Course Project CS648

Randomized Algorithm to Find Exact Median

Course Instructor: Prof. Surender Baswana
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Submitted by:

Siddharth Pathak (211034)

May 7, 2024

Abstract

In computational algorithms, randomized algorithms play a pivotal role in efficiently addressing complex computational problems. This report delves into developing a randomised algorithm to efficiently find the median of a set S in $1.5n + o(n)$ operations. Leveraging the significance of randomly sampled subsets, our algorithm, operating on set S, explores the relationship between a subset A of elements from S and the medians of S.

With a subset size approximately three-quarters that of S ($|A| \approx n^{3/4}$), our algorithm sorts A in sub-linear time $o(n)$, challenging the conventional assumption that the median of A equals that of S. Randomized algorithms are increasingly recognized for solving computational problems efficiently, particularly in scenarios where deterministic approaches fall short due to the problem's complexity or the input data's size.

Through rigorous analysis, we establish that while the medians of A and S may not coincide, they are not significantly distant within S's sorted order. By carefully selecting elements a and b from A, we ensure that the median of S lies within the interval $[a, b]$ with high probability while limiting the number of S elements between a and b to $o(n)$. This algorithmic approach showcases the power of randomized algorithms and provides valuable insights into efficiently approximating the median of S, facilitating faster computation with minimal computational cost.

Contents

Chapter 1

Randomized Algorithm for Exact Median

1.1 Idea and Intuition

Random sampling is fundamental in randomized algorithms, providing a powerful approach to solving complex computational problems efficiently. Algorithms like Monte Carlo and Las Vegas utilize random sampling to approximate solutions and guarantee correctness with varying runtime. Randomized algorithms leverage sampling to navigate large search spaces, mitigate worst-case scenarios, and adapt to diverse input distributions. This controlled randomness enhances efficiency, scalability, and innovation in algorithm design, making randomized algorithms indispensable in modern computational tasks.

The proposed methodology revolves around sampling a random subset, denoted as A, comprising a small fraction of the elements from the set S, typically with a size approximation of $|A| \approx n^{3/4}$. Given this size, sorting A can be accomplished in sub-linear time, denoted as $o(n)$. Initially, one might intuitively assume that the median of A coincides with the median of S. However, such an assumption is improbable. Nonetheless, it's observed that the median of A and the median of S tend not to deviate significantly in the sorted order of S. Let us fix the notation that for any set P, P_i denotes the i^{th} element in the set and $i \in \{1, 2, \ldots, |P|\}$

To elaborate, the algorithm scrutinizes two elements, denoted as a and b, within A, where a represents the $(k/2 - t)^{th}$ element and b represents the $(k/2 + t + 1)^{th}$ element. Here, t, which is chosen as less than $k/2$ in such a way, it ensures that with high probability,

- The median of S lies inclusively between a and b.
- The number of elements of S residing between a and b remains negligibly small, denoted as $o(n)$.

Should these conditions hold, the algorithm proceeds as follows: Firstly, a and b are identified within A through sorting. Subsequently, the rank of a in S is determined via a linear scan of S. A subsequent linear scan is conducted to identify all elements, denoted as Q, lying between a and b within S. Since Q is guaranteed to encompass the median and contains only a negligible number of elements $o(n)$, sorting Q effectively facilitates the determination of the median. We will extensively use Chernoff's bound in our analysis, so it is good to mention it here before we start.

Theorem 1.1.1. *(Chernoff Bounds). Let $X = \sum_{i=1}^{n} X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Let $\mu = \mathbb{E}(X) = \sum_{i=1}^{n} p_i$. Then*
(i) Upper Tail: $\mathbb{P}(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2+\delta}}$ for all $\delta > 0$;
(ii) Lower Tail: $\mathbb{P}(X \leq (1-\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2}}$ for all $0 < \delta < 1$.

1.2 Sketch of Algorithm

In the previous section, we discussed the algorithm verbally. The pseudocode for the algorithm mentioned above is as follows:

Algorithm 1 Randomized Algorithm to Find Exact Median

```
Require: Set S
Ensure: Median of set S
  n = |S|
  k = n^{3/4}
  p = k/n
  A = Random sample by picking each element of S independently with probability p.
  Fix some t
  Sort(A)                                          ▷ o(n) comparisons
  a = (k/2 − t)^{th} element of A
  b = (k/2 + t + 1)^{th} element of A
  R_a = Rank of a in set S                         ▷ n comparisons
  R_b = Rank of b in set S                         ▷ 0.5n + o(n) comparisons
  Q = Set of elements from S lying between a and b
  if R_a < n/2 and R_b > n/2 then
      Sort(Q)                                      ▷ o(n) comparisons
      return Q[n/2 − R_a]
  else
      STOP
  end if
```

