# An Efficient Algorithm for the Approximate Median Selection Problem

S. Battiato, D. Cantone, D. Catalano and G. Cincotti
*Università di Catania, Dipartimento di Matematica,*
*Viale A. Doria 6, I-95125 Catania, Italy*
e-mail:{battiato,cantone,catalano,cincotti}@dipmat.unict.it

M. Hofri
*Department of Computer Science, WPI*
*100 Institute Road, Worcester MA 01609-2280*
e-mail:hofri@cs.wpi.edu

October 28, 2002

**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 good 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.

## 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. [BFPRT73], [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 behavior 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, §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 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 by Rousseeuw and Bassett 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 a central point of interest for us. 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 [RB90] 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, providing both precise (and intractable), and asymptotic versions. Section 5 provides computational results for refinements which are beyond our analyses. Section 6 concludes the paper with suggested directions for additional research.

## 2. The Algorithm

It is convenient to distinguish two cases, when the input size is an exact power of 3, and otherwise. We present two separate versions, but want to draw the reader's attention that the general one performs exactly

**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 A[i], A[j], A[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];
```

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

the same choices as the first, when the input size is right, with a very small amount of overhead, that the first algorithm skips.

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

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).

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 three in several ways. The simplest is as follows: Let the array size be *n*, where

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

Then the first $t$ triplets have their median extracted as before, using TRIPLET_ADJUST, and the $t$ selected medians, as well as the remaindered $k$ elements are forwarded to the next stage. In addition we keep alternating the direction from which we select those first triplets—either left-to-right, or right-to-left—so that no elements are remaindered for more than one iteration. Figure 2 shows pseudo-code for a slightly more elaborate scheme, where only the first $t-1$ triplets are processed as above, and the other, $(3+k)$-tuple, is sorted (using an adaptation of selection-sort) to extract its median. Otherwise they are the same: the algorithm continues iteratively using the results of each stage as input for a new one. This is done until the number of elements falls below a small fixed threshold. We then sort the elements and obtain their median as the final result. The alternation of direction is used here too. Note that we chose to select the second element out of four as the median (2 out of 1..4).

*Note:* The reason we use a terminating tuple of size 4 or 5, rather than the simpler case of 1 or 2, is to simplify keeping the equal spacing of elements surviving one stage to the next, which is needed for efficient in-situ processing.

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.

**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 · Step ....
    for (i = Left ; i < Left + (Size − 1) · Step; i = i + Step)
            Min = i;
            for (j = i + Step; j < Left + Size · Step; j = j + Step)
                    if (A[j].Key < A[min].Key) 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, ..., Size − 1].
    LeftToRight = False;   Left = 0;   Step = 1;
    while (Size > Threshold) do
        LeftToRight = Not (LeftToRight);
        Rem = (Size mod 3);
        if (LeftToRight) then i = Left;
                         else i = Left + (3 + Rem) · Step;
        repeat (Size/3 − 1) times
            Triplet_Adjust (A, i, Step);
            i = i + 3 · Step;
        end repeat;
        if (LeftToRight) then Left = Left + Step;
                         else i = Left;
                              Left = Left + (1 + Rem) · Step;
        SELECTION_SORT (A, i, 3 + Rem, Step);
        if (Rem = 2) then
                if (LeftToRight) then SWAP (A[i + Step], A[i + 2 · Step])
                                 else SWAP (A[i + 2 · Step], A[i + 3 · Step]);
        Step = 3 · Step; Size = Size/3;
    end while;
    SELECTION_SORT (A, Left, Size, Step);
    return A[Left + Step · ⌊(Size − 1)/2⌋];
```

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

# 3. Run-time Analysis

## 3.1 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, while being 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:

**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. □

**Comment:** A corollary of [FR75] is that an upper bound for the expected number of comparisons needed to select the *exact* median is $(3/2+o(1))n$. This bound is shown in [CM89] to be tight.

**Proof:** Consider first the TRIPLET_ADJUST subroutine. In the following table we show the number of comparisons and exchanges, $\mathbb{C}_3$ 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 $\Pr(\mathbb{C}_3 = 2) = 1 - \Pr(\mathbb{C}_3 = 3) = 1/3$, 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 $E[\mathbb{C}_3] = 8/3$.

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 three the solution is immediate: $T(n) = \frac{1}{2}(n-1)$.

Let $\Pi_n$ be the number of possible inputs of size $n$ and let $\Sigma_n$ 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 $\Sigma_n$ we count all the triplets considered for all the $\Pi_n$ 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*).

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 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. □

### 3.2 Extensions

For an input size which is a power of three, the algorithm of Figure 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.

The choice of deriving local medians from triplets is convenient, but arbitrary. We could use any fixed size $s$. This impacts the cost of the algorithm and its precision. Both the number of comparisons and the likelihood of an exchange increase with $s$. However, the number of the computations of these local medians decreases. For example, when $s = 5$, with any procedure to find the median, the mean number of comparisons required, $c$, is more than double the value for $s = 3$. The procedure that achieves the minimal worst-case number of comparisons needed to find the median of five does it in 6 comparisons. Since this procedure performs exactly the same sequence of operations for all permutations, the mean value for it is $c = 6$ comparisons as

*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).

well. With a different procedure, it is possible to find the median with a smaller mean comparison count, $c = 5\frac{13}{15}$, which is 2.3% lower — but then some permutations require 7 comparisons [Knu98, p. 217].

