Randomized selection with tripartitioning

Krzysztof C. Kiwiel*

January 4, 2004

Abstract

We show that several versions of Floyd and Rivest’s algorithm SELECT [Comm. ACM 18 (1975) 173] for finding the $k$th smallest of $n$ elements require at most $n + \min\{k, n - k\} + o(n)$ comparisons on average, even when equal elements occur. This parallels our recent analysis of another variant due to Floyd and Rivest [Comm. ACM 18 (1975) 165–172]. Our computational results suggest that both variants perform well in practice, and may compete with other selection methods, such as Hoare’s FIND or quickselect with median-of-3 pivots.

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 $1 \leq k \leq n$, find the $k$th 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 $x_j \leq x$. The median of $X$ is the $\lceil n/2 \rceil$th smallest element of $X$.

Selection is one of the fundamental problems in computer science; see, e.g., the references in [DHUZ01, DoZ99, DoZ01] and [Knu98, §5.3.3]. Most references concentrate on the number of comparisons between pairs of elements made in selection algorithms. In the worst case, selection needs at least $(2 + \epsilon)n$ comparisons [DoZ01], whereas the algorithm of [BFP+72] makes at most $5.43n$, that of [SPP76] needs $3n + o(n)$, and that in [DoZ99] takes $2.95n + o(n)$. In the average case, for $k \leq \lceil n/2 \rceil$, at least $n + k - O(1)$ comparisons are necessary [CuM89], whereas the best upper bound is $n + k + O(n^{1/2} \ln^{1/2} n)$ [Knu98, Eq. (5.3.3.16)]. The classical algorithm FIND of [Hoa61], also known as quickselect, has an upper bound of $3.39n + o(n)$ for $k = \lceil n/2 \rceil$ in the average case [Knu98, Ex. 5.2.2–32], which improves to $2.75n + o(n)$ for median-of-3 pivots [Grn99, KMP97].

In practice FIND is most popular. One reason is that the algorithms of [BFP+72, SPP76] are much slower on the average [Mus97, Val00], whereas [KMP97] adds that other methods proposed so far, although better than FIND in theory, are not practical because they are difficult to implement, their constant factors and hidden lower order terms are

*Systems Research Institute, Newelska 6, 01–447 Warsaw, Poland (kiwiel@ibspan.waw.pl)
too large, etc. It is quite surprising that these references \cite{KMP97,Mus97,Val00} ignore the algorithm \texttt{SELECT} of \cite{FlR75}, since most textbooks mention that \texttt{SELECT} is asymptotically faster than \texttt{FIND}. In contrast, this paper shows that \texttt{SELECT} can compete with \texttt{FIND} in both theory and practice, even for fairly small values of the input size $n$.

We now outline our contributions in more detail. The initial two versions of \texttt{SELECT} \cite{FlR75} had gaps in their analysis (cf. \cite{Bro76,PRKT83,Knu98,Ex. 5.3.3–24}); the first version was validated in \cite{Kiw03b}, and the second one will be addressed elsewhere. This paper deals with the third version of \texttt{SELECT} from \cite{FlR75}, which operates as follows. Using a small random sample, it finds an element $v$ almost sure to be just above the $k$th if $k < n/2$, or below the $k$th if $k \geq n/2$. Partitioning $X$ about $v$ leaves $\min\{k, n - k\} + o(n)$ elements on average for the next recursive call, in which $k$ is near 1 or $n$ with high probability, so this second call eliminates almost all the remaining elements.

Apparently this version of \texttt{SELECT} has not been analyzed in the literature, even in the case of distinct elements. We first revise it slightly to simplify our analysis. Then, without assuming that the elements are distinct, we show that \texttt{SELECT} needs at most $n + \min\{k, n - k\} + O(n^{2/3} \ln^{1/3} n)$ comparisons on average, with $\ln^{1/3} n$ replaced by $\ln^{1/2} n$ for the original samples of \cite{FlR75}. Thus the average cost of \texttt{SELECT} reaches the lower bounds of $1.5n + o(n)$ for median selection and $1.25n + o(n)$ for selecting an element of random rank. For the latter task, \texttt{FIND} has the bound $2n + o(n)$ when its pivot is set to the median of a random sample of $s$ elements, with $s \to \infty$, $s/n \to \infty$ as $n \to \infty$ \cite{MaR01}; thus \texttt{SELECT} improves upon \texttt{FIND} mostly by using $k$, the rank of the element to be found, for selecting the pivot $v$ in each recursive call.

\texttt{SELECT} can be implemented by using the tripartitioning schemes of \cite{Kiw03,§5}, which include a modified scheme of \cite{BeM93}; more traditional bipartitioning schemes \cite{Kiw03a,§2] can perform quite poorly in \texttt{SELECT} when equal elements occur. We add that the implementation of \cite{FlR75} avoids random number generation by assuming that the input file is in random order, but this results in poor performance on some inputs of \cite{Val00}; hence our implementation of \texttt{SELECT} employs random sampling.

Our computational experience shows that \texttt{SELECT} outperforms even quite sophisticated implementations of \texttt{FIND} in both comparison counts and computing times. To save space, only selected results are reported for the version of \cite{Val00}, but our experience with other versions on many different inputs was similar. \texttt{SELECT} turned out to be more stable than \texttt{FIND}, having much smaller variations of solution times and numbers of comparisons. Quite surprisingly, contrary to the folklore saying that \texttt{SELECT} is only asymptotically faster than \texttt{FIND}, \texttt{SELECT} makes significantly fewer comparisons even for small inputs (cf. Tab. 7.8).

To relate our results with those of \cite{Kiw03}, let’s call \texttt{QSELECT} the quintary method of \cite{Kiw03} stemming from \cite{FlR75,§2.1]. \texttt{QSELECT} eliminates almost all elements on its first call by using two pivots, almost sure to be just below and above the $k$th element, in a quintary partitioning scheme. Thus most work occurs on the first call of \texttt{QSELECT}, which corresponds to the first two calls of \texttt{SELECT}. Hence \texttt{SELECT} and \texttt{QSELECT} share the same efficiency estimates, and in practice make similarly many comparisons. However, \texttt{QSELECT} tends to be slightly faster on median finding: although its quintary scheme is more complex, most of its work is spent on the first pass through $X$, whereas \texttt{SELECT} first partitions $X$ and then the remaining part (about half) of $X$ on its second call to achieve a
similar problem reduction. On the other hand, SELECT makes fewer comparisons on small inputs. Of course, future work should assess more fully the relative merits of SELECT and qSELECT. For now, the tests reported in [Kiw03a, Kiw03b] and in § suggest that both SELECT and qSELECT can compete successfully with refined implementations of FIND.

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. A modification that improves practical performance is introduced 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, E is the mean-value operator and P[·|E] is the probability conditioned on an event E; the complement of E is denoted by E'.

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.

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

Step 1 (Initiation). If n = 1, return x1. Choose the sample size s ≤ n − 1 and gap g > 0.

Step 2 (Sample selection). Pick randomly a sample S := {y1, . . . , ys} from X.

Step 3 (Pivot selection). Let v be the output of SELECT(S, iv), where

\[ iv := \begin{cases} \min \{\lceil ks/n + g\rceil, s\} & \text{if } k < n/2, \\ \max \{\lceil ks/n - g\rceil, 1\} & \text{if } k \geq n/2. \end{cases} \] (2.1)

Step 4 (Partitioning). By comparing each element x of X \ S to v, partition X into the three sets L := {x ∈ X : x < v}, E := {x ∈ X : x = v} and R := {x ∈ X : v < x}.

Step 5 (Stopping test). If |L| < k ≤ |L ∪ E|, return v.

Step 6 (Reduction). If k ≤ |L|, set ˘X := L, ˘n := |˘X| and ˘k := k; else set ˘X := R, ˘n := |˘X| and ˘k := k − |L ∪ E|.

Step 7 (Recursion). Return SELECT(˘X, ˘k).

A few remarks on the algorithm are in order.

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, ˘X and ˘k are chosen so that the kth smallest element of X is the ˘kth smallest element of ˘X, and ˘n < n (since v ̸∈ ˘X). Also |S| < n for the recursive call at Step 3.

(b) When Step 5 returns v, SELECT may also return information about the positions of the elements of X relative to v. For instance, if X is stored as an array, its k smallest
elements may be placed first via interchanges at Step 4 (cf. \[\text{[Hoe63]}\]). Hence Step 4 need only compare \(v\) with the elements of \(X \setminus S\).

(c) The following elementary property is needed in \[\text{[Hoe63]}\]. Let \(c_n\) denote the maximum number of comparisons taken by \textsc{Select} on any input of size \(n\). Since Step 3 makes at most \(c_n\) comparisons with \(s < n\), Step 4 needs at most \(n - s\), and Step 7 takes at most \(c_n\) with \(\bar{n} < n\), by induction \(c_n < \infty\) for all \(n\).

### 3 Sampling deviations

In this section we analyze general features of sampling used by \textsc{Select}. Our analysis hinges on the following bound on the tail of the hypergeometric distribution established in \[\text{[Hoe63]}\] and rederived shortly in \[\text{[Chv79]}\].

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

\[
P[r' \geq ps + g] \leq e^{-2g^2/s} \quad \forall g \geq 0. \tag{3.1}
\]

Denote by \(x_1^* \leq \ldots \leq x_n^*\) and \(y_1^* \leq \ldots \leq y_n^*\) the sorted elements of the input set \(X\) and the sample set \(S\), respectively, so that \(v = y_n^*\). The following result will give bounds on the position of \(v\) in the sorted input sequence.

