Raport Badawczy
Research Report

RB/68/2004

On Floyd and Rivest's SELECT Algorithm

Krzysztof C. Kiwiel

Instytut Badań Systemowych
Polska Akademia Nauk

Systems Research Institute
Polish Academy of Sciences

POLSKA AKADEMIA NAUK

Instytut Badań Systemowych

ul. Newelska 6

01-447 Warszawa

tel.: (+48) (22) 8373578

fax: (+48) (22) 8372772

Kierownik Pracowni zgłaszający pracę:
Prof. dr hab. inż. Krzysztof C. Kiwiel

Warszawa 2004

On Floyd and Rivest's SELECT algorithm

Krzysztof C. Kiwiel*

August 25, 2004

Abstract

We show that several versions of Floyd and Rivest's algorithm SELECT for finding the kth smallest of n elements require at most $n + \min\{k, n-k\} + o(n)$ comparisons on average and with high probability. This rectifies the analysis of Floyd and Rivest, and extends it to the case of nondistinct elements. Our computational results confirm that SELECT may be the best algorithm in practice.

Key words. Selection, medians, partitioning, computational complexity.

1 Introduction

The *selection problem* is defined as follows: Given a set $X := \{x_j\}_{j=1}^n$ of n elements, a total order $<$ on X, and an integer $1 \le k \le n$, find the *kth smallest* element of X, i.e., an element x of X for which there are at most $k-1$ elements $x_j < x$ and at least k elements $x_j \le x$. The *median* of X is the $\lceil n/2 \rceil$th smallest element of X. (Since we are *not* assuming that the elements are distinct, X may be regarded as a multiset).

Selection is one of the fundamental problems in computer science. It is used in the solution of other basic problems such as sorting and finding convex hulls. For good reviews of its literature, see, e.g., [DHUZ01, DoZ99, DoZ01] and [Knu98, §5.3.3]. We only stress that most references employ a comparison model (in which a selection algorithm is charged only for comparisons between pairs of elements), assuming that the elements are distinct. Then, in the worst case, selection needs at least $(2+\epsilon)n$ comparisons [DoZ01], whereas the pioneering algorithm of [BFP+72] makes at most $5.43n$, its first improvement [SPP76] needs $3n + o(n)$, and the most recent improvement in [DoZ99] takes $2.95n + o(n)$. Thus a gap of almost 50% still remains between the best lower and upper bounds in the worst case.

The average case is better understood. Specifically, for $k \le \lceil n/2 \rceil$, at least $n + k - 2$ comparisons are necessary [CuM89], [Knu98, Ex. 5.3.3–25], whereas the best upper bound is $n + k + O(n^{1/2} \ln^{1/2} n)$ [Knu98, Eq. (5.3.3.16)]. Yet this bound holds for a hardly implementable theoretical scheme [Knu98, Ex. 5.3.3–24], whereas a similar frequently cited bound for the algorithm SELECT of [FlR75b] doesn't have a full proof, as noted in [Knu98,

*Systems Research Institute, Newelska 6, 01–447 Warsaw, Poland (kiwiel@ibspan.waw.pl)

Ex. 5.3.3–24] and [PRKT83]. Significantly worse bounds hold for the classical algorithm FIND of [Hoa61], also known as quickselect, which partitions X by using the median of a random sample of size $s \geq 1$. In particular, for $k = \lceil n/2 \rceil$, the upper bound is $3.39n + o(n)$ for $s = 1$ [Knu98, Ex. 5.2.2–32] and $2.75n + o(n)$ for $s = 3$ [Grü99, KMP97], whereas for finding an element of random rank, the average cost is $3n + o(n)$ for $s = 1$, $2.5n + o(n)$ for $s = 3$ [KMP97], and $2n + o(n)$ when $s \to \infty$, $s/n \to 0$ as $n \to \infty$ [MaR01]. In practice FIND is most popular, because the algorithms of [BFP+72, SPP76] are much slower on the average [Mus97, Val00]. For the general case of nondistinct elements, little is known in theory about these algorithms, but again FIND performs well in practice [Kiw03a, Val00].

Our aim is to rekindle theoretical and practical interest in the algorithm SELECT of [FlR75b, §2.1] (the versions of [FlR75b, §2.3] and [FlR75a] are addressed in [Kiw04b, Kiw04a]). We show that SELECT performs very well in both theory and practice, even when equal elements occur. To outline our contributions in more detail, we recall that SELECT operates as follows. Using a small random sample, two elements u and v almost sure to be just below and above the kth are found. The remaining elements are compared with u and v to create a small selection problem on the elements between u and v that is quickly solved recursively. By taking a random subset as the sample, this approach does well against any input ordering, both on average and with high probability.

First, we revise SELECT slightly to simplify our analysis. Then, without assuming that the elements are distinct, we show that SELECT needs at most $n + \min\{k, n-k\} + O(n^{2/3} \ln^{1/3} n)$ comparisons on average; this agrees with the result of [FlR75b, §2.2] which is based on an unproven assumption [PRKT83, §5]. Similar upper bounds are established for versions that choose sample sizes as in [FlR75a, Meh00, Rei85] and [MoR95, §3.3]. Thus the average costs of these versions reach the lower bounds of $1.5n + o(n)$ for median selection and $1.25n + o(n)$ for selecting an element of random rank (yet the original sample size of [FlR75b, §2.2] has the best lower order term in its cost). We also prove that nonrecursive versions of SELECT, which employ other selection or sorting algorithms for small subproblems, require at most $n + \min\{k, n-k\} + o(n)$ comparisons with high probability (e.g., $1 - 4n^{-2\beta}$ for a user-specified $\beta > 0$); this extends and strengthens the results of [GeS03, Thm 1], [Meh00, Thm 2] and [MoR95, Thm 3.5].

Since theoretical bounds alone needn't convince practitioners (who may worry about hidden constants, etc.), a serious effort was made to design a competitive implementation of SELECT. Here, as with FIND and quicksort [Sed77], the partitioning efficiency is crucial. In contrast with the observation of [FlR75b, p. 169] that "partitioning X about both u and v [is] an inherently inefficient operation", we introduce a *quintary* scheme which performs well in practice.

Relative to FIND, SELECT requires only small additional stack space for recursion, because sampling without replacement can be done in place. Still, it might seem that random sampling needs too much time for random number generation. (Hence several popular implementations of FIND don't sample randomly, assuming that the input file is in random order, whereas others [Val00] invoke random sampling only when slow progress occurs.) Yet our computational experience shows that sampling doesn't hurt even on random inputs, and it helps a lot on more difficult inputs (in fact our interest in SELECT was sparked by the poor performance of the implementation of [FlR75a] on several inputs of

[Val00]). Most importantly, SELECT beats quite sophisticated implementations of FIND [Kiw03a, Kiw03b, Val00] in both comparison counts and computing times even for examples with relatively low comparison costs. To save space, only selected results are reported in §7.3 and [Kiw03a, Kiw03b], but our experience on many other inputs was similar. In particular, empirical estimates of the constants hidden in our bounds were always quite small. Further, the performance of SELECT is extremely stable across a variety of inputs, even for small input sizes (cf. §7.3). A theoretical explanation of these features will be undertaken elsewhere. For now, our experience supports the claim of [FlR75b, §1] that "the algorithm presented here is probably the best practical choice".

The paper is organized as follows. A general version of SELECT is introduced in §2, and its basic features are analyzed in §3. The average performance of SELECT is studied in §4. High probability bounds for nonrecursive versions are derived in §5. Partitioning schemes are discussed in §6. Finally, our computational results are reported in §7.

Our notation is fairly standard. $|A|$ denotes the cardinality of a set A. In a given probability space, P is the probability measure, and E is the mean-value operator.

2 The algorithm SELECT

In this section we describe a general version of SELECT in terms of two auxiliary functions $s(n)$ and $g(n)$ (the sample size and rank gap), which will be chosen later. We omit their arguments in general, as no confusion can arise.

SELECT picks a small random sample S from X and two pivots u and v from S such that $u \le x_k^* \le v$ with high probability, where x_k^* is the kth smallest element of X. Partitioning X into elements less than u, equal to u, between u and v, equal to v, and greater than v, SELECT either detects that u or v equals x_k^*, or determines a subset $\hat{X}$ of X and an integer $\hat{k}$ such that x_k^* may be selected recursively as the $\hat{k}$th smallest element of $\hat{X}$.

Below is a detailed description of the algorithm.

Algorithm 2.1.
SELECT(X, k) (Selects the kth smallest element of X, with $1 \le k \le n := |X|$)

Step 1 (*Initiation*). If $n = 1$, return x_1. Choose the sample size $s \le n - 1$ and gap $g > 0$.

Step 2 (*Sample selection*). Pick randomly a sample $S := \{y_1, \ldots, y_s\}$ from X.

Step 3 (*Pivot selection*). Set $i_u := \max\{\lceil ks/n - g \rceil, 1\}$, $i_v := \min\{\lceil ks/n + g \rceil, s\}$. Let u and v be the i_uth and i_vth smallest elements of S, found by using SELECT recursively.

Step 4 (*Partitioning*). By comparing each element x of X to u and v, partition X into $L := \{x \in X : x < u\}$, $U := \{x \in X : x = u\}$, $M := \{x \in X : u < x < v\}$, $V := \{x \in X : x = v\}$, $R := \{x \in X : v < x\}$. If $k < n/2$, x is compared to v first, and to u only if $x < v$ and $u < v$. If $k \ge n/2$, the order of the comparisons is reversed.

Step 5 (*Stopping test*). If $|L| < k \le |L \cup U|$, return u; else if $|L \cup U \cup M| < k \le n - |R|$, return v.

Step 6 (*Reduction*). If $k \le |L|$, set $\hat{X} := L$ and $\hat{k} := k$; else if $n - |R| < k$, set $\hat{X} := R$ and $\hat{k} := k - n + |R|$; else set $\hat{X} := M$ and $\hat{k} := k - |L \cup U|$. Set $\hat{n} := |\hat{X}|$.

Step 7 (*Recursion*). Return SELECT$(\hat{X}, \hat{k})$.

Our revision of the original version of SELECT [FlR75b, §2] has two features. First, the form of pivot ranks i_u and i_v at Step 3 will allow us to handle more general choices of the sample size s and gap g. Second, for distinct keys and $u < v$, the original version worked with just three sets: L, $U \cup M \cup V$ and R; in contrast, partitioning into five sets at Step 4 is needed when equal keys occur. Still, our revision inherits the following general properties, formulated as numbered remarks to ease future references.

Remarks 2.2. (a) The correctness and finiteness of SELECT stem by induction from the following observations. The returns of Steps 1 and 5 deliver the desired element. At Step 6, $\hat{X}$ and $\hat{k}$ are chosen so that the kth smallest element of X is the $\hat{k}$th smallest element of $\hat{X}$, and $\hat{n} < n$ (since $u, v \notin \hat{X}$). Also $|S| < n$ for the recursive calls at Step 3.

(b) When Step 5 returns u (or v), SELECT may also return information about the positions of the elements of X relative to u (or v). For instance, if X is stored as an array, its k smallest elements may be placed first via interchanges at Step 4 (cf. §6). Hence after Step 3 finds u, we may remove from S its first i_u smallest elements before extracting v. Further, Step 4 need only compare u and v with the elements of $X \setminus S$.

(c) The following elementary property is needed in §4. Let c_n denote the maximum number of comparisons taken by SELECT on any input of size n. Since Step 3 makes at most $c_s + c_{s-i_u}$ comparisons with $s < n$, Step 4 needs at most $2(n-s)$, and Step 7 takes at most $c_{\hat{n}}$ with $\hat{n} < n$, by induction $c_n < \infty$ for all n.

3 Preliminary analysis

In this section we analyze general features of sampling used by SELECT.

3.1 Outline of main proof techniques

Since our analysis involves many technicalities, we now outline the main strategy.

