eliminated, as they are too small to be the percentile element. The upper elements of the lower $S_k^{k,l}$ are recycled. The value of i is updated accordingly, i.e., $i \leftarrow i - (k+1)$.

If $t - 1 > r$ then at least one of (ii) and (iii) is applicable, as if $N_0 \le i$ and $N_1 \le N - i + 1$ then $N + t - r = N_0 + N_1 \le N + 1$. If (i) is not applicable then by the factory definition we have $r \le R_{k,l}$. When no one of (i),(ii) and (iii) can be applied we get that $t - 1 \le r \le R_{k,l} = O(k^2)$. 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, $(k+1)2^l$ elements above the percentile element are eliminated. Step (ii) is therefore performed a total of $t_1 \le (1-\alpha)n/(k+1)2^l < (1-\alpha)n/k2^l$ times. Whenever (iii) is performed, $k+1$ elements below the percentile element are eliminated. Step (iii) is therefore performed a total of $t_0 \le \alpha n/(k+1) < \alpha n/k$ times. The factory $G_{k,l}$ generates all together t_0 $S_k^{k,l}$'s whose upper part is recycled, t_1 $S_k^{k,l}$'s whose lower part is recycled and t^* $S_k^{k,l}$'s that are not recycled at all, where $t^* = O(k^2)$ is the final length of the list L.

The total cost of the binary insertions into the list L is at most $(t_0 + t_1 + t^*) \log n = O((n/k + k^2) \log n)$ which is $o(n)$. The cost of the binary insertions is therefore negligible.

The total number of comparisons performed by the factory $G_k^{k,l}$ is

$$\begin{aligned} & I_{k,l} + t_0 \cdot U^0_{k,l} + t_1 \cdot U^1_{k,l} + t^* \cdot U_{k,l} \\ \leq\ & O(k^2) + \frac{\alpha n}{k} \cdot A^0 k + \frac{(1-\alpha)n}{k 2^l} \cdot A^1 k + O(k^3) \\ =\ & \left(\alpha \cdot A^0 + \frac{1-\alpha}{2^l} \cdot A^1\right) \cdot n + O(k^3). \end{aligned}$$

The total number of comparisons performed is therefore $(\alpha \cdot A^0 + \frac{1-\alpha}{2^l} \cdot A^1) \cdot n + o(n)$ as required. □

Using the factories of Theorem 2.3 we obtain the following corollary:

Corollary 3.2 *If $0 < \alpha < 1$ and $l \geq 0$ is a fixed integer then the αn-th element can be selected using at most*

$$\left(1 + (l+2)(\alpha + \frac{1-\alpha}{2^l})\right) \cdot n + o(n)$$

comparisons.

Finally, by choosing an optimal value of l for each α we obtain:

Theorem 3.3 *The αn-th element, among n elements, can be selected using at most*

$$(1 + \alpha \log \frac{1}{\alpha} + \alpha \log\log \frac{1}{\alpha} + O(\alpha)) \cdot n$$

comparisons.

Proof: Letting $l = \lfloor \log \frac{1}{\alpha} + \log\log \frac{1}{\alpha} \rfloor$ in the expression obtained in Corollary 3.2 we get

$$\begin{aligned} \left(1 + (l+2)(\alpha + \frac{1-\alpha}{2^l})\right) &\leq 1 + (\log \frac{1}{\alpha} + \log\log \frac{1}{\alpha} + 2) \cdot (\alpha + \frac{2\alpha(1-\alpha)}{\log \frac{1}{\alpha}}) \\ &= 1 + \alpha \log \frac{1}{\alpha} + \alpha \log\log \frac{1}{\alpha} + O(\alpha) \end{aligned}$$

as required. □

Some remarks are in place. By choosing $l = 0$ our algorithm degenerates to the Schönhage et al. algorithm. By choosing $l = 2$, we get an algorithm whose complexity is $(2 + 3\alpha) \cdot n + o(n)$. This algorithm improves upon the algorithm of Schönhage et al. for every $\alpha < 1/3$. By choosing $l = 3$, we get an $\frac{13+35\alpha}{8} \cdot n + o(n)$ algorithm. This improves upon the previous algorithm whenever $\alpha < 3/11$. These results are summarized in Table 1 brought in Section 6. Curiously, the results obtained using $l = 1$, are superseded by the results obtained using $l = 0$, when $\alpha > 1/3$, and using $l = 2$, when $\alpha < 1/3$.