**Lemma 3.2.** Suppose \(\bar{i} := \max\{1, \min([ks], s)\}\), \(\bar{j}_i := \max\{[kn - gn/s], 1\}\), and \(\bar{j}_r := \min\{[kn + gn/s], n\}\), where \(-g < \kappa s \leq s + g\), \(1 \leq s \leq n\) and \(g \geq 0\). Then:

(a) \(P[y_i^* < x_j^*] \leq e^{-2g^2/s}\) if \(\bar{i} \geq [\kappa s]\).
(b) \(P[x_j^* < y_i^*] \leq e^{-2g^2/s}\) if \(\bar{i} \leq [\kappa s]\).

**Proof.** Note that \(-g < \kappa s \leq s + g\) implies that \(\bar{j}_i \leq n\) and \(\bar{j}_r \geq 1\) are well-defined.

(a) If \(y_i^* < x_j^*\), at least \(\bar{i}\) samples satisfy \(y_i \leq x_j\), where \(r := \max_{x_j^* < x_j} j\). In the setting of Fact 3.1, we have \(r\) red elements \(x_j \leq x_j^*\), \(ps = rs/n\) and \(r' \geq \bar{i}\). Now, \(1 \leq r \leq \bar{j}_i - 1\) implies \(2 \leq \bar{j}_i = [kn - gn/s] < \kappa n - gn/s + 1\), so \(-rs/n > -\kappa s + g\). Hence \(\bar{i} - ps - g > \kappa s - \kappa s + g - g = 0\), i.e., \(r' > ps + g\). Thus \(P[y_i^* < x_j^*] \leq e^{-2g^2/s}\) by (3.1).

(b) If \(x_j^* < y_i\), \(s - \bar{i} + 1\) samples are at least \(x_{j+1}^*\) with \(j := \max_{x_j^* = x_j^*} j\). Thus we have \(r := n - j\) red elements \(x_j \geq x_{j+1}\), \(ps = s - j\) and \(r' \geq s - \bar{i} + 1\). Since \(\bar{i} < \kappa s + 1\) and \(n > j \geq \bar{j}_r \geq \kappa n + gn/s\), we get \(s - \bar{i} + 1 - ps - g > \bar{j}_s/n - \kappa s - g \geq \kappa s + g - \kappa s - g = 0\). Hence \(r' > ps + g\) and \(P[x_j^* < y_i^*] \leq P[r' \geq ps + g] \leq e^{-2g^2/s}\) by (3.1).

We now bound the position of \(v\) relative to \(x_k^*, x_{k_{l}}^*\) and \(x_{k_{r}}^*\), where

\[
k_l := \max\{[k - 2gn/s], 1\} \quad \text{and} \quad k_r := \min\{[k + 2gn/s], n\}. \tag{3.2}
\]

**Corollary 3.3.** (a) \(P[v < x_k^*] \leq e^{-2g^2/s}\) if \(i_v = [ks/n + g]\) and \(k < n/2\).
(b) \(P[x_{k_r}^* < v] \leq e^{-2g^2/s}\) if \(k < n/2\).
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 | 353855        | 568986 |
| $\phi(n)$ | 190.449 | 0.097295 | 0.048648 | 0.023995 | 0.014557      | 0.011725 | 0.007078      | 0.005690 |

(c) $P[x^*_k < v] \leq e^{-2g^2/s}$ if $i_v = \lfloor ks/n - g \rfloor$ and $k \geq n/2$.

(d) $P[v < x^*_k] \leq e^{-2g^2/s}$ if $k \geq n/2$.

(e) If $k < n/2$, then $i_v \neq \lfloor ks/n + g \rfloor$ iff $n < k + gn/s$; similarly, if $k \geq n/2$, then $i_v \neq \lfloor ks/n - g \rfloor$ iff $k \leq gn/s$.

**Proof.** Use Lem. 3.2 with $\kappa s = ks/n + g$ for (a,b), and $\kappa s = ks/n - g$ for (c,d). □

## 4 Average case performance

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

### 4.1 Floyd-Rivest’s samples

For positive constants $\alpha$ and $\beta$, consider choosing $s = s(n)$ and $g = g(n)$ as

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

This form of $g$ gives a probability bound $e^{-2g^2/s} = n^{-2\beta}$ for Cor. 3.3. 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 it will be seen that the recursive call reduces $n$ at least by the factor $4\phi(n)$ on average, 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 $\gamma$ such that

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

**Proof.** 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 $\phi$, we have for $n \geq \bar{n}$

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

$$f(\lfloor \delta f(n) \rfloor) \leq f(\delta f(n)) \leq \delta \phi(\delta f(\bar{n})) f(n). \quad (4.4)$$

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

$$f(\bar{\alpha} f(n)) = \bar{\alpha} \phi(\bar{\alpha} f(\bar{n})) f(n) \leq \bar{\alpha} \phi(\bar{\alpha} f(\bar{n})) f(n).$$
Also for \( n \geq \tilde{n} \), we have \( s = [\alpha f(n)] = \alpha f(n) + \epsilon \) with \( \epsilon \in [0,1) \) in (4.1). Writing \( s = \hat{\alpha} f(n) \) with \( \hat{\alpha} := \alpha + \epsilon / f(n) \in [\alpha, \hat{\alpha}] \), we deduce from (4.1) that
\[
 gn/s = (\beta / \hat{\alpha})^{1/2} f(n) \leq (\beta / \alpha)^{1/2} f(n).
\]
In particular, \( 4 gn/s \leq \delta f(n) \), since \( \delta := 4(\beta / \alpha)^{1/2} \). Next, (4.1) implies
\[
 ne^{-2g^2/s} \leq n^{1-2\beta} = f(n)n^{1/3-2\beta} \ln^{-1/3} n.
\]
Using the monotonicity of \( f \) and \( \phi \), increase \( n \) if necessary to get for all \( n \geq \tilde{n} \)
\[
 2\hat{\alpha} \phi(\hat{\alpha} f(\tilde{n})) + \delta \phi(\delta f(\tilde{n})) + 2n^{1-2\beta} + 2 \max \{ [\delta f(n)]^{2/3-2\beta} n^{-2/3}, n^{-2\beta} \} \leq 0.95. \tag{4.7}
\]
By Rem. 2.2(c), there is \( \gamma \) such that (4.2) holds for all \( n \leq \tilde{n} \); increasing \( \gamma \) if necessary, and using the monotonicity of \( f \) and the assumption \( \beta \geq 1/6 \), we have for all \( n \geq \tilde{n} \)
\[
 2\hat{\alpha} + 2\delta + 5n^{1/3-2\beta} \ln^{-1/3} n + 3 \max \{ \delta^{1-2\beta} f(n)^{-2\beta}, n^{1/3-2\beta} \ln^{-1/3} n \} \leq 0.05\gamma. \tag{4.8}
\]
Let \( n' \geq \tilde{n} \). Assuming (4.2) holds for all \( n \leq n' \), for induction let \( n = n' + 1 \).
We need to consider the following two cases in the first call of Select.

**Left case:** \( k < n/2 \). First, suppose the event \( \mathcal{E}_l := \{ x^*_k \leq v < x^*_k \} \) occurs. By the rules of Steps 4–6, we have \( \hat{X} = L \) (from \( x^*_k \leq v \)), \( \hat{k} = k \) and \( \hat{n} := |\hat{X}| \leq k - 1 \) (from \( v \leq x^*_k \)); since \( k < k < 2 gn/s + 1 \) by (3.2), we get the two (equivalent) bounds
\[
 \hat{n} < k + 2 gn/s \quad \text{and} \quad \hat{n} - \hat{k} < 2 gn/s. \tag{4.9}
\]
Note that if \( i_v := [ks/n + g] \) then, by Cor. 3.3(a,b), the Boole-Benferroni inequality and the choice (4.1), the complement \( \mathcal{E}_l' \) of \( \mathcal{E}_l \) has \( P[\mathcal{E}_l'] \leq 2 e^{-2g^2/s} = 2n^{-2\beta} \). Second, if \( i_v \neq [ks/n + g] \), then \( n < k + gn/s \) (Cor. 3.3(e)) combined with \( k < n/2 \) gives \( n < 2 gn/s \); hence \( \hat{n} - \hat{k} < \hat{n} < n < 2 gn/s \) implies (4.9). Since also \( \mathcal{E}_l \) implies (4.9), we have
\[
 P[\mathcal{A}_l] \leq 2n^{-2\beta} \quad \text{for} \quad \mathcal{A}_l := \{ \hat{n} - \hat{k} < 2 gn/s \}. \tag{4.10}
\]

**Right case:** \( k \geq n/2 \). First, suppose the event \( \mathcal{E}_r := \{ x^*_{k_l} \leq v < x^*_k \} \) occurs. By the rules of Steps 4–6, we have \( \hat{X} = R \) (from \( v \leq x^*_{k_l} \)), \( \hat{n} - \hat{k} = n - k \) and \( \hat{n} := |\hat{X}| \leq n - k_l \) (from \( x^*_{k_l} \leq v \)); since \( k_l \geq k - 2 gn/s \) by (3.2), we get the two (equivalent) bounds
\[
 \hat{n} \leq n - k + 2 gn/s \quad \text{and} \quad \hat{k} \leq 2 gn/s, \tag{4.11}
\]
using \( \hat{n} - \hat{k} = n - k \). If \( i_v := [ks/n - g] \) then, by Cor. 3.3(c,d), the complement \( \mathcal{E}_r' \) of \( \mathcal{E}_r \) has \( P[\mathcal{E}_r'] \leq 2 e^{-2g^2/s} = 2n^{-2\beta} \). Second, if \( i_v \neq [ks/n - g] \), then \( k \leq gn/s \) (Cor. 3.3(e)) combined with \( k \geq n/2 \) gives \( n \leq 2 gn/s \); hence \( \hat{k} \leq \hat{n} < n \leq 2 gn/s \) implies (4.11). Thus
\[
 P[\mathcal{A}_r'] \leq 2n^{-2\beta} \quad \text{for} \quad \mathcal{A}_r := \{ \hat{k} \leq 2 gn/s \}. \tag{4.12}
\]
Since \( k < n - k \) if \( k < n/2 \), \( n - k < k \) if \( k \geq n/2 \), (4.9) and (4.11) yield
\[
 P[\mathcal{B}] \leq 2n^{-2\beta} \quad \text{for} \quad \mathcal{B} := \{ \hat{n} \leq \min\{ k, n - k \} + 2 gn/s \}. \tag{4.13}
\]
Note that \( \min\{k, n - k\} \leq \lfloor n/2 \rfloor \leq n/2 \); this relation will be used implicitly below.

