CS648: Mini Project

Siddharth, Krish

1 Problem: Finding median using $1.5n + o(n)$ comparisons

We need to design, analyse, and implement a median finding algorithm that performs asymptotically $1.5n$ comparisons with high probability.

2 Basic idea and overview:

We will try to reduce the size of the array to $o(n)$ by selecting two points close to and on either side of the median using randomization, using an idea similar to the approximate median algorithm. These two points will be called pivots (the one with a smaller rank called the first pivot and the one with a larger rank called the second pivot). We will then compare all elements of the array with both these pivots (first compared with the first pivot and then with the second pivot in case it is more than first pivot) and obtain the subset of the original array consisting of elements having rank between those of the pivots, as well as the rank of the median in this subset. We can then use sorting to obtain the median from this subset. Let l be the difference of ranks of the pivots (i.e., the size of the subset). Then for $\frac{n-l}{2}$ elements of the original array, only one comparison will be made during partition, and for $\frac{n+l}{2}$ elements, 2 comparisons will be made. Hence, the total comparisons made during partition will be $1 \times \frac{n-l}{2} + 2 \times \frac{n+l}{2} = 1.5n + \frac{l}{2}$. Comparisons made during sorting will be $l \log l$. If $l = o(n)$, say $l = n^a$ $(a < 1)$, the total comparisons made will be $1.5n + o(n)$. Hence, if the median lies in the subarray and its size is $o(n)$, then $1.5n + o(n)$ comparisons will be made. This will hold with high probability for our algorithm.

We can convert this into a Las Vegas algorithm by applying a deterministic $O(n)$ median-finding algorithm on the complete array in case either the median lies outside the subarray or l is not $o(n)$. Let p be the probability of failure; then the expected number of comparisons will be $(1.5n + o(n)) \times (1 - p) + cn \times p = 1.5n + p \times (cn - o(n))$. p will come out to be inverse exponential in n in our algorithm, and hence, this is still $1.5n + o(n)$.

3 Detailed Solution and Algorithm:

Let A be the given array of size n, assumed to be consisting of distinct integers (if they are not, we can break ties by adding small numbers to them). Let Arr$[i]$ denote the element in the array/set/multi-set Arr at rank i. Let us try to reduce the size of the array to n^a $(a < 1)$. To do so, we will select k elements from the array r.u.i. that will help us find elements with ranks near $\frac{n}{2} \pm \frac{n^a}{2}$ in the original array. Let the multi-set formed from these k elements be B. Rank(x) will denote the rank of x present in A in the array A. We kNow that if we select k elements r.u.i. from A, then on an expectation, the number of elements with rank $< \frac{n}{2} - \frac{n^a}{2}$ will be $k \times p$ (a randomly selected element has rank $< \frac{n}{2} - \frac{n}{2n^{1-a}}) = k \times \left(\frac{1}{2} - \frac{1}{2n^{1-a}}\right)$. Thus, this will also be the rank of the element in B which we will select from it to approximate the first pivot. Similarly, the second pivot will be at index $k\left(\frac{1}{2} + \frac{1}{2n^{1-a}}\right)$ in B. If we assume $k = n^b$, the pivots will be $B\left[\frac{n^b}{2} \pm \frac{n^b}{2n^{1-a}}\right]$. Let X_1 be the random variable equal to rank $\left(B\left[\frac{n^b}{2} - \frac{n^b}{2n^{1-a}}\right]\right)$ (rank of the first pivot) and similarly $X_2 = \text{rank}\left(B\left[\frac{n^b}{2} + \frac{n^b}{2n^{1-a}}\right]\right)$ (rank of the second pivot). Let C be the subset of A consisting of those elements with a rank between (and including) X_1 and X_2. If $X_1 \leq \frac{n}{2}$ and $X_2 \geq \frac{n}{2}$, then the median

is $C\left[\frac{n}{2} - X_1 + 1\right]$, which we calculate using sorting C using merge sort.

The pseudo code is written below:

Input: Array A with n elements
Output: Median of A
$k \leftarrow n^b$;
$t \leftarrow \frac{k}{2n^{1-a}}$;
Let B be a subset of A selected randomly and uniformly, containing k elements;
Sort B;
$p_1 \leftarrow B[\frac{k}{2} - t]$;
$p_2 \leftarrow B[\frac{k}{2} + t]$;
$C \leftarrow [\,]$;
$x_1 \leftarrow 0$ //will give rank of first pivot;
$x_2 \leftarrow 0$ //will give rank of second pivot;
for i ***from*** 1 ***to*** n **do**
  if $A[i] < p_1$ **then**
    $x_1 + +$;
    $x_2 + +$;
    continue;
  else
    if $A[i] \leq p_2$ **then**
      $x_2 + +$;
      append $A[i]$ into C;
    else
    end
  end
end
Sort C;
if $x_1 \leq \frac{n}{2} \leq x_2$ **then**
  return $C[\frac{n}{2} - x_1 + 1]$;
else
  $Median \leftarrow$ Compute the exact median by sorting entire array A;
  return $Median$;
end

Algorithm 1: Algorithm to find the median of an array

4 Probability Analysis:

We calculate the probability of failure of the event that the median lies in C and $|C| \leq 2n^{-a}$. Let E be the event of failure. Clearly, $E = E_1 \cup E_2$, where E_1 is the event that the median lies outside C and E_2 be the event $|C| > 2n^a$. Using the union theorem, $p(E) \leq p(E_1) + p(E_2)$. We will show exponential bound on both $p(E_1)$ and $p(E_2)$.

