An Efficient Algorithm for the Approximate Median Selection Problem

S. Battiato[1], D. Cantone[1], D. Catalano[1], G. Cincotti[1], and M. Hofri[2]

[1] Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I–95125 Catania, Italy
{battiato,cantone,catalano,cincotti}@cs.unict.it

[2] Department of Computer Science, WPI
100 Institute Road, Worcester MA 01609-2280, USA
hofri@cs.wpi.edu

Abstract. We present an efficient algorithm for the approximate median selection problem. The algorithm works *in-place*; it is fast and easy to implement. For a large array it returns, with high probability, a very close estimate of the true median. The running time is linear in the length n of the input. The algorithm performs fewer than $\frac{4}{3}n$ comparisons and $\frac{1}{3}n$ exchanges on the average. We present analytical results of the performance of the algorithm, as well as experimental illustrations of its precision.

Keywords: Approximation algorithms, in-place algorithms, median selection, analysis of algorithms.

1. Introduction

In this paper we present an efficient algorithm for the *in-place* approximate median selection problem. There are several works in the literature treating the exact median selection problem (cf. [BFP*73], [DZ99], [FJ80], [FR75], [Hoa61], [HPM97]). Various in-place median finding algorithms have been proposed. Traditionally, the "comparison cost model" is adopted, where the only factor considered in the algorithm cost is the number of key-comparisons. The best upper bound on this cost found so far is nearly $3n$ comparisons in the worst case (cf. [DZ99]). However, this bound and the nearly-as-efficient ones share the unfortunate feature that their nice asymptotic behaviour is "paid for" by extremely involved implementations.

The algorithm described here approximates the median with high precision and lends itself to an immediate implementation. Moreover, it is quite fast: we show that it needs fewer than $\frac{4}{3}n$ comparisons and $\frac{1}{3}n$ exchanges on the average and fewer than $\frac{3}{2}n$ comparisons and $\frac{1}{2}n$ exchanges in the *worst-case*. In addition to its sequential efficiency, it is very easily parallelizable due to the low level of data contention it creates.

The usefulness of such an algorithm is evident for all applications where it is sufficient to find an approximate median, for example in some heapsort variants (cf. [Ros97], [Kat96]), or for median-filtering in image representation. In addition, the analysis of its precision is of independent interest.

We note that the procedure *pseudomed* in [BB96, $\S$7.5] is similar to performing just one iteration of the algorithm we present (using quintets instead of triplets), as an aid in deriving a (precise) selection procedure.

In a companion paper we show how to extend our method to approximate general k-selection.

All the works mentioned above—as well as ours—assume the selection is from values stored in an array in main memory. The algorithm has an additional property which, as we found recently, has led to its being discovered before, albeit for solving a rather different problem. As is apparent on reading the algorithms presented in Section 2, it is possible to perform the selection in this way "on the fly," without keeping all the values in storage. At the extreme case, if the values are read in one-by-one, the algorithm only uses $\approx 4\log_3 n$ positions (including $\lfloor \log_3 n \rfloor$ loop variables). This way of performing the algorithm is described in [RB90], in the context of estimating the median of an unknown distribution. The authors show there that the value thus selected is a consistent estimator of the desired parameter. They need pay no attention (and indeed do not) to the relation between the value the algorithm selects and the actual sample median. The last relation is the center point of interest for us. Curiously, Weide notes in [Wei78] that this approach provides an approximation of the sample median, though no analysis of the bias is provided. See [HM95] for further discussion of the method of Rousseeuw and Bassett, and numerous other treatments of the statistical problem of low-storage quantile (and in particular median) estimation.

In Section 2 we present the algorithm. Section 3 provides analysis of its run-time. In Section 4, to show the soundness of the method, we present a probabilistic analysis of the precision of its median selection. Since it is hard to glean the shape of the distribution function from the analytical results, we provide computational evidence to support the conjecture that the distribution is asymptotically normal. In Section 5 we illustrate the algorithm with a few experimental results, which also demonstrate its robustness. Section 6 concludes the paper with suggested directions for additional research.

An extended version of this paper is available by anonymous `ftp` from `ftp://ftp.cs.wpi.edu/pub/techreports/99-26.ps.gz`.