For the recursive call of Step 7, let \( \hat{s}, \hat{y} \) and \( \hat{i}_v \) denote the quantities generated as in (4.1) and (2.1) with \( n \) and \( k \) replaced by \( \hat{n} \) and \( \hat{k} \); let \( \hat{v} \) be the pivot found at Step 3, and let \( \hat{X}, \hat{n} \) and \( \hat{k} \) correspond to \( X, n \) and \( k \) at Step 7, so that \( \hat{n} := |X| < \hat{n} \).

The cost of selecting \( v \) and \( \hat{v} \) at Step 3 may be estimated as

\[
C_{s_{iv}} + C_{\hat{s}_{iv}} \leq 1.5s + \gamma_f(s) + 1.5\hat{s} + \gamma_f(\hat{s}) \leq 3s + 2\gamma_f(s),
\]

(4.14)
since \( f \) is increasing and (4.2) holds for \( \hat{s} \leq s \leq n - 1 = n' \) (cf. (4.1)) from \( \hat{n} < n \).

Let \( c := n - s \) and \( \hat{c} := \hat{n} - \hat{s} \) denote the costs of Step 4 for the two calls. Since \( 0 \leq \hat{c} < n \) and 
\[
E\hat{c} = E[\hat{c} | \mathcal{B}^c | \mathcal{B}] + E[\hat{c} | \mathcal{B}^c | \mathcal{B}'] \leq E[\hat{c} | \mathcal{B}] + nP[\mathcal{B}'],
\]
by (4.13) we have

\[
c + E\hat{c} \leq n - s + \min\{k, n - k\} + 2gn/s + 2n^{1-2\beta}.
\]

(4.15)

Using (4.2) again with \( \hat{n} < n \), the cost of finishing up at Step 7 is at most

\[
EC_{\hat{n}k} \leq E[1.5\hat{n} + \gamma_f(\hat{n})] = 1.5E\hat{n} + \gamma_f(\hat{n}).
\]

(4.16)

Thus we need suitable bounds for \( E\hat{n} \) and \( \gamma_f(\hat{n}) \), which may be derived as follows.

To generalize (4.13) to the recursive call, consider the events

\[
\mathcal{B} := \{ \hat{n} \leq \min\{ \hat{k}, n - \hat{k}\} + 2\hat{g}\hat{n}/\hat{s} \} \quad \text{and} \quad \mathcal{C} := \{ \hat{n} \leq \lfloor \delta_f(n) \rfloor \}.
\]

(4.17)

By (4.10) and (4.12), \( \mathcal{B} \cap \mathcal{A}_i \) and \( \mathcal{B} \cap \mathcal{A}_r \) imply \( \mathcal{C} \), since \( 2gn/s + 2\hat{g}\hat{n}/\hat{s} \leq \delta_f(n) \) by (4.5) with \( \hat{n} < n \) and \( \delta := 4(\beta/\alpha)^{1/2} \). For the recursive call, proceeding as in the derivation of (4.13) with \( n \) replaced by \( \hat{n} = i, k \) by \( \hat{k} \), etc., shows that, due to random sampling,

\[
P[\mathcal{B}^c | \mathcal{A}_i, \hat{n} = i] \leq 2i^{-2\beta} \quad \text{and} \quad P[\mathcal{B}^c | \mathcal{A}_r, \hat{n} = i] \leq 2i^{-2\beta}.
\]

(4.18)

In the left case of \( k < n/2 \), using \( \hat{n} < n \) and \( P[\mathcal{A}_i^c] \leq 2n^{-2\beta} \) (cf. (4.10)), we get

\[
E\hat{n} = E[\hat{n} | \mathcal{A}_i]P[\mathcal{A}_i] + E[\hat{n} | \mathcal{A}_i^c]P[\mathcal{A}_i^c] \leq E[\hat{n} | \mathcal{A}_i] + n2n^{-2\beta}.
\]

Partitioning \( \mathcal{A}_i \) into the events \( \mathcal{D}_i := \mathcal{A}_i \cap \{ \hat{n} = i \}, i = 0: n - 1 \) (\( \hat{n} < n \) always), we have

\[
E[\hat{n} | \mathcal{A}_i] = \sum_{i=0}^{n-1} E[\hat{n} | \mathcal{D}_i]P[\mathcal{D}_i | \mathcal{A}_i] \leq \max_{i=0:n-1} E[\hat{n} | \mathcal{D}_i],
\]

where \( E[\hat{n} | \mathcal{D}_i] \leq [\delta_f(n)] \) if \( i \leq [\delta_f(n)] + 1 \), because \( \hat{n} < \hat{n} \) always. As for the remaining terms, \( \mathcal{B} \cap \mathcal{A}_i \subset \mathcal{C} \) implies \( P[\mathcal{C} | \mathcal{D}_i] \leq P[\mathcal{B} | \mathcal{D}_i] \leq 2i^{-2\beta} \) by (4.18), where \( \mathcal{C} := \{ \hat{n} \leq [\delta_f(n)] \} \) and \( \hat{n} < \hat{n} = i \) when the event \( \mathcal{D}_i \) occurs, so \( E[\hat{n} | \mathcal{D}_i] \leq [\delta_f(n)] + 2i^{-2\beta} \). Hence

\[
\max_{i=0:n-1} E[\hat{n} | \mathcal{D}_i] \leq [\delta_f(n)] + \max_{i=\lceil \delta_f(n) \rceil + 2n-1}^{2i^{1-2\beta}},
\]

where the final term is omitted if \( [\delta_f(n)] > n - 3 \); otherwise it is at most

\[
2\max\{([\delta_f(n)] + 1)^{1-2\beta}, n^{1-2\beta}\} \leq 2\max\{\delta^{1-2\beta}f(n)^{-2\beta}, n^{1/3-2\beta} \ln^{-1/3} n\} f(n),
\]

7
since \(\max_{i=\lceil\delta f(n)\rceil+1, n} 2i^{1-2\beta}\) is bounded as above (consider \(\beta \geq 1/2\), then \(\beta < 1/2\) and use \(\delta f(n) < \lceil\delta f(n)\rceil + 1\), the monotonicity of \(f\) and \((4.6)\) for the final inequality). Collecting the preceding estimates, we obtain

\[
E\bar{n} \leq \lfloor \delta f(n) \rfloor + 2n^{1-2\beta} + 2 \max \left\{ \delta^{1-2\beta} f(n)^{-2\beta}, n^{1/3-2\beta} \ln^{-1/3} n \right\} f(n). 
\]

(4.19)

Similarly, replacing \(\bar{n}\) by \(f(\bar{n})\) in our derivations and using the monotonicity of \(f\) yields

\[
E f(\bar{n}) \leq f(\lfloor \delta f(n) \rfloor) + 2f(n)n^{-2\beta} + \max_{i=\lceil\delta f(n)\rceil+2:n-1} 2f(i)i^{-2\beta},
\]

(4.20a)

where the final term is omitted if \(\lfloor \delta f(n) \rfloor > n-3\); otherwise it is at most

\[
2 \max \left\{ \frac{f(\lfloor \delta f(n) \rfloor) + 1}{(\lceil \delta f(n) \rceil + 1)^{2\beta}}, \frac{f(n)}{n^{2\beta}} \right\} \leq 2 \max \left\{ \delta f(n) \right\} \frac{1}{n^{2\beta}} \}
\]

(4.20b)

To see this, use the monotonicity of \(f\) and the fact that for \(i \leq n\) (cf. \((4.1)\))

\[
f(i)i^{-2\beta}/f(n) = i^{2/3-2\beta}n^{-2/3}(\ln i/\ln n)^{1/3} \leq i^{2/3-2\beta}n^{-2/3}.
\]

For the right case, replace \(A_i\) by \(A_r\) in the preceding paragraph to get \((4.19)-(4.20)\).