Note that the above algorithm is not complete. We do not guarantee the size of the set Q. Moreover, t is not a parameter we choose arbitrarily. Ideally, $|Q|$ should be very small. it must be $o(n)$. Both t and $|Q|$ are intertwined as follows:

$$\mathbb{E}(|Q|) = \frac{2t}{p} = o(n) \Rightarrow t = o(n^{3/4})$$

An exact expression for t will come from the analysis of the algorithm; for now, t must be $o(n^{3/4})$ to guarantee the $o(n)$ size of set Q. Note that $\mathbb{E}(|Q|) = \mathbb{E}(R_b - R_a)$. So, one must *also ensure that $R_b - R_a \approx \frac{2mt}{p}$ for some constant m. Detailed analysis of all the parameters and error probability is mentioned in later sections.*

1.3 Inference about Parameters

Let us first discuss the size of a random sample A. As discussed, we need a random sample sorted in $o(n)$ time. Moreover, this random sample should be able to give us a and b so that the set of elements from the set S lies between a and b should have cardinality $o(n)$. If we use a generic way to find random samples, i.e., randomly select each element from set S with probability p, uniformly and independently. We will get a random sample of the expected size of np. Let us fix $\mathbb{E}(|A|) = n^{3/4} = k$ (say). This implies our sampling probability p should be $n^{-1/4}$. We will construct a high probability bound on the random sample $|A|$.

Lemma 1.3.1. *$\mathbb{P}(|A| \geq 2k) \leq e^{-k/4}$ and $\mathbb{P}(|A| \leq k/2) \leq e^{-k/8}$.*

Proof. Let X_i be a random variable defined as follows:

$$X_i = \left\{ \begin{array}{ll} 1, & S_i \in A \\ 0, & S_i \notin A \end{array} \right\}, \quad \forall i = 1, 2, \ldots, n$$

Observe that $|A| = \sum_{i=1}^{n} X_i$. X_i's are independent Bernoulli random variables with probability of success being $p = n^{-1/4}$. By Chernoff bound, we have

$$\mathbb{P}(|A| \geq 2k) = \mathbb{P}\Big(\sum_{i=1}^{n} X_i \geq (1+1)k\Big) \leq e^{-k/4}$$

and

$$\mathbb{P}(|A| \leq k/2) = \mathbb{P}\Big(\sum_{i=1}^{n} X_i \leq (1-0.5)k\Big) \leq e^{-k/8}$$

This finishes the proof. □

Note 1.3.2. *Observe that practically, we need just the upper bound on $|A|$ because a small-sized random sample will not cause any trouble as long as it holds the properties of a random sample.*

A good Monte Carlo algorithm ensures the correct output with high accuracy. Note that we did not comment on the value of t till now. Without finding the t, we can not state the algorithm precisely. The probability that a and b fail to capture the median between them can be found very easily as follows:

$$\begin{aligned}\mathbb{P}(a \textit{ and } b \textit{ fails to capture median between them}) &= \mathbb{P}(\textit{Both } a \textit{ and } b \textit{ lies same side of median of } S) \\ &= \mathbb{P}(\textit{Both } a \textit{ and } b \textit{ lies right side of median}) \\ &\quad + \mathbb{P}(\textit{Both } a \textit{ and } b \textit{ lies left side of median})\end{aligned}$$

Now, let L denote the subset of elements in set S that are considered small, referring to those elements not surpassing the median, resulting in a $|L| = n/2$ cardinality. Correspondingly, let U denote the subset of large elements in set S, consisting of those elements surpassing the median. Let us introduce Y as the number of elements in the intersection between L and A, representing the count of small elements selected in A. Notably, the expected value of Y, denoted as $\mathbb{E}(Y)$, equals $k/2$, as each of the $n/2$ elements is chosen with a probability $p = k/n$. As we mentioned before, we will select a and b as $(k/2-t)^{th}$ and $(k/2+t+1)^{th}$ element in sorted random sample A. By employing Chernoff's bounds again, we will establish that with high probability, the median will lie between a and b. let us infer that when both elements a and b will lie on the right (or left) side of the median in sorted S. Note that a ranks $k/2-t$ in a random sample of A. a must belongs to L. If the random sample has less than $k/2-t$ elements, selected from L, then $a \in U$, i.e. if $Y < k/2-t$ then $a \in U$. It is trivial to see that $a < b$, so $b \in U$, i.e. both a and b lie rightwards of the median. Similarly, both a and b will lie leftwards if b lies left of the median, i.e. an element of rank $k/2+t+1$ in A belongs to L. This will happen if random sample A consists of more than $k/2+t$ elements of L, i.e. $Y > k/2+t$. Now, Let us have find the respective probabilities to prove the low probability bound.