The fact that the optimal choice for l is roughly $\log \frac{1}{\alpha} + \log\log \frac{1}{\alpha}$ is somewhat surprising. With this choice, the proportion of elements in the lower parts of the $S_k^{k,l}$'s is about $\alpha' = \alpha / \log \frac{1}{\alpha} < \alpha$. The algorithm starts by eliminating, using step (iii), many element *below* the αn-th element of the original set until it becomes the $\alpha' N$-th element of the remaining set. The algorithm then proceeds by alternately usings steps (ii) and (iii), throwing elements in the right proportion from both above and below the sought element so that it stays the α'-percentile element of the surviving elements until the conclusion of the algorithm.

Finally, we should point out that to get the asymptotic result stated in Theorem 3.3, we do not have to use the factories of Theorem 2.3. It is enough to construct similar factories in which $U^0_{k,l} \sim (l + O(1))k$ and $U^1_{k,l} \sim (2^l + l + O(1))k$. Such factories can be constructed, as will be shown in the next section, using any linear median finding algorithm. The factories of Schönhage et al. are needed however to get the results brought in Table 1.

4 Factories for $S_k^{k,l}$

In this section we describe the efficient factories $G_k^{k,l}$, for the generation of $S_k^{k,l}$'s, whose existence is promised in Theorem 2.3. The new factories $G_k^{k,l}$ use, as sub-production units, the factories G_k obtained by Schönhage et al. whose characteristics are described in Theorem 2.2.

The section is divided into three subsections. In the first subsection we remind the reader what *hyperpairs* are. In the second subsection we sketch the construction of the factory G_k of Schönhage et al. mentioning details that are relevant for our construction, and finally in the third subsection we describe our new factories $G_k^{k,l}$.

4.1 Hyperpairs

Hyperpairs are very natural constructs introduced by Schönhage et al. and used also by us.

Definition 4.1 *A hyperpair P_w, where w is a binary string, is a finite partial order with a distinguished element, the* center*, defined recursively by*

1. *P_λ is a single element (λ here stands for the empty string).*
2. *P_{w1} is obtained from two disjoint P_w's by comparing their centers and taking the higher as the new center. P_{w0} is obtained in the same way but taking the lower of the two centers as the new center.*

Some basic properties of hyperpairs are brought in the following Lemma. The proof of the Lemma is simple and can be found in [SPP76].

Lemma 4.2 *If c is the center of a hyperpair P_w where w contains h zeros and $r-h$ ones then:*

1. *The center c together with the elements greater than it form a P_{0^h} with center c. The elements greater than c form a disjoint set of hyperpairs P_λ, P_0, ... , $P_{0^{h-1}}$.*
2. *The center c together with the elements smaller than it form a $P_{1^{r-h}}$ with center c. The elements smaller than c form a disjoint set of hyperpairs P_λ, P_1, ... , $P_{1^{r-h-1}}$.*
3. *P_w contains an S_k^m with center c for any $m < 2^h$ and $k < 2^{r-h}$.*

4.2 The factory G_k

We now sketch the operation of Schönhage et al.'s factory. Their factory G_k starts by producing hyperpairs from the following special family:

$$\begin{array}{llll} H_0 = P_\lambda & , \quad H_1 = P_0 & & \\ H_{2t} = P_{01(10)^{t-1}} & , \quad H_{2t+1} = P_{01(10)^{t-1}1} & , & \text{for} \quad t \geq 1. \end{array}$$