We wish to show that for sample sizes and gaps such that s, gn/s and $ne^{-2g^2/s}$ are $o(n)$, SELECT needs on average at most $n + \min\{k, n-k\} + o(n)$ comparisons. For an inductive proof, because the cost of Step 3 is at most twice $1.5s + o(n)$ from $s < n$, we only need to show that the cost of Step 4 is at most $n + \min\{k, n-k\} + o(n)$ and the cost of Step 7 is $o(n)$, since adding these three costs yields the desired estimate.

Our bounds on the costs of Steps 4 and 7 stem from bounds on the ranks of u and v in the input set X. Specifically, denote by $x_1^* \le \ldots \le x_n^*$ and $y_1^* \le \ldots \le y_s^*$ the sorted elements of the input set X and the sample set S, respectively. Thus x_k^* is the kth smallest element of X, whereas $u = y_{i_u}^*$ and $v = y_{i_v}^*$ at Step 3. Hence for $i_u \approx ks/n - g$ and $i_v \approx ks/n + g$, the positions of u and v in the sorted input should not deviate much from $k - gn/s$ and $k + gn/s$, respectively. Indeed, for the bounding indices

$$k_l := \max\{\lceil k - 2gn/s \rceil, 1\} \quad \text{and} \quad k_r := \min\{\lceil k + 2gn/s \rceil, n\}, \tag{3.1}$$

each of the *unfavorable* events $u < x_{k_l}^*$, $x_k^* < u$, $v < x_k^*$, $x_{k_r}^* < v$ has probability at most $e^{-2g^2/s}$ (bounded as the tail of the hypergeometric distribution; cf. Fact 3.1 below).

Hence, by the Boole-Benferroni inequality, the *favorable* event $x^*_{k_l} \le u \le x^*_k \le v \le x^*_{k_r}$ has probability at least $1-4e^{-2g^2/s}$. Our bound for Step 4 stems from the fact that for $k < n/2$ and $v \le x^*_{k_r}$, at most $k_r - 2$ elements $x < v$ are compared to u, whereas for $k \ge n/2$ and $x^*_{k_l} \le u$, at most $n - k_l - 1$ elements $x > u$ are compared to v. Similarly, at Step 7 for the favorable event, at most $k_r - k_l - 1$ elements $x^*_{k_l} < x < x^*_{k_r}$ comprise $\hat{X}$. In each case, expected values are bounded via the Chebyshev inequality (cf. Fact 3.2).

Unfortunately technical complications muddle the picture. First, separate treatment is needed for $k \le gn/s$ or $k > n - gn/s$ (when either u or v becomes redundant; cf. Rem. 3.7). Second, to get sharper estimates for specific choices of s and g, our bounds for Steps 4 and 7 employ s, gn/s and $ne^{-2g^2/s}$ instead of the simpler $o(n)$ notation.

3.2 Sampling deviations and expectation bounds

Our analysis hinges on the following bound on the tail of the hypergeometric distribution established in [Hoe63] and rederived shortly in [Chv79].

Fact 3.1. *Let s balls be chosen uniformly at random from a set of n balls, of which r are red, and r' be the random variable representing the number of red balls drawn. Let $p := r/n$. Then*

$$\mathrm{P}[r' \ge ps + g] \le e^{-2g^2/s} \quad \forall g \ge 0. \tag{3.2}$$

We shall also need a simple version of the (left) Chebyshev inequality [Kor78, §2.4.2].

Fact 3.2. *Let z be a nonnegative random variable such that $\mathrm{P}[z \le \zeta] = 1$ for some constant ζ. Then $\mathrm{E}z \le t + \zeta \mathrm{P}[z > t]$ for all nonnegative real numbers t.*

3.3 Pivot ranks

By intepreting the unfavorable events described below (3.1) in the setting of Fact 3.1, we now bound their probabilities via (3.2). Recall that $u = y^*_{i_u}$ and $v = y^*_{i_v}$ for $y^*_1 \le \ldots \le y^*_s$.

Lemma 3.3. (a) $\mathrm{P}[x^*_k < u] \le e^{-2g^2/s}$ *if* $i_u = \lceil ks/n - g \rceil$.
(b) $\mathrm{P}[u < x^*_{k_l}] \le e^{-2g^2/s}$.
(c) $\mathrm{P}[v < x^*_k] \le e^{-2g^2/s}$ *if* $i_v = \lceil ks/n + g \rceil$.
(d) $\mathrm{P}[x^*_{k_r} < v] \le e^{-2g^2/s}$.
(e) $i_u \ne \lceil ks/n - g \rceil$ *iff* $k \le gn/s$; $i_v \ne \lceil ks/n + g \rceil$ *iff* $n < k + gn/s$.

Proof. (a) If $x^*_k < y^*_{i_u}$, at least $s - i_u + 1$ sample elements y_i satisfy

$$y_i \ge x^*_{\bar{\jmath}+1} \quad \text{with} \quad \bar{\jmath} := \max\{j : x^*_j = x^*_k\}.$$

In the setting of Fact 3.1, we have $r := n - \bar{\jmath}$ red elements $x_j \ge x^*_{\bar{\jmath}+1}$, $ps = s - \bar{\jmath}s/n$ and $r' \ge s - i_u + 1$. Since $i_u = \lceil ks/n - g \rceil < ks/n - g + 1$ and $\bar{\jmath} \ge k$, we get

$$r' > ps + (\bar{\jmath} - k)s/n + g \ge ps + g.$$

Hence $\mathrm{P}[x^*_k < u] \le \mathrm{P}[r' \ge ps + g] \le e^{-2g^2/s}$ by (3.2).

(b) If $y^*_{i_u} < x^*_{k_l}$, at least i_u sample elements y_i satisfy

$$y_i \le x^*_r \quad \text{with} \quad r := \max\{j : x^*_j < x^*_{k_l}\}.$$

Thus we have r red elements $x_j \le x^*_r$, $ps = rs/n$ and $r' \ge i_u$. Now, $1 \le r \le k_l - 1$ implies $2 \le r+1 \le k_l = \lceil k - 2gn/s \rceil$ by (3.1) and thus $k_l < k - 2gn/s + 1$, so $-rs/n > -ks/n + 2g$. Hence

$$i_u - ps - g \ge ks/n - g - rs/n - g > 0,$$

i.e., $r' > ps + g$; invoke (3.2) as before.

(c) If $y^*_{i_v} < x^*_k$, at least i_v sample elements are at most x^*_r, where $r := \max_{x^*_j < x^*_k} j$. Thus we have r red elements $x_j \le x^*_r$, $ps = rs/n$ and $r' \ge i_v$. But

$$i_v - ps - g \ge ks/n + g - rs/n - g \ge 0$$

implies $r' \ge ps + g$, so again (3.2) yields the conclusion.

(d) If $x^*_{k_r} < y^*_{i_v}$, at least $s - i_v + 1$ sample elements are at least $x^*_{\bar{j}+1}$, where $\bar{j} := \max_{x^*_j = x^*_{k_r}} j$. Thus we have $r := n - \bar{j}$ red elements $x_j \ge x^*_{\bar{j}+1}$, $ps = s - \bar{j}s/n$ and $r' \ge s - i_v + 1$. Now, $i_v < ks/n + g + 1$ and $\bar{j} \ge k_r \ge k + 2gn/s$ (cf. (3.1)) yield

$$s - i_v + 1 - ps - g \ge \bar{j}s/n - ks/n - g - 1 + 1 - g \ge 0.$$

Thus $x^*_{k_r} < v$ implies $r' \ge ps + g$; hence $\mathrm{P}[x^*_{k_r} < v] \le \mathrm{P}[r' \ge ps + g] \le e^{-2g^2/s}$ by (3.2).

(e) Follows immediately from the properties of $\lceil \cdot \rceil$ [Knu97, §1.2.4]. □

3.4 Partitioning cost

We may now estimate the partitioning cost of Step 4. We assume that only necessary comparisons are made as in Remark 2.2(b) (but it will be seen that up to s extraneous comparisons may be accomodated in our analysis; cf. Rem. 5.4(a)).

Lemma 3.4. *Let c denote the number of comparisons made at Step 4. Then*

$$\mathrm{P}[c \le \bar{c}] \ge 1 - e^{-2g^2/s} \quad \textit{and} \quad \mathrm{E}c \le \bar{c} + 2(n-s)e^{-2g^2/s} \quad \textit{with} \tag{3.3a}$$

$$\bar{c} := n + \min\{k, n-k\} - s + 2gn/s. \tag{3.3b}$$

Proof. Consider the event $\mathcal{A} := \{c \le \bar{c}\}$ and its complement $\mathcal{A}' := \{c > \bar{c}\}$. If $u = v$, the elements of $X \setminus S$ are compared to v (or u) only, so $c = n - s \le \bar{c}$; hence $\mathrm{P}[\mathcal{A}'] = \mathrm{P}[\mathcal{A}' \cap \{u < v\}]$, and we may assume $u < v$ below.

First, suppose $k < n/2$. Then each element x in $X \setminus S$ is compared to v first, and then to u only if $x < v$, so

$$c = n - s + |\{x \in X \setminus S : x < v\}|.$$

In particular, $c \le 2(n-s)$. Since $k < n/2$, $\bar{c} = n + k - s + 2gn/s$. If $v \le x^*_{k_r}$, then

$$\{x \in X \setminus S : x < v\} \subset \{x \in X : x \le v\} \setminus \{u, v\}$$

yields $|\{x \in X \setminus S : x < v\}| \le k_r - 2$, so $c \le n - s + k_r - 2$; since $k_r < k + 2gn/s + 1$ by (3.1), we get

$$c \le n + k - s + 2gn/s - 1 \le \bar{c}.$$

Thus $u < v \le x^*_{k_r}$ implies $\mathcal{A}$. Therefore, $\mathcal{A}' \cap \{u < v\}$ implies $\{x^*_{k_r} < v\} \cap \{u < v\}$, so

$$\mathrm{P}[\mathcal{A}' \cap \{u < v\}] \le \mathrm{P}[x^*_{k_r} < v] \le e^{-2g^2/s}$$

(Lem. 3.3(d)). Hence we have (3.3), since by Fact 3.2 (with $z := c$, $\zeta := 2(n-s)$),

$$\mathrm{E}c \le \bar{c} + 2(n-s)\mathrm{P}[c > \bar{c}] \le \bar{c} + 2(n-s)e^{-2g^2/s}.$$

Next, suppose $k \ge n/2$. Then each element x in $X \setminus S$ is compared to u first, and to v only if $u < x$, so

$$c = n - s + |\{x \in X \setminus S : u < x\}|.$$

If $x^*_{k_l} \le u$, then

$$\{x \in X \setminus S : u < x\} \subset \{x \in X : u \le x\} \setminus \{u, v\}$$

yields $|\{x \in X \setminus S : u < x\}| \le n - k_l - 1$; hence $k_l \ge k - 2gn/s$ (cf. (3.1)) gives

$$c \le n - s + (n-k) + 2gn/s - 1 \le \bar{c}.$$

Thus $\mathcal{A}' \cap \{u < v\}$ implies $\{u < x^*_{k_l}\} \cap \{u < v\}$, so $\mathrm{P}[\mathcal{A}' \cap \{u < v\}] \le \mathrm{P}[u < x^*_{k_l}] \le e^{-2g^2/s}$ (Lem. 3.3(b)), and we get (3.3) as before. □

3.5 Size of the selected set

The following result will imply that, for suitable choices of s and g, the set $\hat{X}$ selected at Step 6 will be "small enough" with high probability and in expectation; we let $\hat{X} := \emptyset$ and $\hat{n} := 0$ if Step 5 returns u or v, but we don't consider this case explicitly.

Lemma 3.5. $\mathrm{P}[\hat{n} < 4gn/s] \ge 1 - 4e^{-2g^2/s}$, *and* $\mathrm{E}\hat{n} \le 4gn/s + 4ne^{-2g^2/s}$.

Proof. The first bound yields the second one by Fact 3.2 (with $z := \hat{n} < n$). In each case below, we define an event $\mathcal{E}$ that implies the event $\mathcal{B} := \{\hat{n} < 4gn/s\}$.

