An Efficient Approximate Algorithm for the k-th Selection Problem

D. Cantone, G. Cincotti
Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I–95125 Catania, Italy
email: {cantone,cincotti}@cs.unict.it

and M. Hofri
Department of Computer Science, WPI
100 Institute Road, Worcester MA 01609-2280, USA
email: hofri@wpi.edu

Abstract

We present an efficient randomized algorithm for the approximate k-th selection problem. It works *in-place* and it is fast and easy to implement. The running time is linear in the length of the input.

For a large input set the algorithm returns, with high probability, an element which is very close to the exact k-th element. The quality of the approximation is analyzed theoretically and experimentally.

Keywords: Selection problem, Median finding, Approximation algorithms, In-place algorithms, Randomized algorithms, Analysis of algorithms.

1. Introduction

Given a multi-set S of n elements from some totally ordered universe and an integer $k \in \{1, \ldots, n\}$, the k-th selection problem consists in finding the k-th smallest element in S. The median selection is a special case of this problem when $k = \lceil \frac{n}{2} \rceil$.

The standard cost metric used to compare selection al;gorithms is the *comparison cost* model, in which the computational complexity of an algorithm is determined by the number of element comparisons. This resulted in the invention of some algorithms that use a small number of comparisons, and "pay" for it by a relatively large overhead of elemnt moves and exchanges. We shall cosider the effect of the total number of operations.

Early in the sixties, Hoare [8] devised an efficient algorithm for the k-th selection problem, namely "Quickselect", of great theoretical and practical importance. The algorithm takes the same approach as in "Quicksort," making a sequence of random choices of pivot elements, executing about $4n$ comparisons on the average. It has a the problem of having a cost in $\mathcal{O}(n^2)$ time in the worst-case.

Floyd and Rivest [7] improved the previous algorithm by choosing the pivot elements through an opportune sampling. They obtained a very practical algorithm requiring $n + \min(k, n-k) + o(n)$, i.e. at most $1.5n + o(n)$, comparisons on the average (cf. [5]).

The first discovered algorithm with linear worst-case cost was given by Blum et al. [2] in 1973, requiring only $5.43n + o(n)$ comparisons in the worst-case. It was an important complexity result because till then the selection problem was assumed to be as difficult as sorting. Incidentally, a lower bound of $2n$ comparisons (in the worst-case) for the median selection problem is due to Bent and John [3].

Three years later, Schönhage et al. [11] concentrated their attention on the median selection problem, devising an algorithm requiring at most $3n + o(n)$ comparisons. Such result was slightly improved in 1995 by Dor and Zwick [6] that obtained a $2.95n + o(n)$ upper bound modifing the algorithm by Schönhage et al.. These algorithms attempts to minimize the number of key comparisons but not the total number of operations, and hence do not lead to faster algorithms in practice.

All the worst-case linear time algorithms cited above require, apart from the space for the elements themselves, at least $\Theta(\log n)$ additional space during the computation. From this point of view, the research community showed interest in finding optimal *in-place* solutions, i.e. algorithms that require only a constant amount of extra space. The first in-place algorithm, due to Lai and Wood [9] in 1988, needs at most $6.83n + o(n)$ comparisons and it was obtained by an implicit emulation of the basic algorithm by Blum et al..

Based on similar ideas as the median selection algorithm by Schönhage et al., Carlsson and Sundstrom [4] in 1995 achieved the $(2.95 + \varepsilon)n + o(n)$ in-place upper bound, for any $\varepsilon > 0$. This is arbitrarily close to the Dor and Zwick's upper bound for the same problem, without space restrictions. It is to be noticed that these algorithms have mainly theoretical interest because their implementations are extremely complicated and not practical.

Selection arises in several applications others than order statistics. In most cases, the selected elements are used as threshold for filtering operations, because it is reasonable to concentrate attention only to the subset of the most important elements with respect to a suitable criterion.

For instance, a computer chess program could quickly evaluate all possible next moves, and then decide to further elaborate only the best p-percentage moves more carefully. Identifying such best moves can be done by a selection and a filtering operation.

Another explicit use of selection arises in filtering outlying elements. More specifically, in dealing with noisy statistical data samples, sometimes it is convenient to eliminate noise throwing out the largest and smallest p-percentage of them. Selection can be used to identify the items defining the p-th and $(100 - p)$-th percentiles and the outliers are then filtered out by comparing each item to the two selected elements.

A crucial observation for the current treatment is that many applications do not require the *exact* solution to an instance of the k-th selection problem, and an *approximate* solution would suffice. In such contexts it becomes important to deal with efficient approximate algorithms for the k-th selection problem.

An extremely efficient algorithm for the approximate median selection problem is described in [1]. The idea behind this algorithm is simple. To simplify the exposition, let the size of the input set be a power of 3, e.g. $n = 3^r$ with $r \in \mathbb{N}$. The algorithm then proceeds in r stages. At each stage it divides the surviving input into subsets of three elements, and calculates the median of each such triplet. The "local medians" survive to the next stage. The algorithm continues recursively, using the local results to compute the approximate median of the initial set. In [1] is also described how to handle efficiently input sizes that are not powers of 3.

In this paper, we present an efficient algorithm for the approximate k-th selection problem. It can be viewed as a clever generalization of the algorithm described in [1]. In particular, it works in-place and its precision is strongly based on the good behavior of the approximate median selection algorithm.

The paper is organized as follows. In Section 2, we first show that the k-th and the median selection problems are equivalent. An algorithm for the approximate k-th selection problem is described in Section 3, and one of its possible implementation is discussed in Section 4. Moreover, in Section 5 we prove the linear computational cost of the algorithm, whereas in Section 6 we develop a probabilistic analysis aimed to establish the quality of the algorithm output. Section 7 collects some experimental results. The last section concludes the paper with some final remarks.

2. Reducing the *k*-th selection to the median selection problem

In this section, we show that though the k-th selection is more general than the median selection problem they are equivalent. In particular, we describe how an instance of the k-th selection problem can be easily reduced to an instance of the median selection problem.

Let $S^{(n)}$ be a multi-set[1] of n elements from a totally ordered universe. In the following, we denote by $-\infty$ (resp. $+\infty$) an element of the universe smaller than the minimum (resp. greater than the maximum) of $S^{(n)}$.

Solving an instance $\langle S^{(n)} \rangle$ of the median selection problem means to find the median, i.e. the $\lceil \frac{n}{2} \rceil$-th, element out of the set $S^{(n)}$. In the more general k-th selection problem, solving an instance $\langle S^{(n)}, k \rangle$, with $k \in \{1, \ldots, n\}$, requires finding the k-th element out of $S^{(n)}$.

Reduction 1. *Given an instance $\langle S^{(n)}, k \rangle$ of the k-th selection problem, it can be reduced to an instance $\langle S_\infty^{(n,m)} \rangle$ of the median selection problem, where the multi-set*

$$S_\infty^{(n,m)} = S^{(n)} \cup \bigcup_{i=1}^{m} \{\infty\}, \qquad \text{with} \qquad m = |n - 2k + 1|,$$

is obtained from $S^{(n)}$ by adding m copies of the ∞ value defined as follows:

$$\infty = \begin{cases} -\infty & \text{if } k \leq \lceil \frac{n}{2} \rceil, \\ +\infty & \text{otherwise}. \end{cases}$$

Obviously, we have $0 \leq m < n$ with $|S_\infty^{(n,m)}| = n + m$, and the following:

Theorem 1. *The Reduction 1 from the k-th selection to the median selection problem is correct.*

<u>*Proof.*</u> When $k \leq \lceil \frac{n}{2} \rceil$, we construct the multi-set $S_\infty^{(n,m)}$ obtained from $S^{(n)}$ by adding $m = n - 2k + 1 \geq 0$ copies of the element $-\infty$. It is immediate to recognize that the k-th element of $S^{(n)}$ is exactly the $m + k = (n - k + 1)$-st element of $S_\infty^{(n,m)}$, namely the median element of $S_\infty^{(n,m)}$, because $|S_\infty^{(n,m)}| = 2n - 2k + 1$ and the median is given by the $\left\lceil \frac{|S_\infty^{(n,m)}|}{2} \right\rceil = \lceil n - k + \frac{1}{2} \rceil = (n - k + 1)$-st element.

[1] A set that can contain multiple copies of the same element.

On the other hand, if $k > \left\lceil \frac{n}{2} \right\rceil$, then we construct the multi-set $S_\infty^{(n,m)}$ by adding $m = 2k - n - 1 > 0$ copies of the element $+\infty$. In this case, the k-th element of $S^{(n)}$ is exactly the k-th element of $S_\infty^{(n,m)}$ yet, namely the median element of $S_\infty^{(n,m)}$, because $|S_\infty^{(n,m)}| = 2k-1$ and the median is given by the $\left\lceil \frac{|S_\infty^{(n,m)}|}{2} \right\rceil = \left\lceil k - \frac{1}{2} \right\rceil = k$-th element. Thus, in both cases the problem of selecting the k-th element of $S^{(n)}$ has been reduced to the one of selecting the median of $S_\infty^{(n,m)}$. ■

By virtue of the above reduction, any algorithm for the median selection can be used for solving the k-th selection problem. In particular, the reduction is useful for both the *exact* and the *approximate* variants of the k-th selection problem, and the latter case may be solved using the efficient and accurate algorithm for the approximate median selection presented in [1].

3. An probabilistic, approximate algorithm

The Reduction 1 described in Section 2 and the idea behind the approximate median selection algorithm are joined together, to get a randomized algorithm for the approximate k-th selection problem, that works *in-place*, without introducing any explicit infinite value.