Some small H_r's are shown in Fig. 3. The curious anomaly in the first two bits of the strings that define the H_r's is essential. An H_i is easily obtained from two H_{i-1}'s using one comparison. By Lemma 4.2, an H_{2h}, where $h = \lceil \log(k+1) \rceil$ contains an S_k^k. When an H_{2h} is generated by the factory it is subjected to a process of *grafting* and then to a process of *pruning*. We do not describe these processes here as their description is not relevant to our algorithm. The result of these two processes in a partial order $\bar{S}_k^k$ on $2k+1$ elements that contains an S_k^k. The elements above the center of this $\bar{S}_k^k$ form a collection of

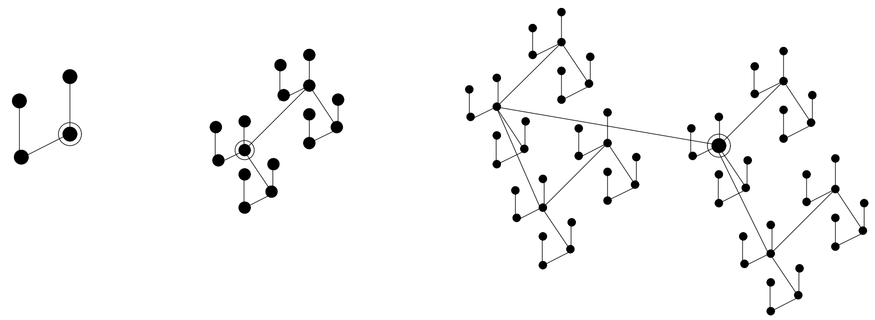

Figure 3: Some small H_r's ($H_2 = P_{01}$, $H_4 = P_{0110}$ and $H_6 = P_{011010}$).

disjoint P_{0^i}'s and the elements below the center form a collection of disjoint P_{1^i}'s. Typical $\bar{S}_k^k$'s that can be produced by such a factory are shown in Fig. 4. When the lower or upper part of an $\bar{S}_k^k$ 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 $\bar{S}_k^k$ and the utility obtained from its lower or upper parts when these parts are recycled. The two $\bar{S}_k^k$'s shown on the left of Fig. 4, for example, would probably have a larger non-amortized production cost than that of the S_k^k shown on the right of Fig. 4. This is compensated however by the fact that when one of the two remaining $\bar{S}_k^k$'s is recycled, some comparisons are saved.

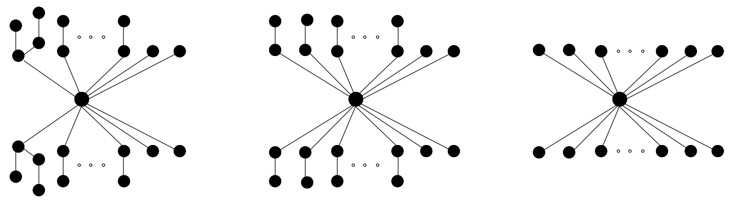

Figure 4: Some typical $\bar{S}_k^k$'s.

Although the $\bar{S}_k^k$'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. 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. We may therefore assume that each $\bar{S}_k^k$ generated by G_k is of a form similar to the one shown in the middle of Fig. 4, i.e., the elements above and below its center form collections of singletons and pairs.

4.3 The factory $G_k^{k,l}$

Our factory $G_k^{k,l}$ is composed of three units, as shown in Fig. 5. The first unit receives as inputs partial orders of the form P_{0^i}, where $0 \le i \le l$, and generates a stream of P_{0^l}'s. The generated P_{0^l}'s, or more accurately, the centers of the generated P_{0^l}'s, are fed into the second unit. The second unit is just Schönhage et al.'s factory G_k for the production of $\bar{S}_k^k$'s. The second unit produces $\bar{S}_k^k \circ P_{0^l}$'s, i.e., $\bar{S}_k^k$'s that each of their elements is also a center of a disjoint P_{0^l}. A typical $\bar{S}_k^k \circ P_{0^l}$ is shown in Fig. 6. The third unit of our factory receives $\bar{S}_k^k \circ P_{0^l}$'s and prunes them to $\bar{S}_k^{k,l}$'s, as shown in Fig. 6. An $\bar{S}_k^{k,l}$ is an $\bar{S}_k^k$ whose elements above the center are centers of disjoint P_{0^l}'s.