Add the costs \((4.14)-(4.15)\) and \((4.16)\), using \((4.19)-(4.20)\), to get

\[
C_{nk} \leq 3s + 2\gamma f(s) + n - s + \min\{k, n - k\} + 2gN/s + 2n^{1-2\beta}
+ 1.5\lfloor \delta f(n) \rfloor + 3n^{1-2\beta} + 3 \max \left\{ \delta^{1-2\beta} f(n)^{-2\beta}, n^{1/3-2\beta} \ln^{-1/3} n \right\} f(n)
+ \gamma f(\lfloor \delta f(n) \rfloor) + 2\gamma f(n)n^{-2\beta} + 2\gamma \max \left\{ \delta f(n) \right\} f(n).
\]

Now, using the bounds \((4.3)-(4.4)\), \(2gN/s \leq \frac{1}{2}f(n)\) (cf. \((4.3)\)) and \((4.6)\) gives

\[
C_{nk} \leq n + \min\{k, n - k\}
+ \left[ 2\tilde{\alpha} + 2\delta + 5n^{1/3-2\beta} \ln^{-1/3} n + 3 \max \left\{ \delta^{1-2\beta} f(n)^{-2\beta}, n^{1/3-2\beta} \ln^{-1/3} n \right\} \right] f(n)
+ \left[ 2\tilde{\alpha}\delta f(\bar{n}) \right] \right\} \gamma f(n).
\]

By \((4.7)-(4.8)\), the two bracketed terms above are at most \(0.05\gamma f(n)\) and \(0.95\gamma f(n)\), respectively; thus \((4.2)\) holds as required. \(\square\)

### 4.2 Other sampling strategies

We now indicate briefly how to adapt the proof of Thm 4.1 to several variations on \((4.1)\); a choice similar to \((4.2)\) below was used in \(\text{FIR75a}\).

**Remarks 4.2.** (a) Theorem 4.1 remains true for \(\beta \geq 1/6\) and \((4.1)\) replaced by

\[
s := \min \left\{ \left\lfloor \alpha n^{2/3} \right\rfloor, n - 1 \right\}, \ g := (\beta s \ln n)^{1/2} \text{ and } f(n) := n^{2/3} \ln^{1/2} n.
\]

(4.21)

Indeed, using \(e^{3/2} - 1 \leq \alpha n^{2/3} \leq n - 1\), \(e^{3/2} \leq \delta f(n)\), \(\alpha := \alpha + n^{-2/3}\) and \(s = \alpha n^{2/3}\) with \(\alpha \in [\alpha, \tilde{\alpha})\) yields \((4.3)-(4.5)\) as before, and \(\ln^{-1/2}\) replaces \(\ln^{-1/3}\) in \((4.6), (4.8)\) and \((4.19)\).
(b) Theorem 4.1 holds for the following modification of (4.1) with \( \epsilon_l > 1 \)
\[
s := \min \{\lceil \alpha f(n) \rceil, n - 1\} \quad \text{and} \quad g := (\beta s \ln \epsilon_l n)^{1/2} \text{ with } f(n) := n^{2/3} \ln \epsilon_l^{1/3} n. \tag{4.22}
\]
First, using \( e^{\epsilon_l} - 1 \leq \alpha f(\bar{n}) \leq \bar{n} - 1 \) and \( e^{\epsilon_l} \leq \delta f(\bar{n}) \) gives (4.3)–(4.5) as before. Next, fix \( \tilde{\beta} \geq 1/6. \) Let \( \beta_n := \beta \ln \epsilon_l - 1_n. \) Increase \( \bar{n} \) if necessary so that \( \beta_i \geq \tilde{\beta} \) for all \( i \geq \min \{\bar{n}, \lceil \delta f(\bar{n}) \rceil\}; \) then replace \( \beta \) by \( \tilde{\beta} \) and \( \ln n^{1/3} \) by \( \ln \epsilon_l n^{1/3} \) in (4.6) and below.

(c) Several other replacements for (4.1) may be analyzed as in [Kiw03b, §§4.1–4.2].

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

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

Remarks 4.3. (a) Suppose Step 2 of \textsc{Select} employs sampling with replacement. Since the tail bound (3.1) remains valid for the binomial distribution [Chv79, Hoe63], Lemma 4.2 is not affected. However, when Step 4 no longer skips comparisons with the elements of \( S_{-s} \) in (4.15) is replaced by 0; the resulting change in the bound on \( C_{nk} \) only needs replacing \( 2\bar{\alpha} \) in (4.8) by \( 3\bar{\alpha} \). Hence the preceding results remain valid.

(b) Of course, sampling with replacement needs additional storage for \( S \). However, the increase in both storage and the number of comparisons may be tolerated because the sample sizes are relatively small.

4.3 Handling small subfiles

Since the sampling efficiency decreases when \( X \) shrinks, consider the following modification. For a fixed cut-off parameter \( n_{\text{cut}} \geq 1 \), let \( s\text{Select}(X, k) \) be a “small-select” routine that finds the \( k \)th smallest element of \( X \) in at most \( C_{\text{cut}} < \infty \) comparisons when \( |X| \leq n_{\text{cut}} \) (even bubble sort will do). Then \textsc{Select} is modified to start with the following

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

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

Another advantage is that even small \( n_{\text{cut}} \) (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 \( n_{\text{cut}} \). For example, for the choice of (4.1) with \( \alpha = 1 \) and \( n_{\text{cut}} = 600 \), at most four recursive levels occur for \( n \leq 2^{31} \approx 2.15 \cdot 10^9 \).

5 A modified version

We now consider a modification inspired by a remark of [Bro76]. For \( k \) close to \( \lfloor n/2 \rfloor \), by symmetry it is best to choose \( v \) as the sample median with \( i_v = \lfloor s/2 \rfloor \), thus attempting to get \( v \) close to \( x^*_k \) instead of \( x^*_v \) or \( x^*_v \); then more elements are eliminated.
Hence we may let
\[ i_v := \begin{cases} 
[k_s/n + g] & \text{if } k < n/2 - gn/s, \\
[s/2] & \text{if } n/2 - gn/s \leq k \leq n/2 + gn/s, \\
[k_s/n - g] & \text{if } k > n/2 + gn/s.
\end{cases} \quad (5.1) \]

Note that (5.1) coincides with (2.11) in the left case of \( k < n/2 - gn/s \) and the right case of \( k > n/2 + gn/s \), but the middle case of \( n/2 - gn/s \leq k \leq n/2 + gn/s \) fixes \( i_v \) at the median position \( \lceil s/2 \rceil \); in fact \( i_v \) is the median of the three values in (5.1):
\[ i_v := \max \{ \min \{ [k_s/n + g], [s/2] \}, [k_s/n - g] \} . \quad (5.2) \]

Corollary 3.3 remains valid for the left and right cases. For the middle case, letting
\[ j_i := \max \{ [n/2 - gn/s], 1 \} \quad \text{and} \quad j_r := \min \{ [n/2 + gn/s], n \}, \quad (5.3) \]
we obtain from Lemma 3.2 with \( \kappa = 1/2 \) the following complement of Corollary 3.3.

**Corollary 5.1.** \( P[v < x_{j_i}^*] \leq e^{-2g^2/s} \) and \( P[x_{j_r}^* < v] \leq e^{-2g^2/s} \) if \( n/2 - gn/s \leq k \leq n/2 + gn/s \).

**Theorem 5.2.** Theorem 4.1 holds for SELECT with Step 3 using (5.1).

**Proof.** We only indicate how to adapt the proof of Thm 4.1 following (4.8). As noted after (4.1), the left case now has \( k < n/2 - gn/s \) and the right case has \( k > n/2 + gn/s \), so we only need to discuss the middle case.

**Middle case:** \( n/2 - gn/s \leq k \leq n/2 + gn/s \). Suppose the event \( \mathcal{E}_m := \{ x_{j_i}^* \leq v \leq x_{j_r}^* \} \) occurs (note that \( P[\mathcal{E}_m^c] \leq 2e^{-2g^2/s} = 2n^{-2\beta} \) by Cor. 5.1). If \( X = L \), then, by the rules of Steps 4–6, we have \( k = k \) and \( \hat{n} \leq j_r - 1 \); since \( j_r < n/2 + gn/s + 1 \) by (5.3), we get \( \hat{n} < n/2 + gn/s \). Hence \( k \geq n/2 - gn/s \) yields \( \hat{n} < k + 2gn/s \) and \( \hat{n} - k < 2gn/s \) as in (4.9). Next, if \( X = R \) then \( \hat{n} - \hat{k} = n - k \) and \( \hat{k} := k - |L \cup E| \), so \( L \cup E = \{ x \in X : x \leq v \} \supseteq x_{j_i}^* \); gives \( \hat{k} \leq k - j_i \). Since \( k \leq n/2 + gn/s \) and \( j_i \leq n/2 - gn/s \) by (5.3), we get \( \hat{k} \leq 2gn/s \) and \( \hat{n} - \hat{k} = n + 2gn/s \) in (4.11); further, \( \hat{n} - j_i \) yields \( \hat{n} \leq n + 2gn/s \). Noticing that \( n - gn/s \leq k \leq n/2 + gn/s \) implies \( n/2 \leq \min\{k, n - k\} + gn/s \), we have \( \hat{n} \leq \min\{k, n - k\} + 2gn/s \) in both cases.

Thus in the middle case we again have (4.13) and hence (4.15); further, by (4.10) and (4.12), the event \( \mathcal{E}_m \subseteq \mathcal{A}_t \cup \mathcal{A}_r \) is partitioned into \( \mathcal{E}_m \cap \mathcal{A}_t \) and \( \mathcal{E}_m \cap \mathcal{A}_r \).

