Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 347 (2005) 214–238
Theoretical Computer Science
www.elsevier.com/locate/tcs

On Floyd and Rivest's SELECT algorithm

Krzysztof C. Kiwiel*

Systems Research Institute, Newelska 6, 01-447 Warsaw, Poland

Received 23 December 2003; received in revised form 1 August 2004; accepted 17 June 2005

Communicated by W. Szpankowski

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.

Keywords: 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 $1 \leqslant k \leqslant n$, 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 $x_j \leqslant x$. 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., [6–8] and [21, Section 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,

* Tel.: +48 22 8364414; fax: +48 22 8372772.
E-mail address: kiwiel@ibspan.waw.pl.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.06.032

selection needs at least $(2+\varepsilon)n$ comparisons [8], whereas the pioneering algorithm of [3] makes at most $5.43n$, its first improvement [29] needs $3n+o(n)$, and the most recent improvement in [7] 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 $k \leqslant \lceil n/2 \rceil$, at least $n+k-2$ comparisons are necessary [5,21, Exercise 5.3.3–25], whereas the best upper bound is $n+k+O(n^{1/2}\ln^{1/2} n)$ [21, Eq. (5.3.3.16)]. Yet this bound holds for a hardly implementable theoretical scheme [21, Exercise 5.3.3–24], whereas a similar frequently cited bound for the algorithm SELECT of [10] does not have a full proof, as noted in [21, Exercise 5.3.3–24] and [27]. Significantly worse bounds hold for the classical algorithm FIND of [13], also known as quickselect, which partitions X by using the median of a random sample of size $s \geqslant 1$. In particular, for $k=\lceil n/2 \rceil$, the upper bound is $3.39n+o(n)$ for $s=1$ [21, Exercise 5.2.2–32] and $2.75n+o(n)$ for $s=3$ [12,15], whereas for finding an element of random rank, the average cost is $3n+o(n)$ for $s=1$, $2.5n+o(n)$ for $s=3$ [15], and $2n+o(n)$ when $s \to \infty$, $s/n \to 0$ as $n \to \infty$ [23]. In practice FIND is most popular, because the algorithms of [3,29] are much slower on the average [26,31]. For the general case of nondistinct elements, little is known in theory about these algorithms, but again FIND performs well in practice [16,31].

Our aim is to rekindle theoretical and practical interest in the algorithm SELECT of [10, Section 2.1] (the versions of [10, Section 2.3] and [9] are addressed in [19,18]). 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 [10, Section 2.2] which is based on an unproven assumption [27, Section 5]. Similar upper bounds are established for versions that choose sample sizes as in [9,24,28] and [25, Section 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 [10, Section 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 [11, Theorem 1], [24, Theorem 2] and [25, Theorem 3.5].

Since theoretical bounds alone need not 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 [30], the partitioning efficiency is crucial. In contrast with the observation of [10, 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 do not sample randomly, assuming that the input file is in random order, whereas others [31] invoke random sampling only when slow progress occurs.) Yet our computational experience shows that sampling does not 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 [9] on several inputs of [31]). Most importantly, SELECT beats quite sophisticated implementations of FIND [16,17,31] in both comparison counts and computing times even for examples with relatively low comparison costs. To save space, only selected results are reported in Section 7.3 and [16,17], 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. Section 7.3). A theoretical explanation of these features will be undertaken elsewhere. For now, our experience supports the claim of [10, Section 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 Section 2, and its basic features are analyzed in Section 3. The average performance of SELECT is studied in Section 4. High probability bounds for nonrecursive versions are derived in Section 5. Partitioning schemes are discussed in Section 6. Finally, our computational results are reported in Section 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 $u \leqslant x_k^* \leqslant v$ 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 $1 \leqslant k \leqslant n := |X|$)
Step 1 (*Initiation*). If $n = 1$, return x_1. Choose the sample size $s \leqslant n - 1$ 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 $k \geqslant n/2$, the order of the comparisons is reversed.
Step 5 (*Stopping test*). If $|L| < k \leqslant |L \cup U|$, return u; else if $|L \cup U \cup M| < k \leqslant n - |R|$, return v.
Step 6 (*Reduction*). If $k \leqslant |L|$, 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 [10, Section 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. Section 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 Section 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 $x_1^* \leqslant \cdots \leqslant x_n^*$ and $y_1^* \leqslant \cdots \leqslant y_s^*$ 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 $\mathrm{e}^{-2g^2/s}$ (bounded as the tail of the hypergeometric distribution; cf. Fact 3.1 below). Hence, by the Boole–Benferroni inequality, the *favorable* event $x_{k_l}^* \leqslant u \leqslant x_k^* \leqslant v \leqslant x_{k_r}^*$ has probability at least $1 - 4\mathrm{e}^{-2g^2/s}$. Our bound for Step 4 stems from the fact that for $k < n/2$ and $v \leqslant x_{k_r}^*$, at most $k_r - 2$ elements $x < v$ are compared to u, whereas for $k \geqslant n/2$ and $x_{k_l}^* \leqslant u$, 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 $k \leqslant gn/s$ or $k > n - gn/s$ (when either u or v becomes redundant; cf. Remark 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 $n\mathrm{e}^{-2g^2/s}$ 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 [14] and rederived shortly in [4].