Recall (cf. Lemma 4.2) that the elements above the center of a P_{0^l} form a disjoint set of hyperpairs

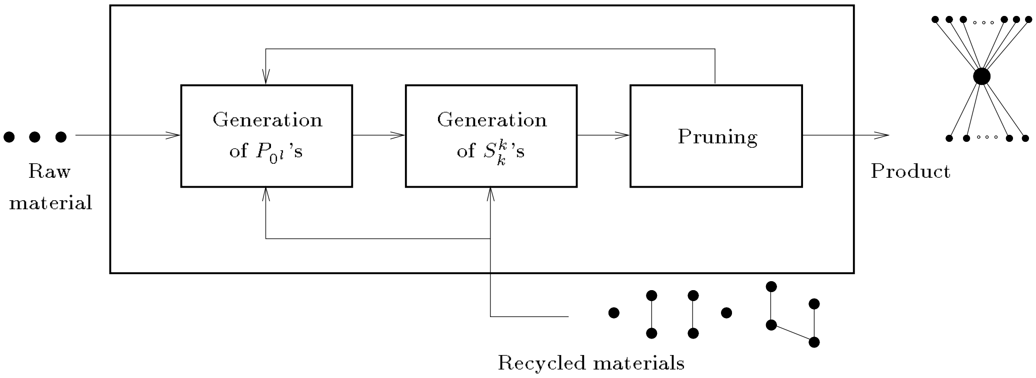

Figure 5: The factory $G_k^{k,l}$.

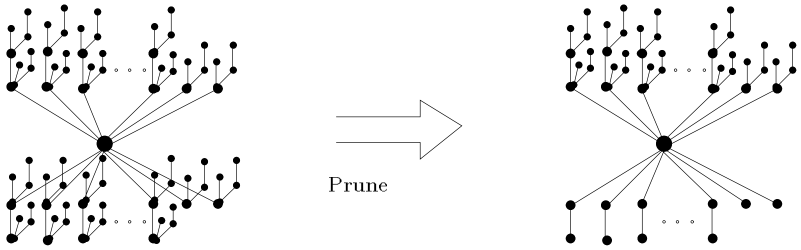

Figure 6: A typical $\bar{S}_k^{k,l}$.

$P_\lambda, P_0, \ldots, P_{0^{l-1}}$. A P_{0^l} is pruned by cutting the l edges that connect its center with the centers of the $P_\lambda, P_0, \ldots, P_{0^{l-1}}$ that are above it. An $\bar{S}_k^k \circ P_{0^l}$ is pruned into an $\bar{S}_k^{k,l}$ by pruning the k P_{0^l}'s that are below its center. Exactly lk edges are cut in this process.

The elements above the center of an $\bar{S}_k^{k,l}$ form a collection of P_{0^l}'s and $P_{0^{l+1}}$'s (pairs of P_{0^l}'s). The elements below the center of an $\bar{S}_k^{k,l}$ form a collection of singletons and pairs. If the upper part of an $\bar{S}_k^{k,l}$ is recycled, the P_{0^l}'s and the $P_{0^{l+1}}$'s are returned directly to the second production unit, i.e., to the factory G_k. If the lower part of an $\bar{S}_k^{k,l}$ is recycled, the singletons and pairs are returned to the first production unit, i.e., to the unit that produces P_{0^l}'s.