2. The Algorithm

It is convenient to distinguish two cases:

2.1 The size of the input is a power of 3: $n = 3^r$

Let $n = 3^r$ be the size of the input array, with an integer r. The algorithm proceeds in r stages. At each stage it divides the 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. To incur the fewest number of exchanges we do not move the chosen elements from their original triplets. This adds some index manipulation operations, but is typically advantageous. (While the order of the elements is disturbed, the contents of the array is unchanged).

```
Approximate Median Algorithm (1)

Triplet_Adjust($A$, $i$, Step)
    Let $j$= $i$+Step and $k$= $i$+2 · Step; this procedure moves the median of a triplet of
    terms at locations $i,\ j,\ k$ to the middle position.
    if ($A[i] < A[j]$)
      then
          if ($A[k] < A[i]$) then Swap($A[i], A[j]$);
          else if ($A[k] < A[j]$) then Swap($A[j]$, $A[k]$);
      else
          if ($A[i] < A[k]$) then Swap($A[i], A[j]$);
          else if ($A[k] > A[j]$) then Swap($A[j]$, $A[k]$);

Approximate_Median($A$, $r$)
    This procedure returns the approximate median of the array $A[\,0,\ ...,\ 3^r - 1]$.
    Step=1;    Size=$3^r$;
    repeat $r$ times
          $i$=(Step$-$1)/2;
          while $i <$ Size do
                Triplet_Adjust($A$, $i$, Step);
                $i$=$i$+(3·Step);
          end while;
          Step = 3·Step;
    end repeat;
    return $A[($Size$ - 1)/2]$;
```

Fig. 1. Pseudo-code for the approximate median algorithm, $n = 3^r, r \in \mathbb{N}$.

In Fig. 1 we show pseudo-code for the algorithm. The procedure *Triplet_Adjust* finds the median of triplets with elements that are indexed by two parameters: one, i, denotes the position of the leftmost element of triplet in the array. The second parameter, *Step*, is the relative distance between the triplet elements. This approach requires that when the procedure returns, the median of the triplet is in the middle position, possibly following an exchange. The *Approximate_Median* algorithm simply consists of successive calls to the procedure.

2.2 The extension of the algorithm to arbitrary-size input

The method described in the previous subsection can be generalized to array sizes which are not powers of 3. The basic idea is similar. Let n be the input size at the current stage, where

$$n = 3 \cdot t + k, \qquad\qquad k \in \{0, 1, 2\}.$$

We divide the input into $(t - 1)$ triplets and a $(3 + k)$-tuple. The $(t - 1)$ triplets are processed by the same *Triplet_Adjust* procedure described above. The last tuple is sorted (using an adaptation of selection-sort) and the median is extracted. The algorithm continues iteratively using the results of each stage as input for a new one. This is done until the number of local medians falls below a small fixed threshold. We then sort the remaining elements and obtain the median. To symmetrize the algorithm, the array is scanned from left to right during the first iteration, then from right to left on the second one, and so on, changing the scanning sense at each iteration. This should reduce the perturbation due to the different way in which the medians from the $(3 + k)$-tuples are selected and improve the precision of the algorithm. Note that we chose to select the second element out of four as the median (2 out of 1..4). We show pseudo-code for the general case algorithm in Fig. 2.

Approximate Median Algorithm (2)

```
Selection_Sort ($A$, Left, Size, Step)
  This procedure sorts Size elements of the array $A$ located at positions Left, $Left + Step$,
  $Left + 2 \cdot Step, \ldots, Left + (Size - 1) \cdot Step$.

  for ($i = Left$ ; $i < Left + (Size - 1) \cdot Step$; $i = i + Step$)
        $Min = i$;
        for ($j = i + Step$; $j < Left + Size \cdot Step$; $j = j + Step$)
              if ($A[j] < A[min]$) then $min = j$;
        end for;
        Swap($A[i]$, $A[min]$);
  end for;
Approximate_Median_AnyN ($A$, Size)
   This procedure returns the approximate median of the array $A[0, \ldots, Size - 1]$.
   $LeftToRight = False$;   $Left = 0$;   $Step = 1$;
   while ($Size > Threshold$) do
         $LeftToRight$ = Not ($LeftToRight$);
         $Rem = (Size \bmod 3)$;
         if ($LeftToRight$) then $i = Left$;
                      else $i = Left + (3 + Rem) \cdot Step$;
         repeat $(Size/3 - 1)$ times
              Triplet_Adjust ($A$, $i$, $Step$);
              $i = i + 3 \cdot Step$;
         end repeat;
         if ($LeftToRight$) then $Left = Left + Step$;
                      else $i = Left$;
                           $Left = Left + (1 + Rem) \cdot Step$;
         Selection_Sort ($A$, $i$, $3 + Rem$, $Step$);
         if ($Rem = 2$) then
                  if ($LeftToRight$) then Swap($A[i + Step]$, $A[i + 2 \cdot Step]$)
                                 else Swap($A[i + 2 \cdot Step]$, $A[i + 3 \cdot Step]$);
         $Step = 3 \cdot Step$; $Size = Size/3$;
   end while;
   Selection_Sort ($A$, Left, Size, Step);
   return $A[Left + Step \cdot \lfloor (Size - 1)/2 \rfloor]$;
```