First, consider the *middle* case of $gn/s < k \le n - gn/s$, where $i_u = \lceil ks/n - g \rceil$ and $i_v = \lceil ks/n + g \rceil$ (Lem. 3.3(e)). Denote the favorable event by

$$\mathcal{E} := \left\{ x^*_{k_l} \le u \le x^*_k \le v \le x^*_{k_r} \right\}.$$

By Lem. 3.3 and the Boole-Benferroni inequality, its complement $\mathcal{E}'$ has $\mathrm{P}[\mathcal{E}'] \le 4e^{-2g^2/s}$, so $\mathrm{P}[\mathcal{E}] \ge 1 - 4e^{-2g^2/s}$. By the rules of Steps 4–6, the bracketing property $u \le x^*_k \le v$ of $\mathcal{E}$ implies $\hat{X} = M$, whereas the bound $x^*_{k_l} \le u \le v \le x^*_{k_r}$ yields $\hat{n} \le k_r - k_l + 1 - 2$; since $k_r < k + 2gn/s + 1$ and $k_l \ge k - 2gn/s$ by (3.1), we get $\hat{n} < 4gn/s$. Hence $\mathcal{E} \subset \mathcal{B}$ and thus $\mathrm{P}[\mathcal{B}] \ge \mathrm{P}[\mathcal{E}]$.

Next, consider the *left* case of $k \le gn/s$, i.e., $i_u \ne \lceil ks/n - g \rceil$ (Lem. 3.3(e)). If $i_v \ne \lceil ks/n + g \rceil$, then $n < k + gn/s$ (Lem. 3.3(e)) gives $\hat{n} < n < k + gn/s \le 2gn/s$; take

Table 4.1: Sample size $f(n) := n^{2/3}\ln^{1/3} n$ and relative sample size $\phi(n) := f(n)/n$.

n	10^3	10^4	10^5	10^6	$5\cdot 10^6$	10^7	$5\cdot 10^7$	10^8
$f(n)$	190.449	972.953	4864.76	23995.0	72287.1	117248	353885	568986
$\phi(n)$	.190449	.097295	.048648	.023995	.014557	.011725	.007078	.005690

$\mathcal{E} := \{n < k + gn/s\}$, a certain event. For $i_v = \lceil ks/n + g\rceil$, let $\mathcal{E} := \{x^*_k \le v \le x^*_{k_r}\}$; again $\mathrm{P}[\mathcal{E}] \ge 1 - 2e^{-2g^2/s}$ by Lem. 3.3(c,d). Now, $x^*_k \le v$ implies $\hat{X} \subset L \cup M$, whereas $v \le x^*_{k_r}$ gives $\hat{n} \le k_r - 1 < k + 2gn/s \le 3gn/s$; therefore $\mathcal{E} \subset \mathcal{B}$.

Finally, consider the *right* case of $k > n - gn/s$, i.e., $i_v \ne \lceil ks/n + g\rceil$. If $i_u \ne \lceil ks/n - g\rceil$, the inequality $k \le gn/s$ gives $\hat{n} < n < 2gn/s$; take $\mathcal{E} := \{k \le gn/s\}$. For $i_u = \lceil ks/n - g\rceil$, the event $\mathcal{E} := \{x^*_{k_l} \le u \le x^*_k\}$ has $\mathrm{P}[\mathcal{E}] \ge 1 - 2e^{-2g^2/s}$ by Lem. 3.3(a,b). Now, $u \le x^*_k$ implies $\hat{X} \subset M \cup R$, whereas $x^*_{k_l} \le u$ yields $\hat{n} \le n - k_l$ with $k_l \ge k - 2gn/s$ and thus $\hat{n} < 3gn/s$. Hence $\mathcal{E} \subset \mathcal{B}$. □

The following stronger version of Lemma 3.5 is needed in §5.

Corollary 3.6. $\mathrm{P}[c \le \bar{c} \text{ and } \hat{n} < 4gn/s] \ge 1 - 4e^{-2g^2/s}$.

Proof. Check that $\mathcal{E}$ implies $\mathcal{A}$ in the proofs of Lems. 3.4 and 3.5; note that $n \le 2gn/s$ yields $c \le 2(n-s) \le \bar{c}$ (cf. (3.3b)) in the left and right subcases. □

The proof of Lemma 3.5 reveals that u plays a relatively minor rôle in the left case; similarly for v in the right case. This motivates the following modification.

Remark 3.7. Suppose Step 3 resets $i_u := i_v$ if $k \le gn/s$, or $i_v := i_u$ if $k > n - gn/s$, finding a single pivot $u = v$ in these cases. The preceding results remain valid.

4 Analysis of the recursive version

In this section we analyze the average performance of SELECT for various sample sizes.

4.1 Floyd-Rivest's samples

For positive constants α and β, consider choosing $s = s(n)$ and $g = g(n)$ as

$$s := \min\{\lceil \alpha f(n)\rceil, n-1\} \text{ and } g := (\beta s \ln n)^{1/2} \text{ with } f(n) := n^{2/3}\ln^{1/3} n. \tag{4.1}$$

This form of g gives a probability bound $e^{-2g^2/s} = n^{-2\beta}$ for Lems. 3.4–3.5. To get more feeling, suppose $\alpha = \beta = 1$ and $s = f(n)$. Let $\phi(n) := f(n)/n$. Then $s/n = g/s = \phi(n)$ and $\hat{n}/n$ is at most $4\phi(n)$ with high probability (at least $1 - 4/n^2$), i.e., $\phi(n)$ is a contraction factor; note that $\phi(n) \approx 2.4\%$ for $n = 10^6$ (cf. Tab. 4.1).

Theorem 4.1. *Let C_{nk} denote the expected number of comparisons made by* SELECT *for s and g chosen as in (4.1) with $\beta \geq 1/6$. There exists a positive constant γ such that*

$$C_{nk} \leq n + \min\{\, k, n-k \,\} + \gamma f(n) \quad \forall 1 \leq k \leq n. \tag{4.2}$$

Proof. The main idea of our inductive proof is simple: add the costs of Steps 3, 4, 7 and simplify to get (4.2). To this end, however, we need a few preliminary facts.

The function $\phi(t) := f(t)/t = (\ln t/t)^{1/3}$ decreases to 0 on $[e, \infty)$, whereas $f(t)$ grows to infinity on $[2, \infty)$. Let $\delta := 4(\beta/\alpha)^{1/2}$. Pick $\bar{n} \geq 3$ large enough so that $e - 1 \leq \alpha f(\bar{n}) \leq \bar{n} - 1$ and $e \leq \delta f(\bar{n})$. Let $\bar{\alpha} := \alpha + 1/f(\bar{n})$. Then, by (4.1) and the monotonicity of f and ϕ, we have for $n \geq \bar{n}$

$$s \leq \bar{\alpha} f(n) \quad \text{and} \quad f(s) \leq \bar{\alpha}\phi(\bar{\alpha} f(\bar{n}))f(n), \tag{4.3}$$

$$f(\delta f(n)) \leq \delta\phi(\delta f(\bar{n}))f(n). \tag{4.4}$$

Indeed, for instance, the first inequality of (4.3) yields $f(s) \leq f(\bar{\alpha} f(n))$, whereas

$$f(\bar{\alpha} f(n)) = \bar{\alpha}\phi(\bar{\alpha} f(n))f(n) \leq \bar{\alpha}\phi(\bar{\alpha} f(\bar{n}))f(n).$$

Also for $n \geq \bar{n}$, we have $s = \lceil \alpha f(n) \rceil = \alpha f(n) + \epsilon$ with $\epsilon \in [0, 1)$ in (4.1). Writing $s = \tilde{\alpha} f(n)$ with $\tilde{\alpha} := \alpha + \epsilon/f(n) \in [\alpha, \bar{\alpha})$, we deduce from (4.1) that

$$gn/s = (\beta/\tilde{\alpha})^{1/2} f(n) \leq (\beta/\alpha)^{1/2} f(n). \tag{4.5}$$

In particular, $4gn/s \leq \delta f(n)$, since $\delta := 4(\beta/\alpha)^{1/2}$. For $\beta \geq 1/6$, (4.1) implies

$$ne^{-2g^2/s} \leq n^{1-2\beta} = f(n) n^{1/3-2\beta} \ln^{-1/3} n. \tag{4.6}$$

Using the monotonicity of ϕ and f on $[e, \infty)$, increase $\bar{n}$ if necessary to get

$$2\bar{\alpha}\phi(\bar{\alpha} f(\bar{n})) + \delta\phi(\delta f(\bar{n})) + 4\phi(\bar{n})\bar{n}^{1/3-2\beta} \ln^{-1/3} \bar{n} \leq 0.95. \tag{4.7}$$

By Rem. 2.2(c), there is γ such that (4.2) holds for all $n \leq \bar{n}$; increasing γ if necessary, we have

$$2\bar{\alpha} + 2\delta + 8\bar{n}^{1/3-2\beta} \ln^{-1/3} \bar{n} \leq 0.05\gamma. \tag{4.8}$$

Let $n' \geq \bar{n}$. Assuming (4.2) holds for all $n \leq n'$, we will inductively prove that it holds for $n = n' + 1$.

The cost of Step 3 can be estimated as follows. We may first apply SELECT recursively to S to find $u = y^*_{i_u}$, and then extract $v = y^*_{i_v}$ from the elements $y^*_{i_u+1}, \ldots, y^*_s$ (assuming $i_u < i_v$; otherwise $v = u$). Since $s \leq n'$, the expected number of comparisons is

$$C_{s,i_u} + C_{s-i_u, i_v - i_u} \leq 1.5s + \gamma f(s) + 1.5(s - i_u) + \gamma f(s - i_u) \leq 3s - 1.5 + 2\gamma f(s). \tag{4.9}$$

The partitioning cost of Step 4 is estimated by (3.3) as

$$\mathrm{E}c \leq n + \min\{\, k, n-k \,\} - s + 2gn/s + 2ne^{-2g^2/s}. \tag{4.10}$$

The cost of finishing up at Step 7 is at most

$$C_{\hat{n}\hat{k}} \le 1.5\hat{n} + \gamma f(\hat{n}).$$

But by Lem. 3.5, $\mathrm{P}[\hat{n} \ge 4gn/s] \le 4e^{-2g^2/s}$, and $\hat{n} < n$, so (cf. Fact 3.2 with $z := 1.5\hat{n} + \gamma f(\hat{n})$)

$$\mathrm{E}\,[\,1.5\hat{n} + \gamma f(\hat{n})\,] \le 1.5 \cdot 4gn/s + \gamma f(4gn/s) + [\,1.5n + \gamma f(n)\,]\,4e^{-2g^2/s}.$$

Since $4gn/s \le \delta f(n)$, f is increasing, and $f(n) = \phi(n) \cdot n$ above, we get

$$\mathrm{E}C_{\hat{n}\hat{k}} \le 6gn/s + \gamma f(\delta f(n)) + [\,1.5 + \gamma\phi(n)\,]\,4ne^{-2g^2/s}. \tag{4.11}$$

Add the costs (4.9), (4.10) and (4.11) to get

$$\begin{aligned} C_{nk} &\le 3s - 1.5 + 2\gamma f(s) + n + \min\{\,k, n-k\,\} - s + 2gn/s + 2ne^{-2g^2/s} \\ &\quad + 6gn/s + \gamma f(\delta f(n)) + [\,1.5 + \gamma\phi(n)\,]\,4ne^{-2g^2/s} \\ &\le n + \min\{\,k, n-k\,\} + \left[\,2s + 8gn/s + 8ne^{-2g^2/s}\,\right] \qquad (4.12a) \\ &\quad + \gamma\left[\,2f(s) + f(\delta f(n)) + 4ne^{-2g^2/s}\phi(n)\,\right]. \qquad (4.12b) \end{aligned}$$

By (4.3)–(4.6), the bracketed term in (4.12a) is at most $0.05\gamma f(n)$ due to (4.8), and that in (4.12b) is at most $0.95f(n)$ from (4.7); thus (4.2) holds as required. □

We now indicate briefly how to adapt the preceding proof to several variations on (4.1); choices similar to (4.13) and (4.17) are used in [Meh00] and [FlR75a], respectively.