We now turn to the analysis of the factory constructed. The analysis is greatly simplified if a clever accounting principle introduced by Schönhage et al. is used. The information we care to remember on the elements that pass through the factory can always be described using an acyclic Hasse diagram. Only elements belonging to different connected components of the Hasse diagram are compared and as a result an edge between these two components is added. 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 that remain in the factory when the production stops. This number is at most the production residue of the factory and it can be attributed to the initial cost.

Looking at our factory, we see that no edges are cut in its first production unit. All edges cut in the second production unit are cut inside the factory G_k, we can therefore rely here on the analysis of G_k made by Schönhage et al. [SPP76]. Finally, exactly lk edges are cut in the third production unit for every $\bar{S}_k^{k,l}$ generated. To keep our books balanced, we should also count the number of edges contained in parts of $\bar{S}_k^{k,l}$'s that are not recycled. The upper part of an $\bar{S}_k^{k,l}$ contains $(k+1)2^l - 1$ edges. If an $\bar{S}_k^{k,l}$ is lower recycled, all the edges in the upper part are cut. The accounting of Schönhage et al. takes into account k of these edges, namely upper edges of the $\bar{S}_k^k$ on which the $\bar{S}_k^{k,l}$ is built. The upper unit cost $U_{k,l}^1$ of $G_k^{k,l}$ is therefore equal to the upper unit cost U_k^1 of G_k plus $lk+(k+1)(2^l-1)$. If an $\bar{S}_k^{k,l}$ is upper recycled, the k edges contained in its lower part are cut, but this is already accounted for in the lower unit cost of G_k. The lower unit cost $U_{k,l}^0$ of $G_k^{k,l}$ is therefore equal to the lower unit cost U_k^0 of G_k plus lk. The unit cost $U_{k,l}$ of $G_k^{k,l}$ is obtained in a similar manner.

Finally, notice that the production residue $R_{k,l}$ and the initial cost $I_{k,l}$ of $G_k^{k,l}$ are at most 2^l times the corresponding values for G_k. As l is constant, we get that $I_{k,l}, R_{k,l} = O(k^2)$ as required. This completes the proof of Theorem 2.3.

The construction just described also establishes the following general result: If there exists a factory F_k for S_k^k with unit cost $U_k \sim Ak$, then there exists a factory $F_{k,l}$ for $S_k^{k,l}$ with lower unit cost $U_{k,l}^0 \sim (l+A)k$ and upper unit cost $U_{k,l}^1 \sim (2^l+l+A-1)k$. Any algorithm that finds the median of n elements using at most $An/2+o(n)$ yields an S_k^k factory with unit cost $U_k \sim Ak$ (such a factory finds medians of $2k+1$ elements). Any linear median algorithm can be used therefore to obtain an $S_k^{k,l}$ factory with $U_{k,l}^0 \sim (l+O(1))k$ and $U_{k,l}^1 \sim (2^l+l+O(1))k$. This, as mentioned at the end of the previous section, is enough in order to imply the asymptotic bound of Theorem 3.3.

5 Implications of Yao's hypothesis

Constructing a partial order S_k^m on a set of $k+m+1$ elements is easily seen to be equivalent to finding the $k+1$-th element of the set. Is it easier to construct an S_k^m if more than $k+m+1$ elements are available? Yao [Yao74] conjectured that additional elements do not help.

Let $g(S_k^m, n)$ be the number of comparisons required to produce a partial order S_k^m given a set of $n \geq m+k+1$ elements. Also let $g(S_k^m) = g(S_k^m, m+k+1)$. Clearly $g(S_k^m, n) \leq g(S_k^m)$, for any $n \geq m+k+1$, as additional elements can always be ignored. Yao conjectured that $g(S_k^m, n) = g(S_k^m)$ for every $k, m \geq 0$ and every $n \geq m+k+1$.