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*

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

We shall also need a simple version of the (left) Chebyshev inequality [22, Section 2.4.2].

Fact 3.2. *Let z be a nonnegative random variable such that $P[z \leqslant \zeta] = 1$ for some constant ζ. Then $Ez \leqslant t + \zeta P[z > t]$ 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 $y_1^* \leqslant \cdots \leqslant y_s^*$.

Lemma 3.3. (a) $P[x_k^* < u] \leqslant \mathrm{e}^{-2g^2/s}$ *if* $i_u = \lceil ks/n - g\rceil$.

(b) $P[u < x_{k_l}^*] \leqslant \mathrm{e}^{-2g^2/s}$.

(c) $P[v < x_k^*] \leqslant \mathrm{e}^{-2g^2/s}$ *if* $i_v = \lceil ks/n + g\rceil$.

(d) $P[x^*_{k_r} < v] \leqslant e^{-2g^2/s}$.
(e) $i_u \neq \lceil ks/n - g \rceil$ *iff* $k \leqslant gn/s$; $i_v \neq \lceil ks/n + g \rceil$ *iff* $n < k + gn/s$.

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

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

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

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

Hence $P[x^*_k < u] \leqslant P[r' \geqslant ps + g] \leqslant e^{-2g^2/s}$ by (3.2).
(b) If $y^*_{i_u} < x^*_{k_l}$, at least i_u sample elements y_i satisfy

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

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

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

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 $x_j \leqslant x^*_r$, $ps = rs/n$ and $r' \geqslant i_v$. But

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

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

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

Thus $x^*_{k_r} < v$ implies $r' \geqslant ps + g$; hence $P[x^*_{k_r} < v] \leqslant P[r' \geqslant ps + g] \leqslant e^{-2g^2/s}$ by (3.2).
(e) Follows immediately from the properties of $\lceil \cdot \rceil$ [20, Section 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. Remark 5.4(a)).

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

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

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

Proof. Consider the event $\mathcal{A} := \{c \leqslant \bar{c}\}$ 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 $c = n - s \leqslant \bar{c}$; hence $P[\mathcal{A}'] = P[\mathcal{A}' \cap \{u < v\}]$, 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, $c \leqslant 2(n-s)$. Since $k < n/2$, $\bar{c} = n + k - s + 2gn/s$. If $v \leqslant x^*_{k_r}$, then

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

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

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

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

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

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

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

Next, suppose $k \geqslant n/2$. 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 $x^*_{k_l} \leqslant u$, then

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

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

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

Thus $\mathcal{A}' \cap \{u < v\}$ implies $\{u < x^*_{k_l}\} \cap \{u < v\}$, so $P[\mathcal{A}' \cap \{u < v\}] \leqslant P[u < x^*_{k_l}] \leqslant \mathrm{e}^{-2g^2/s}$ (Lemma 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 do not consider this case explicitly.