The total expected number of comparisons needed for an array of size $n = 5^r$ is derived from the equation

$$C_5(n) = \frac{n}{5}c + C_5(\frac{n}{5}), \quad C_5(1) = 0,$$

Hence $C_5(n) = c(n-1)/4$. We see that the number of median calculations is exactly one half of the number required for $s = 3$, and the expected number of exchanges—four out of five permutations require one—is $(n-1)/5$. Using the value $c = 6$ we find $C_5(n) \approx (3n)/2$, 12.5% higher than $C_3(n)$. Combined with the smaller number of exchanges, the increase in cost is quite small. The actual increase depends on the relative cost of comparison and exchange. (In fact, we are not comparing costs of these operations: the comparison-count represents also the overhead needed for most of the bookkeeping work the algorithm does). As we show in Section 5, this extra cost provides handsome return in precision of the selection. The trend continues: when we compute local medians for larger and larger basic blocks — we have better precision at increasing cost.

# 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 three). 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*,

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

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

$$v(n) < rank(x) < n - v(n) + 1. \tag{2}$$

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. The initial values are similar, but not identical. Moreover, they are not symmetrical with respect to numbers excluded from the lower and upper ends of the range. The reason is clearly that we need to select medians for even values of $n$.

We choose, arbitrarily, to take the second smallest out of four.

Let us use $\underline{v}(n)$ to denote the number of values excluded at the lower end, and $\overline{v}(n)$ for the number of those excluded from the upper end.

Both functions satisfy the same functional inequality as $v(n)$ does. The initial values for $\underline{v}(n)$ are $\underline{v}(3) = \underline{v}(4) = 1$, $\underline{v}(5) = 2$, and the solution is then:

$$\underline{v}(n) = \begin{cases} 3 \cdot 2^r - 1 & n = 5 \cdot 3^r + j,\ 0 \le j < 3,\ r > 0 \\ 2 & n = 5 \\ 2^{\lfloor \log_3 n \rfloor} - 1 & \text{otherwise} \end{cases} \tag{3}$$

Note that this is not a monotonic function. E.g.:

$$\underline{v}(4) = \underline{v}(6) = 1,\ \ \underline{v}(5) = 2.$$

$$\underline{v}(13) = \underline{v}(14) = \underline{v}(18) = 3,\ \ \underline{v}(15) = \underline{v}(17) = 5.$$

For the number excluded from the upper end of the range the situation is quite similar. The recurrence is again as above, $\overline{v}(n) = 2\overline{v}(\lfloor n/3 \rfloor) + 1$, but our choice of medians from sets of sizes 2 and 4 leads to a more complicated solution (the following holds for selecting `threshold` of Figure 2 in the range 2 to 5). We need to specify the following initial values:

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $\overline{v}(n)$ | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 |

For higher values we observe that any $n \ge 9$ can be uniquely represented as

$$n = k \cdot 3^r + \sum_{i=0}^{r-1} a_i \cdot 3^i, \qquad 3 \le k \le 8,\ \ 0 \le a_i \le 2,\ \ r \ge 1 \tag{4}$$

then $\overline{v}(n) = 2^r \cdot (\overline{v}(k) + 1) - 1$.

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 three. 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

No less important than the efficiency of the algorithm is its precision, which can be expressed by the probability function

$$P(z) = \Pr[zn < rank(x) < (1-z)n+1], \tag{5}$$

for $0 \le z \le 1/2$. This describes the closeness of the selected value to the true median.

The purpose of the following analysis is to calculate this distribution. We consider $n$ which is a power of three.

**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 will turn out that this quite narrow look at the selection process is all we need to characterize it completely.

The reader may find it best, when following the derivation, to imagine the data arranged in a way which is somewhat different than the one actually used by the algorithm: A permutation is seen as an arrangement of the first $n$ natural numbers in $\frac{n}{3} = 3^{r-1}$ successive triplets, that we index by $j$. The $j^{th}$ triplet, in positions $(3j-2,\ 3j-1,\ 3j)$, $\quad 1 \le j \le \frac{n}{3}$, provides one locally selected median-of-three, or three-median, that continues to the next stage. Only such permutations where the integer $a$ is the $b^{th}$ smallest three-median are admissible (contribute to $q_{a,b}^{(r)}$).

We count admissible permutations in the following steps:

(1) Count such permutations where the three-medians come out partially sorted, in increasing order. By partial sorting we mean that the leftmost $b-1$ three-medians are smaller than $a$ and the rightmost $\frac{n}{3} - b$ are larger than $a$.

(2) Account for this restriction: multiply by the number of rearrangements of each permutation constructed as in step (1).

Step (2) is easy to dispose of: step (1) fixes the position of the triplet where $a$ is the three-median, and allows $(b-1)!(\frac{n}{3}-b)!$ orders of the other triplets. Hence step (2) will contribute the factor $\frac{(n/3)!}{(b-1)!(\frac{n}{3}-b)!} = \frac{n}{3}\binom{\frac{n}{3}-1}{b-1}$.

To do (1) we need to account for the partial sortedness of the three-medians, and notice in which ways our arrangements restrict the steps that need to be done (we show below they imply certain restrictions on in-triplet ordering). The relative order of three-medians requires the following:
**A**—in each of the leftmost $b-1$ triplets as above, numbered $j \in [1,\ b)$, there are two values smaller than $a$ ("small values"). This guarantees that each three-median there is smaller than $a$. We call it a small triplet. One more small value must land in the $b^{th}$ triplet.
**B**—in each triplet numbered $j \in (b,\ \frac{n}{3}]$, there are two values larger than $a$ ("large values"). This guarantees that each three-median there is larger than $a$. We call it a large triplet. One more large value is in triplet $b$.

This guarantees the partial sortedness of the permutation. Our assumption that we use the first $n$ natural numbers implies numerical constraints between $a, b, n$:

$$a-1 \geq 2(b-1)+1 \quad \Longrightarrow \quad a \geq 2b, \tag{6}$$

$$n-a \geq 2(\frac{n}{3}-b)+1 \quad \Longrightarrow \quad a \leq \frac{n}{3}+2b-1. \tag{7}$$

This also leads to the same $\nu(n)$, the possible range of the median-selection-procedure we derived.