Schönhage et al. [SPP76] showed that if Yao's conjecture is true, then the median of n elements can be found using at most $2.5n+o(n)$ comparisons. We extend this result and show that Yao's conjecture also implies better upper bounds for selecting the αn-th element, for any $\alpha > 0$. Specifically, we obtain the following:

Theorem 5.1 *Let $0 < \alpha < 1$ and let $l \geq 0$ be a fixed integer. If Yao's conjecture is true then the αn-th element can be found using at most*

$$\left(1+(l+2)\alpha+\frac{1-\alpha}{2^l}\right)\cdot n+o(n)$$

comparisons.

In particular, if $1/3 \leq \alpha \leq 1/2$ and if Yao's conjecture is true, then choosing $l=0$ we get that the αn-th element can be chosen using at most $(2+\alpha)\cdot n+o(n)$ comparisons. Theorem 5.1 implies the following result:

Theorem 5.2 *If Yao's conjecture is true, then the αn-th element can be found using at most*

$$(1+\alpha\log\frac{1}{\alpha}+O(\alpha))\cdot n$$

comparisons.

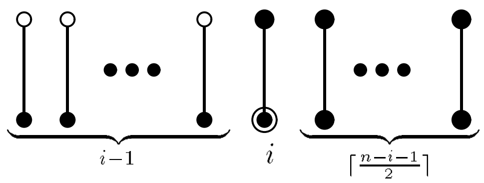

Figure 7: Finding an S_{i-1}^{n-i} when $2\lceil\frac{n+i-1}{2}\rceil$ elements are available.

The proof of both theorems is based on the following simple lemma:

Lemma 5.3 $\quad g(S_{i-1}^{n-i}, 2\lceil\frac{n+i-1}{2}\rceil) \leq \lceil\frac{n+i-1}{2}\rceil + g(S_{i-1}^{\lceil(n-i-1)/2\rceil})$

Proof: Consider the following simple algorithm for finding a partial order S_{i-1}^{n-i} in a set X of $2\lceil\frac{n+i-1}{2}\rceil$ elements. Divide X into pairs, compare each pair and let X' be the set of elements that lost in the comparisons. This takes $\lceil\frac{n+i-1}{2}\rceil$ comparisons. Then, select the i-th element of X' using an optimal algorithm. This takes $g(S_{i-1}^{\lceil(n-i-1)/2\rceil})$ comparisons. If c is the i-th element of X' then, c is larger than $i-1$ elements of X and smaller than $1+2\lceil\frac{n-i-1}{2}\rceil \geq n-i$ elements of X, as can be seen from Fig. 7. □

An immediate corollary of Lemma 5.3 is the following:

Corollary 5.4 *If Yao's conjecture is true then* $g(S_{i-1}^{n-i}) \leq \lceil\frac{n+i-1}{2}\rceil + g(S_{i-1}^{\lceil(n-i-1)/2\rceil})$.

We are now able to prove Theorems 5.1 and 5.2.

Proof of Theorem 5.1: For simplicity, we ignore integrality problems. We also write $S_{\alpha n}^{(1-\alpha)n}$ instead of $S_{\alpha n-1}^{(1-\alpha)n}$. We start by proving the claim of the Theorem for $l=0$. Using Corollary 5.4 and the identity $g(S_k^m) = g(S_m^k)$ we get:

$$g(S_{\alpha n}^{(1-\alpha)n}) \leq \frac{(1+\alpha)n}{2} + g(S_{\alpha n}^{(1-\alpha)n/2}) = \frac{(1+\alpha)n}{2} + g(S_{(1-\alpha)n/2}^{\alpha n})$$

Applying Corollary 5.4 once more we get

$$g(S_{\alpha n}^{(1-\alpha)n}) \leq \frac{(1+\alpha)n}{2} + \left(\frac{n}{2} + g(S_{(1-\alpha)n/2}^{\alpha n/2})\right) = \left(1+\frac{\alpha}{2}\right)\cdot n + g(S_{\alpha n/2}^{(1-\alpha)n/2}).$$