In the following, we denote by $\mathbb{N}$ the set of cardinals, with $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, and by $\mathbb{Z}$ the set of relative integers. Moreover, we define $[n]_k \stackrel{\text{def}}{=} n \mod k$ for any $n, k \in \mathbb{N}$, thus $[n]_k \in \{0, 1, \ldots, k-1\}$.

Let $\langle S_\infty^{(n,m)} \rangle$ be the reduction of an instance $\langle S^{(n)}, k \rangle$ of the k-th selection problem with $n, m \in \mathbb{N}_0$ and $m < n$. For each $i = 0..3$, we calculate the probabilities:

$$p_i^{(n,m)} = \Pr \left\{ \begin{array}{c} \text{A random } \textit{unordered} \text{ triplet } t = \{x,y,z\} \in S_\infty^{(n,m)} \\ \text{contains } i \text{ infinite values} \end{array} \right\},$$

and find

$$p_0^{(n,m)} = \frac{\binom{n}{3}}{\binom{n+m}{3}}, \qquad p_1^{(n,m)} = \frac{m\binom{n}{2}}{\binom{n+m}{3}}, \qquad p_2^{(n,m)} = \frac{n\binom{m}{2}}{\binom{n+m}{3}}, \qquad p_3^{(n,m)} = \frac{\binom{m}{3}}{\binom{n+m}{3}},$$

where $\binom{n+m}{3}$ is the number of possible unordered triplets that can be selected from $S_\infty^{(n,m)}$. At this point, let the set $S_\infty^{(n,m)}$ be partitioned into $\frac{1}{3}(n+m)$ uniformly distributed triplets;[2] the expected number $\mu_i(n,m)$ of unordered triplets in $S_\infty^{(n,m)}$ containing i infinite values is given by:

$$\mu_i(n,m) = \frac{1}{3}(n+m)p_i^{(n,m)}, \qquad \text{for each} \quad i = 0..3\,.$$

[2] At this level of abstraction, it does not matter if $(n+m)$ is not a multiple of 3.

In particular, we obtain:

$$\mu_0(n,m) = \frac{n(n-1)(n-2)}{3(n+m-1)(n+m-2)}, \tag{1}$$

$$\mu_1(n,m) = \frac{mn(n-1)}{(n+m-1)(n+m-2)}, \tag{2}$$

$$\mu_2(n,m) = \frac{m(m-1)n}{(n+m-1)(n+m-2)}, \tag{3}$$

$$\mu_3(n,m) = \frac{m(m-1)(m-2)}{3(n+m-1)(n+m-2)}. \tag{4}$$

Because $\sum_{i=0}^{3} p_i^{(n,m)} = 1$, we get the equality $\sum_{i=0}^{3} \mu_i(n,m) = \frac{1}{3}(n+m)$ and, in particular, the fundamental system of identities:

$$\begin{cases} 3\mu_0(n,m) + 2\mu_1(n,m) + \mu_2(n,m) & = & n\,, \\ \mu_1(n,m) + 2\mu_2(n,m) + 3\mu_3(n,m) & = & m\,. \end{cases} \tag{5}$$

Equations (1) through 4 suggest a randomized algorithm for the approximate k-th selection problem where infinite values are only used for book-keeping, and for this reason we refer to them as *virtual infinities*. A detailed description of the algorithm follows.

1. Let $\langle S, k\rangle$ be the problem instance in input, with $n = |S|$ and $m = |n-2k+1|$. Furthermore, let the boolean variable *PositiveInfinities* be *True* whether $k > \lceil \frac{n}{2} \rceil$ or *False* otherwise.

2. Repeat the following steps until the cardinality n becomes smaller than a fixed *Threshold*:

   (a) Find integers $r, r_\infty \in \mathbb{N}_0$, by means of a predefined criterion, such that:

   $$r + r_\infty = [n+m]_3\,.$$

   These values represent the number of elements in S and the number of virtual infinities, respectively, that do not exactly fit in any triplet.

   (b) Compute values $\overline{\mu}_i \in \mathbb{N}_0$, with $i = 0..3$, such that $\overline{\mu}_i$ is a "good" integer approximation for $\mu_i(n,m)$ and the following integer system:

   $$\begin{cases} 3\overline{\mu}_0 + 2\overline{\mu}_1 + \overline{\mu}_2 & = & n - r\,, \\ \overline{\mu}_1 + 2\overline{\mu}_2 + 3\overline{\mu}_3 & = & m - r_\infty\,, \end{cases} \tag{6}$$

   is satisfied with such values.

   (c) Accordingly to the first equation of system (6), choose randomly in S: $\overline{\mu}_0$ triplets of elements, $\overline{\mu}_1$ couples of elements, and $\overline{\mu}_2 + r$ single elements.

   (d) Perform the following operations on the selected elements of S:

   - remove from S the minimal and maximal elements of each of the $\overline{\mu}_0$ triplets;
   - if *PositiveInfinities* is *True* then remove from S the minimal element of each of the $\overline{\mu}_1$ couples, otherwise remove the maximal element of each of them;

- remove from S each of the $\overline{\mu}_2$ singleton.

(e) Retain or remove the r elements of S with a predefined policy, and set n to the cardinality of the remaining set. Treat analogously the r_∞ virtual infinities, saving or removing them, i.e., respectively, letting $m = \overline{\mu}_2 + \overline{\mu}_3 + r_\infty$ or $m = \overline{\mu}_2 + \overline{\mu}_3$ (see below).

3. Return the exact median of the multi-set obtained from the remaining elements in S joined with m copies of the value $+\infty$ whether *PositiveInfinities* is *True*, or m copies of $-\infty$ otherwise.

Observe that, any reasonable criterion introduced in Step (2a) should ensure $r_\infty \leq r$ because we always have $m < n$. In particular, a satisfying criterion is obtained setting $r = [n+m]_3 - r_\infty$ and

$$r_\infty = \begin{cases} 1 & \text{if } [n+m]_3 = 2 \text{ and } m \geq \frac{3}{4}n\,, \\ 0 & \text{otherwise,} \end{cases}$$

where the value $\frac{3}{4}n$ is arbitrarily chosen and it means a large fraction of n.

Obviously, the system (6) has always integer solutions. Indeed, $\overline{\mu}_2, \overline{\mu}_3$ can be considered as free variables, and $\overline{\mu}_1$ can be obtained from the second equation; now, adding the two equations in (6), we obtain $3\sum_{i=0}^{3}\overline{\mu}_i = n+m-[n+m]_3$, and get $\overline{\mu}_0$ because $n+m-[n+m]_3$ is multiple of 3. Notice that the value $\sum_{i=0}^{3}\overline{\mu}_i$ is the number of *virtual triplets*, i.e. triplets in which virtual infinities can occur, processed in a single iteration.

Each operation listed in Step (2d) corresponds to the handling of virtual triplets containing, respectively, 0, 1 or 2 infinite values.

A good policy to treat the r elements in Step (2e) is addressed in [1]; these elements are processed joining them with one of the $\overline{\mu}_0$ triplets and their local median (i.e., of 3, 4, or 5 elements) survives to the next iteration. Such technique could also include the r_∞ virtual infinities.

Finally, the value m computed in Step (2e) is the number of virtual infinities that survive to the next iteration; specifically, such infinities come out from virtual triplets containing 2 or 3 infinite values. Analogously, in Step (2e) we have $n = \overline{\mu}_0 + \overline{\mu}_1 + r$ or $n = \overline{\mu}_0 + \overline{\mu}_1$ elements coming out from virtual triplets containing 0 or 1 infinite values.

Interval extremes adjustment. The extreme cases of selection, such as the smallest (i.e., $k \approx 1$) or the largest (i.e., $k \approx n$) elements are better treated *ad hoc*, modifying the quantities given in equations (1) through (4).

4. An implementation of the algorithm

In this section we give a possible implementation of the algorithm described above. It is assumed that the selection is from values stored in an array in main memory. In this case, the computations disturb the order of the elements, but the contents of the array is unchanged.

Let n be the size of the input array A, and k be the index of the element that we want to select, and let *Threshold* be an integer value. The algorithm pseudo-code is shown in Fig. 1; it provided the experimental results reported in Section 7.

For the sake ef brevity of this presentation, the value r_∞ introduced in Step (2a) is always set to zero, and the r elements treated in Step (2e) are simply thrown away. Asymptotically, these choices do not decrease the output quality in terms of precision, but it makes a noticeable difference in small arrays.

Pseudo-code of the approximate selection function

```
FUNCTION ApproximateSelection (A, n, k, Threshold) : integer
  This function returns the index of the approximate k-th element of the input array A[1,...,n].

  m = |n − 2k + 1|;
  if (k > ⌈n/2⌉)  then  PositiveInfinities = True;
                  else  PositiveInfinities = False;
  while (n > Threshold) do
      r = [n + m]_3;
      ⟨μ̄_0, μ̄_1, μ̄_2, μ̄_3⟩ = SystemSolve (n, m);
      while (r > 0) do                                       /* Eliminate r random */
          Swap (A[n], A[RandomInteger(1 . . n)] );           /* elements from consideration */
          n = n − 1;     r = r − 1;
      end while;
      i = 1;      j = 1;      m = μ̄_2 + μ̄_3;
      while (j < n) do
          Choice = RandomInteger (1 . . (μ̄_0 + μ̄_1 + μ̄_2) );
          if (Choice ≤ μ̄_2) then                              /*Delete the singleton*/
                  j = j + 1;     μ̄_2 = μ̄_2 − 1;
          elsif (Choice ≤ μ̄_1 + μ̄_2) then                      /* Treat the couple */
                  if (A[j] > A[j+1]) then Swap (A[j], A[j+1]);
                  if (PositiveInfinities = True)  then Swap (A[i], A[j+1]);
                                                  else Swap (A[i], A[j]);
                  j = j + 2;     i = i + 1;     μ̄_1 = μ̄_1 − 1;
          else                                                 /* Treat the triplet */
                  h = TripletMedian(A, j);
                  Swap (A[i], A[h]);
                  j = j + 3;     i = i + 1;     μ̄_0 = μ̄_0 − 1;
      end while;
      n = i − 1;
  end while;
  SelectionSort (A, n);
  if (PositiveInfinities = True) then              /* We consider the m residues virtual infinities */
      if (m ≥ n) then return n;
                 else return ⌈(m + n)/2⌉;
  else
      if (m ≥ n) then return 1;
                 else return ⌈(n − m)/2⌉;
```

Figure 1: Pseudo-code for the approximate k-th selection algorithm.