Counting the ways we can do the above is best done by viewing the arrangement in stages.

First we place the element $a$ in triplet $b$ (say, in location $3b$. Then we select and distribute $b-1$ pairs (and a singleton for triplet $b$) of small values, in the leftmost $b-1$ triplets of places. These elements can be selected in $\binom{a-1}{2b-1}$ ways. Then we scatter them around, which we can do in $(2b-1)!$ ways, for a total of $(a-1)!/(a-2b)!$ arrangements. Similarly for $\frac{n}{3}-b$ pairs of large values (and one in triplet $b$), in $\binom{n-a}{2(\frac{n}{3}-b)+1}$ ways times $(2(\frac{n}{3}-b)+1)!$, or in $(n-a)!/(\frac{n}{3}-a+2b-1)!$ ways.

To visualize the arguments below assume that at this time, each such pair occupies the two leftmost positions in each triplet (triplet $b$ is now filled up).

This distribution, the first stage of step (1), creates therefore

$$\frac{(a-1)!(n-a)!}{(a-2b)!(\frac{n}{3}-a+2b-1)!} \tag{8}$$

arrangements.

Next, we are left with $\frac{n}{3}-1$ elements, $a-2b$ of them are small, and the rest are large. They have to be distributed into the positions left open in $\frac{n}{3}-1$ triplets (all except triplet number $b$). Here appears a complication, the only one in the entire procedure.

It is best shown via an example: Suppose $a=20,\ b=5$ and we look at a small triplet. Further, assume the triplet has so far the entries 1 and 2, from the first distribution.

We compare two possibilities.

In one, we now put there the element 3, one of the $a-2b=10$ surviving small values. Like this triplet 1,2,3 we also get the 2,1,3, if the first distribution reversed the order of the pair. The other four permutations of this three values arise when the first distribution selected either of the pairs 1,3 or 2,3 for this location, and the set is completed in the second step by inserting 2 or 1 respectively. Conclusion: each such insertion accounts for exactly one ordered triplet.

In the second possibility we put there a surviving large value, say 25. In the same way we now have the triplets, 1,2,25 and 2,1,25. The other possible positions of the "25," unlike the first possibility, cannot arise via the way we did the initial distribution. Hence we should multiply the count of such permutations by 3; in other words: each such insertion accounts for exactly three triplets. This observation, that at this step we need to distinguish between small and large values and where they land, leads to the need of using an additional parameter. We select it to be the number of small values, out of the $a-2b$, that get to be inserted into "small" triplets, those in the range $1\ldots b-1$, and denote it by $i$. Further, call a small triplet into which we put a small value 'homogeneous,' and let it be 'heterogeneous' if we put there a large value (and similarly for each of the rightmost $\frac{n}{3}-b$ triplets which gets a large or small value, respectively). With this notation we shall have, for a fixed $i$,

$$\begin{array}{rl} i & \text{small homogeneous triplets} \\ \frac{n}{3}+b-a+i & \text{large homogeneous triplets} \\ b-i-1 & \text{small heterogeneous triplets} \\ a-2b-i & \text{large heterogeneous triplets} \end{array}$$

We need to choose which of the small, and which of the large triplets would be, say, heterogeneous, and this introduces a factor of $\binom{b-1}{i}\binom{\frac{n}{3}-b}{a-2b-i}$.

Next comes the choice of the numbers, out of the $\frac{n}{3}-1$ available, that go into the small and large triplets. Since these need to be put in all possible orders, the selection factors cancel, and we are left with $(a-2b)!(\frac{n}{3}-a+2b-1)!$.

Finally we need to multiply by the factors $6\times 3^{a-b-2i-1}$.
The factor 6 accounts for the possible number of ways we can order the $b$th triplet (since so far there has been no constraint on the locations of its elements), and the next for the contribution of the possible orders of the heterogeneous triplets, as shown above.

Combining it all, with the contribution of step (2) above, we have

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

From relations (6–7) we see that $q_{a,b}^{(r)}$ is nonzero for $0 \le a-2b \le \frac{n}{3}-1$ 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 $u = 3b-a-1, v = \frac{n}{3}+b-a$, but this does not appear to confer any advantage.

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 $p_{a,b}^{(r)} = q_{a,b}^{(r)}/n!$:

$$p_{a,b}^{(r)} = \frac{2}{3}\frac{3^{-b}\binom{\frac{n}{3}-1}{b-1}}{3^{-a}\binom{n-1}{a-1}} \times \sum_i \binom{b-1}{i}\binom{\frac{n}{3}-b}{a-2b-i}\frac{1}{9^i} = \frac{2}{3}\frac{3^{-b}\binom{\frac{n}{3}-1}{b-1}}{3^{-a}\binom{n-1}{a-1}} \times [z^{a-2b}](1+\frac{z}{9})^{b-1}(1+z)^{\frac{n}{3}-b}. \qquad (10)$$

This allows us to calculate our main 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

$$P_a^{(r)} = \sum_{b_r} p_{a,b_r}^{(r)} P_{b_r}^{(r-1)} = \sum_{b_r,b_{r-1},\cdots,b_3} p_{a,b_r}^{(r)} p_{b_r,b_{r-1}}^{(r-1)} \cdots p_{b_3,2}^{(2)}\,, \qquad 2^{j-1} \le b_j \le 3^{j-1}-2^{j-1}+1. \qquad (11)$$

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

$$P_a^{(r)} = \left(\frac{2}{3}\right)^r \frac{3^{a-1}}{\binom{n-1}{a-1}} \sum_{b_r,b_{r-1},\cdots,b_3} \prod_{j=2}^{r} \sum_{i_j \ge 0} \binom{b_j-1}{i_j}\binom{3^{j-1}-b_j}{b_{j+1}-2b_j-i_j}\frac{1}{9^{i_j}} \qquad (12)$$

As above, each $b_j$ takes values in the range $[2^{j-1}\,..\,3^{j-1}-2^{j-1}+1]$, $b_2 = 2$ 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 $\nu(n) < a < n-\nu(n)+1$ 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 a bell-shaped curve, quite similar to the Normal distribution, this is not easy to show analytically — and for a good reason: in Section 4.4 below we analyze the large-sample case, where we see a distribution which is a limit, in a restricted sense, of the distribution of the selected median, and show it is reminiscent of, but not equal to, the Gaussian.