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

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 $gn/s < k \leqslant n - gn/s$, where $i_u = \lceil ks/n - g \rceil$ and $i_v = \lceil ks/n + g \rceil$ (Lemma 3.3(e)). Denote the favorable event by

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

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

Next, consider the *left* case of $k \leqslant gn/s$, i.e., $i_u \neq \lceil ks/n - g \rceil$ (Lemma 3.3(e)). If $i_v \neq \lceil ks/n + g \rceil$, then $n < k + gn/s$ (Lemma 3.3(e)) gives $\hat{n} < n < k + gn/s \leqslant 2gn/s$; take $\mathcal{E} := \{n < k + gn/s\}$, a certain event. For $i_v = \lceil ks/n + g \rceil$, let $\mathcal{E} := \{x^*_k \leqslant v \leqslant x^*_{k_r}\}$; again $P[\mathcal{E}] \geqslant 1 - 2e^{-2g^2/s}$ by Lemma 3.3(c,d). Now, $x^*_k \leqslant v$ implies $\hat{X} \subset L \cup M$, whereas $v \leqslant x^*_{k_r}$ gives $\hat{n} \leqslant k_r - 1 < k + 2gn/s \leqslant 3gn/s$; therefore $\mathcal{E} \subset \mathcal{B}$.

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

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

Corollary 3.6. *$P\left[c \leqslant \bar{c}$ and $\hat{n} < 4gn/s\right] \geqslant 1 - 4e^{-2g^2/s}$.*

Proof. Check that $\mathcal{E}$ implies $\mathcal{A}$ in the proofs of Lemmas 3.4 and 3.5; note that $n \leqslant 2gn/s$ yields $c \leqslant 2(n - s) \leqslant \bar{c}$ (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 $k \leqslant gn/s$, or $i_v := i_u$ if $k > n - gn/s$, finding a single pivot $u = v$ in these cases. The preceding results remain valid.

Table 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	5×10^6	10^7	5×10^7	10^8
$f(n)$	190.449	972.953	4864.76	23995.0	72287.1	117248	353885	568986
$\phi(n)$	0.190449	0.097295	0.048648	0.023995	0.014557	0.011725	0.007078	0.005690

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

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

This form of g gives a probability bound $\mathrm{e}^{-2g^2/s} = n^{-2\beta}$ for Lemmas 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. Table 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 $\beta\geqslant 1/6$. There exists a positive constant γ such that*

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

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 $\bar{n}\geqslant 3$ large enough so that $e-1\leqslant \alpha f(\bar{n})\leqslant \bar{n}-1$ and $e\leqslant \delta f(\bar{n})$. Let $\bar{\alpha} := \alpha + 1/f(\bar{n})$. Then, by (4.1) and the monotonicity of f and ϕ, we have for $n\geqslant\bar{n}$

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

$$f(\delta f(n))\leqslant \delta\phi(\delta f(\bar{n}))\, f(n). \quad (4.4)$$

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

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

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

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

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

$$n\mathrm{e}^{-2g^2/s}\leqslant n^{1-2\beta} = f(n) n^{1/3-2\beta}\ln^{-1/3} n. \quad (4.6)$$

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

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

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

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

Let $n'\geqslant\bar{n}$. Assuming (4.2) holds for all $n\leqslant n'$, 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 $s\leqslant n'$, the expected number of comparisons is

$$\begin{aligned} C_{s,i_u}+C_{s-i_u,i_v-i_u} &\leqslant 1.5s+\gamma f(s)+1.5(s-i_u)+\gamma f(s-i_u)\\ &\leqslant 3s-1.5+2\gamma f(s). \end{aligned} \tag{4.9}$$

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

$$Ec\leqslant n+\min\{k,n-k\}-s+2gn/s+2n\mathrm{e}^{-2g^2/s}. \tag{4.10}$$

The cost of finishing up at Step 7 is at most

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

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

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

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

$$EC_{\hat{n}\hat{k}}\leqslant 6gn/s+\gamma f(\delta f(n))+\left[1.5+\gamma\phi(n)\right]4n\mathrm{e}^{-2g^2/s}. \tag{4.11}$$

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

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

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 [24] and [9], respectively.

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

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

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

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

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

$$n\mathrm{e}^{-2g^2/s} = n(\theta s)^{-2\beta} \leqslant f(n)(\alpha\theta)^{-2\beta} n^{(1-4\beta)/3} \ln^{-(1+2\beta)/3} n, \quad (4.15)$$

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):

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

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

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

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

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

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

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

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 $k \leqslant n$*

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

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 Theorem 4.1. Indeed, picking $\bar{n} \geqslant 1$ such that $\alpha\bar{n}^{\varepsilon_s} \leqslant \bar{n} - 1$, for $n \geqslant \bar{n}$ we may use