Remarks 4.2. (a) Theorem 4.1 holds for the following modification of (4.1):

$$s := \min\{\,\lceil \alpha f(n) \rceil, n-1\,\} \text{ and } g := (\beta s \ln \theta s)^{1/2} \text{ with } f(n) := n^{2/3}\ln^{1/3} n, \tag{4.13}$$

provided that $\beta \ge 1/4$, where $\theta > 0$. Indeed, the analogue of (4.5) (cf. (4.1), (4.13))

$$gn/s = (\beta/\tilde{\alpha})^{1/2} f(n)(\ln \theta s/\ln n)^{1/2} \le (\beta/\alpha)^{1/2} f(n)(\ln \theta s/\ln n)^{1/2} \tag{4.14}$$

works like (4.5) for large n (since $\lim_{n\to\infty} \frac{\ln \theta s}{\ln n} = 2/3$), whereas replacing (4.6) by

$$ne^{-2g^2/s} = n(\theta s)^{-2\beta} \le f(n)(\alpha\theta)^{-2\beta} n^{(1-4\beta)/3} \ln^{-(1+2\beta)/3} n, \tag{4.15}$$

we may replace $\bar{n}^{1/3-2\beta}$ by $(\alpha\theta)^{-2\beta}\bar{n}^{(1-4\beta)/3}$ in (4.7)–(4.8).

(b) Theorem 4.1 holds for the following modification of (4.1):

$$s := \min\{\,\lceil \alpha f(n) \rceil, n-1\,\} \text{ and } g := (\beta s \ln^{\epsilon_l} n)^{1/2} \text{ with } f(n) := n^{2/3}\ln^{\epsilon_l/3} n, \tag{4.16}$$

provided either $\epsilon_l = 1$ and $\beta \ge 1/6$, or $\epsilon_l > 1$. Indeed, since (4.16)=(4.1) for $\epsilon_l = 1$, suppose $\epsilon_l > 1$. Clearly, (4.3)–(4.5) hold with $\phi(t) := f(t)/t$. For an arbitrary $\beta > 0$, choosing $\tilde{\beta} \ge 1/6$, for n large enough we have $g^2/s = \beta \ln^{\epsilon_l} n \ge \tilde{\beta} \ln n$; hence, replacing 2β by $2\tilde{\beta}$ and $\ln^{-1/3}$ by $\ln^{-\epsilon_l/3}$ in (4.6)–(4.8), we may use the proof of Thm 4.1.

(c) Theorem 4.1 remains true if we use $\beta \ge 1/6$,

$$s := \min\left\{\left\lceil \alpha n^{2/3} \right\rceil, n-1\right\},\ g := (\beta s \ln n)^{1/2} \text{ and } f(n) := n^{2/3}\ln^{1/2} n. \tag{4.17}$$

Again (4.3)–(4.5) hold with $\phi(t) := f(t)/t$, and $\ln^{-1/2}$ replaces $\ln^{-1/3}$ in (4.6)–(4.8).

(d) None of these choices gives $f(n)$ better than that in (4.1) for the bound (4.2).

4.2 Reischuk's samples

For positive constants α and β, consider using

$$s := \min\{\lceil \alpha n^{\epsilon_s} \rceil, n-1\} \quad \text{and} \quad g := (\beta s n^{\epsilon})^{1/2} \quad \text{with} \tag{4.18a}$$

$$\eta := \max\{1 + (\epsilon - \epsilon_s)/2, \epsilon_s\} < 1 \quad \text{for some fixed } 0 < \epsilon < \epsilon_s. \tag{4.18b}$$

Theorem 4.3. *Let C_{nk} denote the expected number of comparisons made by* SELECT *for s and g chosen as in (4.18). There exists a positive constant γ_η such that for all $k \le n$*

$$C_{nk} \le n + \min\{k, n-k\} + \gamma_\eta f_\eta(n) \quad \textit{with} \quad f_\eta(n) := n^\eta. \tag{4.19}$$

Proof. We only show how to modify the proof of Theorem 4.1.

The function $f_\eta(t) := t^\eta$ grows to ∞ on $(0,\infty)$, whereas $\phi_\eta(t) := f_\eta(t)/t = t^{\eta-1}$ decreases to 0, so f_η and ϕ_η may replace f and ϕ in the proof of Thm 4.1. Indeed, picking $\bar{n} \ge 1$ such that $\alpha \bar{n}^{\epsilon_s} \le \bar{n} - 1$, for $n \ge \bar{n}$ we may use

$$s = \tilde{\alpha} n^{\epsilon_s} \le \bar{\alpha} f_\eta(n) \quad \text{with} \quad \alpha \le \tilde{\alpha} \le \bar{\alpha} := 1 + 1/\bar{n}^{\epsilon_s}$$

to get analogues of (4.3)–(4.4) and the following analogue of (4.5)

$$gn/s = (\beta/\tilde{\alpha})^{1/2} n^{1+(\epsilon-\epsilon_s)/2} \le (\beta/\alpha)^{1/2} f_\eta(n). \tag{4.20}$$

Since $g^2/s = \beta n^\epsilon$ by (4.18), and $te^{-2\beta t^\epsilon}/t^\eta$ decreases to 0 for $t \ge t_\eta := \left(\frac{1-\eta}{2\beta\epsilon}\right)^{1/\epsilon}$, we may replace (4.6) by

$$ne^{-2g^2/s} = ne^{-2\beta n^\epsilon} \le \bar{n}^{1-\eta} e^{-2\beta \bar{n}^\epsilon} f_\eta(n) \quad \forall n \ge \bar{n} \ge t_\eta. \tag{4.21}$$

Hence, with $\bar{n}^{1-\eta} e^{-2\beta\bar{n}^\epsilon}$ replacing $\bar{n}^{1/3-2\beta} \ln^{-1/3} \bar{n}$ in (4.7)–(4.8), the proof goes through. □

We now compare Floyd and Rivest's choice of (4.1) with Reischuk's choice of (4.18).

Remarks 4.4. (a) For a fixed $\epsilon \in (0,1)$, minimizing η in (4.18b) yields the *optimal* sample size parameter

$$\epsilon_s := (2+\epsilon)/3 \quad \text{with} \quad \eta = \epsilon_s > 2/3 \quad \text{and} \quad f_\eta(n) = n^{(2+\epsilon)/3}; \tag{4.22}$$

note that if $s = \alpha n^{\epsilon_s}$ in (4.18a), then $g = (\alpha\beta)^{1/2} n^{\epsilon_g}$ with $\epsilon_g := (1+2\epsilon)/3$. To compare the bounds (4.2) and (4.19) for this optimal choice, let $\Phi_\epsilon(t) := (t^\epsilon/\ln t)^{1/3}$, so that $\Phi_\epsilon(t) = f_\eta(t)/f(t) = \phi_\eta(t)/\phi(t)$. Since $\lim_{n\to\infty} \Phi_\epsilon(n) = \infty$, the choice (4.1) is asymptotically superior to (4.18). However, $\Phi_\epsilon(n)$ grows quite slowly, and $\Phi_\epsilon(n) < 1$ even for fairly large n when ϵ is small (cf. Tab. 4.2). On the other hand, for small ϵ and $\beta = 1$, the probability bound $e^{-2g^2/s} = e^{-2n^\epsilon}$ of (4.18) is weak relative to $e^{-2g^2/s} = n^{-2}$ ensured by (4.1).

(b) A tightly related variant of (4.18) consists in using

$$s := \min\{\lceil \alpha n^{\epsilon_s} \rceil, n-1\} \quad \text{and} \quad g := (\alpha\beta)^{1/2} n^{\epsilon_g}$$

Table 4.2: Relative sample sizes $\Phi_\epsilon(n)$ and probability bounds e^{-2n^ϵ}.

		$\Phi_\epsilon(n) := (t^\epsilon/\ln t)^{1/3}$				$\exp(-2n^\epsilon)$			
n		10^5	10^6	$5\cdot10^6$	10^7	10^5	10^6	$5\cdot10^6$	10^7
	1/4	1.16	1.32	1.45	1.52	$3.6\cdot10^{-16}$	$3.4\cdot10^{-28}$	$8.4\cdot10^{-42}$	$1.4\cdot10^{-49}$
ϵ	1/6	.840	.898	.946	.969	$1.2\cdot10^{-6}$	$2.1\cdot10^{-9}$	$4.4\cdot10^{-12}$	$1.8\cdot10^{-12}$
	1/9	.678	.695	.711	.719	$7.6\cdot10^{-4}$	$9.3\cdot10^{-5}$	$1.5\cdot10^{-5}$	$6.2\cdot10^{-6}$

with $0 < \epsilon_g < \epsilon_s$ such that

$$\epsilon := 2\epsilon_g - \epsilon_s > 0 \quad \text{and} \quad \eta := \max\{1 + \epsilon_g - \epsilon_s, \epsilon_s\} < 1.$$

Theorem 4.3 covers this choice. Indeed, the equality $1 + \epsilon_g - \epsilon_s = 1 + (\epsilon - \epsilon_s)/2$ shows that (4.18b) remains valid, and we have the following analogues of (4.20) and (4.21)

$$gn/s = (\alpha\beta)^{1/2} n^{1+(\epsilon-\epsilon_s)/2}/\tilde{\alpha} \le (\beta/\alpha)^{1/2} f_\eta(n), \tag{4.23}$$

$$ne^{-2g^2/s} \le \bar{n}^{1-\eta} e^{-2(\alpha\beta/\bar{\alpha})\bar{n}^\epsilon} f_\eta(n) \quad \forall n \ge \bar{n} \ge [(1-\eta)\bar{\alpha}/(2\alpha\beta\epsilon)]^{1/\epsilon}, \tag{4.24}$$

so compatible modifications of (4.7)–(4.8) suffice for the rest of the proof. Note that $\eta \ge (2+\epsilon)/3$ by (a); for the choice $\epsilon_s = \frac{1}{2}$, $\epsilon_g = \frac{7}{16}$ of [Rei85], $\epsilon = \frac{3}{8}$ and $\eta = \frac{15}{16}$.

4.3 Handling small subfiles

Since the sampling efficiency decreases when X shrinks, consider the following modification. For a fixed cut-off parameter $n_{\text{cut}} \ge 1$, let sSelect(X, k) be a "small-select" routine that finds the kth smallest element of X in at most $C_{\text{cut}} < \infty$ comparisons when $|X| \le n_{\text{cut}}$ (even bubble sort will do). Then SELECT is modified to start with the following

Step 0 (*Small file case*). If $n := |X| \le n_{\text{cut}}$, return sSelect$(X, k)$.

Our preceding results remain valid for this modification. In fact it suffices if C_{cut} bounds the *expected* number of comparisons of sSelect(X, k) for $n \le n_{\text{cut}}$. For instance, (4.2) holds for $n \le n_{\text{cut}}$ and $\gamma \ge C_{\text{cut}}$, and by induction as in Rem. 2.2(c) we have $C_{nk} < \infty$ for all n, which suffices for the proof of Thm 4.1.

Another advantage is that even small n_{cut} (1000 say) limits nicely the stack space for recursion. Specifically, the tail recursion of Step 7 is easily eliminated (set $X := \hat{X}$, $k := \hat{k}$ and go to Step 0), and the calls of Step 3 deal with subsets whose sizes quickly reach n_{cut}. For example, for the choice of (4.1) with $\alpha = 1$ and $n_{\text{cut}} = 600$, at most four recursive levels occur for $n \le 2^{31} \approx 2.15 \cdot 10^9$.

5 Analysis of nonrecursive versions

In this section we prove that nonrecursive versions of SELECT, in which Steps 3 and 7 employ other selection or sorting algorithms with suitable worst-case performance, require

at most $n + \min\{k, n-k\} + o(n)$ comparisons with high probability. In this setting our analysis is simpler than that of §4, because the costs of Step 3 are deterministic, whereas Corollary 3.6 yields high probability bounds for the outcomes of Steps 4 and 7.