The values $\overline{\mu}_i$, with $i = 0..3$, required in Step (2b) to solve the integer system (6), are computed by the function *SystemSolve*; its pseudo-code is given in Fig. 2. In this function, the real variables x, y are initialized by means of functions (3) and (4), using the value $(n - r)$ because the r elements are removed at all. Furthermore, the best first choice for the couple in Γ is represented by $\langle \text{Round}(x), \text{Round}(y) \rangle$.

For convenience, the value m computed in Step (2e) is calculated before Step (2c).

Steps (2c) and (2d) are executed by scanning the array from left to right through the index j. For every single step of each scan, we flip a coin with three faces, i.e. the integer variable *Choice*, and depending on the result, we process one, two, or three consecutive elements of the array starting at position j. Notice that, so many calls to the random number generator can be avoided introducing further efficient pseudo-randomization methods as in the style of [1]. All the elements that survive to the next step are gathered on the leftmost portion of the input array through the index i. We show in Fig. 2 the function *TripletMedian* that finds the index of the median of the triplet formed by consecutive elements starting at position i of the array A.

The algorithm continues iteratively using the results of each stage as input for a new one. This is done until the number of the surviving elements falls below the *Threshold* value. Then, the procedure *SelectionSort*, shown in Fig. 2, sorts the remaining elements and the index of exact median of the set described in Step (3) is returned. In particular, if an infinite should be returned, i.e. $m \geq n$, it is replaced with the remaining element closest to it.

5. Run-time analysis

To evaluate the computational cost of our algorithm, we not only consider the usual metric of key comparisons, but also look briefly at the number of element swaps in this particular implementation, given in Section 4.

Let $\delta_t = \frac{8}{3}$ be the average number of key comparisons needed to determine the median element of a triplet (cf. [1]) and let $\delta_c = 1$ be the average number of key comparisons needed to determine the smallest or the largest element of a couple.

The following recurrence, where n and m stand for the number of real keys and virtual infinities respectively, describes the evolution of the expected number of key comparisons from stage to stage:

$$C(n,m) = \begin{cases} 0 & \text{if } n = 1, \\ c(n,m) + C(\mu_0(n,m) + \mu_1(n,m),\ \mu_2(n,m) + \mu_3(n,m)) & \text{if } n > 1. \end{cases} \tag{7}$$

Here $c(n,m) = \delta_t \mu_0(n,m) + \delta_c \mu_1(n,m)$ is the average number of comparisons needed to process the $\mu_0(n,m)$ triplets and the $\mu_1(n,m)$ couples in a single stage of the algorithm; the recursive term adds the average number of comparisons needed to handle the real and infinite keys surviving to the next stage. The average number of key comparisons performed by the algorithm when the input size is n and we are searching for the k-th element is given by $C(n, |n-2k+1|)$.

It is easy to see that our algorithm has running-time which is linear in the input size n. Indeed, let us consider the following two bounding cases: $k = \lceil \frac{n}{2} \rceil$ and $k = 1$ (or symmetrically $k = n$).

In the first case, when n is odd, we have $m = 0$ and equation (7) can be rewritten as follows:

$$C(n,0) = \frac{8}{9}n + C(\frac{n}{3}, 0).$$

This recurrence depends of a single variable and its exact solution, when n is a power of 3, is $C(n,0) = \frac{4}{3}(n-1)$.

In the second case, we have $m = n-1$ but, for the purpose of the analysis, we make the following approximations $m \approx n$, $\mu_0(n,m) \approx \mu_3(n,m) \approx \frac{n}{12}$, $\mu_1(n,m) \approx \mu_2(n,m) \approx \frac{n}{4}$ and equation (7) can then be

Pseudo-code for SystemSolve, TripletMedian, and SelectionSort

FUNCTION **SystemSolve** (n, m) : integers quadruple

This function returns four integer values satisfing conditions given in Step (2b).

$r = [n+m]_3$;
$x = \mu_2(n-r, m)$;
$y = \mu_3(n-r, m)$;
$\Gamma = \{\langle\lfloor x\rfloor, \lfloor y\rfloor\rangle, \quad \langle\lfloor x\rfloor, \lceil y\rceil\rangle, \quad \langle\lceil x\rceil, \lfloor y\rfloor\rangle, \quad \langle\lceil x\rceil, \lceil y\rceil\rangle\}$;

$$- \left[\begin{array}{l} \textit{Choose an ordered couple } \langle\overline{\mu}_2, \overline{\mu}_3\rangle \in \Gamma \textit{ such that the following integer values:} \\ \quad \overline{\mu}_1 = m - 2\overline{\mu}_2 - 3\overline{\mu}_3\,, \\ \quad \overline{\mu}_0 = (n - r - 2\overline{\mu}_1 - \overline{\mu}_2)/3\,, \\ \textit{are non-negatives, i.e. } \overline{\mu}_i \in \mathbb{N}_0 \textit{ with } i = 0..3; \end{array}\right.$$

return $\langle\overline{\mu}_0, \overline{\mu}_1, \overline{\mu}_2, \overline{\mu}_3\rangle$;

FUNCTION **TripletMedian** (A, i) : integer

This function returns the index of the median of the triplet $(A[i], A[i+1], A[i+2])$.

```
if (A[i] < A[i+1])  then
        if (A[i+2] < A[i])          then return i;
        elsif (A[i+2] < A[i+1])     then return (i+2);
        else  return (i+1);
else
        if (A[i] < A[i+2])          then return i;
        elsif (A[i+2] > A[i+1])     then return (i+2);
        else  return (i+1);
```

PROCEDURE **SelectionSort** $(A, Size)$

This procedure rearranges the elements of the array $A[1,\ldots,Size]$ *in ascending sorted order.*

```
i = 1;
while (i < Size) do
        min = i;
        j = i + 1;
        while (j ≤ Size) do
                if (A[j] < A[min]) then min = j;
                j = j + 1;
        end while;
        Swap (A[i], A[min]);
        i = i + 1;
end while;
```

Figure 2: Pseudo-code for the procedures used by the approximate k-th selection algorithm.

rewritten as follows:

$$C(n,n) = \frac{17}{36}n + C(\frac{n}{3}, \frac{n}{3}),$$

with the exact solution, when n is a power of 3, given by $C(n,n) = \frac{17}{24}(n-1)$.

Asymptotically, the leading coefficients of the above solutions does not change when n is not a power of 3. On the other hand, it is easy to verify that, among all the possible k values, the first case above is the most expensive in terms of key comparisons—since comparisons are used optimally when triplets have infinities. Hence can state the following:

Theorem 2. *The average number of key comparisons performed by the algorithm when the input size is n, and the k-th element is searched for, is no more than $\frac{4}{3}n$.*

The above reason underpins the way k influences the number of comparisons required by the algorithm. Unforunately, there is little hope of securing a general, analytic solution of the recurrence 7, but one can numerically verify that the function $C(n,m)$ is monotonically decreasing in m, with $0 \le m < n$ and n fixed; in particular, the minimum bound is reached in the above second case.

The worst-case analysis can be analogously handled considering the maximum number of key comparisons $\delta_t = 3$ required to find the median of three elements. We get:

Theorem 3. *The number of key comparisons performed by the algorithm when the input size is n, and the k-th element is searched for, is no more than $\frac{3}{2}n$.*

Let us consider the number of element moves. Unlike the procedure developed in [1], it is not easy here to avoid doing as we do in Section 4, and move each surviving element (or swap it with another element in the array, if we need to preserve its content). Therefore the number of elements swaps, both in the average- and in the worst-case, is given by equation (7) setting $\delta_t = \delta_c = 1$. Again, not much hope for an exact solution, but it is a immediate to verify that $C(n,m) < n$. Moreover, one can numerically verify that $C(n,m) < \frac{3}{4}n$ for all $0 \le m < n$, so we can conclude that the number of element swaps performed by the algorithm given in Fig. 1, when the input size is n and we are searching for the k-th element, is no more than $\frac{3}{4}n$.

6. Probabilistic analysis

In this section we give an exact probabilistic analysis of the k-th selection algorithm implemented in Section 4, which returns the value *Output*. Specifically, for any $n,k \in \mathbb{N}$, with $1 \le k \le n$, we calculate the following probability distribution:

$$\mathcal{P}_{n,k}(z) = \Pr\{\, |\mathrm{rank}(Output) - k| \le \lfloor zn \rfloor \,\}, \tag{8}$$

of the relative selection error, measured by $z \in [0,1]$.

W.l.o.g., we can assume that the input array of size n is a permutation of the integers $1,\ldots,n$. The set of the $n!$ permutations is denoted by S_n, sometimes called the free group (it is a group, under composition, but we shall not use this fact here).