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

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

$$gn/s = (\beta/\tilde{\alpha})^{1/2} n^{1+(\varepsilon-\varepsilon_s)/2} \leqslant (\beta/\alpha)^{1/2} f_\eta(n). \quad (4.20)$$

Table 2
Relative sample sizes $\Phi_\varepsilon(n)$ and probability bounds $\mathrm{e}^{-2n^\varepsilon}$

n		$\Phi_\varepsilon(n) := (t^\varepsilon/\ln t)^{1/3}$				$\exp(-2n^\varepsilon)$			
		10^5	10^6	5×10^6	10^7	10^5	10^6	5×10^6	10^7
ε	1/4	1.16	1.32	1.45	1.52	3.6×10^{-16}	3.4×10^{-28}	8.4×10^{-42}	1.4×10^{-49}
	1/6	0.840	0.898	0.946	0.969	1.2×10^{-6}	2.1×10^{-9}	4.4×10^{-12}	1.8×10^{-12}
	1/9	0.678	0.695	0.711	0.719	7.6×10^{-4}	9.3×10^{-5}	1.5×10^{-5}	6.2×10^{-6}

Since $g^2/s = \beta n^\varepsilon$ by (4.18), and $t\mathrm{e}^{-2\beta t^\varepsilon}/t^\eta$ decreases to 0 for $t \geqslant t_\eta := ((1-\eta)/2\beta\varepsilon)^{1/\varepsilon}$, we may replace (4.6) by

$$n\mathrm{e}^{-2g^2/s} = n\mathrm{e}^{-2\beta n^\varepsilon} \leqslant \bar{n}^{1-\eta}\mathrm{e}^{-2\beta\bar{n}^\varepsilon} f_\eta(n) \quad \forall n \geqslant \bar{n} \geqslant t_\eta. \tag{4.21}$$

Hence, with $\bar{n}^{1-\eta}\mathrm{e}^{-2\beta\bar{n}^\varepsilon}$ 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 $\varepsilon \in (0, 1)$, minimizing η in (4.18b) yields the *optimal* sample size parameter

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

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

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

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

with $0 < \varepsilon_g < \varepsilon_s$ such that

$$\varepsilon := 2\varepsilon_g - \varepsilon_s > 0 \quad \text{and} \quad \eta := \max\{1 + \varepsilon_g - \varepsilon_s, \varepsilon_s\} < 1.$$

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

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

$$n\mathrm{e}^{-2g^2/s} \leqslant \bar{n}^{1-\eta}\mathrm{e}^{-2(\alpha\beta/\tilde{\alpha})\bar{n}^\varepsilon} f_\eta(n) \quad \forall n \geqslant \bar{n} \geqslant [(1-\eta)\tilde{\alpha}/(2\alpha\beta\varepsilon)]^{1/\varepsilon}, \tag{4.24}$$

so compatible modifications of (4.7)–(4.8) suffice for the rest of the proof. Note that $\eta \geqslant (2+\varepsilon)/3$ by (a); for the choice $\varepsilon_s = 1/2$, $\varepsilon_g = 7/16$ of [28], $\varepsilon = 3/8$ and $\eta = 15/16$.

4.3. Handling small subfiles

Since the sampling efficiency decreases when X shrinks, consider the following modification. For a fixed cut-off parameter $n_{\text{cut}} \geqslant 1$, 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 $|X| \leqslant n_{\text{cut}}$ (even bubble sort will do). Then SELECT is modified to start with the following:

Step 0 (*Small file case*): If $n := |X| \leqslant n_{\text{cut}}$, 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 $n \leqslant n_{\text{cut}}$. For instance, (4.2) holds for $n \leqslant n_{\text{cut}}$ and $\gamma \geqslant C_{\text{cut}}$, and by induction as in Remark 2.2(c) we have $C_{nk} < \infty$ for all n, which suffices for the proof of Theorem 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 $n \leqslant 2^{31} \approx 2.15 \times 10^9$.