First, consider a nonrecursive version of SELECT in which Steps 3 and 7, instead of SELECT, employ a linear-time routine (e.g., PICK [BFP+72]) that finds the ith smallest of m elements in at most $\gamma_P m$ comparisons for some constant $\gamma_P > 2$.

Theorem 5.1. *Let c_{nk} denote the number of comparisons made by the nonrecursive version of* SELECT *for a given choice of s and g. Suppose $s < n - 1$.*

(a) *For the choice of* (4.1) *with $f(n) := n^{2/3} \ln^{1/3} n$, we have*

$$\mathrm{P}\left[\, c_{nk} \le n + \min\{\, k, n-k \,\} + \hat{\gamma}_P f(n) \,\right] \ge 1 - 4n^{-2\beta} \quad \textit{with} \tag{5.1a}$$

$$\hat{\gamma}_P := (4\gamma_P + 2)(\beta/\alpha)^{1/2} + (2\gamma_P - 1)\,[\alpha + 1/f(n)]\,, \tag{5.1b}$$

also with $f(n)$ in (5.1b) *replaced by $f(3) > 2$ (since $n \ge 3$). Moreover, if $\beta \ge 1/6$, then*

$$\mathrm{E}c_{nk} \le n + \min\{\, k, n-k \,\} + (\,\hat{\gamma}_P + 4\gamma_P + 2\,)\, f(n). \tag{5.2}$$

(b) *For the choice of* (4.13), *if $\theta s \le n$, then* (5.1a) *holds with $n^{-2\beta}$ replaced by $(\alpha\theta)^{-2\beta} n^{-4\beta/3} \ln^{-2\beta/3} n$. Moreover, if $\beta \ge 1/4$, then* (5.2) *holds with $4\gamma_P + 2$ replaced by $(4\gamma_P + 2)(\alpha\theta)^{-2\beta}$.*

(c) *For the choice of* (4.18), (5.1) *holds with $f(n)$ replaced by $f_\eta(n) := n^\eta$ and $n^{-2\beta}$ by $e^{-2\beta n^\epsilon}$. Moreover, if $n^{1-\eta} e^{-2\beta n^\epsilon} \le 1$, then* (5.2) *holds with f replaced by f_η.*

Proof. We start with some general estimates. Since Step 3 takes at most $2\gamma_P s$ comparisons, Step 4 makes c comparisons, and Step 7 takes at most $\gamma_P \hat{n}$ comparisons, the total cost satisfies

$$c_{nk} \le 2\gamma_P s + c + \gamma_P \hat{n}.$$

By Cor. 3.6, the event $\mathcal{C} := \{c \le \bar{c}, \hat{n} < 4gn/s\}$ has probability $\mathrm{P}[\mathcal{C}] \ge 1 - 4e^{-2g^2/s}$. If the event $\mathcal{C}$ occurs, bounding c by $\bar{c}$ (cf. (3.3b)) and $\hat{n}$ by $4gn/s$ above gives

$$\begin{aligned} c_{nk} &\le n + \min\{\, k, n-k \,\} - s + 2gn/s + 2\gamma_P s + \gamma_P \lfloor 4gn/s \rfloor \\ &\le n + \min\{\, k, n-k \,\} + (\,4\gamma_P + 2\,)\, gn/s + (\,2\gamma_P - 1\,)\, s. \end{aligned} \tag{5.3}$$

Similarly, the average total cost satisfies

$$\mathrm{E}c_{nk} \le 2\gamma_P s + \mathrm{E}c + \gamma_P \mathrm{E}\hat{n},$$

so bounding $\mathrm{E}c$ via Lemma 3.4 and $\mathrm{E}\hat{n}$ via Lemma 3.5 yields

$$\mathrm{E}c_{nk} \le n + \min\{\, k, n-k \,\} + (\,4\gamma_P + 2\,)\, gn/s + (\,2\gamma_P - 1\,)\, s + (\,4\gamma_P + 2\,)\, ne^{-2g^2/s}. \tag{5.4}$$

We now spell out consequences of (5.3)–(5.4) for the three cases of our theorem.

(a) Since $e^{-2g^2/s} = n^{-2\beta}$, $s = \lceil \alpha f(n) \rceil \le \bar{\alpha} f(n)$ from $s < n - 1$ and (4.3), and gn/s is bounded by (4.5), (5.3) implies (5.1). Then (5.2) follows from (4.6) and (5.4).

(b) Proceed as for (a), invoking (4.14)–(4.15) instead of (4.5) and (4.6).

(c) Argue as for (a), using the proof of Thm 4.3, in particular (4.20)–(4.21). □

Corollary 5.2. *The nonrecursive version of* SELECT *requires* $n + \min\{k, n-k\} + o(n)$ *comparisons with probability at least* $1 - 4n^{-2\beta}$ *for the choice of* (4.1), *at least* $1 - 4(\alpha\theta)^{-2\beta}n^{-4\beta/3}$ *for the choice of* (4.13), *and at least* $1 - 4e^{-2\beta n^{\epsilon}}$ *for the choice of* (4.18).

The following remarks sketch extensions of our preceding results to several modifications of SELECT, including the use of sorting algorithms at Steps 3 and 7.

Remarks 5.3. (a) Suppose Steps 3 and 7 simply sort S and $\hat{X}$ by any algorithm that takes at most $\gamma_S(s \ln s + \hat{n} \ln \hat{n})$ comparisons for a constant γ_S. This cost is at most $(s + \hat{n})\gamma_S \ln n$, because $s, \hat{n} < n$, so we may replace $2\gamma_P$ by $\gamma_S \ln n$ and $4\gamma_P$ by $4\gamma_S \ln n$ in (5.3)–(5.4), and hence in (5.1)–(5.2). For the choice of (4.1), this yields

$$\mathrm{P}\left[c_{nk} \le n + \min\{k, n-k\} + \hat{\gamma}_S f(n) \ln n\right] \ge 1 - 4n^{-2\beta} \quad \text{with} \tag{5.5a}$$

$$\hat{\gamma}_S := (4\gamma_S + 2\ln^{-1} n)(\beta/\alpha)^{1/2} + (\gamma_S - \ln^{-1} n)\left[\alpha + 1/f(n)\right], \tag{5.5b}$$

$$\mathrm{E}c_{nk} \le n + \min\{k, n-k\} + \left(\hat{\gamma}_S + 4\gamma_S + 2\ln^{-1} n\right) f(n) \ln n, \tag{5.6}$$

where $\ln^{-1} n$ may be replaced by $\ln^{-1} 3$, and (5.6) still needs $\beta \ge 1/6$; for the choices (4.13) and (4.18), we may modify (5.5)–(5.6) as in Thm 5.1(b,c). Corollary 5.2 remains valid.

(b) The bound (5.2) holds if Steps 3 and 7 employ a routine (e.g., FIND [Hoa61], [AHU74, §3.7]) for which the expected number of comparisons to find the ith smallest of m elements is at most $\gamma_P m$ (then $\mathrm{E}c_{nk} \le 2\gamma_P s + \mathrm{E}c + \gamma_P \mathrm{E}\hat{n}$ is bounded as before).

(c) Suppose Step 6 returns to Step 1 if $\hat{n} \ge 4gn/s$. By Cor. 3.6, such loops are finite with probability 1, and don't occur with high probability, for n large enough.

(d) Our results improve upon [GeS03, Thm 1], which only gives an estimate like (5.1a), but with $4n^{-2\beta}$ replaced by $O(n^{1-2\beta/3})$, a much weaker bound. Further, the approach of [GeS03] is restricted to distinct elements.

We now comment briefly on the possible use of sampling with replacement.

Remarks 5.4. (a) Suppose Step 2 of SELECT employs sampling with replacement. Since the tail bound (3.2) remains valid for the binomial distribution [Chv79, Hoe63], Lemma 3.3 is not affected. However, when Step 4 no longer skips comparisons with the elements of S, $-s$ in (3.3) and (4.10) is replaced by 0 (cf. the proof of Lem. 3.4), $2s$ in (4.12a) by $3s$ and $2\bar{\alpha}$ in (4.8) by $3\bar{\alpha}$. Similarly, adding s to the right sides of (5.3)–(5.4) boils down to omitting -1 in (5.1b) and $-\ln^{-1} n$ in (5.5b). Hence the preceding results remain valid.

(b) Of course, sampling with replacement needs additional storage for S. This is inconvenient for the recursive version, but tolerable for the nonrecursive ones because the sample sizes are relatively small (hence also (3.3) with $-s$ omitted is not too bad).

(c) Our results improve upon [MoR95, Thm 3.5], corresponding to (4.18) with $\epsilon = 1/4$ and $\beta = 1$, where the probability bound $1 - O(n^{-1/4})$ is weaker than our $1 - 4e^{-2n^{1/4}}$, sampling is done with replacement and the elements are distinct.

(d) Our results subsume [Meh00, Thm 2], which gives an estimate like (5.2) for the choice (4.13) with $\beta = 1$, using quickselect (cf. Rem. 5.3(b)) and sampling with replacement in the case of distinct elements.

6 Ternary and quintary partitioning

In this section we discuss ways of implementing SELECT when the input set is given as an array $x[1\colon n]$. We need the following notation to describe its operations in more detail.

Each stage works with a segment $x[l\colon r]$ of the input array $x[1\colon n]$, where $1 \le l \le r \le n$ are such that $x_i < x_l$ for $i = 1\colon l-1$, $x_r < x_i$ for $i = r+1\colon n$, and the kth smallest element of $x[1\colon n]$ is the $(k-l+1)$th smallest element of $x[l\colon r]$. The task of SELECT is *extended*: given $x[l\colon r]$ and $l \le k \le r$, SELECT(x, l, r, k, k_-, k_+) permutes $x[l\colon r]$ and finds $l \le k_- \le k \le k_+ \le r$ such that $x_i < x_k$ for all $l \le i < k_-$, $x_i = x_k$ for all $k_- \le i \le k_+$, $x_i > x_k$ for all $k_+ < i \le r$. The initial call is SELECT$(x, 1, n, k, k_-, k_+)$.

A vector swap denoted by $x[a\colon b] \leftrightarrow x[b{+}1\colon c]$ means that the first $d := \min(b{+}1{-}a, c{-}b)$ elements of array $x[a\colon c]$ are exchanged with its last d elements in arbitrary order if $d > 0$; e.g., we may exchange $x_{a+i} \leftrightarrow x_{c-i}$ for $0 \le i < d$, or $x_{a+i} \leftrightarrow x_{c-d+1+i}$ for $0 \le i < d$.

6.1 Ternary partitions

For a given pivot $v := x_k$ from the array $x[l\colon r]$, the following *ternary* scheme partitions the array into three blocks, with $x_m < v$ for $l \le m < a$, $x_m = v$ for $a \le m \le d$, $x_m > v$ for $d < m \le r$. The basic idea is to work with the five inner parts of the array

$x<v$	$x=v$	$x<v$	?	$x>v$	$x=v$	$x>v$
l	$\bar{l}$	p	$i\ \ j$	q	$\bar{r}$	r

(6.1)

until the middle part is empty or just contains an element equal to the pivot

$x=v$	$x<v$	$x=v$	$x>v$	$x=v$
$\bar{l}$	$p\ \ j$		i	$q\ \ \bar{r}$

(6.2)

(i.e., $j = i-1$ or $j = i-2$), then swap the ends into the middle for the final arrangement

$x<v$	$x=v$	$x>v$
$\bar{l}$	$a\ \ d$	$\bar{r}$

(6.3)

A1. [Initialize.] Set $v := x_k$ and exchange $x_l \leftrightarrow x_k$. Set $i := \bar{l} := l$, $p := l+1$, $q := r-1$ and $j := \bar{r} := r$. If $v < x_r$, set $\bar{r} := q$. If $v > x_r$, exchange $x_l \leftrightarrow x_r$ and set $\bar{l} := p$.

A2. [Increase i until $x_i \ge v$.] Increase i by 1; then if $x_i < v$, repeat this step.

A3. [Decrease j until $x_j \le v$.] Decrease j by 1; then if $x_j > v$, repeat this step.