It follows now by induction that

$$g(S_{\alpha n}^{(1-\alpha)n}) \leq (2+\alpha)n + o(n).$$

Suppose now that $l > 1$. By applying Corollary 5.4 l times we obtain

$$\begin{aligned} g(S_{\alpha n}^{(1-\alpha)n}) &\leq \frac{(1+\alpha)n}{2} + g(S_{\alpha n}^{(1-\alpha)n/2}) \\ &\leq \frac{(1+\alpha)n}{2} + \frac{(1+3\alpha)n}{4} + g(S_{\alpha n}^{(1-\alpha)n/4}) \\ &\leq \left(\frac{1+\alpha}{2} + \frac{1+3\alpha}{4} + \cdots + \frac{1+(2^l-1)\alpha}{2^l}\right)\cdot n + g(S_{\alpha n}^{(1-\alpha)n/2^l}) \\ &= \left(l\alpha + (1-\frac{1}{2^l})(1-\alpha)\right)\cdot n + g(S_{\alpha n}^{(1-\alpha)n/2^l}). \end{aligned}$$

Letting now $n' = \left(\alpha + \frac{1-\alpha}{2^l}\right) \cdot n$ and $\alpha' = \frac{\alpha n}{n'}$ we get, using the claim of the Theorem with $l = 0$ that

$$g(S_{\alpha n}^{(1-\alpha)n/2^l}) = g(S_{\alpha' n'}^{(1-\alpha')n'}) \le (2+\alpha')n' + o(n) = \left(3\alpha + \frac{1-\alpha}{2^{l-1}}\right) \cdot n + o(n).$$

Combining the last two inequalities we get that

$$\begin{aligned} g(S_{\alpha n}^{(1-\alpha)n}) &\le \left[\left(l\alpha + (1 - \frac{1}{2^l})(1-\alpha)\right) \cdot n\right] + \left[\left(3\alpha + \frac{1-\alpha}{2^{l-1}}\right) \cdot n + o(n)\right] \\ &= \left(1 + (l+2)\alpha + \frac{1-\alpha}{2^l}\right) \cdot n + o(n) \end{aligned}$$

as required. □

Proof of Theorem 5.2: It is easy to check that the optimal choice for l in Theorem 5.1, ignoring again integrality problems, is $l = \log(\frac{1}{\alpha} - 1) - \log\log e$. Substituting this value into the bound of Theorem 5.1, we obtain

$$g(S_{\alpha n}^{(1-\alpha)n}) \le \left(1 + \log(\frac{1}{\alpha} - 1) + (2 + \log e - \log\log e)\alpha\right) \cdot n + o(n) \le (1 + \alpha \log \frac{1}{\alpha} + 2.914\alpha) \cdot n + o(n).$$

The integrality problems are solved by taking $l = \lfloor \log(\frac{1}{\alpha} - 1) - \log\log e \rfloor$. This only affects the $O(\alpha)$ term in the above expression. □

Even if Yao's conjecture is true, the proofs of Theorems 5.1 and 5.2 do not describe algorithms that achieve the bounds stated. They just prove that such algorithms exist. We can however find the following constructive use of Lemma 5.3. *Without* relying on Yao's hypothesis we obtain the following:

Theorem 5.5

$$g(S_{\alpha n}^{(1-\alpha)n}, (1+\alpha)n) \le (2 + 2\alpha) \cdot n + o(n).$$

Proof: Using Lemma 5.3, and using the $3n + o(n)$ median algorithm of Schönhage et al. we get

$$g(S_{\alpha n}^{(1-\alpha)n}, (1+\alpha)n) \le \frac{(1+\alpha)n}{2} + g(S_{\alpha n}^{(1-\alpha)n/2}) \le \frac{(1+\alpha)n}{2} + 3 \cdot \frac{(1+\alpha)n}{2} + o(n) = (2+2\alpha) \cdot n + o(n).$$