We make two reasonable uniformity assumptions. One concerns the distribution of the input set: all permutations are assumed equally likely. The second refers to the random choice of triplets, couples and singletons during the execution of the algorithm, which we assume are all independent of the array content and of each other.

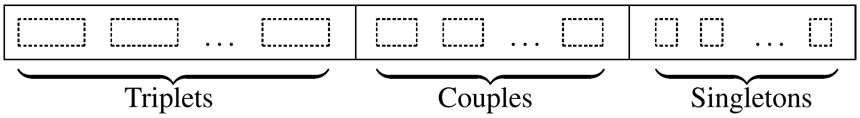


Figure 3: Looking at the array elements as a sequence of sub-sequences.

While the algorithm creates the successive triplets probabilistically, for the following exposition it is convenient to view the input array as a sequence of consecutive triplets followed by consecutive couples and by consecutive singletons as shown in Fig. 3, which does not show of course the virtual infinities.

W.l.o.g. in the analysis, we consider the case $1 \le k \le \lceil \frac{n}{2} \rceil$, with the symmetric one, $\lceil \frac{n}{2} \rceil < k \le n$, being completely analogous. This means that the infinities are negative, and the surviving element of any couple is its smaller element.

We use of the following notation and identity for the multinomial coefficient:

$$\binom{N}{\{k\}^m} \stackrel{\text{def}}{=} \binom{N}{\underbrace{k,k,\ldots,k}_{m\text{ times}}} = \prod_{i=0}^{m-1} \binom{N-ki}{k} = \frac{N!}{(k!)^m (N-km)!}, \tag{9}$$

for all $N,k,m \in \mathbb{N}_0$, with $N \ge km$.

We construct our solution for the probability distribution (8), from the answer to the question "what is the probability that when the array has the n values $1,\ldots,n$ in random order, the value a is selected, and becomes the b^{th} smallest among the surviving elements." We decompose this calculation to a sequence of combinatorial problems.

A sequence of l two-position boxes is given. We place in them a random permutation of distinct integers, $\pi \in S_{2l}$. The smallest element in each box is selected, giving us l surviving elements.

Definition 1. *Given $l \in \mathbb{N}$ and $a,b \in \mathbb{N}_0$, we denote by $Q_l^{[2]}(a,b)$ the number of permutations π, out of the $(2l)!$ possible ones, in which exactly b elements survive among the a smallest elements in π.*

For convenience, we extend Def. 1 for $l = 0$ too, setting:

$$Q_0^{[2]}(a,b) = \begin{cases} 1 & \text{if } a = b = 0\,, \\ 0 & \text{otherwise.} \end{cases}$$

The following notation will simplify our discussion. For any permutation π and an integer a, we denote by "$\bullet$" each element among the a smallest ones in π, and denote by "$\circ$" the others.

Lemma 4. *Let $a,b,l \in \mathbb{N}_0$. The number of permutations $Q_l^{[2]}(a,b)$ is given by*

$$Q_l^{[2]}(a,b) = \begin{cases} 2^{(2b-a)} l!b!(l-b)! \binom{2l-a}{l-b}\binom{l-a+b}{2b-a}\binom{a}{a-b} & \text{if } b \le l\,, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. It is immediate to verify that any contribution to $Q_l^{[2]}(a,b)$ can be given only if the condition $b \le a \le 2b \le 2l$ is satisfied.

In a permutation π, among the $2l$ elements, there are a "$\bullet$" and $2l-a$ "$\circ$". We count permutations that result in selections of b "$\bullet$" and $l-b$ "$\circ$". All the surviving "$\circ$" must come from the $l-b$ couples of type $\{\circ,\circ\}$ in π, and thus $(2l-a)-2(l-b)=2b-a$ unordered couples of type $\{\circ,\bullet\}$ occur in π. Consequently, $[a-(2b-a)]/2=a-b$ couples of type $\{\bullet,\bullet\}$ are present in π.

We need to count the permutations π that satisfy the above constraints. First we count the number of ways to select occupants for the unmixed boxes: We have $\frac{1}{(a-b)!}\binom{a}{\{2\}^{a-b}}$ ways to select $a-b$ pairs of "$\bullet$" from the a "$\bullet$" in π, and $\frac{1}{(l-b)!}\binom{2l-a}{\{2\}^{l-b}}$ ways to select $(l-b)$ pairs of "$\circ$" from the $(2l-a)$ available "$\circ$" in π. Then, we have $\frac{1}{(2b-a)!}\prod_{i=0}^{2b-a-1}(2b-a-i)^2=(2b-a)!$ ways to match the remaining $(2b-a)$ "$\circ$" and $(2b-a)$ "$\bullet$" with the $(2b-a)$ boxes containing mixed couples.

Finally, to complete the specification of π, a factor 2^l accounts for the order inside each box, and an $l!$ accounts for the order of the boxes. Multiplying these factors and using the identity (9), we get:

$$Q_l^{[2]}(a,b)=\begin{cases} 2^{(2b-a)}l!\frac{a!(2l-a)!}{(a-b)!(l-b)!(2b-a)!} & \text{if } b\le a\le 2b\le 2l\,, \\ 0 & \text{otherwise,}\end{cases} \tag{10}$$

and rewriting it using the binomial coefficients we get the desired result. ■

Notice that, for any $l\in\mathbb{N}_0$, we have $Q_l^{[2]}(0,0)=(2l)!$ and $Q_l^{[2]}(a,0)=0$ for all $a\neq 0$.

Now we consider the following consequence of the above result:

Definition 2. *Given $a,b,l\in\mathbb{N}$, we denote by $\overline{Q}_l^{[2]}(a,b)$ the number of permutations π, out of the $(2l)!$ possible ones, in which the a^{th} smallest element in π is selected and becomes the b^{th} smallest among the surviving elements.*

Lemma 5. *Let $a,b,l\in\mathbb{N}$. The number of permutations $\overline{Q}_l^{[2]}(a,b)$ is given by*

$$\overline{Q}_l^{[2]}(a,b)=2l(2l-a)Q_{l-1}^{[2]}(a-1,b-1).$$

Proof. Recycling the notation of the previous proof, we observe that the a^{th} element in π is the largest "$\bullet$" and given that it is selected, it came from a mixed couple. Any permutation that contributes here was counted in $Q_l^{[2]}(a,b)$, if a was among those selected.

Since the candidate permutations have $2b-a$ mixed couples, then among the permutations we counted above only in $(2b-a)/a$ of them we find the largest "$\bullet$" in a mixed couple, that is,

$$\overline{Q}_l^{[2]}(a,b)=\frac{2b-a}{a}Q_l^{[2]}(a,b)=2l(2l-a)Q_{l-1}^{[2]}(a-1,b-1),$$

where the second equality follows from the explicit form above in Eq. (10). ■

By convention, we extend Def. 2 to the case $l=0$ and we let $\overline{Q}_0^{[2]}(a,b)=0$ for any $a,b\in\mathbb{N}$.

We deal now with the the same combinatorial problem with respect to triplets of actual values, instead of couples. A random permutation π from S_{3l} is placed in a sequence of l three-position boxes, as before. We select the median element in each box, and get l surviving elements.

Definition 3. *Given $l\in\mathbb{N}$ and $a,b\in\mathbb{N}_0$, we denote by $Q_l^{[3]}(a,b)$ the number of permutations π, out of the $(3l)!$ possible ones, in which exactly b elements survive among the a smallest elements in π.*

As above, it will be convenient to extend Def. 3 to $l=0$, setting:

$$Q_0^{[3]}(a,b)=\begin{cases} 1 & \text{if } a=b=0\,, \\ 0 & \text{otherwise.}\end{cases}$$

Lemma 6. *Let $a,b,l \in \mathbb{N}_0$. The number of permutations $Q_l^{[3]}(a,b)$ is given by*

$$Q_l^{[3]}(a,b) = \begin{cases} 3^{a-b} a!(3l-a)! \binom{l}{b} \sum_{i=0}^{b} \frac{1}{9^i} \binom{b}{i} \binom{l-b}{a-2b-i} & \text{if } a \le 3l, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Proof. Clearly this configuration requires $0 \le b \le a \le 3l$. As above, we denote the a smallest elements in the array by "$\bullet$" and the other $(3l-a)$ entries by "$\circ$". With the notation introduced before, four types of triplets can occur in π, namely $t_0 : \{\bullet,\bullet,\bullet\}$, $t_1 : \{\bullet,\bullet,\circ\}$, $t_2 : \{\bullet,\circ,\circ\}$ and $t_3 : \{\circ,\circ,\circ\}$, where the order of the elements inside each triplet is not relevant at the moment.

Let denote by $\langle\pi\rangle_i$ the number of triplets in π of type t_i, with $0 \le i \le 3$. All the permutations π contributing to $Q_l^{[3]}(a,b)$ need to satisfy the following constraints:

$$\begin{cases} 3\langle\pi\rangle_0 + 2\langle\pi\rangle_1 + \langle\pi\rangle_2 = a\,, \\ \langle\pi\rangle_1 + 2\langle\pi\rangle_2 + 3\langle\pi\rangle_3 = 3l - a\,, \\ \langle\pi\rangle_0 + \langle\pi\rangle_1 = b\,. \end{cases} \tag{12}$$

The four numbers are not independent, of course, since they must all sum to l; however, since adding the first two rows yields this sum, $\sum_{i=0}^{3} \langle\pi\rangle_i = l$, the system (12) is undetermined. Specifically, we can describe the space of solutions as spanned by fixing $\langle\pi\rangle_(0)$ at any value in the interval $[0,b]$.