### 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. 4.3.

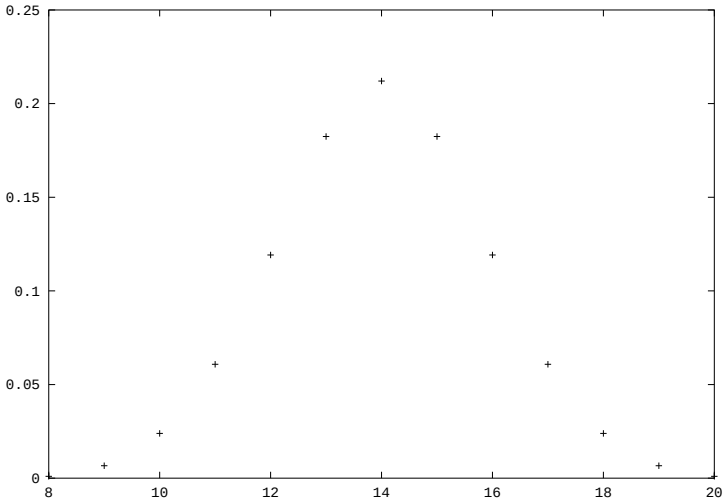


Figure 3: Plot of the median probability distribution for n=27

By comparison, with a larger array ($r = 5,\ n = 243$, Figure 4) we notice the relative concentration of the likely range of selection around the true median. In terms of these probabilities the relation (5) is:

$$\sum_{\lfloor zn \rfloor < a < \lceil (1-z)n \rceil + 1} P_a^{(r)} \quad \text{where } 0 \le z < \frac{1}{2}. \tag{13}$$

We chose to present the effectiveness of the algorithm by computing directly from Eq.(12) 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 $\mu_d$, the mean of $|D_n|$ and its standard deviation, denoted by $\sigma_d$.

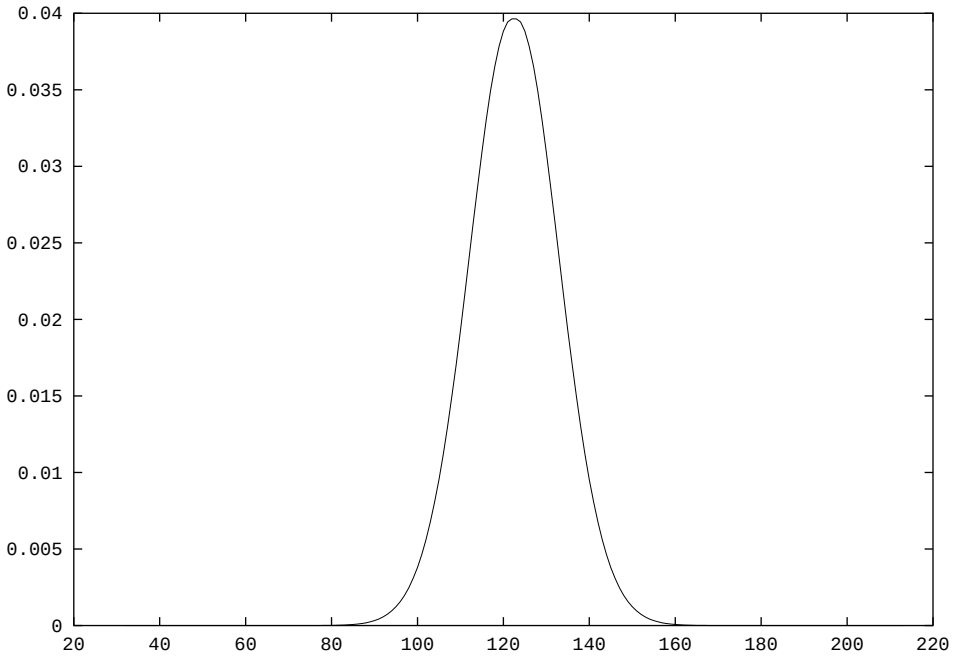


Figure 4: Plot of the median probability distribution for n=243

Numerical computations produced the results in Table 1; note the trend in the two rightmost columns. This trend appears here arbitrary, but as the next subsection shows, the ratios approach a limit.