Fig. 2. Pseudo-code for the approximate median algorithm, any $n \in \mathbb{N}$.

Note: The reason we use a terminating tuple of size 4 or 5, rather than 1 or 2, is to keep the equal spacing of elements surviving one stage to the next.

The procedure $Selection\_Sort$ takes as input four parameters: the array A, its size and two integers, *Left* and *Step*. At each iteration *Left* points to the leftmost element of the array which is in the current input, and *Step* is the distance between any two successive elements in this input.

There are several alternatives to this approach for arbitrary-sized input. An attractive one is described in [RB90], but it requires *additional storage* of approximately $4\log_3 n$ memory locations.

3. Run-time Analysis: Counting Moves and Comparisons

Most of the work of the algorithm is spent in *Triplet_Adjust*, comparing values and exchanging elements within triplets to locate their medians. We compute now the number of comparisons and exchanges performed by the algorithm *Approximate_Median.*

Like all reasonable median-searching algorithms, ours has running-time which is linear in the array size. It is distinguished by the simplicity of its code, and hence it is extremely efficient. We consider first the algorithm described in Fig. 1.

Let $n = 3^r$, $r \in \mathbb{N}$, be the size of a randomly-ordered input array. We have the following elementary results:

Theorem 1. *Given an input of size n, the algorithm Approximate_Median performs fewer than $\frac{4}{3}n$ comparisons and $\frac{1}{3}n$ exchanges on the average.* □

Proof: Consider first the *Triplet_Adjust* subroutine. In the following table we show the number of comparisons and exchanges, $\mathcal{C}_3$ and $\mathcal{E}_3$, for each permutation of three distinct elements:

A[i]	A[i+Step]	A[i+2*Step]	Comparisons	Exchanges
1	2	3	3	0
1	3	2	3	1
2	1	3	2	1
2	3	1	2	1
3	1	2	3	1
3	2	1	3	0

Clearly, assuming all orders equally likely, we find Pr$(\mathcal{C}_3 = 2) = 1 -$ Pr$(\mathcal{C}_3 = 3) = 1/3$, and similarly Pr$(\mathcal{E}_3 = 0) = 1 -$ Pr$(\mathcal{E}_3 = 1) = 1/3$, with expected values $E[\mathcal{E}_3] = 2/3$ and $E[\mathcal{C}_3] = 8/3$.

To find the work of the entire algorithm with an input of size n, we multiply the above by $T(n)$, the number of times the subroutine *Triplet_Adjust* is executed. This number is deterministic. We have $T(1) = 0$ and $T(n) = \frac{n}{3} + T(\frac{n}{3})$, for $n > 1$; for n which is a power of 3 the solution is immediate: $T(n) = \frac{1}{2}(n-1)$.

Let Π_n be the number of possible inputs of size n and let Σ_n be the total number of comparisons performed by the algorithm on all inputs of size n.

The average number of comparisons for all inputs of size n is:

$$C(n) = \frac{\Sigma_n}{\Pi_n} = \frac{16 \cdot T(n) \cdot n!/3!}{n!} = \frac{4}{3}(n-1).$$

To get Σ_n we count all the triplets considered for all the Π_n inputs, i.e. $n! \cdot T(n)$; for each triplet we consider the cost over its 3! permutations (the factor 16 is the cost for the 3! permutations of each triplet[1]).

The average number of exchanges can be shown analogously, since two out of three permutations require an exchange.