Lemma 1.3.3. *$\mathbb{P}(Y < k/2 - t) \le e^{-t^2/k}$ and $\mathbb{P}(Y > k/2 + t) \le e^{-2t^2/3k}$.*

Proof. Let us define RV Y_i as following:

$$Y_i = \left\{ \begin{array}{ll} 1, & L_i \in A \\ 0, & L_i \notin A \end{array} \right\}, \quad \forall i = 1, 2, .., n/2$$

Observe that $Y = \sum_{i=1}^{n/2} Y_i$. Since each element in S (so in L) is selected randomly uniformly Independently, We can use the result from Chernoff's bound here. By Chernoff's bound (Lower tail), we get

$$\mathbb{P}(Y < k/2 - t) = \mathbb{P}\left(\sum_{i=1}^{n/2} Y_i < \Big(1 - \frac{2t}{k}\Big) k/2 \right) \le e^{-t^2/k}$$

By Chernoff's bound (Upper tail) and observing that $2t/k < 1$, we get

$$\mathbb{P}(Y < k/2 + t) = \mathbb{P}\left(\sum_{i=1}^{n/2} Y_i < \Big(1 + \frac{2t}{k}\Big) k/2 \right) \le e^{-2t^2/3k}$$

This ends the proof of the above lemma. □

$$\begin{aligned} \mathbb{P}(a \textit{ and } b \textit{ fails to capture median between them}) &= \mathbb{P}(Y < k/2 - t) + \mathbb{P}(Y > k/2 + t) \\ &\le e^{-t^2/k} + e^{-2t^2/3k} \\ &< 2e^{-2t^2/3k} \\ &= \frac{2}{n^c} \qquad \textit{for } t = \sqrt{1.5ck\log(n)} \end{aligned}$$

Hence, for choosing $t = \sqrt{1.5ck\log(n)}$, we can manage to have an inverse polynomial bound on the error probability of the algorithm.

Note 1.3.4. *It is important to note that we can not choose c as large as we want to make the error probability significantly less. If we increase c greater than one, our algorithm will not remain consistent for smaller inputs. We decided to choose a to be* $(k/2-t)^{th}$ *so, this index must be positive, i.e.*

$$k/2 - \sqrt{1.5ck\log(n)} > 0$$

$$\sqrt{k} - 2\sqrt{1.5c\log(n)} > 0$$

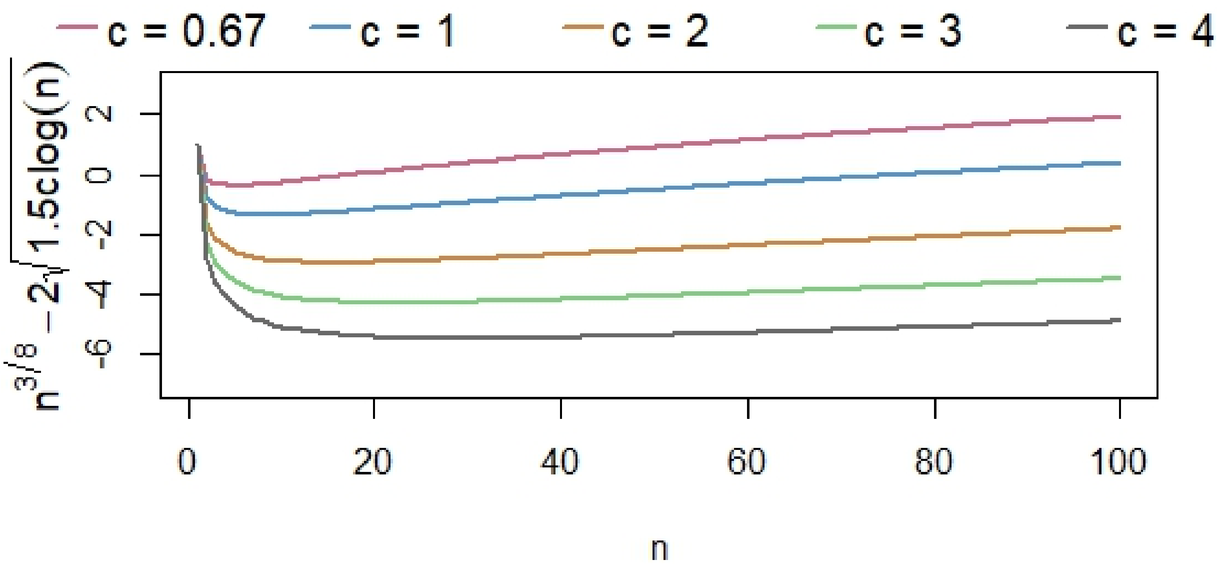

Figure 1.1: Constraint on parameter c

Now, we will ensure the upper bound on the size of Q*. As mentioned, the expected value of* $|Q|$ *is* $2t/p$*. We need to ABORT the algorithm if* $|Q|$ *deviates significantly from its expected size. One must ensure theoretically that the probability of deviation of* $|Q|$ *from its expected size is small. We show that with high probability, the ranks,* R_a *and* R_b *are close to* $n/2$*.*

Lemma 1.3.5. $\mathbb{P}(R_a < n/2 - \mathbb{E}(|Q|)) \leq exp\big(-t^2/2(k-4t\big)$ *and* $\mathbb{P}(R_b > n/2 + \mathbb{E}(|Q|)) \leq exp\big(-t^2/2(k-4t\big)$*.*