The ratio $\mu_d/\mathcal{M}_d(n)$ can be viewed as a measure of the expected relative error of the approximate median selection algorithm. The trend of this ratio $\mu_d/\mathcal{M}_d(n)$ can be then seen as the improvement of the selection effectiveness with increasing (initial) array size $n$. According to the last claim this ratio decreases as $n^{\log_3 2-1} = (2/3)^r = 1/n^{0.36907}$.

| $n$ | $r = \log_3 n$ | $\mu_d$ | $\sigma_d$ | $\mu_d/n^{\log_3 2}$ | $\sigma_d/n^{\log_3 2}$ | $\mu_d/\mathcal{M}_d(n)$ |
|---|---|---|---|---|---|---|
| 9 | 2 | 0.428571 | 0.494872 | 0.107143 | 0.123718 | 0.107143 |
| 27 | 3 | 1.475971 | 1.184262 | 0.184496 | 0.148033 | 0.113536 |
| 81 | 4 | 3.617240 | 2.782263 | 0.226077 | 0.173891 | 0.090431 |
| 243 | 5 | 8.096189 | 6.194667 | 0.253006 | 0.193583 | 0.066911 |
| 729 | 6 | 17.377167 | 13.282273 | 0.271518 | 0.207545 | 0.047739 |
| 2187 | 7 | 36.427027 | 27.826992 | 0.284586 | 0.217398 | 0.033328 |
| 6561 | 8 | 75.255332 | 57.452918 | 0.293966 | 0.224425 | 0.022944 |

Table 1: Statistics of the median selection bias $|D_n|$ as function of array size

### 4.4 Extensions

Since we find the probabilities given in Eq.(12) hard to compute with, we approximate that PMF with another distribution that is easier to handle.

Let $\Xi = (\xi_1, \xi_2, \ldots, \xi_n)$ be $n$ independent identical $U(0,1)$ variates. We denote these values, when sorted, by $\xi_{(1)}, \xi_{(2)}, \ldots, \xi_{(n)}$. The ranks of the elements of $\Xi$ form a permutation of the integers 1 through $n$. If we used our Section 2 algorithm on this permutation, and it returned the result $X_n$, then using the algorithm on $\Xi$ itself would return the order statistic $\xi_{(X_n)}$, which we denote by $Y_n$. Since the $\xi_i$ are independent, unlike the elements of a permutation, it is much easier to calculate using these variates.

We first show that the $Y_n$ provide a useful approximation as we claimed. More precisely, we use the distribution of $Y_n - \frac{1}{2}$ to approximate the distribution of $D_n/n$.

The probability density function of the order statistic $\xi_{(k)}$ of $U(0,1)$ is given by

$$f_k(x) = \binom{n}{k} k x^{k-1} (1-x)^{n-k}, \tag{14}$$

hence

$$E[\xi_{(k)}] = \frac{k}{n+1}, \qquad V[\xi_{(k)}] = \frac{k(n+1-k)}{(n+1)^2(n+2)}. \tag{15}$$

Therefore, the mean square distance between $Y_n - \frac{1}{2}$ and $D_n/n$, over all the permutations for which, say, $X_n = k$ (which implies $Y_n = \xi_{(k)}$), can be estimated as follows:

$$E\left[\left(Y_n - \frac{1}{2}\right) - \left(\frac{X_n - \frac{n+1}{2}}{n}\right)\right]^2 = E\left[Y_n - \frac{X_n - 1/2}{n}\right]^2 = E\left[\xi_{(k)} - \frac{k-1/2}{n}\right]^2 \lesssim \frac{1}{4n}. \tag{16}$$

Since the bound is independent of $k$, it holds for all samples. Table 1 suggests that $V(|D_n|/n)$ is in $\Omega(n^{\log_3 4 - 2})$); since $\log_3 4 \approx 1.262$, this is larger than the bound in Eq.(16), and we can say that $Y_n - 1/2$ is a good approximation for $D_n/n$.

Next we derive a recursion for the distribution of $Y_n - 1/2$. We continue to consider only values of $n$ which are powers of 3:

$$F_r(x) \equiv \Pr(Y_n - 1/2 \le x), \qquad -1/2 \le x \le 1/2, \qquad n = 3^r. \tag{17}$$

We start the recursion with $F_0(x) = x + 1/2$. Since $Y_{3n}$ is obtained by taking the median of 3 independent values each of which has the distribution of $Y_n$, $F_r(\cdot)$, then $Y_{3n}$ has the distribution $F_{r+1}(\cdot)$,

$$F_{r+1}(x) = \Pr(Y_{3n} \le x + 1/2) = 3F_r^2(x)(1 - F_r(x)) + F_r^3(x) = 3F_r^2(x) - 2F_r^3(x). \tag{18}$$

A slightly simpler form is obtained by shifting $F_r(\cdot)$ by 1/2;

$$G_r(x) \equiv F_r(x) - 1/2 \quad \Longrightarrow \quad G_0(x) = x, \quad G_{r+1}(x) = \frac{3}{2} G_r(x) - 2G_r^3(x). \tag{19}$$

The limiting function for this recursion exists[†] but it is not an interesting one: it has the values $-(1/2)$ and 1/2, in the subintervals $[-1/2,0)$ and $(0,1/2]$ respectively. At the origin and the end points it has the same values as $G_0(\cdot)$.

While this is a consistent result, it is hardly useful. It merely tells us that as $n$ increases $Y_n - \frac{1}{2}$ (and $D_n/n$) converge to zero, as Figures 3 and 4, and Table 1 suggest. To get a non-degenerate result we need to introduce a change of scale.

The clue to such possible scale is given by Eq.(16), which implies that $\mu^{2r} E\left[\left(Y_n - \frac{1}{2}\right) - D_n/n\right]^2 \longrightarrow 0$ as $r$ (and $n = 3^r$) increase, so long as $\mu$ is not too large. A simple calculation shows we need $\mu \in [0,\ \sqrt{3})$. With such a $\mu$ we could still track $\mu^r(D_n/n)$ with $\mu^r(Y_n - 1/2)$ effectively. Note that $\mu^r(D_n/n) = D_n/n^{1-\log_3 \mu}$. In particular, for the value $\mu = 3/2$, which is in the allowed interval, we have the following result:

**Theorem 3:** [SJ99] Let $n = 3^r$, $r \in \mathbb{N}$, and $X_n$ be the approximate median of a random permutation of $1,\ldots,n$ (with all permutations assumed equally likely) computed with the *Approximate_Median* algorithm. Then a random variable $X$ exists, such that

$$\left(\frac{3}{2}\right)^r \frac{X_n - \frac{n+1}{2}}{n} \quad \longrightarrow \quad X, \tag{20}$$

where $X$ has the distribution $F(\cdot)$, determined by the equations

$$F(x) \equiv G(x) + 1/2, \qquad G(\frac{3}{2}x) = \frac{3}{2}G(x) - 2G^3(x), \qquad -\infty < x < \infty \tag{21}$$

The distribution function $F(\cdot)$ is strictly increasing throughout.