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 Section 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 [3]) 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*

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

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

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

$$Ec_{nk} \leqslant n + \min\{k, n-k\} + (\hat{\gamma}_P + 4\gamma_P + 2) f(n). \tag{5.2}$$

(b) *For the choice of* (4.13), *if $\theta s \leqslant n$, 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 $\beta \geqslant 1/4$, 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 $\mathrm{e}^{-2\beta n^\varepsilon}$. Moreover, if $n^{1-\eta}\mathrm{e}^{-2\beta n^\varepsilon} \leqslant 1$, 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

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

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

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

Similarly, the average total cost satisfies

$$Ec_{nk} \leqslant 2\gamma_P s + Ec + \gamma_P E\hat{n},$$

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

$$\begin{aligned} Ec_{nk} &\leqslant n + \min\{k, n-k\} + \big(4\gamma_P + 2\big)\, gn/s + \big(2\gamma_P - 1\big)\, s \\ &\quad + \big(4\gamma_P + 2\big)\, n e^{-2g^2/s}. \end{aligned} \tag{5.4}$$

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

(a) Since $e^{-2g^2/s} = n^{-2\beta}$, $s = \lceil \alpha f(n) \rceil \leqslant \bar{\alpha} f(n)$ 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 Theorem 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* $1 - 4e^{-2\beta n^{\varepsilon}}$ *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

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

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

$$Ec_{nk} \leqslant n + \min\{k, n-k\} + (\hat{\gamma}_S + 4\gamma_S + 2\ln^{-1} n)\, f(n) \ln n, \tag{5.6}$$

where $\ln^{-1} n$ may be replaced by $\ln^{-1} 3$, and (5.6) still needs $\beta \geqslant 1/6$; for the choices (4.13) and (4.18), we may modify (5.3)–(5.6) as in Theorem 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 [1, Section 3.7], [13]) for which the expected number of comparisons to find the ith smallest of m elements is at most $\gamma_P m$ (then $Ec_{nk} \leqslant 2\gamma_P s + Ec + \gamma_P E\hat{n}$ is bounded as before).

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

(d) Our results improve upon [11, Theorem 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 [11] 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 [4,14], 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 Lemma 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 [25, Theorem 3.5], corresponding to (4.18) with $\varepsilon = 1/4$ and $\beta = 1$, where the probability bound $1 - O(n^{-1/4})$ is weaker than our $1 - 4\mathrm{e}^{-2n^{1/4}}$, sampling is done with replacement and the elements are distinct.

(d) Our results subsume [24, Theorem 2], which gives an estimate like (5.2) for the choice (4.13) with $\beta = 1$, using quickselect (cf. Remark 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 $x[1{:}n]$. We need the following notation to describe its operations in more detail.

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

A vector swap denoted by $x[a{:}b] \leftrightarrow x[b+1{:}c]$ means that the first $d := \min(b+1-a, c-b)$ elements of array $x[a{:}c]$ 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 $0 \leqslant i < d$, or $x_{a+i} \leftrightarrow x_{c-d+1+i}$ for $0 \leqslant i < d$.

6.1. Ternary partitions

For a given pivot $v := x_k$ from the array $x[l{:}r]$, the following *ternary* scheme partitions the array into three blocks, with $x_m < v$ for $l \leqslant m < a$, $x_m = v$ for $a \leqslant m \leqslant d$, $x_m > v$ for

$d < m \leqslant r$. The basic idea is to work with the five inner parts of the array

$$\begin{array}{|c|c|c|c|c|c|c|}\hline x<v & x=v & x<v & ? & x>v & x=v & x>v \\ \hline l & \bar{l} & p & i\ j & q & \bar{r} & r \end{array} \tag{6.1}$$

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

$$\begin{array}{|c|c|c|c|c|c|}\hline x=v & x<v & x=v & x>v & x=v \\ \hline \bar{l} & p & j & i & q & \bar{r} \end{array} \tag{6.2}$$

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

$$\begin{array}{|c|c|c|c|}\hline x<v & x=v & x>v \\ \hline \bar{l} & a & d & \bar{r} \end{array}\ . \tag{6.3}$$

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 $x_i \geqslant v$.] Increase i by 1; then if $x_i < v$, repeat this step.

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

A4. [Exchange.] (Here $x_j \leqslant v \leqslant x_i$.) 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 $x[\bar{l}: p-1] \leftrightarrow x[p: j]$ and $x[i: q] \leftrightarrow x[q+1: \bar{r}]$.