By picking the "worst" rows in the table given in the proof of Theorem 1, it is straightforward to verify also the following:

Theorem 2. *Given an input of size $n = 3^r$, the algorithm Approximate_Median performs fewer than $\frac{3}{2}n$ comparisons and $\frac{1}{2}n$ exchanges in the worst-case.* □

For an input size which is a power of 3, the algorithm of Fig. 2 performs nearly the same operations as the simpler algorithm – in particular, it makes the same key-comparisons, and selects the same elements. For $\log_3 n \notin \mathbb{N}$, their performance only differs on one tuple per iteration, hence the leading term (and its coefficient) in the asymptotic expression for the costs is the same as in the simpler case.

The non-local algorithm described in [RB90] performs exactly the same number of comparisons as above but always moves the selected median. The overall run-time cost is very similar to our procedure.

4. Probabilistic Performance Analysis

4.1 Range of Selection

It is obvious that not all the input array elements can be selected by the algorithm — *e.g.*, the smallest one is discarded in the first stage. Let us consider first the algorithm of Fig. 1 (*i.e.* when n is a power of 3). Let $v(n)$ be the number of elements from the lower end (alternatively – upper end, since the *Approximate_Median* algorithm has bilateral symmetry) of the input which *cannot be selected* out of an array of n elements. It is easy to verify (by observing the tree built with the algorithm) that $v(n)$ obeys the following recursive inequality:

$$v(3) = 1 \quad , \qquad v(n) \geq 2v(n/3) + 1. \tag{1}$$

Moreover, when $n = 3^r$, the equality holds. The solution of the recursive *equation*,

$$\{v(3) = 1; \quad v(n) = 2v(n/3) + 1\}$$

[1] Alternatively, we can use the following recurrence: $C(1) = 0$ and $C(n) = \frac{n}{3} \cdot c + C(\frac{n}{3})$, for $n > 1$, where $c = \frac{16}{6}$ is the average number of comparisons of *Triplet_Adjust* (because all the 3! permutations are equally likely).

is the following function

$$v(n) = n^{\log_3 2} - 1 = 2^{\log_3 n} - 1.$$

Let x be the output of the algorithm over an input of n elements. From the definition of $v(n)$ it follows that

$$v(n) < rank(x) < n - v(n) + 1. \qquad (2)$$

The second algorithm behaves very similarly (they perform the same operations when $n = 3^r$) and the range function $v(n)$ obeys the same recurrence.

Unfortunately not many entries get thus excluded. The range of possible selection, as a fraction of the entire set of numbers, increases promptly with n. This is simplest to illustrate with n that is a power of 3. Since $v(n)$ can be written as $2^{\log_3 n} - 1$, the ratio $v(n)/n$ is approximately $(2/3)^{\log_3 n}$. Thus, for $n = 3^3 = 27$, where the smallest (and largest) 7 numbers cannot be selected, 52% of the range is excluded; the comparable restriction is 17.3% for $n = 3^6 = 729$ and only 1.73% for $n = 3^{12} = 531441$.

The true state of affairs, as we now proceed to show, is much better: while the possible range of choice is wide, the algorithm zeroes in, with overwhelming probability, on a very small neighborhood of the true median.

4.2 Probabilities of Selection

The most telling characteristic of the algorithms is their precision, which can be expressed via the probability function

$$P(z) = \Pr[zn < rank(x) < (1-z)n + 1], \qquad (3)$$

for $0 \leq z \leq 1/2$, which describes the closeness of the selected value to the true median.

The purpose of the following analysis is to show the behavior of this distribution. We consider n which is a power of 3.

Definition 1. *Let $q_{a,b}^{(r)}$ be the number of permutations, out of the $n! = 3^r!$ possible ones, in which the entry which is the a^{th} smallest in the set is: (1) selected, and (2) becomes the b^{th} smallest in the next set, which has $\frac{n}{3} = 3^{r-1}$ entries.*

It turns out that this quite narrow look at the selection process is all we need to characterize it completely.

It can be shown that

$$q_{a,b}^{(r)} = 2n(a-1)!(n-a)!\binom{\frac{n}{3}-1}{b-1}3^{a-b-1} \times \sum_i \binom{b-1}{i}\binom{\frac{n}{3}-b}{a-2b-i}\frac{1}{9^i} \qquad (4)$$

(for details, see [BCC*99]).