**Proof:** The ingredients of the theorem satisfy the data for the Lemma of Appendix A, with $\phi(x) = (3/2)x - 2x^3,\ \mu = 3/2,\ A = 2,\ a = 1/2$. Hence $G_r(x(2/3)^r)$ as defined in Eq.(19) converges to a function $G(x)$ which satisfies Eq.(21). □

There is no obvious way to solve Eq.(21), but it can still be very informative. First, we see that $G_0(x)$ and $\phi(x)$ are odd functions, hence all the $G_r(x)$, as well as $G(x)$ are necessarily odd as well. We can write then $F(x) = \frac{1}{2} + \sum_{k\geq 1} b_k x^{2k-1}$. This Taylor expansion around the origin can be computed by extracting successive coefficients from Eq.(21), using the recurrence

$$b_j = \frac{2}{\mu(1-\mu^{2j-2})} \sum_{i=1}^{j-1} b_i \sum_{k=1}^{j-i} b_k b_{j-i-k+1}, \qquad j \geq 1 \qquad b_0 = 1, \qquad \mu = 3/2. \tag{22}$$

A direct calculation showed the procedure is extremely well-behaved numerically. Table 2 contains a few of the coefficients. We used these coefficients (in fact, the first 150 of them) to compute values for $F(x)$.

[†]That the iteration converges to a limit function is immediate from the fact that the function $a(z) = (3/2)z - 2z^3$ is increasing (in absolute value) for $z \in (-1/2, 1/2)$. At the excluded end points of the interval and at $z = 0$, the sequence $\{G_r(\cdot)\}$ is stationary.

| $k$ | $b_k$ |
|---|---|
| 1 | $1.00000000000000 \times 10^{+00}$ |
| 2 | $-1.06666666666667 \times 10^{+00}$ |
| 3 | $1.05025641025641 \times 10^{+00}$ |
| 4 | $-8.42310905468800 \times 10^{-01}$ |
| 5 | $5.66391554459281 \times 10^{-01}$ |
| 6 | $-3.29043692201665 \times 10^{-01}$ |
| 7 | $1.69063219329527 \times 10^{-01}$ |
| 8 | $-7.82052123482121 \times 10^{-02}$ |
| 9 | $3.30170547707520 \times 10^{-02}$ |
| 10 | $-1.28576608229956 \times 10^{-02}$ |
| 20 | $-4.33903859413399 \times 10^{-08}$ |
| 30 | $-3.20126276232555 \times 10^{-15}$ |
| 40 | $-1.94773425996709 \times 10^{-23}$ |
| 60 | $-4.03988860877434 \times 10^{-42}$ |
| 80 | $-5.63050454255617 \times 10^{-63}$ |
| 100 | $-1.88810747562091 \times 10^{-85}$ |
| 125 | $2.50253570235335 \times 10^{-115}$ |
| 150 | $-8.16299422374440 \times 10^{-147}$ |

Table 2: Coefficients for the expansion $G(x) = \sum_{k \geq 1} b_k x^{2k-1}$.

The calculation showed that the distribution is very close to a Normal distribution with standard deviation of $\sigma = 1/\sqrt{2\pi}$, but it is not exactly Normal. The largest *relative* difference between $F(x)$ and $N(0, \sigma^2)$ over the interval $[0, 1.3963]$—which amounts to 3.5 standard deviations—is close to 0.14%.

Equation (20) can be written as $D_n \longrightarrow Xn/\mu^r = X \times 2^r = X \times n^{\log_3 2}$. An immediate corollary is that asymptotically, this determines the rate of growth of the moments of $D_n$ (or of $|D_n|$) with $n$: every time $n$ is trebled, $\mu_d$ and $\sigma_d$ should double. The numbers in Table 1 show how for small values of $n$ the rate of increase is even faster, as the influence of the "taboo" values, as shown in §4.1, decrease.

**Notes:** 1. The choice of $\sigma$ above is consistent with the first term of the expansion of the corresponding Gaussian density. A choice of a slightly larger standard deviation can reduce the maximal relative "misfit" to less than 0.05%.

2. We tested for normality by observing the ratio $-\frac{(2i-1)b_i}{(2i+1)b_{i+1}}$. If $F(\cdot)$ were Gaussian, this ratio would be constant, and equal to the variance of the distribution. However, we find that the ratio changes by a factor of

twenty over the range of the table...