Step A1 ensures that $x_l \leqslant v \leqslant x_r$, so steps A2 and A3 do not need to test whether $i \leqslant j$; thus their loops can run faster than those in the schemes of [2, Prog. 6] and [20, Exercise 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 $l \leqslant i \leqslant r_s := l + s - 1$, 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 SELECT$(x, l, r_s, k_u, k_u^-, k_u^+)$ returns $k_u^+ \geqslant k_v$, 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 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:

$$\begin{array}{|c|c|c|c|c|c|}\hline x<u & x=u & u<x<v & x=v & x>v & ? \\ \hline l & k_u^- \ k_u^+ & & k_v^- \ k_v^+ & r_s & r \end{array}\ . \tag{6.4}$$

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

$$\begin{array}{|c|c|c|c|c|c|}\hline x<u & x=u & u<x<v & ? & x=v & x>v \\ \hline l & \bar{l} & \bar{p} & k_v^- \ \bar{q} & \bar{r} & r \end{array}. \tag{6.5}$$

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 Section 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 $l \leqslant m < a$, $x_m = u$ for $a \leqslant m < b$, $u < x_m < v$ for $b \leqslant m \leqslant c$, $x_m = v$ for $c < m \leqslant d$, $x_m > v$ for $d < m \leqslant r$. The basic idea is to work with the six-part array stemming from (6.5)

$$\begin{array}{|c|c|c|c|c|c|}\hline x=u & u<x<v & x<u & ? & x>v & x=v \\ \hline \bar{l} & \bar{p} & p & i\ j & q & \bar{r} \end{array} \tag{6.6}$$

until i and j cross

$$\begin{array}{|c|c|c|c|c|}\hline x=u & u<x<v & x<u & x>v & x=v \\ \hline \bar{l} & \bar{p} & p & j\ i & q & \bar{r} \end{array}, \tag{6.7}$$

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

$$\begin{array}{|c|c|c|c|c|c|}\hline x=u & x<u & u<x<v & x>v & x=v \\ \hline \bar{l} & \bar{p} & b & c\ i & q & \bar{r} \end{array} \tag{6.8}$$

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

$$\begin{array}{|c|c|c|c|c|c|}\hline x<u & x=u & u<x<v & x=v & x>v \\ \hline \bar{l} & a & b & c & d & \bar{r} \end{array}. \tag{6.9}$$

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

B2. [Increase i until $x_i \geqslant v$.] Increase i by 1. If $x_i \geqslant v$, go to B3. If $x_i < u$, repeat this step. (At this point, $u \leqslant x_i < v$.) 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 $i \geqslant j$, 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 $x[\bar{p}: p-1] \leftrightarrow x[p: j]$, $x[\bar{l}: \bar{p}-1] \leftrightarrow x[\bar{p}: b-1]$, and finally $x[i: q] \leftrightarrow x[q+1: \bar{r}]$.

For the case of $k \geqslant \lfloor (r+l)/2 \rfloor$ 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

$$\begin{array}{|c|c|c|c|c|c|} \hline x=u & x<u & ? & x>v & u<x<v & x=v \\ \hline \bar{l} & p & i\ j & q & \bar{q} & \bar{r} \end{array}, \tag{6.10}$$

$$\begin{array}{|c|c|c|c|c|c|} \hline x=u & x<u & x>v & u<x<v & x=v \\ \hline \bar{l} & p & j\ i & q & \bar{q} & \bar{r} \end{array}, \tag{6.11}$$

$$\begin{array}{|c|c|c|c|c|c|} \hline x=u & x<u & u<x<v & x>v & x=v \\ \hline \bar{l} & p & j\ b & c & \bar{q} & \bar{r} \end{array}. \tag{6.12}$$

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

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 $x_j \leqslant u$.] Decrease j by 1. If $x_j \leqslant u$, go to C4. If $x_j > v$, repeat this step. (At this point, $u < x_j \leqslant v$.) 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 $i \geqslant j$, 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 $x[\bar{l}: p-1] \leftrightarrow x[p: j]$, $x[i: q] \leftrightarrow x[q+1: \bar{q}]$ and finally $x[c+1: \bar{q}] \leftrightarrow x[\bar{q}+1: \bar{r}]$.

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 $a \leqslant k$, then $l := d+1$ if $c < k$; $r := c$ if $k \leqslant d$, then $r := a-1$ if $k < b$. If $l \geqslant r$, 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. Section 4.3).

A simple version of sSelect is obtained if Steps 2 and 3 choose $u := v := x_k$ when $r-l+1 \leqslant n_{\text{cut}}$ (this choice of [9] 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 $n \leqslant n_{\text{cut}}$ were handled by Steps 2 and 3 choosing $u := v := x_k$; the resulting version of

sSelect (cf. Sections 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 [9]; future work should test other sample sizes and parameters.