It can also be seen that $q_{a,b}^{(r)}$ is nonzero for $0 \leq a - 2b \leq \frac{n}{3} - 1$ only. The sum is expressible as a Jacobi polynomial, $\left(\frac{8}{9}\right)^{a-2b} P_{a-2b}^{(u,v)}\left(\frac{5}{4}\right)$, where $u = 3b - a - 1, v = \frac{n}{3} + b - a$, and a simpler closed form is unlikely.

Let $p_{a,b}^{(r)}$ be the probability that item a gets to be the b^{th} smallest among those selected for the next stage. Since the $n! = 3^r!$ permutations are assumed to be equally likely, we have $p_{a,b}^{(r)} = q_{a,b}^{(r)}/n!$:

$$p_{a,b}^{(r)} = \frac{2}{3}\frac{3^{-b}\binom{\frac{n}{3}-1}{b-1}}{3^{-a}\binom{n-1}{a-1}} \times \sum_i \binom{b-1}{i}\binom{\frac{n}{3}-b}{a-2b-i}\frac{1}{9^i}$$
$$= \frac{2}{3}\frac{3^{-b}\binom{\frac{n}{3}-1}{b-1}}{3^{-a}\binom{n-1}{a-1}} \times [z^{a-2b}](1+\frac{z}{9})^{b-1}(1+z)^{\frac{n}{3}-b}. \tag{5}$$

This allows us to calculate the center of our interest: The probability $P_a^{(r)}$, of starting with an array of the first $n = 3^r$ natural numbers, and having the element a ultimately chosen as approximate median. It is given by

$$P_a^{(r)} = \sum_{b_r} p_{a,b_r}^{(r)} P_{b_r}^{(r-1)} = \sum_{b_r, b_{r-1}, \cdots, b_3} p_{a,b_r}^{(r)} p_{b_r,b_{r-1}}^{(r-1)} \cdots p_{b_3,2}^{(2)} \, , \tag{6}$$

where $2^{j-1} \le b_j \le 3^{j-1} - 2^{j-1} + 1$, for $j = 3, 4, \ldots, r$.

Some telescopic cancellation occurs when the explicit expression for $p_{a,b}^{(r)}$ is used here, and we get

$$P_a^{(r)} = \left(\frac{2}{3}\right)^r \frac{3^{a-1}}{\binom{n-1}{a-1}} \sum_{b_r, b_{r-1}, \cdots, b_3} \prod_{j=2}^{r} \sum_{i_j \ge 0} \binom{b_j - 1}{i_j}\binom{3^{j-1} - b_j}{b_{j+1} - 2b_j - i_j}\frac{1}{9^{i_j}} \, . \tag{7}$$

As above, each b_j takes values in the range $[2^{j-1} \ldots 3^{j-1} - 2^{j-1} + 1]$, $b_2 = 2$ and $b_{r+1} \equiv a$ (we could let all b_j take all positive values, and the binomial coefficients would produce nonzero values for the required range only). The probability $P_a^{(r)}$ is nonzero for $v(n) < a < n - v(n) + 1$ only.

This distribution has so far resisted our attempts to provide an analytic characterization of its behavior. In particular, while the examples below suggest very strongly that as the input array grows, it approaches the normal distribution, this is not easy to show analytically. (See Section 4.4 of [BCC*99] for an approach to gain further information about the large-sample distribution.)

4.3 Examples

We computed $P_a^{(r)}$ for several values of r. Results for a small array ($r = 3,\ n = 27$) are shown in Fig.3. By comparison, with a larger array ($r = 5,\ n = 243$, Fig. 4) we notice the relative concentration of the likely range of selection around the true median. In terms of these probabilities the relation (3) is:

$$\sum_{\lfloor zn \rfloor < a < \lceil (1-z)n \rceil + 1} P_a^{(r)} \tag{8}$$

where $0 \le z < \frac{1}{2}$.