3. Another way of relating the distribution $F(\cdot)$ to the Gaussian is by comparing the (asymptotic) rate of decay of the tail of the distributions. This turns out to be informative here. Eq.(18) can be rewritten as

$$F_{r+1}(x/\mu^r) = 3F_r^2(x/\mu^r) - 2F_r^3(x/\mu^r). \tag{23}$$

Using Theorem 3 we find that $F(\cdot)$, the limiting distribution of $\mu^r D_n/n$, satisfies

$$F(x\mu) = 3F^2(x) - 2F^3(x), \qquad \mu = \frac{3}{2}. \tag{24}$$

Let $g(x)$ for positive $x$ be the tail probability, $g(x) \equiv 1 - F(x) = F(-x)$, and the last equality means that $g(\cdot)$ also satisfies the equation

$$g(x\mu) = 3g^2(x) - 2g^3(x) \Longrightarrow 3g(x\mu) = (3g(x))^2 \left(1 - \frac{2}{3}g(x)\right). \tag{25}$$

Since $0 < g(x) < 1$ for all finite $x$, we find

$$\frac{1}{3}(3g(x))^2 < 3g(x\mu) < (3g(x))^2. \tag{26}$$

The right-hand side inequality, written for $h(x) \equiv 3g(x)$, produces $h\left((3/2)^k x\right) < h^{2^k}(x)$. Taking logarithm of both sides, and fixing $x = 1$ we find $\ln h\left((3/2)^k\right) < 2^k \ln h(1)$. Let $-c \equiv \ln h(1) < 0$, and $t \equiv (3/2)^k$. We evaluate $h(1)$ using the Taylor development, and find $-c \approx \ln(0.0206756894) = -3.87879669$ and $k = \frac{\ln t}{\ln(3/2)}$. This leads to $2^k = \exp(k \ln 2) = \exp\left(\ln t \frac{\ln 2}{\ln 3/2}\right)$. Hence

$$\ln h(t) < -c \exp\left(\ln t \frac{\ln 2}{\ln 3/2}\right) \Longrightarrow h(t) < e^{-ct^v}, \tag{27}$$

where $v = \ln 2/\ln(3/2) \approx 1.7095113\ldots$. And relation (26) leads immediately to

$$\frac{1}{3}e^{-ct^v} < h(t) < e^{-ct^v} \qquad c \approx 3.87879669, \quad v \approx 1.7095113. \tag{28}$$

The choice $x = 1$ was arbitrary, and the last equation provides the estimate for any value $t$, not only $t = (3/2)^k$ for some integer $k$. Further, looking at the left-hand side of inequality (26) we see it was designed to hold for all $x$. However, since $g(x)$ is monotonically increasing, Eq.(25) suggests that for any $\varepsilon > 0$ we can find $x_\varepsilon$ such that $h(x\mu) > (1 - h(x_\varepsilon))\, h^2(x)$, which suggests that in fact

$$g(t) = 1 - F(t) = \frac{1}{3}h(t) \sim \frac{1}{3}e^{-ct^v}, \quad t \to \infty, \qquad c \approx 3.87879669, \quad v \approx 1.70951129135. \tag{29}$$

The tails of the Normal distribution $N(0, 1/2\pi)$, on the other hand, decay quite faster; they satisfy, for large $x$, $1 - \Phi_\sigma(x) \approx e^{-x^2/2}/(2\pi x)$.

## 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 three, $n = 3^r$, 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 bias of the approximate median. For each returned result we compute its distance from the correct median $\mathcal{M}_d(n) = \frac{n+1}{2}$. The first table presents the results in both actual, or "absolute" values, and in "relative" units. Only relative units are used in the other tables. The relative units, $d_{\%}$, are computed as $D_n$ divided by the value of the correct median (actually, by the distance of the median from the extreme terms): $d_{\%} = 100 \times \frac{D_n}{(n-1)/2}$. The extremes of $d_{\%}$ are 0, when the true median is returned – and it would have been 100 if it were possible to return the smallest (or largest) elements. (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. All experiments used a sample size of 5000, throughout.

| | $\mu_d$ | | $\sigma$ | | $\mu_d + 2\,\sigma$ | | Rng(95%) | | Min-Max | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | rel. | abs. | rel. | abs. | rel. | abs. | rel. | abs. | rel. | abs. |
| 50 | 10.27 | 2.516 | 7.89 | 1.933 | 26.05 | 6.382 | 24.49 | 6 | 0–44.90 | 0–11 |
| 100 | 8.45 | 4.183 | 6.63 | 3.282 | 21.70 | 10.74 | 20.20 | 10 | 0–48.48 | 0–24 |
| $3^5$ (243) | 6.74 | 8.189 | 5.13 | 6.233 | 17.00 | 20.66 | 16.53 | 20 | 0–38.02 | 0–46 |
| 500 | 5.80 | 14.47 | 4.41 | 11.00 | 14.63 | 36.50 | 14.03 | 35 | 0–29.06 | 0–73 |
| $3^6$ (729) | 4.83 | 17.58 | 3.71 | 13.50 | 12.26 | 44.63 | 12.09 | 44 | 0–24.73 | 0–90 |
| 1000 | 4.70 | 23.48 | 3.66 | 18.28 | 12.02 | 60.04 | 11.61 | 58 | 0–23.62 | 0–118 |
| $3^7$ (2187) | 3.32 | 36.29 | 2.54 | 27,76 | 8.41 | 91.92 | 8.05 | 88 | 0–16.83 | 0–184 |
| 5000 | 2.71 | 81.29 | 2.10 | 62.99 | 6.91 | 207.3 | 6.72 | 202 | 0–17.20 | 0–516 |
| $3^8$ (6561) | 2.31 | 75.77 | 1.75 | 57.40 | 5.81 | 190.6 | 5.67 | 186 | 0–11.95 | 0–392 |
| 10000 | 2.53 | 126.5 | 1.86 | 92.99 | 6.24 | 312.0 | 6.04 | 302 | 0–11.38 | 0–569 |
| $3^9$ (19683) | 1.58 | 155.5 | 1.18 | 116.1 | 3.94 | 387.7 | 3.86 | 380 | 0–6.78 | 0–667 |

Table 3: Simulation results for $d_{\%}$ and $D_n$, bias of the approximate median. Sample size = 5000

The columns are: $n$ – array size; $\mu_d$ – the average of the absolute value of the bias over the sample; $\sigma$ –

the sample standard-error of $d_{\%}$ or $D_n$; Rng(95%) – half the size of an interval symmetric around zero that contains 95% of the returned values; the last column gives the observed extreme values of $d_{\%}$ or $D_n$. In the rows that correspond to those of Table 1, the agreement of the $\mu_d$ and $\sigma_d$ columns is excellent (the relative differences are under 0.5%).