For comparisons we employed the implementation of QUICKSELECT from [16, Section 6.1], with median-of-3 pivots and the binary partitioning scheme of [16, Sections 2.1 and 4]. Apparently our implementation represents the state-of-the-art; see [16, Section 6.3] for comparisons with other partitioning schemes, and note that the results of [17, Section 7.3] suggest that the RISELECT algorithm of [31] tends to be less efficient.

7.2. Testing examples

We used minor modifications of the input sequences of [31], 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 [30]. 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 [26] were modified to become difficult when the middle element comes from position k instead of $k + 1$).

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 3, 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 $\ln n$.

Table 3
Performance of SELECT on randomly generated inputs

Sequence	Size n	Time (m s)			Comparisons (n)			γ_{avg}	L_{avg} (n)	P_{avg} ($\ln n$)	N_{avg} ($\ln n$)	p_{avg}	s_{avg} ($\%n$)
		Avg	Max	Min	Avg	Max	Min						
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

Finally, s_{avg} is the average sum of sample sizes; $s_{avg}/f(n)$ drops from 0.68 for $n = 50$K to 0.56 for $n = 16$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 could not be measured accurately), and the differences between maximum and minimum times are fairly small (and also partly 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 4 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 5 and 6. On the random sequences, the expected value of C_{avg} is $2.75n + o(n)$ [15]. Twenty random instances of each size yield fairly accurate estimates, since the values of C_{avg} in Table 5 are within 7% of $2.75n$; Table 7 shows what happens when 1000 instances are used for each size. The results for the onezero sequences confirm that binary partitioning may handle

Table 4
Performance of SELECT on deterministic inputs

Sequence	Size n	Time (m s)			Comparisons (n)			γ_{avg}	L_{avg} (n)	P_{avg} ($\ln n$)	N_{avg} ($\ln n$)	p_{avg}	s_{avg} ($\%n$)
		Avg	Max	Min	Avg	Max	Min						
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

equal keys quite efficiently [30]. The results for the remaining inputs are quite good, since some versions of quickselect may behave poorly on these inputs [31].

As always, limited testing does not 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 Fig. 1. Note that SELECT is not 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

Table 5
Performance of QUICKSELECT on randomly generated inputs

Sequence	Size n	Time (m s)			Comparisons (n)			L_{avg} (n)	P_{avg} ($\ln n$)
		Avg	Max	Min	Avg	Max	Min		
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

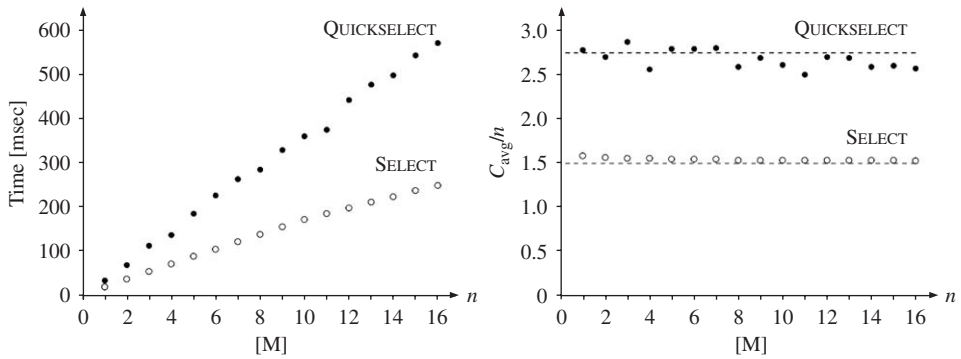

Fig. 1. Average running times and comparisons per element on random inputs.

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.