Let us define the following equivalence relation[3] on the set S_{3l}, of the $(3l)!$ permutations:

$$\pi \sim \pi' \Longleftrightarrow \langle\pi\rangle_i = \langle\pi'\rangle_i\,, \qquad \text{for each } i \in [0,3],$$

for any $\pi,\pi' \in S_{3l}$. We denote by $\widetilde{\pi} \in S_{3l}/\sim$ the $\sim$-equivalence class of a permutation $\pi \in S_{3l}$. Moreover, by solving the system (12) in terms of $\langle\pi\rangle_0$ we see that all the permutations $\pi \in S_{3l}$ contributing to $Q_l^{[3]}(a,b)$ can be gathered together in the following set of equivalence classes

$$\widetilde{\Pi}_{a,b}^{(l)} = \left\{ \widetilde{\pi} \in S_{3l}/\sim \;\middle|\; 0 \le \langle\pi\rangle_0 \le b \quad \text{and} \quad \begin{pmatrix} \langle\pi\rangle_1 = & b - \langle\pi\rangle_0 \\ \langle\pi\rangle_2 = & a - 2b - \langle\pi\rangle_0 \\ \langle\pi\rangle_3 = & l - a + b + \langle\pi\rangle_0 \end{pmatrix} \right\}. \tag{13}$$

The note about the solutions above implies that $\left|\widetilde{\Pi}_{a,b}^{(l)}\right| = b+1$. We start by computing the size $|\widetilde{\pi}|$ for each equivalence class $\widetilde{\pi} \in S_{3l}/\sim$:

Lemma 7.

$$|\widetilde{\pi}| = l! \times 6^l \times \prod_{i=0}^{3} C_i(\widetilde{\pi})\,,$$

[3] We do not use any of the standard properties of such a relation, except that its classes partition S_{3l}.

where

$$C_0(\widetilde{\pi}) = \frac{1}{\langle\pi\rangle_0!}\binom{a}{\{3\}^{\langle\pi\rangle_0}} = \frac{1}{\langle\pi\rangle_0!}\prod_{i=0}^{\langle\pi\rangle_0-1}\binom{3\langle\pi\rangle_0+2\langle\pi\rangle_1+\langle\pi\rangle_2-3i}{3},$$
$$C_1(\widetilde{\pi}) = \frac{1}{\langle\pi\rangle_1!}\binom{2\langle\pi\rangle_1+\langle\pi\rangle_2}{\{2\}^{\langle\pi\rangle_1}}\times\langle\pi\rangle_1! = \frac{1}{\langle\pi\rangle_1!}\prod_{i=0}^{\langle\pi\rangle_1-1}\binom{2\langle\pi\rangle_1+\langle\pi\rangle_2-2i}{2}(\langle\pi\rangle_1-i),$$
$$C_2(\widetilde{\pi}) = \frac{1}{\langle\pi\rangle_2!}\binom{\langle\pi\rangle_1+2\langle\pi\rangle_2}{\{2\}^{\langle\pi\rangle_2}}\times\langle\pi\rangle_2! = \frac{1}{\langle\pi\rangle_2!}\prod_{i=0}^{\langle\pi\rangle_2-1}\binom{\langle\pi\rangle_1+2\langle\pi\rangle_2-2i}{2}(\langle\pi\rangle_2-i),$$
$$C_3(\widetilde{\pi}) = \frac{1}{\langle\pi\rangle_3!}\binom{3l-a}{\{3\}^{\langle\pi\rangle_3}} = \frac{1}{\langle\pi\rangle_3!}\prod_{i=0}^{\langle\pi\rangle_3-1}\binom{\langle\pi\rangle_1+2\langle\pi\rangle_2+3\langle\pi\rangle_3-3i}{3}.$$

Proof. The $C_i(\widetilde{\pi})$ have the following interpretation:
$C_0(\widetilde{\pi})$ counts the ways to distribute $3\langle\pi\rangle_0$ "$\bullet$" elements out of the a available into $\langle\pi\rangle_0$ boxes containing triplets of type t_0; the factor $\frac{1}{\langle\pi\rangle_0!}$ is inserted because we are not considering now the order among boxes. Similarly, $C_3(\widetilde{\pi})$ enumerates the ways to distribute $3\langle\pi\rangle_3$ "$\circ$" elements out of $(3l-a)$ into $\langle\pi\rangle_3$ unordered boxes containing triplets of type t_3.
In $C_1(\widetilde{\pi})$ we count the ways to populate $\langle\pi\rangle_1$ unordered boxes, each with 2 of the $(a-3\langle\pi\rangle_0)$ remaining "$\bullet$" per box and one of the $\langle\pi\rangle_1$ remaining "$\circ$" elements.
$C_2(\widetilde{\pi})$ is defined in a completely analogous manner.
A factor 6^l takes into account the order of the elements inside each box, and a factor $l!$ is finally introduced to complete the specification of the permutation by accounting for the order of the boxes. ■

To return to proving lemma 6 we apply identity (9) and get after several cancellations:

$$\prod_{i=0}^{3} C_i(\widetilde{\pi}) = \frac{a!(3l-a)!}{2^{\langle\pi\rangle_1+\langle\pi\rangle_2}3^{\langle\pi\rangle_0+\langle\pi\rangle_3}\prod_{i=0}^{3}(\langle\pi\rangle_i!)}$$

Now we can calculate the quantity of our interest; by the definition of $\widetilde{\Pi}_{a,b}^{(l)}$ we get:

$$\begin{aligned} Q_l^{[3]}(a,b) &= |\{\pi\in S_{3l}\mid\widetilde{\pi}\in\widetilde{\Pi}_{a,b}^{(l)}\}| = \sum_{\widetilde{\pi}\in\widetilde{\Pi}_{a,b}^{(l)}}|\widetilde{\pi}| \\ &= l!a!(3l-a)!\sum_{\widetilde{\pi}\in\widetilde{\Pi}_{a,b}^{(l)}}\frac{3^{\langle\pi\rangle_1+\langle\pi\rangle_2}}{\prod_{i=0}^{3}(\langle\pi\rangle_i!)}, \end{aligned}$$

We use the relation in Eq. (13) to rewrite the terms the last row, and since each $\widetilde{\pi}\in\widetilde{\Pi}_{a,b}^{(l)}$ can be seen as corresponding to a value of $\langle\pi\rangle_0$ in $[0,b]$, the sum there can be changed to one on this index, and we get the desired result. ■
Notice that, for any $l\in\mathbb{N}_0$, we have $Q_l^{[3]}(0,0)=(3l)!$ and $Q_l^{[3]}(a,0)$ is not necessarily null when $a\neq 0$.

Again we derive a useful consequence of the last result:

Definition 4. *Given $a,b,l\in\mathbb{N}$, we denote by $\overline{Q}_l^{[3]}(a,b)$ the number of permutations π, out of the $(3l)!$ possible ones, in which the a^{th} smallest element in π is selected and becomes the b^{th} smallest among the surviving elements.*

Lemma 8. *Let $a,b,l \in \mathbb{N}$. The number of permutations $\overline{Q}_l^{[3]}(a,b)$ is given by*

$$\overline{Q}_l^{[3]}(a,b) = 6l(a-1)(3l-a)Q_{l-1}^{[3]}(a-2,b-1).$$

Proof. All the permutations contributing to $\overline{Q}_l^{[3]}(a,b)$ are among those we counted in $Q_l^{[3]}(a,b)$, but we only need those in which a ends up being selected. as before, since a element is the largest "$\bullet$" in π it gets selected if π puts it in a triplet of type t_1. Such a triplet has 2 "$\bullet$" elements, for a total of $2\langle\pi\rangle_1$ out of a such elements. In the formula (11) we assign $\langle\pi\rangle_0 = i$, hence $\langle\pi\rangle_1 = b-i$, and therefore

$$\overline{Q}_l^{[3]}(a,b) = 3^{a-b}a!(3l-a)!\binom{l}{b}\sum_{i=0}^{b}\frac{1}{9^i}2\frac{b-i}{a}\binom{b}{i}\binom{l-b}{a-2b-i} \tag{14}$$

Rearranging the terms above we obtain the needed result. ∎

By convention, we extend Def. 4 letting $\overline{Q}_0^{[3]}(a,b) = 0$ for any $a,b \in \mathbb{N}$.

We are ready to define the next step in our analysis, combining all the results obtained so far.

An ordered sequence of consecutive l_0,l_1,l_2 boxes is given, where each box contains exactly either three, two or one ordered elements, respectively (see Fig. 3). Initially, $n = 3l_0+2l_1+l_2$ totally ordered elements are uniformly distributed in the boxes, creating a random permutation π. Following the process of our algorithm, we select the median element in each triplet (cf. Def. 3) and the smallest element in each box containing a couple (cf. Def. 1). Thus, (l_0+l_1) elements survive at this stage, including none of the singletons contained in the last l_2 boxes, as shown in Fig. 4.

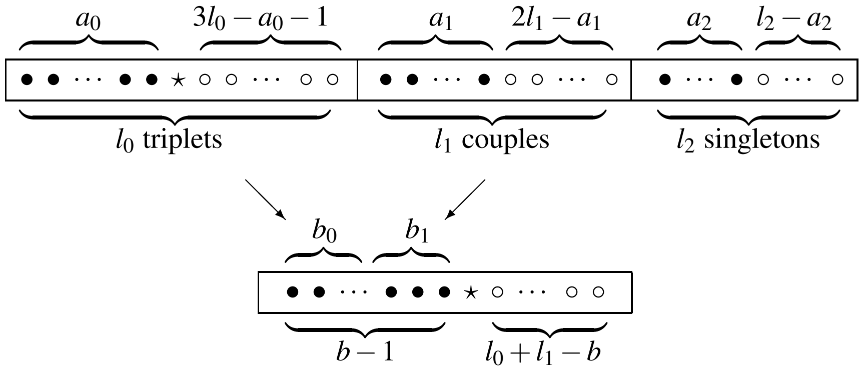


Figure 4: Looking at the elements array as a sequence of triplets, couples and singletons.