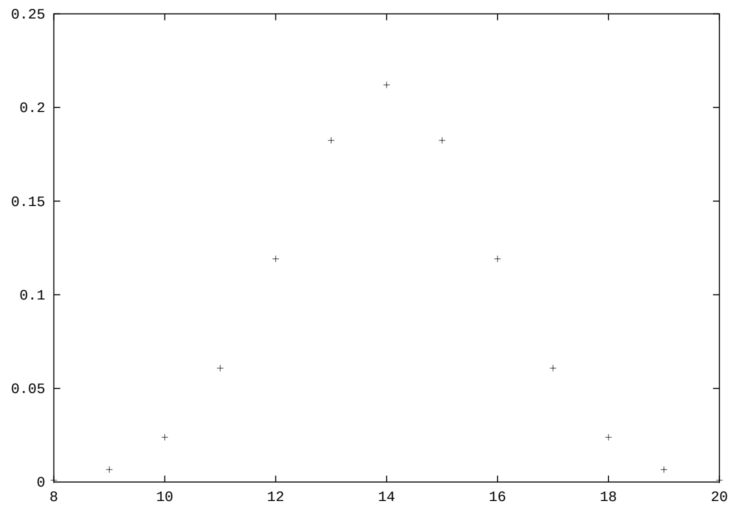

Fig. 3. Plot of the median probability distribution for n=27.

We chose to present the effectiveness of the algorithm by computing directly from equation (7) the statistics of the absolute value of the bias of the returned approximate median, $D_n \equiv |X_n - \mathcal{M}_d(n)|$ (where $\mathcal{M}_d(n)$ is the true median, $(n+1)/2$). We compute its mean (Avg.) and standard deviation, denoted by σ_d.

A measure of the improvement of the selection effectiveness with increasing (initial) array size n is seen from the variance ratio $\sigma_d/\mathcal{M}_d(n)$. This ratio may be viewed as a measure of the expected relative error of the approximate median selection algorithm.

Numerical computations produced the numbers in Table 1; note the trend in the rightmost column. (This trend is the basis for the approach examined in Section 4.4 of [BCC*99].)

n	$r = \log_3 n$	Avg.	σ_d	$\sigma_d/\sqrt{n}$
9	2	0.428571	0.494872	0.164957
27	3	1.475971	1.184262	0.227911
81	4	3.617240	2.782263	0.309140
243	5	8.096189	6.194667	0.397388
729	6	17.377167	13.282273	0.491958
2187	7	36.427027	27.826992	0.595034

Table 1. Statistics of the median selection as function of array size.

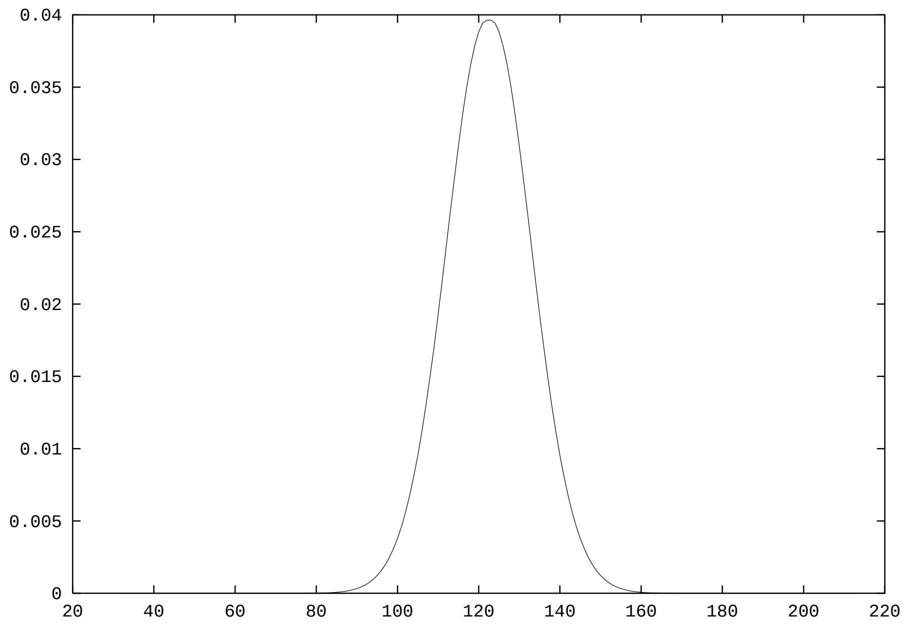

Fig. 4. Plot of the median probability distribution for n=243.