A4. [Exchange.] (Here $x_j \le v \le x_i$.) If $i < j$, exchange $x_i \leftrightarrow x_j$; then if $x_i = v$, exchange $x_i \leftrightarrow x_p$ and increase p by 1; if $x_j = v$, exchange $x_j \leftrightarrow x_q$ and decrease q by 1; return to A2. If $i = j$ (so that $x_i = x_j = v$), increase i by 1 and decrease j by 1.

A5. [Cleanup.] Set $a := \bar{l} + j - p + 1$ and $d := \bar{r} - q + i - 1$. Exchange $x[\bar{l}\colon p-1] \leftrightarrow x[p\colon j]$ and $x[i\colon q] \leftrightarrow x[q+1\colon \bar{r}]$.

Step A1 ensures that $x_l \le v \le x_r$, so steps A2 and A3 don't need to test whether $i \le j$; thus their loops can run faster than those in the schemes of [BeM93, Prog. 6] and [Knu97, Ex. 5.2.2–41] (which do need such tests, since, e.g., there may be no element $x_i > v$).

6.2 Preparing for quintary partitions

At Step 1, $r-l+1$ replaces n in finding s and g. At Step 2, it is convenient to place the sample in the initial part of $x[l\!:\!r]$ by exchanging $x_i \leftrightarrow x_{i+\mathrm{rand}(r-i)}$ for $l \le i \le r_s := l+s-1$, where $\mathrm{rand}(r-i)$ denotes a random integer, uniformly distributed between 0 and $r-i$.

Step 3 uses $k_u := \max\{\lceil l-1+is/m-g\rceil, l\}$ and $k_v := \min\{\lceil l-1+is/m+g\rceil, r_s\}$ with $i := k-l+1$ and $m := r-l+1$ for the recursive calls. If $\textsc{Select}(x, l, r_s, k_u, k_u^-, k_u^+)$ returns $k_u^+ \ge k_v$, we have $v := u := x_{k_u}$, so we only set $k_v^- := k_v$, $k_v^+ := k_u^+$ and reset $k_u^+ := k_v - 1$. Otherwise the second call $\textsc{Select}(x, k_u^+ + 1, r_s, k_v, k_v^-, k_v^+)$ produces $v := x_{k_v}$.

After u and v have been found, our array looks as follows

$x<u$	$x=u$	$u<x<v$	$x=v$	$x>v$	?
l	k_u^- k_u^+		k_v^- k_v^+	r_s	r

(6.4)

Setting $\bar{l} := k_u^-$, $\bar{p} := k_u^+ + 1$, $\bar{r} := r - r_s + k_v^+$, $\bar{q} := \bar{r} - k_v^+ + k_v^- - 1$, we exchange $x[k_v^+ + 1\!:\!r_s] \leftrightarrow x[r_s + 1\!:\!r]$ and then $x[k_v^-\!:\!k_v^+] \leftrightarrow x[k_v^+ + 1\!:\!\bar{r}]$ to get the arrangement

$x<u$	$x=u$	$u<x<v$	?	$x=v$	$x>v$
l	$\bar{l}$	$\bar{p}$	k_v^- $\bar{q}$	$\bar{r}$	r

(6.5)

The third part above is missing precisely when $u = v$; in this case (6.5) reduces to (6.1) with initial $p := \bar{p}$, $q := \bar{q}$, $i := p-1$ and $j := q+1$. Hence the case of $u = v$ is handled via the ternary partitioning scheme of §6.1, with step A1 omitted.

6.3 Quintary partitions

For the case of $k < \lfloor (r+l)/2 \rfloor$ and $u < v$, Step 4 may use the following *quintary* scheme to partition $x[l\!:\!r]$ into five blocks, with $x_m < u$ for $l \le m < a$, $x_m = u$ for $a \le m < b$, $u < x_m < v$ for $b \le m \le c$, $x_m = v$ for $c < m \le d$, $x_m > v$ for $d < m \le r$. The basic idea is to work with the six-part array stemming from (6.5)

$x=u$	$u<x<v$	$x<u$	?	$x>v$	$x=v$
$\bar{l}$	$\bar{p}$	p	i j	q	$\bar{r}$

(6.6)

until i and j cross

$x=u$	$u<x<v$	$x<u$	$x>v$	$x=v$
$\bar{l}$	$\bar{p}$	p j	i q	$\bar{r}$

; (6.7)

we may then swap the second part with the third one to bring it into the middle

$x=u$	$x<u$	$u<x<v$	$x>v$	$x=v$
$\bar{l}$	$\bar{p}$	b c	i q	$\bar{r}$

, (6.8)

and finally swap the extreme parts with their neighbors to get the desired arrangement

$x<u$	$x=u$	$u<x<v$	$x=v$	$x>v$
$\bar{l}$	a	b c	d	$\bar{r}$

. (6.9)

B1. [Initialize.] Set $p := k_v^-$, $q := \bar{q}$, $i := p - 1$ and $j := q + 1$.

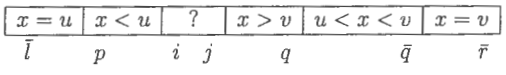
B2. [Increase i until $x_i \geq v$.] Increase i by 1. If $x_i \geq v$, go to B3. If $x_i < u$, repeat this step. (At this point, $u \leq x_i < v$.) If $x_i > u$, exchange $x_i \leftrightarrow x_p$; otherwise exchange $x_i \leftrightarrow x_p$ and $x_p \leftrightarrow x_{\bar{p}}$ and increase $\bar{p}$ by 1. Increase p by 1 and repeat this step.

B3. [Decrease j until $x_j < v$.] Decrease j by 1. If $x_j > v$, repeat this step. If $x_j = v$, exchange $x_j \leftrightarrow x_q$, decrease q by 1 and repeat this step.

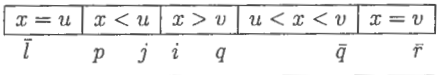
B4. [Exchange.] If $i \geq j$, go to B5. Exchange $x_i \leftrightarrow x_j$. If $x_i > u$, exchange $x_i \leftrightarrow x_p$ and increase p by 1; otherwise if $x_i = u$, exchange $x_i \leftrightarrow x_p$ and $x_p \leftrightarrow x_{\bar{p}}$ and increase $\bar{p}$ and p by 1. If $x_j = v$, exchange $x_j \leftrightarrow x_q$ and decrease q by 1. Return to B2.

B5. [Cleanup.] Set $a := \bar{l} + i - p$, $b := a + \bar{p} - \bar{l}$, $d := \bar{r} - q + j$ and $c := d - \bar{r} + q$. Swap $x[\bar{p}\colon p-1] \leftrightarrow x[p\colon j]$, $x[\bar{l}\colon \bar{p}-1] \leftrightarrow x[\bar{p}\colon b-1]$, and finally $x[i\colon q] \leftrightarrow x[q+1\colon \bar{r}]$.

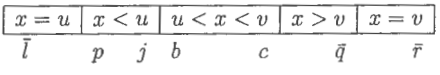
For the case of $k \geq \lfloor (r+l)/2 \rfloor$ and $u < v$, Step 4 may use the following quintary scheme, which is a symmetric version of the preceding one obtained by replacing (6.6)–(6.8) with



, (6.10)



, (6.11)



. (6.12)

C1. [Initialize.] Set $p := \bar{p}$, $q := \bar{q} - k_v^- + k_u^+ + 1$, $i := p - 1$ and $j := q + 1$, and swap $x[\bar{p}\colon k_v^- - 1] \leftrightarrow x[k_v^-\colon \bar{q}]$.

C2. [Increase i until $x_i > u$.] Increase i by 1. If $x_i < u$, repeat this step. If $x_i = u$, exchange $x_i \leftrightarrow x_p$, increase p by 1 and repeat this step.

C3. [Decrease j until $x_j \leq u$.] Decrease j by 1. If $x_j \leq u$, go to C4. If $x_j > v$, repeat this step. (At this point, $u < x_j \leq v$.) If $x_j < v$, exchange $x_j \leftrightarrow x_q$; otherwise exchange $x_j \leftrightarrow x_q$ and $x_q \leftrightarrow x_{\bar{q}}$ and decrease $\bar{q}$ by 1. Decrease q by 1 and repeat this step.

C4. [Exchange.] If $i \geq j$, go to C5. Exchange $x_i \leftrightarrow x_j$. If $x_i = u$, exchange $x_i \leftrightarrow x_p$ and increase p by 1. If $x_j < v$, exchange $x_j \leftrightarrow x_q$ and decrease q by 1; otherwise if $x_j = v$, exchange $x_j \leftrightarrow x_q$ and $x_q \leftrightarrow x_{\bar{q}}$ and decrease $\bar{q}$ and q by 1. Return to C2.

C5. [Cleanup.] Set $a := \bar{l} + i - p$, $b := a + p - \bar{l}$, $d := \bar{r} - q + j$ and $c := d - \bar{r} + \bar{q}$. Swap $x[\bar{l}\colon p-1] \leftrightarrow x[p\colon j]$, $x[i\colon q] \leftrightarrow x[q+1\colon \bar{q}]$ and finally $x[c+1\colon \bar{q}] \leftrightarrow x[\bar{q}+1\colon \bar{r}]$.

To make (6.3) and (6.9) compatible, the ternary scheme may set $b := d+1$, $c := a-1$. After partitioning l and r are updated by setting $l := b$ if $a \leq k$, then $l := d+1$ if $c < k$; $r := c$ if $k \leq d$, then $r := a-1$ if $k < b$. If $l \geq r$, SELECT may return $k_- := k_+ := k$ if $l = r$, $k_- := r+1$ and $k_+ := l-1$ if $l > r$. Otherwise, instead of calling SELECT recursively, Step 6 may jump back to Step 1, or Step 0 if sSelect is used (cf. §4.3).

A simple version of sSelect is obtained if Steps 2 and 3 choose $u := v := x_k$ when $r - l + 1 \leq n_{\text{cut}}$ (this choice of [FlR75a] works well in practice, but more sophisticated pivots could be tried); then the ternary partitioning code can be used by sSelect as well.

7 Experimental results

7.1 Implemented algorithms

An implementation of SELECT was programmed in Fortran 77 and run on a notebook PC (Pentium M 755 2 GHz, 1.5 GB RAM) under MS Windows XP. The input set X was specified as a double precision array. For efficiency, the tail recursion was removed and small arrays with $n \le n_{\text{cut}}$ were handled by Steps 2 and 3 choosing $u := v := x_k$; the resulting version of sSelect (cf. §§4.3 and 6.3) typically required less than $3.5n$ comparisons. The choice of (4.1) was employed, with the parameters $\alpha = 0.5$, $\beta = 0.25$ and $n_{\text{cut}} = 600$ as proposed in [FlR75a]; future work should test other sample sizes and parameters.

For comparisons we employed the implementation of QUICKSELECT from [Kiw03a, §6.1], with median-of-3 pivots and the binary partitioning scheme of [Kiw03a, §§2.1 and 4]. Apparently our implementation represents the state-of-the-art; see [Kiw03a, §6.3] for comparisons with other partitioning schemes, and note that the results of [Kiw03b, §7.3] suggest that the RISELECT algorithm of [Val00] tends to be less efficient.

7.2 Testing examples

We used minor modifications of the input sequences of [Val00], defined as follows:

random A random permutation of the integers 1 through n.

onezero A random permutation of $\lceil n/2 \rceil$ ones and $\lfloor n/2 \rfloor$ zeros.

sorted The integers 1 through n in increasing order.

rotated A sorted sequence rotated left once; i.e., $(2, 3, \ldots, n, 1)$.

organpipe The integers $(1, 2, \ldots, n/2, n/2, \ldots, 2, 1)$.

m3killer Musser's "median-of-3 killer" sequence with $n = 4j$ and $k = n/2$:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & \ldots & k-2 & k-1 & k & k+1 & \ldots & 2k-2 & 2k-1 & 2k \\ 1 & k+1 & 3 & k+3 & \ldots & 2k-3 & k-1 & 2 & 4 & \ldots & 2k-2 & 2k-1 & 2k \end{pmatrix}.$$

twofaced Obtained by randomly permuting the elements of an m3killer sequence in positions $4\lfloor \log_2 n \rfloor$ through $n/2 - 1$ and $n/2 + 4\lfloor \log_2 n \rfloor - 1$ through $n - 2$.