Proof. Since $\mathbb{E}(|Q|) = o(n)$, by proving the above lemma, we will establish the fact that $R_b - R_a = |Q| = o(n)$ with high probability. Let us consider T_s as the set of small elements with a rank of less than $n/2 - \mathbb{E}(|Q|)$ in set S if sorted. Similarly, T_l denotes the set of large elements with a rank greater than $n/2 + \mathbb{E}(|Q|)$ in set S if sorted. Define

$$Y_s := |T_s \cap A|$$

$$Y_l := |T_l \cap A|$$

Finding the probability $R_a < n/2 - \mathbb{E}(|Q|)$ is equivalent to finding the probability that the number of elements sampled from T_s is greater than or equal to $k/2 - t$, i.e. $Y_s > k/2 - t$. We get

$$\mathbb{P}(R_a < n/2 - \mathbb{E}(|Q|)) = \mathbb{P}(Y_s \geq k/2 - t)$$

We randomly selected each element of T_s with probability p, uniformly and independently. Let us define RV $Y_s^{(i)}$ as follows:

$$Y_s^{(i)} = \left\{ \begin{array}{ll} 1, & T_{si} \in A \\ 0, & T_{si} \notin A \end{array} \right\}, \quad \forall i = 1, 2, \ldots, n/2 - \mathbb{E}(|Q|)$$

Observe that $Y_s = \sum_{i=1}^{n/2-\mathbb{E}(|Q|)} Y_s^{(i)}$. Each of $Y_s^{(i)}$ is independently distributed RV with $Bernoulli(p)$ distribution. By linearity of expectation, we have

$$\mathbb{E}(Y_s) = \mathbb{E}\left(\sum_{i=1}^{n/2-\mathbb{E}(|Q|)} Y_s^{(i)}\right) = p(n/2 - \mathbb{E}(|Q|)) = p(n/2 - 2t/p) = k/2 - 2t$$

By applying Chernoff's bound on Y_s, we have

$$\begin{aligned}
\mathbb{P}(Y_s \geq k/2 - t) &= \mathbb{P}\left(\sum_{i=1}^{n/2-\mathbb{E}(|Q|)} Y_s^{(i)} \geq (k/2 - 2t)\left(1 + \frac{2t}{k-4t}\right)\right) \\
&\leq exp\left(-\frac{\left(\frac{2t}{k-4t}\right)^2 (k/2 - 2t)}{4}\right) \\
&= exp\left(-\frac{t^2}{2(k-4t)}\right)
\end{aligned}$$

Similarly, finding the probability $R_b > n/2 + \mathbb{E}(|Q|)$ is equivalent to finding the probability that the number of elements sampled from T_l is more significant than $|A| - k/2 - t \approx k/2 - t$, i.e. $Y_l \geq k/2 - t$. We get

$$\mathbb{P}(R_b > n/2 + \mathbb{E}(|Q|)) = \mathbb{P}(Y_l \geq k/2 - t)$$

Note that we randomly select each element of T_l with probability p, uniformly and independently. Let us define RV $Y_l^{(i)}$ as follows:

$$Y_l^{(i)} = \left\{ \begin{array}{ll} 1, & T_{li} \in A \\ 0, & T_{li} \notin A \end{array} \right\}, \quad \forall i = 1, 2, \ldots, n/2 - \mathbb{E}(|Q|)$$

Observe that $Y_l = \sum_{i=1}^{n/2-\mathbb{E}(|Q|)} Y_l^{(i)}$. Each of $Y_l^{(i)}$ is independently distributed RV with $Bernoulli(p)$ distribution. By linearity of expectation, we have

$$\mathbb{E}(Y_l) = \mathbb{E}\left(\sum_{i=1}^{n/2-\mathbb{E}(|Q|)} Y_l^{(i)}\right) = p(n/2 - \mathbb{E}(|Q|)) = p(n/2 - 2t/p) = k/2 - 2t$$

By applying Chernoff's bound on Y_l, we have

$$\begin{aligned}
\mathbb{P}(Y_l \geq k/2 - t) &= \mathbb{P}\Bigg(\sum_{i=1}^{n/2-\mathbb{E}(|Q|)} Y_l^{(i)} \geq (k/2 - 2t)\Big(1 + \frac{2t}{k-4t}\Big)\Bigg) \\
&\leq exp\Bigg(- \frac{\Big(\frac{2t}{k-4t}\Big)^2 (k/2-2t)}{4}\Bigg) \\
&= exp\Bigg(- \frac{t^2}{2(k-4t)}\Bigg)
\end{aligned}$$

This Completes the proof $\square$

Note 1.3.6. *From the proof of the above lemma, we get some more constraints in t such that $t^2 > k$ and $k > 4t$. Note that $t = \sqrt{k\log(n)} = \sqrt{n^{3/4}\log(n)}$ satisfies all the constraint. So, we are using $t = \sqrt{n^{3/4}\log(n)}$ in our final algorithm and implementation also.*

Theorem 1.3.7. *Error Probability of the algorithm is inverse in input size n.*

Proof. Our algorithm will fail to output the median if the random sample contains some problematic properties. These properties are:

- $|A| > 2k$
- $R_a > n/2$ or $R_a < n/2 - 2t/p$
- $R_b < n/2$ or $R_b < n/2 + 2t/p$