Next, reasoning as before, we see that (4.18) and hence (4.19)–(4.20) remain valid in the left and right cases, whereas in the middle case we have
\[ P[\hat{\mathcal{B}} | \mathcal{E}_m, \mathcal{A}_t, \hat{n} = \hat{i}] \leq 2i^{-2\beta} \quad \text{and} \quad P[\hat{\mathcal{B}} | \mathcal{E}_m, \mathcal{A}_r, \hat{n} = \hat{i}] \leq 2i^{-2\beta} . \quad (5.4) \]

In the middle case, \( \hat{E} \hat{n} = E[\hat{n} | \mathcal{E}_m] \hat{P}[\mathcal{E}_m] + E[\hat{n} | \mathcal{E}_m^c] \hat{P}[\mathcal{E}_m^c] \) is bounded by \( E[\hat{n} | \mathcal{E}_m] + 2n^{-1-2\beta} \), since \( \hat{P}[\mathcal{E}_m^c] \leq 2n^{-2\beta} \) and \( \hat{n} < n \) always. Next, partitioning \( \mathcal{E}_m \) into \( \mathcal{E}_m \cap \mathcal{A}_t \) and \( \mathcal{E}_m \cap \mathcal{A}_r \), we obtain \( E[\hat{n} | \mathcal{E}_m] \leq \max\{E[\hat{n} | \mathcal{E}_m, \mathcal{A}_t], E[\hat{n} | \mathcal{E}_m, \mathcal{A}_r]\} \), where \( E[\hat{n} | \mathcal{E}_m, \mathcal{A}_t] \) and \( E[\hat{n} | \mathcal{E}_m, \mathcal{A}_r] \) may be bounded like \( E[\hat{n} | \mathcal{A}_t] \) and \( E[\hat{n} | \mathcal{A}_r] \) in the left and right cases to get (4.19). Then (4.20) is obtained similarly, and the conclusion follows as before. □
6 Ternary partitions

In this section we discuss ways of implementing \textsc{Select} when the input set is given as an array \(x[1:n]\). We employ the following notation.

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

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 \(0 \leq i < d\), or \(x_{a+i} \leftrightarrow x_{c-d+1+i}\) for \(0 \leq i < d\).

6.1 Tripartitioning schemes

For a given pivot \(v := x_l\) from the array \(x[l]\), the following ternary scheme \cite[§5.1]{Kiw03a} partitions the array into three blocks, with \(x_m < v\) for \(l \leq m < a\), \(x_m = v\) for \(a \leq m \leq b\), \(x_m > v\) for \(b < m \leq r\). After comparing the pivot \(v\) to \(x_r\) to produce the initial setup

\[
\begin{array}{cccccc}
  x = v & x < v & ? & x > v & x = v \\
  l & p & i & j & q & r \\
\end{array}
\]

(6.1)

with \(i := l\) and \(j := r\), we work with the three inner blocks of the array

\[
\begin{array}{cccccc}
  x = v & x < v & ? & x > v & x = v \\
  l & p & i & j & q & r \\
\end{array}
\]

(6.2)

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

\[
\begin{array}{cccccc}
  x = v & x < v & x = v & x > v & x = v \\
  l & p & j & i & q & r \\
\end{array}
\]

(6.3)

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

\[
\begin{array}{cccc}
  x < v & x = v & x > v \\
  l & a & b & r \\
\end{array}
\]

(6.4)

\textbf{Scheme A (Safeguarded ternary partition).}

\textbf{A1.} [Initialize.] Set \(i := l\), \(p := i+1\), \(j := r\) and \(q := j-1\). If \(v > x_j\), exchange \(x_i \leftrightarrow x_j\) and set \(p := i\); else if \(v < x_j\), set \(q := j\).

\textbf{A2.} [Increase \(i\) until \(x_i \geq v\).] Increase \(i\) by 1; then if \(x_i < v\), repeat this step.

\textbf{A3.} [Decrease \(j\) until \(x_j \leq v\).] Decrease \(j\) by 1; then if \(x_j > v\), repeat this step.
Scheme B (Double-index controlled ternary partition).

B1. [Initialize.] Set $i := p := l + 1$ and $j := q := r$.

B2. [Increase $i$ until $x_i > v$.] If $i \leq j$ and $x_i < v$, increase $i$ by 1 and repeat this step. If $i < j$ and $x_i = v$, exchange $x_i$ with the pivot $v$, increase $p$ and $i$ by 1, and repeat this step.

B3. [Decrease $j$ until $x_j < v$.] If $i < j$ and $x_j > v$, decrease $j$ by 1 and repeat this step. If $i < j$ and $x_j = v$, exchange $x_j$ with the pivot $v$, decrease $j$ and $q$ by 1, and repeat this step. If $i \geq j$, set $j := i - 1$ and go to B3.

B4. [Exchange.] Exchange $x_i \leftrightarrow x_j$ until $i > 1$, decrease $j$ by 1, and return to B2.

B5. [Cleanup.] Set $a := l + j - p + 1$ and $b := r - q + i - 1$. Exchange $x[l:p-1] \leftrightarrow x[p:j]$ and $x[i:q] \leftrightarrow x[q+1:r]$.

Step A1 ensures that $x_l \leq v \leq x_r$, so steps A2 and A3 don't need to test whether $i \leq j$. This scheme makes two extraneous comparisons (only one when $i = j$ at A4). Spurious comparisons are avoided in the following modification [Kiw03a §5.3] of the scheme of BeM93 (cf. Knu98 Ex. 5.2.2–41), for which $i = j + 1$ in (6.3).

6.2 Preparing for ternary 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 $x_i \leftrightarrow x_{i+\text{rand}(r-i)}$ for $l \leq i \leq r_s := l + s - 1$, where $\text{rand}(r-i)$ denotes a random integer, uniformly distributed between 0 and $r - i$.

Step 3 uses $i := k - l + 1$ and $m := r - l + 1$ instead of $k$ and $n$ to find the pivot position

$$k_v := \begin{cases} \min \{ [l - 1 + is/m + g], r_s \} & \text{if } i < m/2, \\ \max \{ [l - 1 + is/m - g], l \} & \text{if } i \geq m/2, \end{cases} \quad (6.5)$$

so that the recursive call of $\text{SELECT}(x, l, r, k_v, k_v^-, k_v^+)$ produces $v := x_{k_v}$.

After $v$ has been found, our array looks as follows

$$\begin{array}{c|c|c|c|c} x < v & x = v & x > v & \text{?} \\
\hline l & k_v^- & k_v^+ & r_s & r \end{array} \quad (6.6)$$

Setting $\bar{l} := k_v^-$ and $\bar{r} := r - r_s + k_v^+$, we swap $x[k_v^+ + 1 : r_s] \leftrightarrow x[r_s + 1 : r]$ in (6.6) to get

$$\begin{array}{c|c|c|c|c} x < v & x = v & \text{?} & x > v \\
\hline l & \bar{l} & k_v^+ & \bar{r} & r \end{array} \quad (6.7)$$
If \( k^+= r_s \), we use scheme \( \mathbf{A} \) with \( l \) replaced by \( k^+ \) in \( \mathbf{A1} \) (cf. (6.1)) and by \( \bar{l} \) in \( \mathbf{A5} \) (cf. (6.3)); for \( k^+ < r_s \), we set \( i := k^+_w, p := i + 1, j := \bar{r} + 1, q := \bar{r}, \) omit \( \mathbf{A1} \) and replace \( l, r \) by \( l, \bar{r} \) in \( \mathbf{A5} \). Similarly, for scheme \( \mathbf{B} \) we replace \( l, r \) by \( k^+ \), \( \bar{r} \) in \( \mathbf{B1} \), and by \( \bar{l}, \bar{r} \) in \( \mathbf{B5} \).

After partitioning \( l \) and \( r \) are updated by setting \( l := b + 1 \) if \( a \leq k \), \( r := a - 1 \) if \( k \leq b \).

If \( l \geq r \), \texttt{Select} may return \( k^- := r_s \) if \( l = r \), \( k^- := r + 1 \) and \( k^+ := l - 1 \) if \( l > r \). Otherwise, instead of calling \texttt{Select} recursively, Step 6 may jump back to Step 1, or to Step 0 if \texttt{sSelect} is used (cf. §4.3).

A simple version of \texttt{sSelect} is obtained if Steps 2 and 3 choose \( v := x_k \) when \( r - l + 1 \leq n_{\text{cut}} \) (this choice of \cite{FLR75a} works well in practice, but more sophisticated pivots could be tried); then the ternary partitioning code can be used by \texttt{sSelect} as well.

7 Experimental results

7.1 Implemented algorithms

An implementation of \texttt{Select} was programmed in Fortran 77 and run on a notebook PC (Pentium 4M 2 GHz, 768 MB RAM) under MS Windows XP. The input set \( X \) was specified as a double precision array. For efficiency, the recursion was removed and small arrays with \( n \leq n_{\text{cut}} \) were handled as if Steps 2 and 3 chose \( v := x_k \); the resulting version of \texttt{sSelect} (cf. §§4.3 and 6.2) typically required less than 3.5\( n \) comparisons. The choice of (4.21) was employed, with the parameters \( \alpha = 0.5, \beta = 0.25 \) and \( n_{\text{cut}} = 600 \) as proposed in \cite{FLR75a}; future work should test other sample sizes and parameters.

7.2 Testing examples

As in \cite{Kiw03b}, we used minor modifications of the input sequences of \cite{Va00}:

- **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 \( 4\lceil \log_2 n \rceil \) through \( n/2 - 1 \) and \( n/2 + 4\lceil \log_2 n \rceil - 1 \) through \( n - 2 \).