For each input sequence, its (lower) median element was selected for $k := \lceil n/2 \rceil$.

These input sequences were designed to test the performance of selection algorithms under a range of conditions. In particular, the onezero sequences represent inputs containing many duplicates [Sed77]. The rotated and organpipe sequences are difficult for many implementations of quickselect. The m3killer and twofaced sequences are hard for implementations with median-of-3 pivots (their original versions [Mus97] were modified to become difficult when the middle element comes from position k instead of $k + 1$).

Table 7.1: Performance of SELECT on randomly generated inputs.

Sequence	Size	Time [msec]			Comparisons [n]			γ_{avg}	L_{avg}	P_{avg}	N_{avg}	p_{avg}	s_{avg}
	n	avg	max	min	avg	max	min		[n]	[$\ln n$]	[$\ln n$]		[%n]
random	50K	0	0	0	1.81	1.85	1.77	5.23	1.22	0.46	1.01	7.62	4.11
	100K	0	0	0	1.72	1.76	1.65	4.50	1.15	0.45	0.99	8.05	3.20
	500K	8	10	0	1.62	1.63	1.60	4.14	1.08	0.59	1.27	7.59	1.86
	1M	18	20	10	1.59	1.60	1.57	3.93	1.06	0.64	1.35	8.18	1.47
	2M	36	40	30	1.57	1.58	1.56	3.73	1.04	0.76	1.59	7.67	1.16
	4M	70	81	60	1.56	1.56	1.55	3.61	1.03	0.94	1.94	7.21	0.91
	8M	137	141	130	1.54	1.55	1.54	3.45	1.03	0.98	1.99	7.45	0.72
	16M	247	251	240	1.53	1.54	1.53	3.44	1.02	0.99	2.02	7.55	0.57
onezero	50K	0	0	0	1.51	1.52	1.50	0.24	1.02	0.28	0.27	1.17	3.41
	100K	2	10	0	1.51	1.51	1.50	0.23	1.01	0.26	0.25	1.14	2.72
	500K	9	11	0	1.51	1.51	1.51	0.26	1.01	0.23	0.23	1.17	1.61
	1M	18	20	10	1.51	1.51	1.51	0.26	1.01	0.22	0.22	1.20	1.29
	2M	35	41	30	1.51	1.51	1.50	0.26	1.01	0.28	0.27	1.14	1.03
	4M	72	80	70	1.50	1.50	1.50	0.26	1.00	0.33	0.26	1.16	0.83
	8M	142	151	140	1.50	1.50	1.50	0.26	1.00	0.38	0.25	1.11	0.66
	16M	270	281	260	1.50	1.50	1.50	0.26	1.00	0.36	0.24	1.11	0.53
twofaced	50K	1	10	0	1.80	1.85	1.74	4.99	1.21	0.46	1.01	7.53	4.11
	100K	0	0	0	1.73	1.76	1.69	4.67	1.16	0.43	0.96	8.23	3.20
	500K	9	10	0	1.62	1.63	1.61	4.07	1.08	0.61	1.30	7.85	1.87
	1M	18	20	10	1.59	1.60	1.58	3.82	1.06	0.67	1.40	7.86	1.47
	2M	37	41	30	1.57	1.58	1.56	3.66	1.04	0.75	1.58	7.98	1.16
	4M	71	80	70	1.56	1.56	1.55	3.60	1.03	0.95	1.96	7.36	0.92
	8M	136	141	130	1.54	1.55	1.54	3.48	1.03	0.96	1.98	7.48	0.72
	16M	251	251	241	1.53	1.54	1.53	3.38	1.02	1.00	2.06	7.74	0.57

7.3 Computational results

We varied the input size n from 50,000 to 16,000,000. For the random, onezero and twofaced sequences, for each input size, 20 instances were randomly generated; for the deterministic sequences, 20 runs were made to measure the solution time.

The performance of SELECT on randomly generated inputs is summarized in Table 7.1, where the average, maximum and minimum solution times are in milliseconds, and the comparison counts are in multiples of n; e.g., column six gives C_{avg}/n, where C_{avg} is the average number of comparisons made over all instances. Thus $\gamma_{\text{avg}} := (C_{\text{avg}} - 1.5n)/f(n)$ estimates the constant γ in the bound (4.2); moreover, we have $C_{\text{avg}} \approx 1.5L_{\text{avg}}$, where L_{avg} is the average sum of sizes of partitioned arrays. Further, P_{avg} is the average number of SELECT partitions, whereas N_{avg} is the average number of calls to sSelect and p_{avg} is the average number of sSelect partitions per call; both P_{avg} and N_{avg} grow slowly with $\ln n$. Finally, s_{avg} is the average sum of sample sizes; $s_{\text{avg}}/f(n)$ drops from 0.68 for $n = 50\text{K}$ to 0.56 for $n = 16\text{M}$ on the random and twofaced inputs, and from 0.57 to 0.52 on the onezero inputs, whereas the initial $s/f(n) \approx \alpha = 0.5$. The average solution times grow linearly with n (except for small inputs whose solution times couldn't be measured accurately), and the differences between maximum and minimum times are fairly small (and also partly

Table 7.2: Performance of SELECT on deterministic inputs.

Sequence	Size	Time [msec]			Comparisons [n]			γ_{avg}	L_{avg}	P_{avg}	N_{avg}	p_{avg}	s_{avg}
	n	avg	max	min	avg	max	min		[n]	[$\ln n$]	[$\ln n$]		[%n]
sorted	50K	1	20	0	1.80	1.88	1.71	4.92	1.21	0.44	0.98	7.80	4.08
	100K	3	30	0	1.73	1.76	1.71	4.76	1.16	0.44	0.97	7.83	3.21
	500K	6	11	0	1.62	1.63	1.61	4.09	1.08	0.60	1.27	7.91	1.86
	1M	11	20	10	1.60	1.61	1.58	4.02	1.06	0.63	1.34	8.05	1.46
	2M	20	20	10	1.57	1.58	1.57	3.75	1.04	0.77	1.60	7.46	1.16
	4M	35	40	30	1.56	1.56	1.55	3.59	1.03	0.95	1.95	7.45	0.91
	8M	58	61	50	1.54	1.55	1.53	3.50	1.03	0.99	2.03	7.55	0.72
	16M	105	111	100	1.53	1.54	1.53	3.37	1.02	1.00	2.04	7.65	0.57
rotated	50K	4	30	0	1.80	1.91	1.71	4.99	1.21	0.44	0.98	7.90	4.08
	100K	2	30	0	1.74	1.76	1.70	4.83	1.16	0.44	0.96	7.91	3.21
	500K	6	10	0	1.62	1.63	1.61	4.09	1.08	0.60	1.28	8.01	1.86
	1M	11	20	10	1.60	1.60	1.59	4.03	1.06	0.64	1.35	8.14	1.47
	2M	18	21	10	1.57	1.58	1.56	3.74	1.04	0.76	1.59	7.54	1.16
	4M	30	31	30	1.56	1.56	1.55	3.59	1.03	0.94	1.93	7.26	0.91
	8M	58	61	50	1.54	1.55	1.53	3.47	1.03	0.99	2.02	7.43	0.72
	16M	104	111	100	1.53	1.54	1.53	3.35	1.02	1.00	2.04	7.61	0.57
organpipe	50K	1	10	0	1.80	1.84	1.70	5.04	1.21	0.46	1.01	7.59	4.11
	100K	5	30	0	1.74	1.76	1.71	4.88	1.16	0.45	0.98	8.03	3.22
	500K	5	10	0	1.62	1.63	1.60	4.04	1.08	0.62	1.32	7.75	1.87
	1M	14	20	10	1.59	1.60	1.57	3.87	1.06	0.66	1.39	7.72	1.47
	2M	27	30	20	1.57	1.58	1.56	3.69	1.04	0.74	1.56	7.66	1.16
	4M	50	51	50	1.56	1.56	1.55	3.57	1.03	0.97	1.99	7.22	0.92
	8M	97	101	90	1.55	1.55	1.54	3.58	1.03	0.97	1.99	7.38	0.72
	16M	169	171	160	1.53	1.54	1.53	3.39	1.02	0.99	2.02	7.68	0.57
m3killer	50K	3	30	0	1.84	2.27	1.76	5.61	1.23	0.47	1.04	7.69	4.21
	100K	3	10	0	1.74	1.77	1.70	4.83	1.16	0.44	0.97	7.79	3.21
	500K	8	20	0	1.63	1.64	1.61	4.24	1.08	0.58	1.23	7.79	1.86
	1M	15	20	10	1.59	1.60	1.58	3.92	1.06	0.67	1.40	7.87	1.47
	2M	31	40	30	1.57	1.58	1.56	3.67	1.04	0.75	1.57	7.85	1.16
	4M	60	61	60	1.56	1.56	1.55	3.64	1.03	0.96	1.96	7.33	0.92
	8M	117	120	110	1.54	1.55	1.54	3.51	1.03	0.96	1.97	7.39	0.72
	16M	219	221	210	1.53	1.54	1.53	3.37	1.02	0.97	1.98	7.64	0.57

due to the operating system). Except for the smallest inputs, the maximum and minimum numbers of comparisons are quite close, and C_{avg} nicely approaches the theoretical lower bound of $1.5n$; this is reflected in the values of γ_{avg}. Note that the results for the random and twofaced sequences are almost identical, whereas the onezero inputs only highlight the efficiency of our partitioning.

Table 7.2 exhibits similar features of SELECT on the deterministic inputs. The results for the sorted and rotated sequences are almost the same, whereas the solution times on the organpipe and m3killer sequences are between those for the sorted and random sequences.

The performance of QUICKSELECT on the same inputs is described in Tables 7.3 and 7.4. On the random sequences, the expected value of C_{avg} is $2.75n+o(n)$ [KMP97]. Twenty random instances of each size yield fairly accurate estimates, since the values of C_{avg} in

Table 7.3: Performance of QUICKSELECT on randomly generated inputs.

Sequence	Size	Time [msec]			Comparisons [n]			L_{avg}	P_{avg}
	n	avg	max	min	avg	max	min	[n]	[$\ln n$]
random	50K	1	10	0	2.60	4.07	1.56	2.60	1.44
	100K	2	10	0	2.69	3.98	1.63	2.69	1.51
	500K	11	20	0	2.61	4.04	1.78	2.61	1.51
	1M	32	41	20	2.78	4.04	1.77	2.78	1.56
	2M	67	100	50	2.70	3.92	1.91	2.70	1.51
	4M	135	180	90	2.56	3.46	1.70	2.56	1.59
	8M	283	411	200	2.59	3.96	1.78	2.59	1.64
	16M	568	751	431	2.57	3.46	1.93	2.57	1.57
onezero	50K	1	10	0	2.72	2.85	2.67	2.72	1.77
	100K	1	10	0	2.74	2.88	2.68	2.74	1.79
	500K	12	20	10	2.70	2.73	2.68	2.70	1.82
	1M	30	40	20	2.75	2.88	2.68	2.75	1.84
	2M	69	80	60	2.71	2.85	2.68	2.71	1.84
	4M	148	171	140	2.73	3.21	2.68	2.73	1.84
	8M	307	330	300	2.73	2.92	2.68	2.73	1.86
	16M	621	631	610	2.70	2.79	2.68	2.70	1.87
twofaced	50K	3	31	0	2.65	4.43	1.72	2.65	1.50
	100K	0	0	0	2.62	3.71	1.75	2.62	1.53
	500K	12	20	10	2.63	4.18	1.79	2.63	1.51
	1M	29	41	20	2.66	4.41	1.76	2.66	1.56
	2M	67	90	40	2.67	3.71	1.73	2.67	1.57
	4M	144	190	100	2.77	3.83	2.02	2.77	1.57
	8M	300	481	190	2.86	4.83	1.68	2.86	1.56
	16M	572	921	370	2.60	4.62	1.66	2.60	1.68

Table 7.3 are within 7% of $2.75n$; Table 7.5 shows what happens when 1000 instances are used for each size. The results for the onezero sequences confirm that binary partitioning may handle equal keys quite efficiently [Sed77]. The results for the remaining inputs are quite good, since some versions of quickselect may behave poorly on these inputs [Val00].