Table 6
Performance of QUICKSELECT on deterministic inputs

Sequence	Size n	Time (m s)			Comparisons (n)			L_{avg} (n)	P_{avg} ($\ln n$)
		Avg	Max	Min	Avg	Max	Min		
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

Table 7
Numbers of comparisons per element made on small random inputs

Size		1000	2500	5000	7500	10,000	12,500	15,000	17,500	20,000	25,000
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

Acknowledgements

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 in improving our presentation.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.
[2] J.L. Bentley, M.D. McIlroy, Engineering a sort function, Software-Practice Experience 23 (1993) 1249–1265.
[3] M.R. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, R.E. Tarjan, Time bounds for selection, J. Comput. System Sci. 7 (1972) 448–461.
[4] V. Chvátal, The tail of the hypergeometric distribution, Discrete Math. 25 (1979) 285–287.
[5] W. Cunto, J.I. Munro, Average case selection, J. ACM 36 (1989) 270–279.
[6] D. Dor, J. Håstad, S. Ulfberg, U. Zwick, On lower bounds for selecting the median, SIAM J. Discrete Math. 14 (2001) 299–311.
[7] D. Dor, U. Zwick, Selecting the median, SIAM J. Comput. 28 (1999) 1722–1758.
[8] D. Dor, U. Zwick, Median selection requires $(2+\varepsilon)N$ comparisons, SIAM J. Discrete Math. 14 (2001) 312–325.
[9] R.W. Floyd, R.L. Rivest, The algorithm SELECT—for finding the ith smallest of n elements (Algorithm 489), Comm. ACM 18 (1975) 173.
[10] R.W. Floyd, R.L. Rivest, Expected time bounds for selection, Comm. ACM 18 (1975) 165–172.
[11] A.V. Gerbessiotis, C.J. Siniolakis, Randomized selection in $n + C + o(n)$ comparisons, Inform. Proc. Lett. 88 (2003) 95–100.
[12] R. Grübel, On the median-of-k version of Hoare's selection algorithm, Theoret. Inform. Appl. 33 (1999) 177–192.
[13] C.A.R. Hoare, Algorithm 65: FIND, Comm. ACM 4 (1961) 321–322.
[14] W. Hoeffding, Probability inequalities for sums of bounded random variables, J. Amer. Statist. Assoc. 58 (1963) 13–30.
[15] P. Kirschenhofer, C. Martínez, H. Prodinger, Analysis of Hoare's FIND algorithm with median-of-three partition, Random Struct. Algorithms 10 (1997) 143–156.
[16] K.C. Kiwiel, Partitioning schemes for quicksort and quickselect, Technical report, Systems Research Institute, Warsaw, 2003, available at the URL http://arxiv.org/abs/cs.DS/0312054.
[17] K.C. Kiwiel, Randomized selection with quintary partitions, Technical report, Systems Research Institute, Warsaw, 2003, available at the URL http://arxiv.org/abs/cs.DS/0312055.
[18] K.C. Kiwiel, Improved randomized selection, Technical report, Systems Research Institute, Warsaw, 2004, available at the URL http://arxiv.org/abs/cs.DS/0402005.
[19] K.C. Kiwiel, Randomized selection with tripartitioning, Technical report, Systems Research Institute, Warsaw, 2004, available at the URL http://arxiv.org/abs/cs.DS/0401003.
[20] D.E. Knuth, The Art of Computer Programming, Fundamental Algorithms, third ed., Vol. 1, Addison-Wesley, Reading, MA, 1997.
[21] D.E. Knuth, The Art of Computer Programming, Sorting and Searching, second ed., Vol. 3, Addison-Wesley, Reading, MA, 1998.
[22] V.S. Koroliuk (Ed.), Handbook on Probability Theory and Mathematical Statistics, Naukova Dumka, Kiev, 1978 (Russian).
[23] C. Martínez, S. Roura, Optimal sampling strategies in quicksort and quickselect, SIAM J. Comput. 31 (2001) 683–705.
[24] 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.

[25] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, Cambridge, England, 1995.
[26] D.R. Musser, Introspective sorting and selection algorithms, Software-Practice Experience 27 (1997) 983–993.
[27] J.T. Postmus, A.H.G. Rinnooy Kan, G.T. Timmer, An efficient dynamic selection method, Comm. ACM 26 (1983) 878–881.
[28] R. Reischuk, Probabilistic parallel algorithms for sorting and selection, SIAM J. Comput. 14 (1985) 396–409.
[29] A. Schönhage, M. Paterson, N. Pippenger, Finding the median, J. Comput. System Sci. 13 (1976) 184–199.
[30] R. Sedgewick, Quicksort with equal keys, SIAM J. Comput. 6 (1977) 240–287.
[31] J.D. Valois, Introspective sorting and selection revisited, Software-Practice Experience 30 (2000) 617–638.