For each input sequence, its (lower) median element was selected for \( k := \lceil n/2 \rceil \).
Table 7.1: Performance of SELECT on randomly generated inputs.

| Sequence   | Size | Time [msec] | Comparisons [n] | \( \gamma_{\text{avg}} \) | \( L_{\text{avg}} \) | \( P_{\text{avg}} \) | \( N_{\text{avg}} \) | \( p_{\text{avg}} \) | \( s_{\text{avg}} \) |
|------------|------|-------------|-----------------|-----------------|-------------|-------------|-------------|-------------|-------------|
|            |      | avg | max | min | avg | max | min | avg | min | avg | min | avg | min | avg | min | avg | min | avg | min |
| random     | 50K  | 2   | 10  | 0   | 1.66 | 1.77 | 1.61 | 1.74 | 1.65 | 0.46 | 0.55 | 8.33 | 2.59 |
|            | 100K | 3   | 10  | 0   | 1.63 | 1.71 | 1.55 | 1.76 | 1.63 | 0.60 | 0.69 | 7.58 | 2.12 |
|            | 500K | 13  | 20  | 10  | 1.56 | 1.61 | 1.54 | 1.36 | 1.56 | 0.67 | 0.74 | 8.05 | 1.19 |
|            | 1M   | 23  | 30  | 20  | 1.52 | 1.58 | 1.00 | 1.74 | 1.52 | 0.66 | 0.73 | 8.32 | 0.91 |
|            | 2M   | 46  | 51  | 40  | 1.54 | 1.56 | 1.52 | 1.22 | 1.54 | 0.75 | 0.82 | 8.38 | 0.72 |
|            | 4M   | 88  | 91  | 80  | 1.53 | 1.55 | 1.52 | 1.18 | 1.53 | 0.86 | 0.92 | 8.22 | 0.57 |
|            | 8M   | 172 | 181 | 160 | 1.52 | 1.53 | 1.51 | 1.13 | 1.52 | 0.92 | 0.98 | 8.54 | 0.44 |
|            | 16M  | 336 | 341 | 320 | 1.52 | 1.53 | 1.51 | 1.06 | 1.52 | 0.95 | 1.01 | 8.41 | 0.35 |
| onezero    | 50K  | 2   | 10  | 0   | 1.28 | 1.51 | 1.00 | 0.00 | 1.28 | 0.24 | 0.18 | 1.26 | 1.91 |
|            | 100K | 3   | 10  | 0   | 1.25 | 1.51 | 1.00 | 0.00 | 1.25 | 0.26 | 0.15 | 1.20 | 1.49 |
|            | 500K | 15  | 20  | 10  | 1.33 | 1.50 | 1.00 | 0.00 | 1.33 | 0.29 | 0.17 | 1.34 | 0.93 |
|            | 1M   | 30  | 41  | 20  | 1.33 | 1.50 | 1.00 | 0.00 | 1.33 | 0.27 | 0.15 | 1.20 | 0.73 |
|            | 2M   | 60  | 71  | 41  | 1.30 | 1.50 | 1.00 | 0.00 | 1.30 | 0.26 | 0.14 | 1.29 | 0.56 |
|            | 4M   | 109 | 131 | 90  | 1.20 | 1.50 | 1.00 | 0.00 | 1.20 | 0.22 | 0.13 | 1.18 | 0.41 |
|            | 8M   | 219 | 261 | 190 | 1.20 | 1.50 | 1.00 | 0.00 | 1.20 | 0.22 | 0.13 | 1.31 | 0.32 |
|            | 16M  | 436 | 501 | 370 | 1.25 | 1.50 | 1.00 | 0.00 | 1.25 | 0.20 | 0.11 | 1.21 | 0.27 |
| twofaced   | 50K  | 1   | 10  | 0   | 1.67 | 1.77 | 1.59 | 1.87 | 1.67 | 0.47 | 0.56 | 8.24 | 2.63 |
|            | 100K | 3   | 11  | 0   | 1.62 | 1.73 | 1.56 | 1.67 | 1.62 | 0.60 | 0.69 | 7.61 | 2.11 |
|            | 500K | 12  | 20  | 10  | 1.56 | 1.59 | 1.53 | 1.23 | 1.56 | 0.63 | 0.71 | 8.33 | 1.18 |
|            | 1M   | 24  | 31  | 20  | 1.55 | 1.57 | 1.53 | 1.23 | 1.55 | 0.69 | 0.76 | 8.22 | 0.92 |
|            | 2M   | 45  | 51  | 40  | 1.54 | 1.57 | 1.52 | 1.23 | 1.54 | 0.78 | 0.85 | 8.36 | 0.73 |
|            | 4M   | 88  | 91  | 80  | 1.53 | 1.54 | 1.52 | 1.17 | 1.53 | 0.88 | 0.94 | 8.05 | 0.57 |
|            | 8M   | 170 | 180 | 160 | 1.52 | 1.53 | 1.51 | 1.12 | 1.52 | 0.90 | 0.97 | 8.51 | 0.44 |
|            | 16M  | 332 | 341 | 320 | 1.52 | 1.53 | 1.51 | 1.04 | 1.52 | 0.96 | 1.02 | 8.55 | 0.35 |

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 \( C_{\text{avg}}/n \), where \( C_{\text{avg}} \) is the average number of comparisons made over all instances. Thus \( \gamma_{\text{avg}} := (C_{\text{avg}} - 1.5n)/f(n) \) estimates the constant \( \gamma \) in the bound (4.2); moreover, we have \( C_{\text{avg}} \approx L_{\text{avg}} \), where \( L_{\text{avg}} \) is the average sum of sizes of partitioned arrays. Further, \( P_{\text{avg}} \) is the average number of SELECT partitions, whereas \( N_{\text{avg}} \) is the average number of calls to sSelect and \( p_{\text{avg}} \) is the average number of sSelect partitions per call; both \( P_{\text{avg}} \) and \( N_{\text{avg}} \) grow slowly with \( \ln n \) (linearly on the onezero inputs). Finally, \( s_{\text{avg}} \) is the average sum of sample sizes; \( s_{\text{avg}}/n^{2/3} \) drops from 0.95 for \( n = 50K \) to 0.88 for \( n = 16M \) on the random and twofaced inputs, and oscillates about 0.7 on the onezero inputs, whereas the initial \( s/n^{2/3} \approx \alpha = 0.5 \). The results for the random and twofaced sequences are very similar: the average solution times grow linearly with \( n \) (except for small inputs whose solution times couldn’t be
Table 7.2: Performance of SELECT on deterministic inputs.