As always, limited testing doesn't warrant firm conclusions, but a comparison of SELECT and QUICKSELECT is in order, especially for the random sequences, which are most frequently used in theory and practice for evaluating sorting and selection algorithms. On the random inputs, the ratio of the expected numbers of comparisons for QUICKSELECT and SELECT is asymptotically $2.75/1.5 \approx 1.83$, whereas the ratio of their computing times approaches 2.3 in Figure 7.1. Note that SELECT isn't just asymptotically faster; in fact QUICKSELECT is about 80% slower even on middle-sized inputs. Similar slow-downs occur on the onezero and twofaced sequences. The slow-downs are less pronounced on the organpipe and m3killer inputs, but they are still significant even for the "easiest" sorted and rotated inputs. Note that, relative to QUICKSELECT, the solution times and comparison counts of SELECT are much more stable across all the inputs. This feature may be important in applications.

Acknowledgment. I would like to thank Olgierd Hryniewicz, Roger Koenker, Ronald L. Rivest and John D. Valois for useful discussions. Two anonymous referees helped a lot

Table 7.4: Performance of QUICKSELECT on deterministic inputs.

Sequence	Size	Time [msec]			Comparisons [n]			L_{avg}	P_{avg}
	n	avg	max	min	avg	max	min	[n]	[$\ln n$]
sorted	50K	0	0	0	2.94	3.68	2.27	2.94	1.55
	100K	0	0	0	2.89	4.63	2.23	2.89	1.62
	500K	8	10	0	2.88	4.56	1.96	2.88	1.63
	1M	13	20	0	2.96	4.44	1.82	2.96	1.59
	2M	29	41	20	3.02	4.44	2.06	3.02	1.54
	4M	44	70	30	2.76	4.10	1.99	2.76	1.56
	8M	88	120	60	2.80	3.62	1.89	2.80	1.65
	16M	175	241	120	2.75	3.74	1.87	2.75	1.63
rotated	50K	0	0	0	2.82	3.95	1.87	2.82	1.57
	100K	9	30	0	2.77	3.79	1.84	2.77	1.55
	500K	8	20	0	2.80	4.39	1.74	2.80	1.68
	1M	13	20	10	2.87	4.68	1.92	2.87	1.62
	2M	23	31	20	2.55	3.44	1.75	2.55	1.56
	4M	45	70	20	2.72	4.22	1.61	2.72	1.57
	8M	92	161	60	2.85	5.16	1.89	2.85	1.59
	16M	177	251	110	2.78	3.97	1.65	2.78	1.57
organpipe	50K	8	30	0	2.60	3.71	1.73	2.60	1.52
	100K	2	10	0	2.71	3.62	2.03	2.71	1.60
	500K	9	10	0	2.76	4.77	1.72	2.76	1.53
	1M	21	30	10	2.74	4.77	2.00	2.74	1.49
	2M	46	60	30	2.83	4.18	1.85	2.83	1.62
	4M	92	110	70	2.74	3.73	2.17	2.74	1.54
	8M	181	250	120	2.64	4.10	1.77	2.64	1.53
	16M	372	470	270	2.61	3.49	1.94	2.61	1.62
m3killer	50K	1	10	0	2.60	3.47	1.88	2.60	1.60
	100K	2	10	0	2.89	3.96	1.85	2.89	1.50
	500K	10	20	0	2.83	4.90	1.83	2.83	1.59
	1M	24	31	10	2.79	3.85	1.90	2.79	1.55
	2M	54	70	40	3.06	4.47	1.93	3.06	1.65
	4M	102	130	60	2.81	4.06	1.63	2.81	1.60
	8M	193	261	150	2.75	4.43	1.87	2.75	1.63
	16M	409	480	320	2.87	3.94	1.87	2.87	1.58

in improving our presentation.

References

[AHU74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[BeM93] J. L. Bentley and M. D. McIlroy, *Engineering a sort function*, Software–Practice and Experience **23** (1993) 1249–1265.

[BFP+72] M. R. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan, *Time bounds for selection*, J. Comput. System Sci. **7** (1972) 448–461.

[Bro76] T. Brown, *Remark on Algorithm 489*, ACM Trans. Math. Software **3** (1976) 301–304.

[Chv79] V. Chvátal, *The tail of the hypergeometric distribution*, Discrete Math. **25** (1979) 285–287.

Table 7.5: Numbers of comparisons per element made on small random inputs.

Size		1000	2500	5000	7500	10000	12500	15000	17500	20000	25000
SELECT	avg	2.80	2.52	2.25	2.15	2.09	2.05	1.99	1.97	1.94	1.90
	max	4.16	3.34	2.79	2.55	2.86	2.33	2.25	2.47	2.14	2.29
	min	2.05	1.99	1.91	1.86	1.79	1.83	1.77	1.77	1.79	1.76
QUICKSELECT	avg	2.75	2.72	2.75	2.71	2.73	2.72	2.73	2.76	2.75	2.72
	max	5.28	5.01	6.00	5.35	5.61	4.66	5.38	5.65	5.42	4.80
	min	1.00	1.59	1.58	1.61	1.60	1.59	1.60	1.55	1.59	1.59

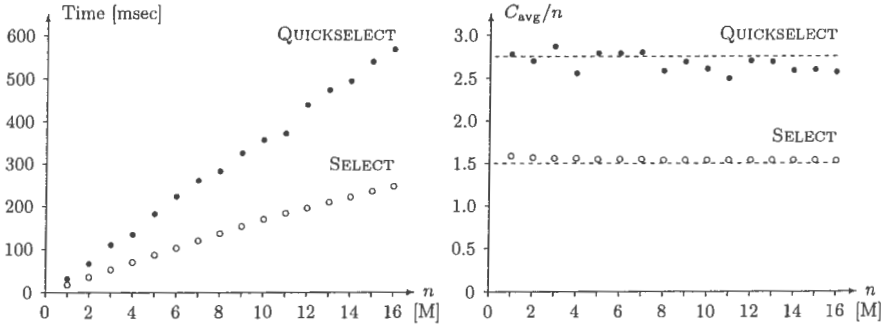


Figure 7.1: Average running times and comparisons per element on random inputs.

[CuM89] W. Cunto and J. I. Munro, *Average case selection*, J. of the ACM **36** (1989) 270–279.

[DHUZ01] D. Dor, J. Håstad, S. Ulfberg and U. Zwick, *On lower bounds for selecting the median*, SIAM J. Discrete Math. **14** (2001) 299–311.

[DoZ99] D. Dor and U. Zwick, *Selecting the median*, SIAM J. Comput. **28** (1999) 1722–1758.

[DoZ01] ———, *Median selection requires* $(2+\epsilon)N$ *comparisons*, SIAM J. Discrete Math. **14** (2001) 312–325.

[FlR73] R. W. Floyd and R. L. Rivest, *Bounds on the expected time for median computations*, in Courant Computer Science Symposium, R. Rustin, ed., vol. 9, Algorithmic Press, New York, NJ, 1973, pp. 69–76.

[FlR75a] ———, *The algorithm SELECT—for finding the ith smallest of* n *elements (Algorithm 489)*, Comm. ACM **18** (1975) 173.

[FlR75b] ———, *Expected time bounds for selection*, Comm. ACM **18** (1975) 165–172.

[GeS96] A. V. Gerbessiotis and C. J. Siniolakis, *Concurrent heaps on the BSP model*, Tech. Report PRG-TR-14-96, Oxford University Computing Lab., Oxford, UK, 1996.

[GeS03] ———, *Randomized selection in* $n + C + o(n)$ *comparisons*, Information Proc. Letters **88** (2003) 95–100.

[Grü99] R. Grübel, *On the median-of-k version of Hoare's selection algorithm*, Theor. Inform. Appl. **33** (1999) 177–192.

[Hoa61] C. A. R. Hoare, *Algorithm 65:* FIND, Comm. ACM **4** (1961) 321–322.

[Hoe63] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc. **58** (1963) 13–30.

[JoK69] N. L. Johnson and S. Kotz, *Distributions in Statistics: Discrete Distributions*, Houghton Mifflin, Boston, 1969.

[Kiw02] K. C. Kiwiel, *Randomized selection revisited*, Tech. report, Systems Research Institute, Warsaw, 2002. Available at the URL http://arxiv.org/abs/cs.DS/0204033.

[Kiw03a] ———, *Partitioning schemes for quicksort and quickselect*, Tech. report, Systems Research Institute, Warsaw, 2003. Available at the URL http://arxiv.org/abs/cs.DS/0312054.

[Kiw03b] ———, *Randomized selection with quintary partitions*, Tech. report, Systems Research Institute, Warsaw, 2003. Available at the URL http://arxiv.org/abs/cs.DS/0312055.

[Kiw04a] ———, *Improved randomized selection*, Tech. report, Systems Research Institute, Warsaw, 2004. Available at the URL http://arxiv.org/abs/cs.DS/0402005.

[Kiw04b] ———, *Randomized selection with tripartitiong*, Tech. report, Systems Research Institute, Warsaw, 2004. Available at the URL http://arxiv.org/abs/cs.DS/0401003.

[KMP97] P. Kirschenhofer, C. Martínez and H. Prodinger, *Analysis of Hoare's* FIND *algorithm with median-of-three partition*, Random Stuctures and Algorithms **10** (1997) 143–156.

[Knu97] D. E. Knuth, *The Art of Computer Programming. Volume I: Fundamental Algorithms*, third ed., Addison-Wesley, Reading, MA, 1997.

[Knu98] ———, *The Art of Computer Programming. Volume III: Sorting and Searching*, second ed., Addison-Wesley, Reading, MA, 1998.

[Kor78] V. S. Koroliuk, ed., *Handbook on Probability Theory and Mathematical Statistics*, Naukova Dumka, Kiev, 1978 (Russian).

[MaR01] C. Martínez and S. Roura, *Optimal sampling strategies in quicksort and quickselect*, SIAM J. Comput. **31** (2001) 683–705.

[Meh00] K. Mehlhorn, *Foundations of Data Structures and Algorithms: Selection*, Lecture notes, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2000. Available at the URL http://www.mpi-sb.mpg.de/~mehlhorn/Informatik5.html.

[MoR95] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, England, 1995.

[Mus97] D. R. Musser, *Introspective sorting and selection algorithms*, Software–Practice and Experience **27** (1997) 983–993.

[PRKT83] J. T. Postmus, A. H. G. Rinnooy Kan and G. T. Timmer, *An efficient dynamic selection method*, Comm. ACM **26** (1983) 878–881.

[Raj01] S. Rajasekaran, *Selection algorithms for parallel disk systems*, J. Parallel Distributed Comp. **61** (2001) 536–544.

[Rei85] R. Reischuk, *Probabilistic parallel algorithms for sorting and selection*, SIAM J. Comput. **14** (1985) 396–409.

[Rob55] H. Robbins, *A remark on Stirling's formula*, Amer. Math. Monthly **62** (1955) 26–29.

[Sed77] R. Sedgewick, *Quicksort with equal keys*, SIAM J. Comput. **6** (1977) 240–287.

[Sib99] J. F. Sibeyn, *External selection*, in STACS 99, Proc. of 16th Annual Symposium on Theoretical Aspects of Computer Science, C. Meinel and S. Tison, eds., Lecture Notes in Computer Science 1563, Springer, Berlin, 1999, pp. 291–301.

[SPP76] A. Schönhage, M. Paterson and N. Pippenger, *Finding the median*, J. Comput. System Sci. **13** (1976) 184–199.

[Val00] J. D. Valois, *Introspective sorting and selection revisited*, Software–Practice and Experience **30** (2000) 617–638.