These experiments allow us to gauge the quality of the information we have about the distribution of the approximate median bias. We use one entry in the last table for this. The table states that for $n = 1000$, 95% of the values of $D_{1000}$ fell in the interval $[-58, 58]$. According to Eq.(20), we have the approximate relation $D_n \sim nX/\mu^r$. Now $n/\mu^r = 3^r/(3/2)^r = n^{\ln(2)/\ln(3)} \approx n^{0.63092975}$. We find

$$\Pr[|D_n| \le d] \approx \Pr[|X| \le d\mu^r/n] \Longrightarrow \Pr[|D_{1000}| \le 58] \approx \Pr[|X| \le 0.7424013] \approx 0.934543.$$

This value was calculated as $2G(0.742\ldots)$ using the Taylor coefficients. To use the tail approximation we would similarly find

$$\Pr[|D_{1000}| \le 58] = 1 - 2g(0.7424013) \approx 1 - \frac{2}{3}\exp\left(-3.87879669 \times 0.7424013^{1.7095113}\right) \approx 0.935205.$$

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 | $\mu_d$ | $\sigma_d$ | Rng(95%) | $\mu_d$ | $\sigma_d$ | Rng(95%) | $\mu_d$ | $\sigma_d$ | 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 4: Quality of selection as a function of threshold value using the algorithm of Figure 2

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. The results are consistent with the discussion in Section 3.2:

| | $\mu_d$ | $\sigma_d$ | $\mu_d + 2\,\sigma$ | 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 5: 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 needs to be extended to find ways to compute with the exact probabilities, as given in Eq.(12). Also, the limiting distribution of the bias $D_n$ with respect to the true median – while we have quite effective computational tools for it, we still have no manageable analytical characterization beyond the unwieldy non-local cubic recurrence. In particular, even the seemingly simple matter of calculating the moments of the limiting distribution, leading to good approximations of moments of $D_n$, is open.

We have also left the analytic evaluation of the effects of using larger sets than triplets, such as quintets or septets, and of a threshold beyond which the remaining numbers are simply sorted, for future work.

Of special interests are the various ways in which this algorithm can be extended to the more general selection of the $k/n$ fractile of an arbitrary set of numbers. We hope to report on some such work soon.

## Acknowledgments

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

## Appendix

**Lemma 1:** Let $a \in (0,\infty)$ and a mapping $\phi$ of $[0,a]$ into $[0,a]$ be available. Further, for $x > a$ we define $\phi(x) = x$. Assume

$(i)$ $\quad \phi(0) = 0$

(*ii*) $\phi(a) = a$

(*iii*) $\phi(x) > x$, for all $x \in (0, a)$.

(*iv*) $\phi'(0) = \mu > 1$, and continuous there; $\phi(\cdot)$ is continuous and strictly increasing on $[0, a)$.

(*v*) $\phi(x) < \mu x, \quad x \in (0, a)$.

Then

$$\text{as } r \longrightarrow \infty, \qquad \phi_r(x/\mu^r) \longrightarrow \psi(x), \qquad x \geq 0$$

where $\phi_r(\cdot)$ is the $r$th iterate of $\phi(\cdot)$. The function $\psi(x)$ is well defined and strictly monotonic increasing for all $x$, increases from 0 to $a$, and satisfies the equation $\psi(\mu x) = \phi(\psi(x))$.

**Proof:** From Property $(v)$: $\phi(x/\mu^{r+1}) < x/\mu^r$,
Since iteration preserves monotonicity, $\phi_{r+1}(x/\mu^{r+1}) = \phi_r(\phi(x/\mu^{r+1})) < \phi_r(x/\mu^r)$.
Since $\phi(\cdot)$ and its iterates are nonnegative, this monotonic decrease implies convergence. We denote the limit by $\psi(\cdot)$.

We note that the properties of $\psi(x)$ depend on the behavior of $\phi(\cdot)$ near $x = 0$. In particular, since $\phi'(x)$ is continuous at $x = 0$, $\psi(\cdot)$ is continuous throughout. Since it is bounded, the convergence is uniform on $[0, \infty]$. Hence, since $\phi(\cdot)$ and all its iterates are strictly monotonic, so is $\psi(\cdot)$ itself.

To find more information about $\psi(\cdot)$, we use the monotonicity we showed and $(v)$ again, to produce

$$\psi(x) \;<\; \phi_r(x/\mu^r) \;<\; \phi(x/\mu) \;<\; x. \tag{30}$$

For any positive $x$ there is a minimal $r$ such that $x/\mu^r \leq a$. Since $\phi_r(\cdot)$ maps $[0, a]$ to itself, then Eq.(30) implies $\psi(x) \leq a$, for all $x \geq 0$. We now show that $\psi(\cdot)$ achieves the value $a$. We create an increasing sequence $\{x_j, j \geq 1\}$, such that

$$x_{j-1}/\mu^{j-1} < x_j/\mu^j \longrightarrow a^-, \tag{31}$$

that is, the values $x_j/\mu^j$ increase as well, and approach $a$ from below. Property $(iii)$ iterates to $\phi_r(x) > x$, hence also $\phi_j(x_j/\mu^j) > x_j/\mu^j \longrightarrow a$. Since $\psi(\infty) = \lim_{j\to\infty} \psi(x_j)$, necessarily this limit achieves the value $a$. □

## 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.

[BFPRT73] 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 3*n* 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.

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

[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.

[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).