| Sequence | Size | Time [msec] | Comparisons [n] | $\gamma_{avg}$ | $L_{avg}$ | $P_{avg}$ | $N_{avg}$ | $p_{avg}$ | $s_{avg}$ |
|----------|------|-------------|-----------------|---------------|-------|-------|-------|-------|-------|
|          | n   | avg | max | min | avg | max | min |       |       |       |
| sorted   | 50K | 1   | 10  | 0   | 1.67 | 1.76 | 1.59 | 1.85 | 1.66 | 0.48 | 0.57 | 7.24 | 2.65 |
|          | 100K| 2   | 10  | 0   | 1.62 | 1.69 | 1.55 | 1.70 | 1.62 | 0.60 | 0.69 | 6.76 | 2.12 |
|          | 500K| 8   | 10  | 0   | 1.56 | 1.62 | 1.53 | 1.35 | 1.56 | 0.67 | 0.74 | 7.52 | 1.19 |
|          | 1M  | 15  | 20  | 10  | 1.54 | 1.58 | 1.53 | 1.19 | 1.54 | 0.68 | 0.75 | 7.87 | 0.92 |
|          | 2M  | 27  | 31  | 20  | 1.54 | 1.56 | 1.52 | 1.23 | 1.54 | 0.74 | 0.81 | 7.61 | 0.73 |
|          | 4M  | 51  | 61  | 40  | 1.53 | 1.55 | 1.52 | 1.19 | 1.53 | 0.87 | 0.93 | 7.34 | 0.57 |
|          | 8M  | 98  | 111 | 90  | 1.52 | 1.53 | 1.51 | 1.10 | 1.52 | 0.89 | 0.95 | 8.03 | 0.44 |
|          | 16M | 186 | 200 | 170 | 1.52 | 1.52 | 1.51 | 1.04 | 1.52 | 0.95 | 1.01 | 7.99 | 0.35 |
| rotated  | 50K | 1   | 10  | 0   | 1.67 | 1.78 | 1.59 | 1.86 | 1.66 | 0.48 | 0.57 | 9.45 | 2.64 |
|          | 100K| 2   | 10  | 0   | 1.63 | 1.73 | 1.58 | 1.76 | 1.63 | 0.61 | 0.69 | 9.12 | 2.12 |
|          | 500K| 8   | 10  | 0   | 1.56 | 1.62 | 1.54 | 1.39 | 1.56 | 0.65 | 0.73 | 10.03 | 1.18 |
|          | 1M  | 15  | 20  | 10  | 1.55 | 1.58 | 1.53 | 1.29 | 1.55 | 0.69 | 0.76 | 9.56 | 0.92 |
|          | 2M  | 27  | 31  | 20  | 1.54 | 1.55 | 1.52 | 1.19 | 1.54 | 0.78 | 0.84 | 8.69 | 0.72 |
|          | 4M  | 51  | 60  | 50  | 1.53 | 1.54 | 1.52 | 1.18 | 1.53 | 0.87 | 0.94 | 8.92 | 0.57 |
|          | 8M  | 98  | 111 | 90  | 1.52 | 1.53 | 1.51 | 1.12 | 1.52 | 0.89 | 0.96 | 9.29 | 0.44 |
|          | 16M | 185 | 210 | 170 | 1.52 | 1.53 | 1.51 | 1.04 | 1.52 | 0.93 | 0.99 | 8.96 | 0.35 |
| organpipe| 50K | 1   | 10  | 0   | 1.67 | 1.78 | 1.59 | 1.94 | 1.67 | 0.45 | 0.55 | 8.21 | 2.62 |
|          | 100K| 3   | 10  | 0   | 1.62 | 1.69 | 1.57 | 1.68 | 1.62 | 0.60 | 0.69 | 7.61 | 2.11 |
|          | 500K| 10  | 10  | 10  | 1.57 | 1.60 | 1.54 | 1.43 | 1.56 | 0.67 | 0.75 | 8.18 | 1.19 |
|          | 1M  | 20  | 20  | 10  | 1.55 | 1.58 | 1.52 | 1.24 | 1.55 | 0.70 | 0.77 | 8.21 | 0.93 |
|          | 2M  | 37  | 41  | 30  | 1.53 | 1.55 | 1.52 | 1.15 | 1.53 | 0.78 | 0.85 | 8.48 | 0.72 |
|          | 4M  | 68  | 80  | 60  | 1.53 | 1.54 | 1.52 | 1.13 | 1.53 | 0.84 | 0.91 | 8.21 | 0.57 |
|          | 8M  | 130 | 140 | 120 | 1.52 | 1.54 | 1.51 | 1.07 | 1.52 | 0.88 | 0.94 | 8.64 | 0.44 |
|          | 16M | 240 | 260 | 230 | 1.52 | 1.53 | 1.51 | 1.02 | 1.52 | 0.94 | 1.00 | 8.44 | 0.35 |
| m3killer | 50K | 1   | 10  | 0   | 1.67 | 1.76 | 1.60 | 1.89 | 1.67 | 0.47 | 0.55 | 8.82 | 2.62 |
|          | 100K| 4   | 10  | 0   | 1.63 | 1.71 | 1.57 | 1.80 | 1.63 | 0.60 | 0.69 | 7.69 | 2.13 |
|          | 500K| 11  | 20  | 10  | 1.57 | 1.62 | 1.53 | 1.44 | 1.57 | 0.66 | 0.73 | 8.61 | 1.19 |
|          | 1M  | 20  | 20  | 10  | 1.55 | 1.59 | 1.52 | 1.40 | 1.55 | 0.72 | 0.79 | 8.33 | 0.93 |
|          | 2M  | 38  | 41  | 30  | 1.54 | 1.56 | 1.52 | 1.25 | 1.54 | 0.78 | 0.85 | 8.30 | 0.73 |
|          | 4M  | 73  | 81  | 70  | 1.53 | 1.54 | 1.52 | 1.28 | 1.53 | 0.87 | 0.94 | 8.22 | 0.57 |
|          | 8M  | 137 | 150 | 130 | 1.52 | 1.53 | 1.51 | 1.05 | 1.52 | 0.91 | 0.97 | 8.37 | 0.44 |
|          | 16M | 248 | 260 | 230 | 1.52 | 1.52 | 1.51 | 0.96 | 1.52 | 0.92 | 0.97 | 8.42 | 0.35 |

measured accurately), and the differences between maximum and minimum times are quite 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 $\gamma_{avg}$. The results for the onezero inputs essentially average two cases: the first pass eliminates either almost all or about half of the elements.

Table 7.2 exhibits similar features of SELECT on the deterministic inputs. The results for the sorted and rotated sequences are very similar, whereas the solution times on the organpipe and m3killer sequences are between those for the sorted and random sequences. The results of Tabs. 7.1–7.2 were obtained with scheme A of §6.2; to save space, Table 7.3 gives only selected results for scheme B, whereas Table 7.3 presents results for the
Table 7.3: Performance of SELECT with ternary scheme B.

| Sequence | Size | Time [msec] | Comparisons [n] | $\gamma_{avg}$ | $L_{avg}$ | $P_{avg}$ | $N_{avg}$ | $p_{avg}$ | $s_{avg}$ |
|----------|------|-------------|-----------------|----------------|-----------|-----------|-----------|-----------|-----------|
|          | n    | avg | max | min | avg | max | min | [n] | [ln n] | [ln n] | [% n] |
| random   | 2M   | 43  | 51  | 40  | 1.53 | 1.54 | 1.52 | 1.02 | 1.53 | 0.76 | 0.83 | 8.31 | 0.72 |
|          | 4M   | 93  | 101 | 90  | 1.53 | 1.55 | 1.52 | 1.09 | 1.53 | 0.85 | 0.92 | 8.42 | 0.57 |
|          | 8M   | 177 | 190 | 170 | 1.52 | 1.54 | 1.51 | 1.03 | 1.52 | 0.87 | 0.93 | 8.15 | 0.44 |
|          | 16M  | 343 | 350 | 340 | 1.51 | 1.53 | 1.51 | 0.88 | 1.51 | 0.91 | 0.97 | 8.50 | 0.35 |
| onezero  | 2M   | 82  | 91  | 70  | 1.30 | 1.50 | 1.00 | 0.00 | 1.30 | 0.26 | 0.14 | 1.29 | 0.56 |
|          | 4M   | 149 | 180 | 130 | 1.20 | 1.50 | 1.00 | 0.00 | 1.20 | 0.22 | 0.13 | 1.18 | 0.41 |
|          | 8M   | 304 | 351 | 270 | 1.20 | 1.50 | 1.00 | 0.00 | 1.20 | 0.22 | 0.13 | 1.31 | 0.32 |
|          | 16M  | 621 | 711 | 531 | 1.25 | 1.50 | 1.00 | 0.00 | 1.25 | 0.20 | 0.11 | 1.21 | 0.27 |
| sorted   | 2M   | 23  | 30  | 20  | 1.54 | 1.55 | 1.52 | 1.18 | 1.54 | 0.78 | 0.85 | 7.61 | 0.72 |
|          | 4M   | 43  | 50  | 40  | 1.53 | 1.54 | 1.51 | 1.18 | 1.53 | 0.86 | 0.92 | 7.76 | 0.57 |
|          | 8M   | 82  | 90  | 80  | 1.52 | 1.53 | 1.51 | 1.10 | 1.52 | 0.89 | 0.95 | 8.01 | 0.44 |
|          | 16M  | 156 | 160 | 150 | 1.52 | 1.53 | 1.51 | 1.04 | 1.52 | 0.97 | 1.03 | 8.12 | 0.35 |

The hybrid scheme I of [Kiw03a, §5.6], which combines some features of schemes A and B. The hybrid scheme is quite competitive, although slower than scheme A on the onezero inputs.

The preceding results were obtained with the modified choice (5.1) of $i_v$. For brevity, Table 7.5 gives results for SELECT with scheme A and the standard choice (2.1) of $i_v$ on the random inputs only, since these inputs are most frequently used in theory and practice for evaluating sorting and selection methods. The modified choice typically requires fewer comparisons for small inputs, but its advantages are less pronounced for larger inputs. A similar behavior was observed for SELECT with scheme B.

For comparison, Table 7.6 extracts from [Kiw03b] some results of qSELECT for the samples (4.1). As noted in §1, qSELECT is slightly faster than SELECT on larger inputs because most of its work occurs on the first partition (cf. $L_{avg}$ in Tabs. 7.1 and 7.6). In Table 7.7 we give corresponding results for riSELECT, a Fortran version of the algorithm of Val00. For these inputs, riSELECT behaves like FIND with median-of-3 pivots (because the average numbers of randomization steps, $N_{rnd}$, are negligible); hence the expected value of $C_{avg}$ is of order 2.75$n$ [KMP97].

Our final Table 7.8 shows that SELECT beats its competitors with respect to the numbers of comparisons made on small random inputs (100 instances for each input size $n$).

Our computational results, combined with those in [Kiw03a, Kiw03b], suggest that both SELECT and qSELECT may compete with FIND in practice.

Acknowledgment. I would like to thank Olgierd Hryniewicz, Roger Koenker, Ronald L. Rivest and John D. Valois for useful discussions.

References

[BEM93] J. L. Bentley and M. D. McIlroy, Engineering a sort function, Software–Practice and Experience 23 (1993) 1249–1265.