5. Experimental Results

In this section we present empirical results, demonstrating the effectiveness of the algorithms – also for the cases which our analysis does not handle directly. 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 lists were permuted using the pseudo-random number generator suggested by Park and Miller in 1988 and updated in 1993 [PM88]. The algorithm was run on random arrays of sizes that were powers of 3, $n = 3^r$, with $r \in \{3,\ 4, \ldots,\ 11\}$, and others. The entry keys were always the integers $1, \ldots, n$.

The following tables present results of such runs. They report the statistics of D_n, the absolute value of the bias of the approximate median. For each returned result we compute its distance from the correct media $\frac{n+1}{2}$. The units we use in the tables are "normalized" values of D_n, denoted by $d_{\%}$; these are percentiles of the possible range of error of the algorithm: $d_{\%} = 100 \times \frac{D_n}{(n-1)/2}$. The extremes are $d_{\%} = 0$ when the true median is returned – and it would have been 100 if it were possible to return the smallest (or largest) elements. (But relation (2) shows that $d_{\%}$ can get arbitrarily close to 100 as n increases, but never quite reach it). Moreover, the probability distributions of the last section suggest, as illustrated in Figure 4, that such deviations are extremely unlikely. Table 2 shows selected results, using a threshold value of 8. All experiments used a sample size of 5000, throughout.

n	Avg.	σ	Avg. + 2 σ	Rng(95%)	(Min-Max)
50	10.27	7.89	26.05	24.49	0.00–44.90
100	8.45	6.63	21.70	20.20	0.00–48.48
3^5	6.74	5.13	17.00	16.53	0.00–38.02
500	5.80	4.41	14.63	14.03	0.00–29.06
3^6	4.83	3.71	12.26	12.09	0.00–24.73
1000	4.70	3.66	12.02	11.61	0.00–23.62
3^7	3.32	2.54	8.41	8.05	0.00–16.83
5000	2.71	2.10	6.91	6.72	0.00–17.20
3^8	2.31	1.75	5.81	5.67	0.00–11.95
10000	2.53	1.86	6.24	6.04	0.00–11.38
3^9	1.58	1.18	3.94	3.86	0.00–6.78

Table 2. Simulation results for $d_{\%}$, fractional bias of approximate median. Sample size = 5000.

The columns are: n – array size; Avg. – the average of $d_{\%}$ over the sample; σ – the sample standard-error of $d_{\%}$; Rng. (95%) – the size of an interval symmetric around Avg. that contains 95% of the returned values; the last column gives the extremes of $d_{\%}$ that were observed. In the rows that correspond to those of Table 1, the agreement of the Avg. and σ columns is excellent (the relative differences are under 0.5%). Columns 4 and 5 suggest the closeness of the median distribution to the Gaussian, as shown above.

All the entries show the critical dependence of the quality of the selection on the size of the initial array. In the following table we report the data for different values of n with sample size of 5000, varying the threshold.

	t=8			t=26			t=80		
n	Avg.	σ	Rng. (95%)	Avg.	σ	Rng. (95%)	Avg.	σ	Rng. (95%)
100	8.63	6.71	22.22	6.80	5.31	16.16	4.64	3.61	12.12
500	5.80	4.41	14.43	4.40	3.30	10.82	3.22	2.40	7.62
1000	4.79	3.67	12.01	3.80	2.88	9.41	2.98	2.27	7.41
10000	2.54	1.87	6.05	1.67	1.28	4.14	1.40	1.06	3.44

Table 3. Quality of selection as a function of threshold value.

As expected, increasing the threshold—the maximal size of an array which is sorted, to produce the exact median of the remaining terms—provides better selection, at the cost of rather larger processing time. For large n, threshold values beyond 30 provide marginal additional benefit. Settling on a correct trade-off here is a critical step in tuning the algorithm for any specific application.

Finally we tested for the relative merit of using quintets rather than triplets when selecting for the median. In this case $n = 1000$, Threshold=8, and sample size=5000.

	Avg.	σ	Avg. + 2 σ	Rng(95%)	(Min-Max)
Triplets	4.70	3.66	12.02	11.61	0.00–23.62
Quintets	3.60	2.74	9.08	9.01	0.00–16.42

Table 4. Comparing selection via triplets and quintets.