Definition 5. *Given $l_0,l_1,l_2 \in \mathbb{N}_0$ and $a,b \in \mathbb{N}$, we denote by $\overline{Q}_{l_0,l_1,l_2}(a,b)$ the number of permutations π, out of the $n!$ possible ones, in which the a^{th} smallest element in π is selected and becomes the b^{th} smallest among the surviving elements.*

Lemma 9. *Let $l_0,l_1,l_2 \in \mathbb{N}_0$ and $a,b \in \mathbb{N}$. The number of permutations $\overline{Q}_{l_0,l_1,l_2}(a,b)$ is given by*

$$\overline{Q}_{l_0,l_1,l_2}(a,b) = l_2! \sum_{\substack{a_i \in [0..(3-i)l_i],\, i=0..2 \\ a_0+a_1+a_2=a-1 \\ b_i \in [0..min(a_i,l_i)],\, i=0,1 \\ b_0+b_1=b-1}} \binom{a-1}{a_0}\binom{a-a_0-1}{a_1}(S_1+S_2)\,,$$

where,

$$S_1 = \binom{n-a}{3l_0-a_0-1}\binom{n-a-3l_0+a_0+1}{2l_1-a_1}\cdot\overline{Q}^{[3]}_{l_0}(a_0+1,b_0+1)\cdot Q^{[2]}_{l_1}(a_1,b_1)\,,$$
$$S_2 = \binom{n-a}{3l_0-a_0}\binom{n-a-3l_0+a_0}{2l_1-a_1-1}\cdot Q^{[3]}_{l_0}(a_0,b_0)\cdot\overline{Q}^{[2]}_{l_1}(a_1+1,b_1+1)\,.$$

Proof. In a generic permutation π, among the n elements, there are a "$\bullet$" distributed into the boxes containing triplets, couples and singletons, and $(n-a)$ "$\circ$". In particular, excluding the a^{th} element "$\star$", a_0 "$\bullet$" occurs in the l_0 triplets (with $0 \le a_0 \le 3l_0$), a_1 "$\bullet$" occurs in the l_1 couples (with $0 \le a_1 \le 2l_1$) and a_2 "$\bullet$" forms the l_2 singletons (with $0 \le a_2 \le l_2$), such that $a_0+a_1+a_2 = a-1$. This situation is depicted in Fig. 4, where each rectangle does not preserve the order of the elements inside. In order to get the element "$\star$" to be selected and to be the b^{th} among the surviving elements, we need b_0 "$\bullet$" survived from triplets (with $0 \le b_0 \le \min(a_0,l_0)$) and b_1 "$\bullet$" survived from couples (with $0 \le b_1 \le \min(a_1,l_1)$), such that $b_0+b_1 = b-1$.

To evaluate $\overline{Q}_{l_0,l_1,l_2}(a,b)$ we just need to count the permutations that satisfy the above constraints. First we count the ways to fill up the a_0 positions in the triplets with $(a-1)$ "$\bullet$" and the a_1 positions in the couples with the remaining $(a-a_0-1)$ "$\bullet$". Secondly, we evaluate the quantities S_1 and S_2 corresponding to the case of "$\star$" belonging to a triplet or to a couple, respectively. In S_1 we count how to fill up the $(3l_0-a_0-1)$ positions in the triplets with $(n-a)$ "$\circ$" and the $(2l_1-a_1)$ positions in the couples with the remaining $(n-a-3l_0+a_0+1)$ "$\circ$"; "$\star$" is the $(a_0+1)^{th}$ smallest element in the triplets and we need it becomes the $(b_0+1)^{th}$ after the triplets are processed; at the same time, we need b_1 "$\bullet$" selected among the a_1 smallest elements in the couples. The quantity S_2 is derived analogously.

Finally, the factor $l_2!$ takes into account the remaining elements forming the l_2 singletons. ∎

The performance of the algorithm is characterized by the probability $P_a^{(n,m)}$, that the a^{th} smallest element of the input array of size n is returned in output as final winner, when the k-th element is searched for, and then $m = |n-2k+1|$.

Let n_i, m_i, for each $i \in [0,f]$ be the sequence of values computed during the $(f+1)$ iterations of the "while" loop of the algorithm, on an input array of size $n = n_0$ with $m_0 = |n_0-2k+1|$; the value $f \in \mathbb{N}_0$ is the smallest index such that $n_{f+1} \le \mathit{Threshold}$.

The previous values are computed by

$$n_{i+1} = \overline{\mu}^{(i)}_0 + \overline{\mu}^{(i)}_1\,, \qquad m_{i+1} = \overline{\mu}^{(i)}_2 + \overline{\mu}^{(i)}_3\,, \qquad \text{for each } i = 0..f\,,$$

where $\langle \overline{\mu}^{(i)}_0, \overline{\mu}^{(i)}_1, \overline{\mu}^{(i)}_2, \overline{\mu}^{(i)}_3 \rangle$ are the integer values returned by SystemSolve (n_i, m_i).

By Lemma 9, we are able to analyze a single iteration of the algorithm. Indeed, for any values of the integer variables n_i, m_i, we can derive the probability $p_{a,b}^{(n_i,m_i)}$ that the a^{th} smallest element at i-th step becomes the b^{th} smallest element in the next $(i+1)$-st step obtained applying rules in Step (2d) of Section 3:

$$p_{a,b}^{(n_i,m_i)} = \frac{1}{n_i!}\cdot\overline{Q}_{\overline{\mu}^{(i)}_0,\overline{\mu}^{(i)}_1,\overline{\mu}^{(i)}_2+[n_i+m_i]_3}(a,b)\,.$$

A recurrence equation can be established to evaluate the probability of our interest:

$$P_a^{(n_i,m_i)} = \sum_{b_i\in[1..a]} p_{a,b_i}^{(n_i,m_i)}\cdot P_{b_i}^{(n_{i+1},m_{i+1})}\,, \qquad \text{for all } i = 0..f\,,$$

and expanding, we get:

$$P_a^{(n,m)} = \sum_{b_0,b_1,\cdots,b_{f-1},b_f} p_{a,b_0}^{(n_0,m_0)} \cdot p_{b_0,b_1}^{(n_1,m_1)} \cdots p_{b_{f-1},b_f}^{(n_f,m_f)} \cdot P_{b_f}^{(n_{f+1},m_{f+1})} , \tag{15}$$

where,

$$P_a^{(n_{f+1},m_{f+1})} = \begin{cases} 1 \quad \text{if} & \left(\begin{array}{c} n_{f+1} \leq m_{f+1} \text{ and } a = 1 \\ \text{or} \\ n_{f+1} > m_{f+1} \text{ and } a = \left\lceil \frac{n_{f+1}-m_{f+1}}{2} \right\rceil \end{array} \right), \\ 0 \quad \text{otherwise.} \end{cases} \tag{16}$$

It is straightforward that probabilities given by (15) and (16) can be symmetrically defined for $\lceil \frac{n}{2} \rceil < k \leq n$, so that we can define the probability distribution given by (8), for each $1 \leq k \leq n$:

$$\mathcal{P}_{n,k}(z) = \sum_{\substack{1 \leq a \leq n \\ |a-k| \leq \lfloor zn \rfloor}} P_a^{(n,|n-2k+1|)} , \qquad \text{with } z \in [0,1] . \tag{17}$$

While these expressions for the distribution of the returned value are far too complex to allow for an analytic characterization, generating numbers from them is possible, as we show in the next section, which also demonstrates how closely they track the actual performance of the algorithm.

7. Experimental results

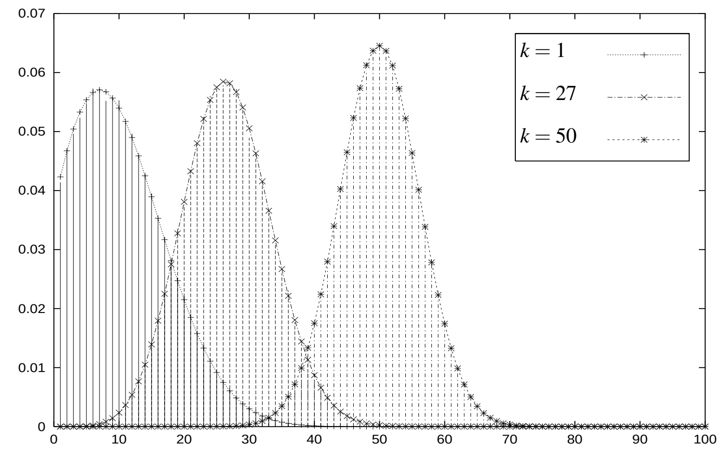
In this section we present empirical results, demonstrating the effectiveness of the algorithm described in Section 4. Our implementation is in standard C (GNU C compiler v2.7). All the experiments were carried out on a PC Pentium II 350Mhz with the Linux (Red Hat distribution) operating system. The arrays were permuted using the pseudo-random number generator suggested by Park and Miller in 1988 and updated in 1993 [10]. The algorithm was run on random arrays of sizes $n = 10^i$, with $i = 2..5$, whose entry keys were always the integers $1, \ldots, n$.