□

A few things are worth noticing. First, note the striking similarity between the bound of Corollary 3.2, obtained without relying on Yao's conjecture, and the bound of Theorem 5.1. Second, note the proximity of the bound just obtained in the proof of Theorem 5.2 with the lower bound $(1 + H(\alpha)) \cdot n = (1 + \alpha \log \frac{1}{\alpha} + \alpha/\ln 2 + O(\alpha^2)) \cdot n$ of Bent and John [BJ85]. In fact, our initial motivation for carrying out the computations of this section was an attempt to disprove Yao's conjecture. Finally, note that Theorem 5.5 gives an explicit algorithm for the generation of an $S_{\alpha n}^{(1-\alpha)n}$, when about $(1+\alpha)n$ elements are available, using only $(2+2\alpha) \cdot n + o(n)$ comparisons. If only n elements are available, and $\alpha \ge 1/3$, the best known method for generating an $S_{\alpha n}^{(1-\alpha)n}$ requires $3n + o(n)$ comparisons. This confronts Yao's conjecture with a concrete challenge.

6 Concluding remarks

We have extended the results of Schönhage et al. [SPP76] and of Blum et al. [BFP+73] and obtained better algorithms for the selection of the αn-th largest element where $\alpha < 1/3$. Our algorithm is almost optimal

l	New result	Optimal range	Assuming Yao	Optimal range
0	$3n$	$\frac{1}{3} \leq \alpha \leq \frac{1}{2}$	$(2+\alpha)\cdot n$	$\frac{1}{3} \leq \alpha \leq \frac{1}{2}$
1	$\frac{5+3\alpha}{2}\cdot n$	$\alpha = \frac{1}{3}$	$\frac{3+5\alpha}{2}\cdot n$	$\frac{1}{5} \leq \alpha \leq \frac{1}{3}$
2	$(2+3\alpha)\cdot n$	$\frac{3}{11} \leq \alpha \leq \frac{1}{3}$	$\frac{5+15\alpha}{4}\cdot n$	$\frac{1}{9} \leq \alpha \leq \frac{1}{5}$
3	$\frac{13+35\alpha}{8}\cdot n$	$\frac{1}{5} \leq \alpha \leq \frac{3}{11}$	$\frac{9+39\alpha}{8}\cdot n$	$\frac{1}{17} \leq \alpha \leq \frac{1}{9}$
4	$\frac{11+45\alpha}{8}\cdot n$	$\frac{5}{37} \leq \alpha \leq \frac{1}{5}$	$\frac{17+95\alpha}{16}\cdot n$	$\frac{1}{33} \leq \alpha \leq \frac{1}{17}$

Table 1: New bounds for the selection of the αn-th element

when α is small. We have also shown that a conjecture of Yao [Yao74] implies that an even better selection algorithm can be obtained. Table 1 summarizes some of new bounds we have obtained.

The two most challenging open problems related to selection are perhaps proving or disproving Yao's conjecture and significantly narrowing the gap between the lower and upper bounds on the number of comparisons required for the selection of the median.

Acknowledgement

We would like to thank Mike Paterson for several helpful discussions.

References

[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.

[Car83] L. Carroll. Lawn tennis turnaments. *St. James's Gazette*, pages 5–6, August 1883. Reprinited in "The complete stories of Lewis Carroll", Magpie Books Ltd., London, 1993, Pages 775-782.

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

[DZ95] D. Dor and U. Zwick. Selecting the median. In *Proceedings of the 6rd Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[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.

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

[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.

[Knu73] Donald E. Knuth. *Sorting and searching*, volume 3 of *The art of computer programming*. Addison-Wesley, 1973.

[Mot82] T. Motoki. A note on upper bounds for selection problems. *Information Processing Letters*, 15(5):214–219, December 1982.

[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.

[Yao74] F. Yao. On lower bounds for selection problems. Technical Report MAC TR-121, Mass. Inst. of Technology, 1974.

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