6. Conclusion

We have presented an approximate median finding algorithm, and an analysis of its characteristics. Both can be extended. In particular, the algorithm can be adapted to select an approximate k^{th}-element – for any $k \in [1, n]$. The analysis of Section 4 can be extended to show how to compute with the exact probabilities, as given in equation (7). Also, the limiting distribution of the bias D with respect to the true median – while we know it is extremely close to a gaussian distribution, we have no efficient representation for it yet.

Acknowledgments

The derivations of Section 4.4 in [BCC*99] are largely due to Svante Janson. The references [HM95], [RB90], and [Wei78] were provided by Reza Modaress. We are grateful for their contributions.

The authors wish also to thank an anonymous referee for some useful comments.

References

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

[BB96] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, Englewood Cliffs, NJ 1996.

[BCC*99] S. Battiato, D. Cantone, D. Catalano, G. Cincotti, and M. Hofri. *An Efficient Algorithm for the Approximate Median Selection Problem*. Technical Report WPI-CS-TR-99-26, Worcester Polytechnic Institute, October 1999. Available from `ftp://ftp.cs.wpi.edu/pub/techreports/99-26.ps.gz`.

[BFP*73] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and Systems Sciences*, 7(4):448–461, 1973.

[CS87] S. Carlsson, M. Sundstrom. Linear-time In-place Selection in Less than $3n$ Comparisons - Division of Computer Science, Lulea University of Technology, S-971 87 LULEA, Sweden.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

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

[Dra67] Alvin W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill, 1967.

[DZ99] D. Dor, and U. Zwick. Selecting the Median. *SIAM Jour. Comp.*, 28(5):1722–1758, 1999.

[FJ80] G.N. Frederickson, D.B. Johnson. Generalized Selection and Ranking. *Proceedings STOC-SIGACT,* Los Angeles CA, 12:420–428, 1980.

[Fre90] G.N. Frederickson. The Information Theory Bound is Tight for selection in a heap. *Proceedings STOC-SIGACT*, Baltimore MD, 22:26–33, 1990.

[FR75] R.W. Floyd, R.L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

[Hoa61] C.A.R. Hoare. Algorithm 63(partition) and algorithm 65(find). *Communications of the ACM*, 4(7):321–322, 1961.

[Hof95] M. Hofri, *Analysis of Algorithms: Computational Methods & Mathematical Tools*, Oxford University Press, New York (1995).

[HM95] C. Hurley and Reza Modarres: Low-Storage quantile estimation, *Computational Statistics*, 10:311–325, 1995.

[HPM97] P. Kirschenhofer, C. Martinez, and H. Prodinger. Analysis of Hoare's Find algorithm with median-of-three partition. *Random Structures and Algorithms*, 10:143–156, 1997.

[Kat96] J. Katajainen. *The Ultimate Heapsort*, DIKU Report 96/42, Department of Computer Science, Univ. of Copenhagen, 1996.

[Knu98] D.E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, 2nd Ed. 1999.

[LW88] T.W. Lai, and D. Wood. Implicit Selection. *Scandinavian Workshop on Algorithm Theory (SWAT88)*:18–23, LNCS 38 Springer-Verlag, 1988.

[Meh84] K. Mehlhorn. *Sorting and Searching, Data Structures and Algorithms*, volume 1. Springer-Verlag, 1984.

[Noz73] A. Nozaky. Two Entropies of a Generalized Sorting Problems. *Journal of Computer and Systems Sciences*, 7(5):615–621, 1973.

[PM88] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988. Updated in *Communications of the ACM*, 36(7):108–110, 1993.

[Ros97] L. Rosaz.*Improving Katajainen's Ultimate Heapsort*, Technical Report N.1115, Laboratoire de Recherche en Informatique, Université de Paris Sud, Orsay, 1997.

[RB90] P.J. Rousseeuw and G.W. Bassett: The remedian: A robust averaging method for large data sets. *Jour. Amer. Statist. Assoc*, 409:97–104, 1990.

[SJ99] Savante Janson – Private communication, June 1999.

[SPP76] A. Schonhage, M. Paterson, and N. Pippenger. Finding the median. *Journal of Computer and Systems Sciences*, 13:184–199, 1976.

[Wei78] B. W. Weide. Space efficient on-line selection algorithm. *Proceedings of the 11th symposium of Computer Science and Statistics, on the interface*, 308–311. (1978).