Let us define a A as a **bad random sample** if it has any of the above-mentioned properties. If the algorithm goes through the bad random sample, it will $ABORT$ immediately and does not output the correct median. So,

$$\begin{aligned}
\text{Error Probability} &= \mathbb{P}(\text{Algorithm gone through bad random sample}) \\
&< \mathbb{P}(|A| > 2k) + \mathbb{P}(R_a > n/2 \text{ or} R_a < n/2 - 2t/p) + \mathbb{P}(R_b < n/2 \text{ or } R_b > n/2 + 2t/p) \\
&< \mathbb{P}(|A| > 2k) + \mathbb{P}(R_a > n/2) + \mathbb{P}(R_a < n/2 - 2t/p) \\
&\qquad + \mathbb{P}(R_b < n/2) + \mathbb{P}(R_b > n/2 + 2t/p) \\
&\leq e^{-k/4} + e^{-t^2/k} + e^{-t^2/2(k-4t)} + e^{-2t^2/3k} + e^{-t^2/2(k-4t)} \\
&< e^{-k/4} + e^{-t^2/k} + e^{-t^2/2k} + e^{-2t^2/3k} + e^{-t^2/2k} \\
&< e^{-k/4} + 4e^{-t^2/k} \\
&= e^{-k/4} + \frac{4}{n^{2/3}} = O(1/n^{O(1)})
\end{aligned}$$

for $t = \sqrt{k\log(n)}$. This completes the proof. $\square$

1.4 The Final Algorithm

The final exact algorithm based on the above discussion is as follows:

Algorithm 2 Randomized Algorithm to Find Exact Median

Require: Set S
Ensure: Median of set S
$n = |S|$
if $(n < 10)$ **then**
  return median of S by sorting S
end if
$p = n^{1/4}$
$A =$ Random sample by picking each element of S independently with probability p.
Fix $k = n^{3/4}$
if $(|A| > 2k)$ **then** ▷ $O(1)$ comparisons
  return STOP, we have gone through bad sample
end if
Fix $t = \sqrt{k \log(n)}$
Sort(A) ▷ $O(k \log(k)) = o(n)$ comparisons
$a = (k/2 - t)^{th}$ element of A
$b = (k/2 + t + 1)^{th}$ element of A
$(R_a, S') =$ (Rank of a in set S, Set of elements from S greater than s) ▷ n comparisons
if $(R_a > n/2$ or $R_a < n/2 - 2t/p)$ **then** ▷ $O(1)$ comparisons
  return STOP, we have gone through bad sample
end if
$(R_b, Q) =$ (Rank of b in set S,Set of elements from S' smaller than b) ▷ $0.5n + o(n)$ comparisons
if $(R_b < n/2$ or $R_b > n/2 + 2t/p)$ **then** ▷ $O(1)$ comparisons
  return STOP, we have gone through bad sample
end if
Sort(Q) ▷ $O\left(\frac{4t}{p} \log\left(\frac{4t}{p}\right)\right) = o(n)$ comparisons
$Q =$ Set of elements from S lying between a and b
return $Q[n/2 - R_a]$

For very small input, it will be tedious to compute all these constants, and

Theorem 1.4.1. *Above algorithm takes $1.5n + o(n)$ comparison and returns the median of the set S with high probability.*

Proof. High probability bound on the correctness of the result from the algorithm can be seen as:

$$\mathbb{P}(\text{Algorithm returns correct median}) = 1 - \text{Eror Probability} = 1 - O(1/n^{O(1)})$$

This is a decent bound for the algorithm. The total number of comparisons in the algorithm can be found by adding the number of comparisons in different steps. From the above algorithm

$$\begin{aligned}
\text{No. of comaprison} &= O(1) + O(k\log(k)) + n + O(1) + 0.5 + O(4t/p) + O(1) + O\Big(\frac{4t}{p}\log\big(\frac{4t}{p}\big)\Big) \\
&= O(n^{0.75}\log(n^{0.75})) + 1.5n + O(n^{5/8}\log(n^{5/8})) \\
&= 1.5n + O(n^{0.75}\log(n)) \\
&= 1.5n + o(n)
\end{aligned}$$

Hence, our algorithm computes the median of the given set S in just $1.5n+o(n)$ comparisons. This completes the proof. □

Chapter 2

Practical Verification & Empirical Results

2.1 Experiments

We have implemented our algorithm in R language and used RStudio software to run the algorithm. To test the consistency and error probability of the algorithm, we ran it on the random inputs of size $n = 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8$ *each* 2000 *times. Below, we represent the fraction of time we got the correct output from our algorithm.*