[BFPr72] 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.
Table 7.4: Performance of SELECT with the hybrid scheme of \[\text{Kiw03a} \; \S 5.6].

| Sequence | Size | Time [msec] | Comparisons [n] | \(\gamma_{avg}\) | \(L_{avg}\) [n] | \(P_{avg}\) [ln n] | \(N_{avg}\) [ln n] | \(p_{avg}\) | \(s_{avg}\) [%] |
|----------|------|-------------|-----------------|-----------------|----------------|----------------|----------------|--------|-----------|
| random   | 2M   | 44          | 50              | 1.53            | 1.54           | 1.52           | 1.03           | 1.33   | 0.76      | 0.83     | 8.31     | 0.72     |
|          | 4M   | 86          | 100             | 1.53            | 1.55           | 1.52           | 1.10           | 1.53   | 0.85      | 0.92     | 8.42     | 0.57     |
|          | 8M   | 163         | 171             | 1.52            | 1.54           | 1.51           | 1.03           | 1.52   | 0.87      | 0.93     | 8.15     | 0.44     |
|          | 16M  | 317         | 321             | 1.51            | 1.53           | 1.51           | 0.88           | 1.51   | 0.91      | 0.97     | 8.50     | 0.35     |
| onezero  | 2M   | 74          | 80              | 1.30            | 1.50           | 1.00           | 0.00           | 1.30   | 0.26      | 0.14     | 1.29     | 0.56     |
|          | 4M   | 141         | 151             | 1.20            | 1.50           | 1.00           | 0.00           | 1.20   | 0.22      | 0.13     | 1.18     | 0.41     |
|          | 8M   | 285         | 301             | 1.20            | 1.50           | 1.00           | 0.00           | 1.20   | 0.22      | 0.13     | 1.31     | 0.32     |
|          | 16M  | 578         | 621             | 1.25            | 1.50           | 1.00           | 0.00           | 1.25   | 0.20      | 0.11     | 1.21     | 0.27     |
| sorted   | 2M   | 23          | 30              | 1.54            | 1.55           | 1.52           | 1.18           | 1.54   | 0.78      | 0.85     | 7.61     | 0.72     |
|          | 4M   | 42          | 50              | 1.53            | 1.54           | 1.51           | 1.19           | 1.53   | 0.86      | 0.92     | 7.76     | 0.57     |
|          | 8M   | 80          | 80              | 1.52            | 1.53           | 1.51           | 1.11           | 1.52   | 0.89      | 0.95     | 8.01     | 0.44     |
|          | 16M  | 153         | 170             | 1.52            | 1.53           | 1.51           | 1.04           | 1.52   | 0.97      | 1.03     | 8.12     | 0.35     |

Table 7.5: Performance of SELECT with the standard choice of \(i_v\).

| Sequence | Size (n) | Time [msec] | Comparisons [n] | \(\gamma_{avg}\) | \(L_{avg}\) [n] | \(P_{avg}\) [ln n] | \(N_{avg}\) [ln n] | \(p_{avg}\) | \(s_{avg}\) [%] |
|----------|----------|-------------|-----------------|-----------------|----------------|----------------|----------------|--------|-----------|
| random   | 50K      | 4           | 10              | 1.83            | 1.97           | 1.74           | 3.73           | 1.83   | 0.57      | 0.67     | 8.49     | 2.96     |
|          | 100K     | 4           | 10              | 1.73            | 1.83           | 1.61           | 3.13           | 1.73   | 0.73      | 0.82     | 7.80     | 2.32     |
|          | 500K     | 14          | 20              | 1.65            | 1.69           | 1.61           | 3.25           | 1.65   | 0.82      | 0.90     | 8.40     | 1.30     |
|          | 1M       | 25          | 30              | 1.61            | 1.65           | 1.58           | 2.83           | 1.60   | 0.89      | 0.97     | 8.28     | 0.99     |
|          | 2M       | 46          | 50              | 1.59            | 1.61           | 1.56           | 2.92           | 1.59   | 0.99      | 1.06     | 8.01     | 0.77     |
|          | 4M       | 90          | 100             | 1.56            | 1.58           | 1.54           | 2.61           | 1.56   | 1.15      | 1.22     | 8.34     | 0.60     |
|          | 8M       | 174         | 181             | 1.55            | 1.57           | 1.54           | 2.70           | 1.55   | 1.21      | 1.27     | 8.09     | 0.47     |
|          | 16M      | 341         | 351             | 1.54            | 1.56           | 1.53           | 2.68           | 1.54   | 1.21      | 1.28     | 8.33     | 0.36     |

Table 7.6: Performance of quintary QSELECT on random inputs.

| Sequence | Size (n) | Time [msec] | Comparisons [n] | \(\gamma_{avg}\) | \(L_{avg}\) [n] | \(P_{avg}\) [ln n] | \(N_{avg}\) [ln n] | \(p_{avg}\) | \(s_{avg}\) [%] |
|----------|----------|-------------|-----------------|-----------------|----------------|----------------|----------------|--------|-----------|
| random   | 50K      | 3           | 10              | 1.81            | 1.85           | 1.77           | 5.23           | 1.22   | 0.46      | 1.01     | 7.62     | 4.11     |
|          | 100K     | 4           | 10              | 1.72            | 1.76           | 1.65           | 4.50           | 1.15   | 0.45      | 0.99     | 8.05     | 3.20     |
|          | 500K     | 14          | 20              | 1.62            | 1.63           | 1.60           | 4.14           | 1.08   | 0.59      | 1.27     | 7.59     | 1.86     |
|          | 1M       | 24          | 30              | 1.59            | 1.60           | 1.57           | 3.93           | 1.06   | 0.64      | 1.35     | 8.18     | 1.47     |
|          | 2M       | 46          | 50              | 1.57            | 1.58           | 1.56           | 3.73           | 1.04   | 0.76      | 1.59     | 7.67     | 1.16     |
|          | 4M       | 86          | 91              | 1.56            | 1.56           | 1.55           | 3.61           | 1.03   | 0.94      | 1.94     | 7.21     | 0.91     |
|          | 8M       | 163         | 171             | 1.54            | 1.55           | 1.54           | 3.45           | 1.03   | 0.98      | 1.99     | 7.45     | 0.72     |
|          | 16M      | 316         | 321             | 1.53            | 1.54           | 1.53           | 3.44           | 1.02   | 0.99      | 2.02     | 7.55     | 0.57     |
Table 7.7: Performance of riSELECT on random inputs.

| Sequence | Size | Time [msec] | Comparisons [n] | \( L_{\text{avg}} \) | \( P_{\text{avg}} \) | \( N_{\text{rnd}} \) |
|----------|------|-------------|-----------------|-----------------|----------------|----------------|
| random   | 50K  | 2           | 10              | 3.10 4.32 1.88  | 3.10 1.63     | 0.45           |
|          | 100K | 4           | 10              | 2.61 4.19 1.77  | 2.61 1.60      | 0.20           |
|          | 500K | 17          | 20              | 2.91 4.45 1.69  | 2.91 1.57      | 0.25           |
|          | 1M   | 33          | 41              | 2.81 3.79 1.84  | 2.81 1.57      | 0.40           |
|          | 2M   | 62          | 90              | 2.60 3.57 1.83  | 2.60 1.61      | 0.35           |
|          | 4M   | 135         | 191             | 2.86 4.38 1.83  | 2.86 1.65      | 0.55           |
|          | 8M   | 249         | 321             | 2.60 3.48 1.80  | 2.60 1.58      | 0.40           |
|          | 16M  | 553         | 762             | 2.99 4.49 1.73  | 2.99 1.58      | 0.40           |

Table 7.8: Numbers of comparisons per element made on small random inputs.

| Size | 1000 | 2500 | 5000 | 7500 | 10000 | 12500 | 15000 | 17500 | 20000 | 25000 |
|------|------|------|------|------|-------|-------|-------|-------|-------|-------|
| \( \text{SELECT} \) | avg  | 2.48 | 2.06 | 1.93 | 1.87  | 1.81  | 1.79  | 1.77  | 1.76  | 1.74  | 1.71  |
|      | max  | 4.25 | 3.03 | 2.28 | 2.22  | 2.09  | 2.05  | 1.95  | 1.93  | 1.93  | 1.93  |
|      | min  | 1.55 | 1.96 | 1.03 | 1.64  | 1.62  | 1.61  | 1.64  | 1.63  | 1.59  | 1.60  |
| \( \text{QSELECT} \) | avg  | 2.86 | 2.55 | 2.24 | 2.16  | 2.07  | 2.03  | 1.98  | 1.98  | 1.94  | 1.90  |
|      | max  | 3.97 | 3.55 | 2.57 | 2.38  | 2.28  | 2.21  | 2.16  | 2.13  | 2.11  | 2.31  |
|      | min  | 2.29 | 1.97 | 1.98 | 1.95  | 1.87  | 1.86  | 1.82  | 1.83  | 1.82  | 1.75  |
| \( \text{riSELECT} \) | avg  | 2.72 | 2.65 | 2.66 | 2.71  | 2.72  | 2.83  | 2.78  | 2.75  | 2.75  | 2.84  |
|      | max  | 4.40 | 4.51 | 4.69 | 4.43  | 4.62  | 4.76  | 4.64  | 4.40  | 5.10  | 4.77  |
|      | min  | 1.68 | 1.83 | 1.75 | 1.59  | 1.70  | 1.77  | 1.78  | 1.67  | 1.90  | 1.71  |

[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.

[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.

[FIR75a] R. W. Floyd and R. L. Rivest, *The algorithm SELECT—for finding the \(i\)th smallest of \(n\) elements (Algorithm 489)*, Comm. ACM 18 (1975) 173.

[FIR75b] _____, *Expected time bounds for selection*, Comm. ACM 18 (1975) 165–172.

[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.

[Kiw03a] K. C. Kiwiel, *Partitioning schemes for quicksort and quickselect*, Tech. report, Systems Research Institute, Warsaw, 2003. Available at the URL [http://arxiv.org/abs/cs.DS/0312054](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](http://arxiv.org/abs/cs.DS/0312055).
[KMP97] P. Kirschenhofer, C. Martínez and H. Prodinger, *Analysis of Hoare’s find algorithm with median-of-three partition*, Random Structures and Algorithms 10 (1997) 143–156.

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

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

[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.

[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.