Our first experiment aims to evaluate how the theoretical probability distribution (15) fits with the empirical one. In order to do this, we chose a sample of $100,000$ random input arrays of size $n = 100$ and run the algorithm on it for some k values chosen between 1 and $\lceil \frac{n}{2} \rceil$, with a fixed *Threshold* equal to $\lfloor \sqrt{n} \rfloor$. We compute the histograms of the relative frequencies of the algorithm outputs so that theoretical and empirical distributions are compared in Fig. 5, where, for each chosen k, we reported the average μ_k and the standard deviation σ_k of both the distributions.

A more interesting experiment was done with variable input sizes $n = 10^i$, $i = 2..5$, with *Threshold* equal to $\lfloor \sqrt{n} \rfloor$. Also in this case we fixed some k values among $\lfloor \frac{n}{8} \rfloor$, $\lfloor \frac{3n}{8} \rfloor$, $\lfloor \frac{n}{2} \rfloor$ and $\lfloor \frac{3n}{4} \rfloor$, with the only exception for $n = 100$, and we built the histograms of the relative frequencies of the algorithm outputs. Each single histogram, shown in Fig. 6, has been obtained on a different sample of size $100,000$.

The *Threshold* value $\lfloor \sqrt{n} \rfloor$ until now used was completely arbitrary. In Fig. 7 we report some histograms referred to the relative frequencies of algorithm outputs for three different *Threshold* values, i.e. 3, $\lfloor \sqrt{n} \rfloor$, $2\lfloor \sqrt{n} \rfloor$. Data are obtained from a fixed sample of $100,000$ arrays of size $n = 1,000$ and for some fixed k values. By examining the standard deviation σ_k values, it is clear that larger is the *Threshold*

Theoretical vs. empirical



		$n = 100$	
k	Data	μ_k	(σ_k)
5	*Theor.*	11.00	(6.93)
	Exper.	11.02	(6.91)
27	*Theor.*	26.63	(6.76)
	Exper.	26.60	(6.78)
50	*Theor.*	49.99	(6.19)
	Exper.	50.03	(6.21)

Figure 5: Comparing the theoretical probability distributions with the experimental results histograms.

value, larger is the computational cost of the selection and better is the quality of the selection. For this reason, depending on the specific application, the *Threshold* value has to be appropriately tuned.

In the previous experiments, we worked on some representative k values arbitrarily chosen. Now, we concentrate our attention on the percentage error made by the algorithm, independently by k. For each $i = 2..5$, we fixed a sample of $1,000$ arrays of size $n = 10^i$ and we used a *Threshold* value $\lfloor\sqrt{n}\rfloor$. We run the algorithm on the input sample for each k value, such that $1 \leq k \leq n$, with the only exception for $i = 4,5$ for which only $1,000$ uniformly distributed k values were examined, i.e. values $k = 1 + j \cdot 10^{i-3}$ with $j = 0..999$. For each single algorithm output, corresponding to a value k and to an array of the sample, we calculate the percentage error distance

$$d_k = \frac{|k - \mathrm{rank}(Output)|}{n-1} \cdot 100, \qquad \text{with } 1 \leq k \leq n\,,$$

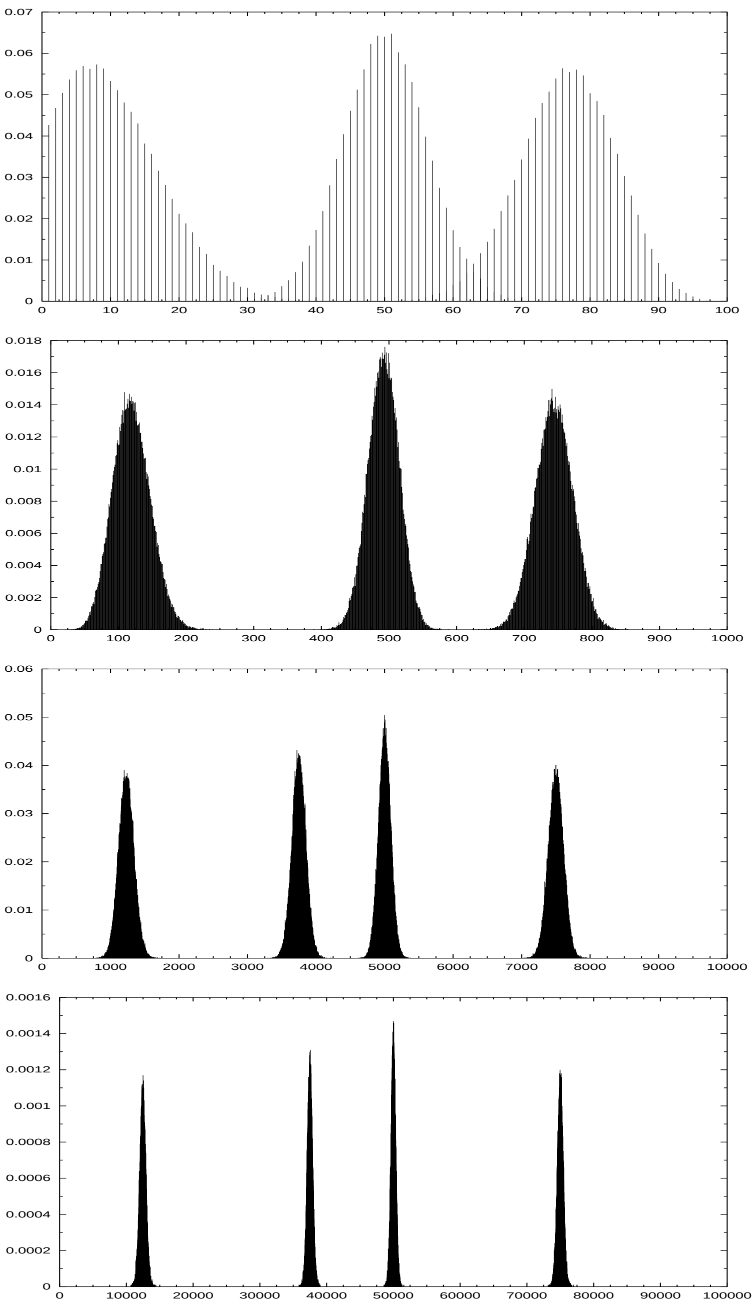
from the correct k-th element. The extreme values assumed by d_k can be 0%, when the k-th element is returned by the algorithm, and 100% when the algorithm returns the largest element in the array whereas the smallest one is searched for, or viceversa. In Fig. 8, for each single k, we plotted the maximum error $M_k = \max(d_k)$, the average error $\varepsilon_k = E[d_k]$ and the standard deviation $\sigma_k = \sigma[d_k]$ of the $1,000$ d_k values.

Finally, in Fig. 9 we give a plot that summarizes some of the statistical indexes obtained from the previous data-sets. It well illustrates the good quality of the algorithm output when the input size n is sufficiently large. More specifically, the trend of the standard deviation $E[\sigma_k]$ may be viewed as a measure of the improvement of the selection effectiveness for increasing n.

8. Conclusion

We have presented an approximate algorithm for the k-th selection problem, based on a statistical approach. It is a generalization of the algorithm for the approximate median selection problem described in

Relative frequencies histograms



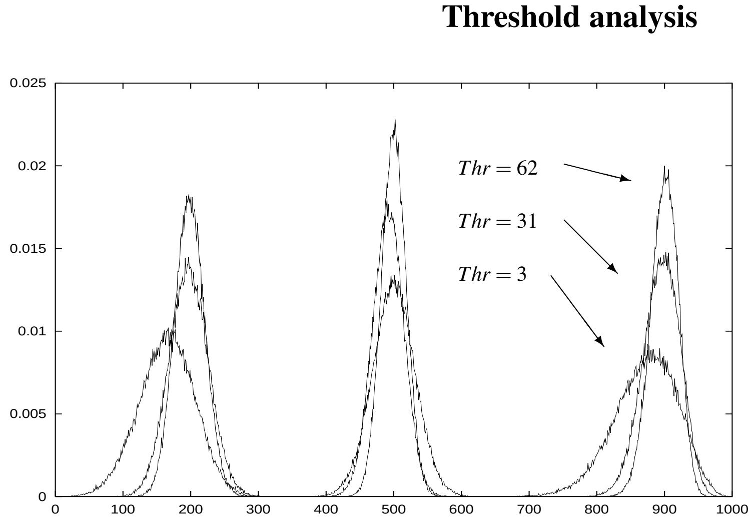
$n = 100$		
k	μ_k	(σ_k)
5	11.0	(6.9)
50	50.0	(6.2)
75	76.7	(7.0)

$n = 1{,}000$		
k	μ_k	(σ_k)
125	119.5	(27.8)
500	492.1	(23.2)
750	743.8	(28.2)

$n = 10{,}000$		
k	μ_k	(σ_k)
1,250	1,230.8	(110.0)
3,750	3,750.0	(100.2)
5,000	5,000.7	(89.1)
7,500	7,501.4	(109.1)

$n = 100{,}000$		
k	μ_k	(σ_k)
12,500	12,460.5	(429.7)
37,500	37,499.7	(379.1)
50,000	50,001.5	(322.8)
75,000	75,003.0	(418.1)

Figure 6: Relative frequencies for $n = 10^i, i = 2..5$ and some fixed k values.



$n = 1,000$			
k	Thr	μ_k	(σ_k)
200	3	164.5	(41.0)
	31	197.0	(29.0)
	62	198.6	(22.3)
500	3	500.0	(31.5)
	31	491.8	(23.1)
	62	500.0	(18.3)
900	3	872.0	(46.6)
	31	895.3	(27.5)
	62	899.4	(20.5)

Figure 7: Quality of selection as a function of the *Threshold* value.