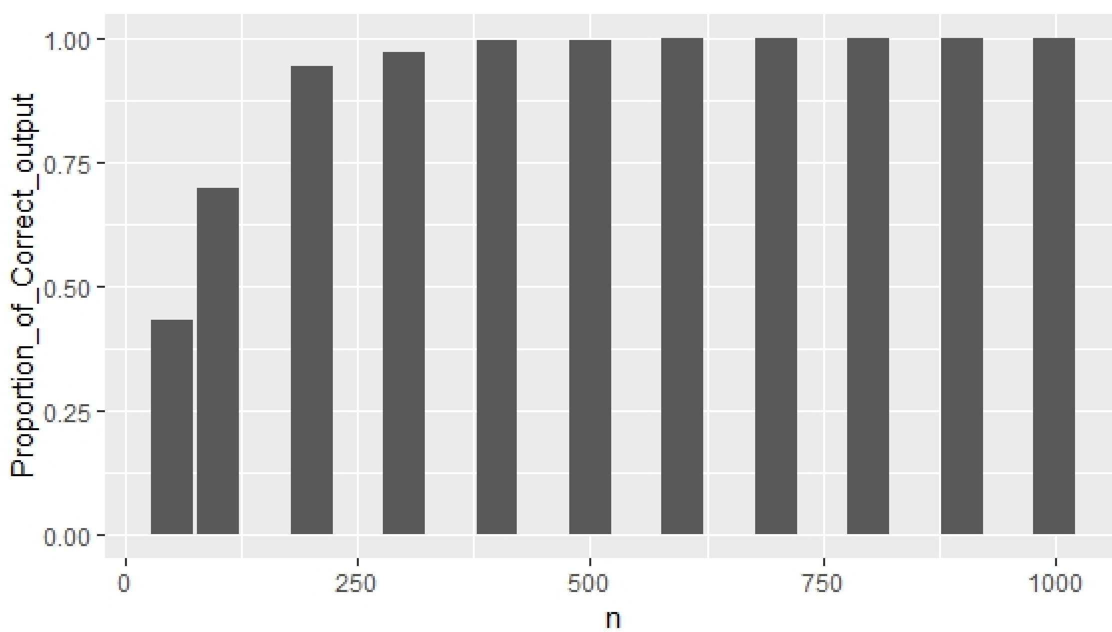

Figure 2.1: Proportion of correct output vs n

Datatable for the result of the experiment is as follows:

Table 2.1: Input Size vs Accuracy

Input size (n)	Number of correct runs out of 2000	Proportion
50	858	0.4290
100	1394	0.6970
200	1888	0.9440
300	1946	0.9730
400	1991	0.9955
500	1996	0.9980
600	1999	0.9995
700	2000	1.0000
800	1999	0.9995
900	2000	1.0000
1000	2000	1.0000

For inputs having a size greater than 1000*, our algorithm gives the correct output almost indeed. These results align with our expectations, as we established that the error probability of our algorithm is just* $O(1/n)$*.*

2.2 Time Efficiency Analysis

Now, we will discuss the speed of our algorithm with the built-in algorithm (Probably the best deterministic algorithm available). The best deterministic algorithm to find the median takes $2.95n$ *comparisons. We have found the difference between the time taken by the algorithm i.e.*

$$\Delta t = T_{randomized\ algorithm} - T_{built-in}$$

Where $T_{randomized\ algorithm}$ *is the time taken by our given algorithm and* $T_{built-in}$ *is the time taken by the In-built median() function in R. Since for each* n*, we have compared both the algorithms* 2000 *times, we have sufficient samples to make inferences about the* Δt*. Common statistics for* Δt *is as follows:*

Table 2.2: Statistics of Δt from samples

Input size (n)	mean of Δt	median of Δt	mode of Δt	SD of Δt
50	0.000070	0.00	0.00	0.001896549
100	0.000115	0.00	0.00	0.002596179
1000	0.000185	0.00	0.00	0.003524484
10000	0.000270	0.00	0.00	0.003799252
100000	0.001125	0.00	0.00	0.008355793
1000000	0.005370	0.01	0.00	0.012591361
10000000	0.046355	0.04	0.03	0.034781732
100000000	0.194850	0.19	0.18	0.217733280

One can observe that Δt is a continuous random variable because the time taken by our randomized Monte Carlo algorithm is random. From the gained samples, we have estimated the approximate density function for the RV Δt for different values of n. Significant results are as follows:

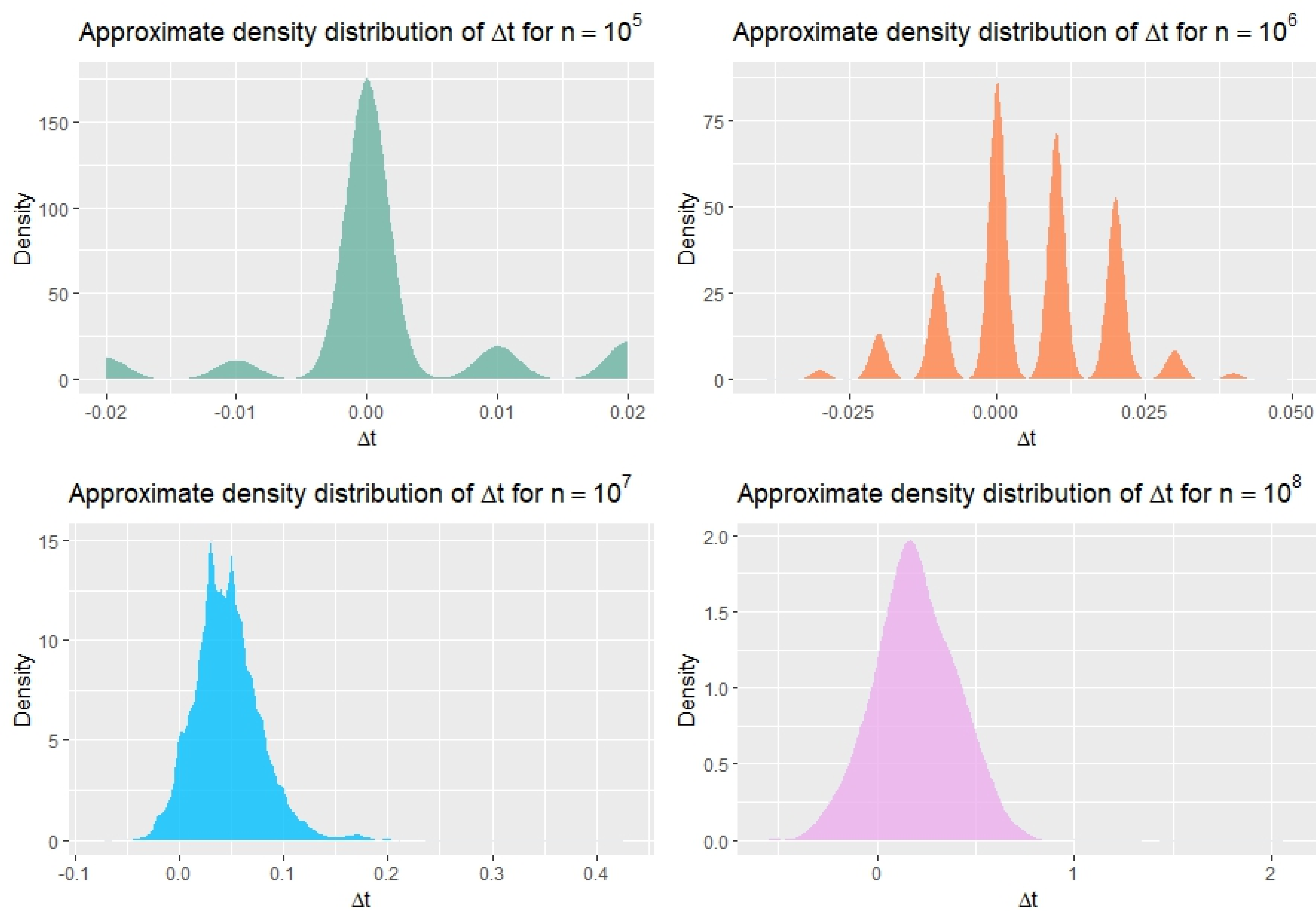

Figure 2.2: Approximate Density function for Δt

After examining the above results, we can say that our algorithm is performing no better than the $In - Built$ algorithm to find the median. The potential reason for this fact is that we need to make a copy of the subset 2 times in the algorithm. This makes our algorithm a little bit more time-consuming than the $built - in$ function. We have also computed the average time taken by the given algorithm, and we got the linear dependence of elapsed time on n as expected. The obtained data is shown below:

Table 2.3: Average time taken by Proposed Algorithm for different sized outputs

n	50	100	200	300	400	500	600	700	800	900	10^3	10^4	10^5	10^6	10^7	10^8
Time (ms)	1.4	0.5	0.2	0.1	0.0	0.0	0.2	0.6	0.3	0.4	1.3	3.9	5.0	29.7	271.3	2528.8

Now, we will compare the number of comparisons made in the inbuilt algorithm and our implementation. For this purpose, we are using C++. The obtained results are as follows:

Input size (n)	Randomized ($p = n^{-0.25}$)	Randomized ($p = n^{-0.25}$)	Deterministic
100	322	273	277
200	971	524	725
400	2753	1438	1201
600	3737	4613	1090
800	4841	4963	1553
900	5368	5664	1798
1000	5927	6249	1825
10000	42877	49877	30830
100000	318251	345892	234259
1000000	2481772	2652757	2675274
10000000	20502494	20927133	28961496
100000000	179674957	185461673	289289681

Table 2.4: Number of Comparisons

Plots for the number of comparisons for both approaches are shown below.