4.1 Showing bound on $p(E_1)$

Let X be the random variable denoting the number of elements from B with rank $< \frac{n}{2}$ in A. Now, clearly E_1 is equivalent to the event: $X \geq \frac{n^b}{2} + \frac{n^b}{2n^{1-a}} \cup X \leq \frac{n^b}{2} - \frac{n^b}{2n^{1-a}}$, or $|X - E[X]| \geq d$, $d = \frac{n^b}{2n^{1-a}}$. Now, clearly

X is a Bernoulli random variable with $p = \frac{1}{2}$, hence using Chernoff's bound, we have

$$P(E_1) = P(|X - E[X]| \geq d) \leq 2e^{-\left(\frac{d}{E[X]}\right)^2 \frac{E[X]}{3}}.$$

$d = \frac{n^b}{2n^{1-a}}$, $E[X] = \frac{n^b}{2}$, hence, $\frac{d}{E[X]} = \frac{1}{n^{1-a}} = n^{a-1}$

$$\implies P(E_1) \leq 2e^{-n^{2a-2} \times \frac{n^b}{6}} = 2e^{-\frac{n^{2a+b-2}}{6}} = 2e^{-g(n)} \text{ (say)}$$

So, we will select a and b such that they satisfy $2a + b > 2$.

4.2 Showing bound on $p(E_2)$

We have $E_2 \subseteq E_{2,1} \cup E_{2,2}$, where $E_{2,1}$ is the event $X_1 \leq \frac{n}{2} - n^a$ and $E_{2,2}$ is the event $X_2 \geq \frac{n}{2} + n^a$, since if $X_1 > \frac{n}{2} - n^a$ and $X_2 < \frac{n}{2} + n^a$, then $|X_1 - X_2| < 2n^a$, hence if E_2 occurs, at least one of $E_{2,1}$ or $E_{2,2}$ occurs (note that $X_1 \leq X_2$). Let Y denote the random variable denoting the number of elements in B with rank $\leq \frac{n}{2} - n^a$ in A. Clearly, $E_{2,1}$ is equal to the event $Y \geq \frac{n^b}{2} - \frac{n^b}{2n^{1-a}}$. Also, Y is a Bernoulli random variable with $p = \frac{1}{2} - \frac{1}{n^{1-a}}$ and $E[Y] = \frac{n^b}{2} - \frac{n^b}{n^{1-a}}$. Using Chernoff's bound,

$$P[Y \geq (1+\delta) E[Y]] \leq e^{-\frac{\delta^2 E[Y]}{3}}.$$

Keeping $(1+\delta)\left(\frac{n^b}{2} - \frac{n^b}{n^{1-a}}\right)$ to be $\frac{n^b}{2} - \frac{n^b}{2n^{1-a}}$, we have,

$$(1+\delta)\left(\frac{n^b}{2} - \frac{n^b}{n^{1-a}}\right) = \frac{n^b}{2} - \frac{n^b}{2n^{1-a}}$$

$$\implies \delta = \frac{1}{n^{1-a} - 2} \geq \frac{1}{n^{1-a}}$$

$$\implies \delta E[Y] = \frac{n^b}{2n^{1-a}}$$

Hence,

$$P(E_{2,1}) = P\left[Y \geq \frac{n^b}{2} - \frac{n^b}{2n^{1-a}}\right] \leq e^{-\delta\left(\frac{E[Y]\delta}{3}\right)}$$

$$\leq e^{-\left(\frac{1}{n^{1-a}}\right)\frac{n^b}{6n^{1-a}}} = e^{-\frac{n^{2a+b-2}}{6}} = e^{-g(n)}$$

$$\implies P(E_{2,1}) \leq e^{-g(n)}.$$

4.3 Final Result

By symmetry, $P(E_{2,2}) \leq e^{-g(n)}$ and using the union theorem,

$$P(E_2) \leq P(E_{2,1}) + P(E_{2,2}) \leq 2e^{-g(n)}$$

So,

$$P(E) \leq P(E_1) + P(E_2) \leq 4e^{-g(n)}$$

Hence, with probability more than $1 - 4e^{-g(n)}$, the number of comparisons is

$$\leq \underbrace{1.5n + \frac{2n^a}{2}}_{\text{Partition}} + \underbrace{n^b \log n^b}_{\text{Sorting multiset B}} + \underbrace{2n^a \log 2n^a}_{\text{Sorting sub-array C}}$$

$$= 1.5n + o(n)$$

and with probability $4e^{-g(n)}$ it is cn for some c. Here $g(n) = \frac{n^{2a+b-2}}{6}$.

5 Experimental results

As a programming optimization, we use modified QuickSelect algorithm in our code to find the pivots from B and median from C instead of sorting (which practically takes significant comparisons, although theoretically it is still $o(n)$). The algorithm takes around $1.6n$ comparisons for $a = b = 0.7$ and $n = 3 \times 10^7$.