[1]. Its precision is strongly based on the good behavior of the approximate median selection algorithm. As the experimental section shows, the quality of the algorithm output becomes better and better for larger input sizes.

In order to get an improving in the quality of results, our future research is oriented to find efficient ways to manipulate larger blocks rather than triplets when selecting for the k-th element. In particular, an alternative method to treat the extremes of the range of selection to get higher precision is under investigation. We are also studying how to apply the technique presented here for the linear case to other higher dimension domains.

References

[1] S. Battiato, D. Cantone, D. Catalano, G. Cincotti, and M. Hofri. An efficient algorithm for the approximate median selection problem. Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC 2000, Rome, Italy). *Lecture Notes in Computer Science*, Vol. 1767, pp. 226–238, 2000.

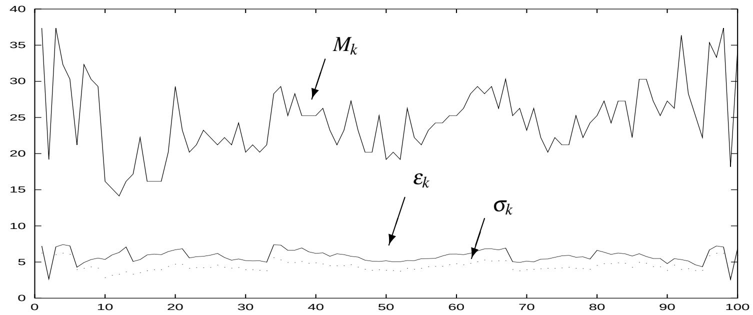
[2] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, Vol. 7, pp. 448–461, 1973.

[3] S. Bent and J. John. Finding the median requires $2n$ comparisons. Proceedings of the 17th ACM Symposium on Theory of Computing, pp. 213–216, 1985.

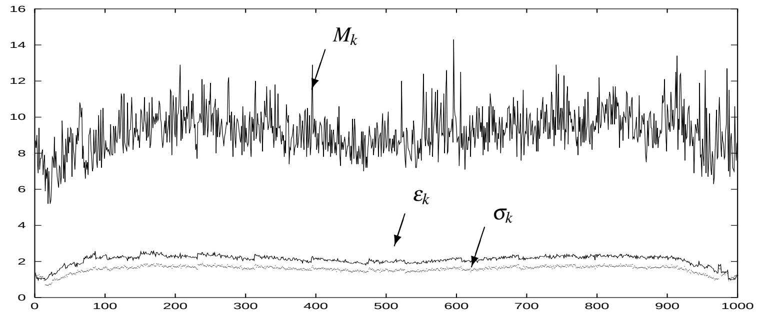
[4] S. Carlsson and M. Sundstrom. Linear-time in-place selection in less than $3n$ comparisons. Proceedings of the 6th International Symposium on Algorithms and Computation (ISAAC 95, Cairns, Australia). *Lecture Notes in Computer Science*, Vol. 1004, pp. 245–253, 1995.

[5] W. Cunto and J.I. Munro. Average case selection. *Journal of the ACM*, Vol. 36, pp. 270–279, 1989.

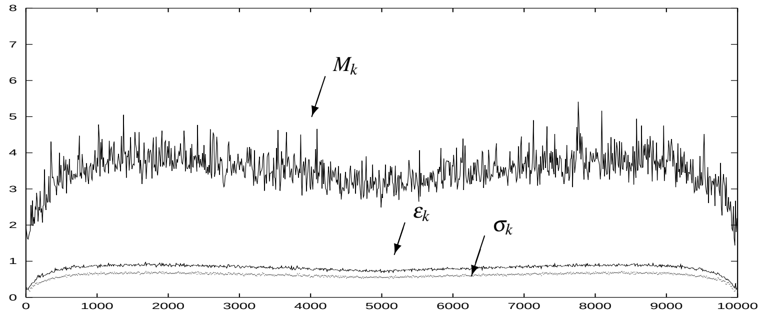
Percentage distance error



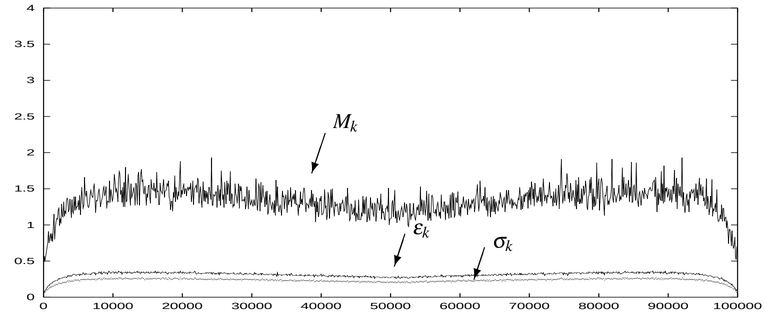
$n = 100$	
$E[\varepsilon_k]$	5.81401
$\sigma[\varepsilon_k]$	(0.85255)
$E[\sigma_k]$	4.41050
$\sigma[\sigma_k]$	(0.71881)
$\max(M_k)$	37.37



$n = 1{,}000$	
$E[\varepsilon_k]$	2.08990
$\sigma[\varepsilon_k]$	(0.28975)
$E[\sigma_k]$	1.57783
$\sigma[\sigma_k]$	(0.19871)
$\max(M_k)$	14.30



$n = 10{,}000$	
$E[\varepsilon_k]$	0.80927
$\sigma[\varepsilon_k]$	(0.10794)
$E[\sigma_k]$	0.61076
$\sigma[\sigma_k]$	(0.07992)
$\max(M_k)$	5.41



$n = 100{,}000$	
$E[\varepsilon_k]$	0.30998
$\sigma[\varepsilon_k]$	(0.03797)
$E[\sigma_k]$	0.23406
$\sigma[\sigma_k]$	(0.02825)
$\max(M_k)$	1.93

Figure 8: Percentage distance error for $n = 10^i$, with $i = 2..5$.

[6] D. Dor and U. Zwick. Selecting the median. Proceedings of the 6th ACM-SIAM Symposium on

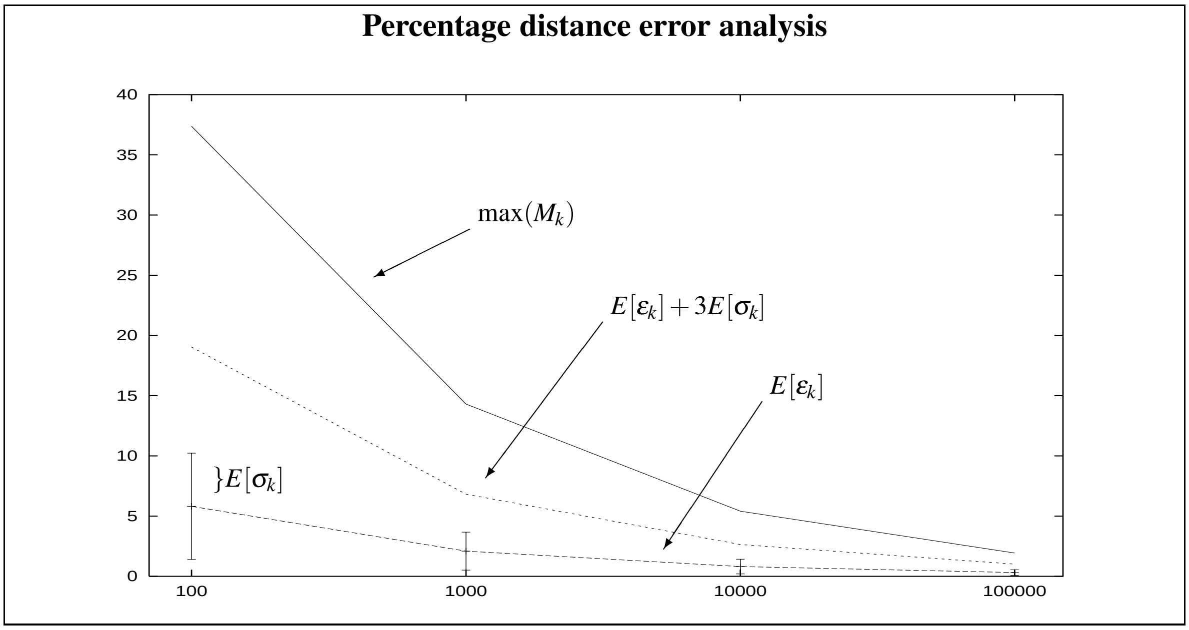


Figure 9: Asymptotically behaviour of percentage distance error w.r.t. n.

Discrete Algorithms (SODA 95, San Francisco, California). *SIAM Journal on Computing*, Vol. 28, pp. 1722–1758, 1999.

[7] R.W. Floyd and R.L. Rivest. Expected time bounds for selection. *Communications of the ACM*, Vol. 18, pp. 165–172, 1975.

[8] C.A.R. Hoare. Algorithm 63 (partition) and algorithm 65 (find). *Communications of the ACM*, Vol. 4, pp. 321–322, 1961.

[9] T.W. Lai and D. Wood. Implicit Selection. Proceedings of the Scandinavian Workshop on Algorithm Theory (SWAT 88, Halmstad, Sweden). *Lecture Notes in Computer Science*, Vol. 318, pp. 18–23, 1988.

[10] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, Vol. 31, pp. 1192–1201, 1988. Updated in *Communications of the ACM*, Vol. 36, pp. 108–110, 1993.

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