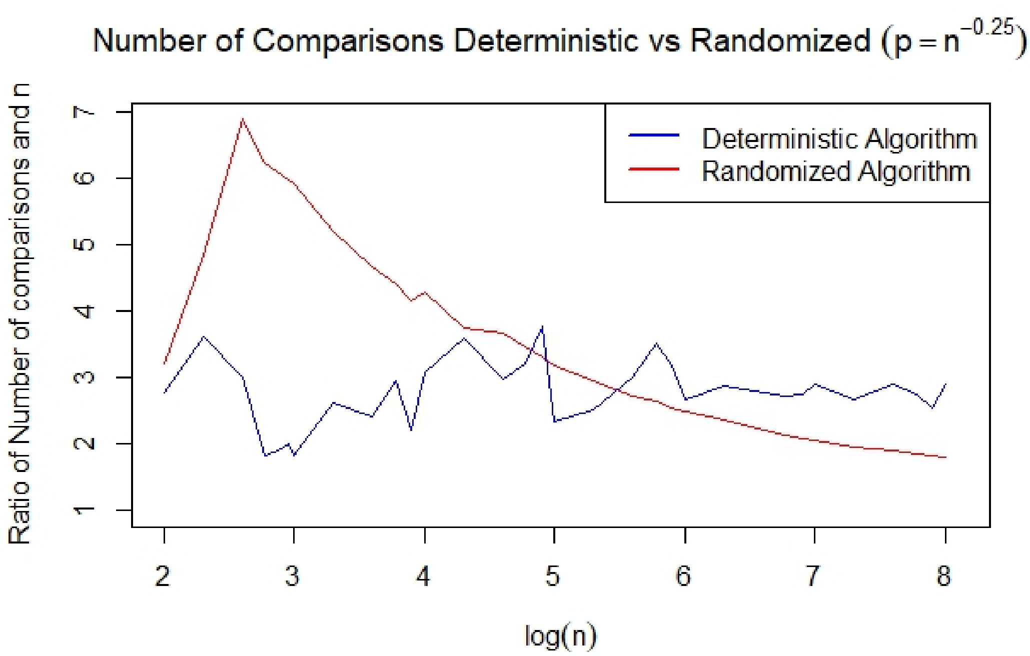

Figure 2.3: Deterministic vs Randomized for $p = n^{-0.25}$

Thus, we can say that our algorithm verified all the theoretical results we obtained during

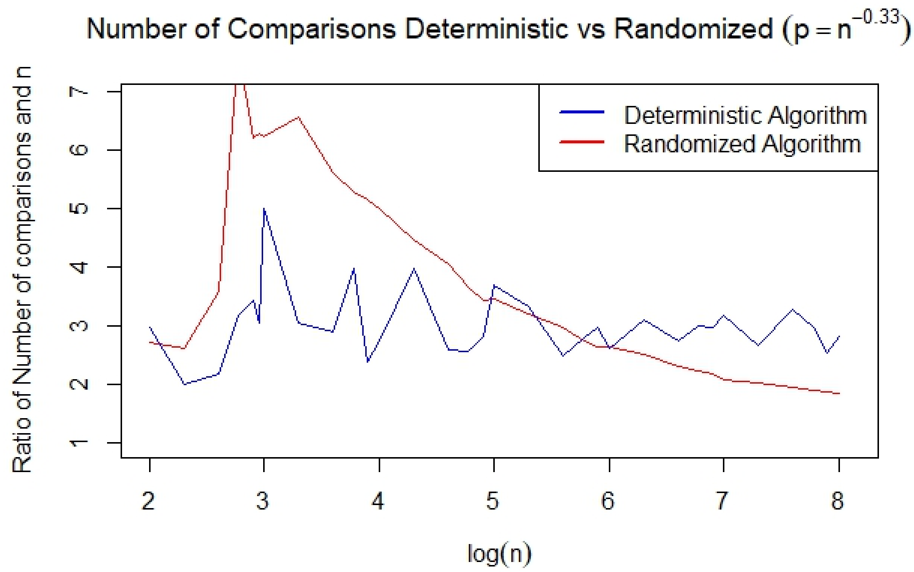

Figure 2.4: Deterministic vs Randomized for $p = n^{-0.33}$

the analysis. It returns the median in $1.5n + o(n)$ *comparisons which is much better than the best deterministic algorithm available that takes* $2.95n$ *comparison to compute the median. Moreover, it has been proven that any deterministic algorithm must take* 2.01 *comparisons to compute the median. So, our proposed algorithm will asymptotically perform better if somehow we manage the copying of the subset in the new set.*

This ends the empirical testing of the algorithm.

Chapter 3

Conclusion & Acknowledgement

In conclusion, this project has delved into the realm of randomized algorithms, particularly focusing on the efficient computation of the median using random sampling techniques. Through rigorous analysis and exploration, we have demonstrated the viability of leveraging random subsets to approximate the median of a given set efficiently. By sampling a small random subset of elements and carefully analyzing their relationship with the original set, we have elucidated a method that offers promising results in terms of computational complexity and accuracy.

Furthermore, we express our sincere gratitude to our instructor for their invaluable guidance, support, and expertise throughout this project. Their dedication, insights, and willingness to share knowledge have been instrumental in shaping our understanding and approach. We deeply appreciate the opportunity provided by our instructor to engage in meaningful discussions, receive hints, and navigate the intricacies of the project. Their mentorship has been instrumental in our learning journey and has enriched our understanding of randomized algorithms.

In essence, this project not only sheds light on the significance of randomized algorithms in computational tasks but also underscores the importance of mentorship and collaboration in academic endeavours. As we conclude this project, we reflect on the knowledge gained, the challenges overcome, and the invaluable contributions of our instructor. We look forward to applying these insights in future endeavours and continuing our exploration of computational algorithms.

Bibliography

[1] Lecture Slides $CS648$

[2] Crucial hints and multiple discussions from the Instructor