Randomized Median and Quick Select

Source: Dasgupta et al.

Defn: The i^{th} order statistic of a set of n elements is the i^{th} smallest element.

The minimum of a given set of elements is the 1^{st} order statistic and the maximum is the n^{th} order statistic

Both min|max can be found in $\Theta(n)$ time (with some improvements)

Median: (the mid/halfway) is the 50^{th} percentile.
If n is odd, the median is uniquely occuring at $i = \left(\frac{n+1}{2}\right)$ when all elements are sorted.

When n is even, the median occurs at $i = n/2$ and $i = \frac{n}{2} + 1$

The i^{th} order statistic of an array of n elements in a sorted order is $A[i]$, where all elements $A[1,2,\ldots,i-1]$ are smaller and $A[i+1,\ldots,n]$ are greater than $A[i]$. (assuming wlog that all elements are distinct)

Problem: Find the i^{th} order Statistic?

Naive: $\underbrace{\text{Sort the array and return}}_{\Omega(n\lg n) \text{ for comparison-based sorts}}$ $A[i]$

[Quick] Selection is a simpler problem than Sorting.
We should be able to do better!

<u>QuickSelect</u> : a randomized modification of Quicksort that gives $O(n)$ on average.

Recall, that in Quicksort, we find the <u>pivot</u> and then recursively sort subarrays on both sides of the pivot.

The partition algorithm in Quicksort, when returns gives two partitions :

Left : those items less or equal to the pivot.
Right : those items greater than the pivot.

Randomized QuickSelect is based on the same idea: Only since we're interested in the i^{th} order statistic, we perform <u>recursion on just one side of the partition.</u>

<u>Example</u> (from Dasgupta et al)

Given an array A with the following elements in an arbitrary order:

Index	1	2	3	4	5	6	7	8	9	10	11
A[i]	2	36	5	21	8	13	11	20	5	4	1

Suppose, we randomly choose the pivot $v = 5$, then partition A into three groups ($O(n)$ scan)

A_L : 2 4 1 (all elements smaller than 5 (v))
A_v : 5 5 (all elements same as v)
A_R : 36 21 8 13 11 20 (all elements greater than v)

Now, if our desired i^{th} statistic is less than pivot v, we'll search in A_L or if it is greater than the pivot, we search A_R.

The recursion then becomes:

$$\text{Selection}(A, i): \begin{cases} \text{Selection}(A_L, i), & \text{if } i \leq |A_L| \\ v, & \text{if } |A_L| < i \leq |A_L| + |A_v| \\ \text{Selection}(A_R, i - |A_L| - |A_v|) \leftarrow \text{why?}, & \text{if } i > |A_L| + |A_v| \end{cases}$$

<u>Example</u>: $A = [4, 8, 3, 9, 15, 11, 2\}$
let $i = 5$ (5^{th} ordeer statistic)

Let $v = 4$ (first element) as the pivot

then $A_L = \{3, 2\}$, $A_v = \{4\}$, $A_R = \{8, 9, 15, 11\}$

$$\because \underbrace{i}_{=5} > \underbrace{|A_L|}_{=2} + \underbrace{|A_v|}_{=1}$$

$$\therefore \text{Selection}(A_R, \underbrace{\underbrace{i}_{=5} - \underbrace{|A_L|}_{=2} - \underbrace{|A_v|}_{=1}}_{=2})$$

ie $A_L' = \{8, 9\}$, $A_v = \{11\}$, $A_R = \{15\}$ (arbitrerily choosing pivot)

$\because i' = i - |A_L| - |A_v| = 2$ and $i' \leq |A_L'|$

$\therefore$ Run Selection (A_L', i')

Hence, we get 9

↓

Sorted A: $[2, 3, 4, 8, \underset{\textcircled{5}}{9}, 11, 15]$

Analysis of Quick Select (Source: Dasgupta et al.)

Given an array A of n elements, we can compute subarrays A_L, A_V, A_R in $O(n)$ time and in-place.

Effect: Shrink $|A|$ into at most $\max(|A_L|, |A_R|)$.

Recall that $\underbrace{\text{our partition algorithm}}_{\text{finding a random pivot and splitting}}$ works in $O(n)$.

The recurrence can be written as:

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n) = \underset{\downarrow}{\Theta(n)} \quad \left[\text{Solve using unrolling}\right]$$

If $|A_L|, |A_R| \simeq \frac{1}{2}|A|$, we get the best-case (highly unlikely)

Worst-case: The running time of Quick Select depends upon the choice of the pivot. It is possible though highly unlikely that we always end up with a split where we shrink the input array as one element at a time!

i.e., $T(n) = T(n-1) + \cdots + T\left(\frac{n}{2}\right) + \cdots$

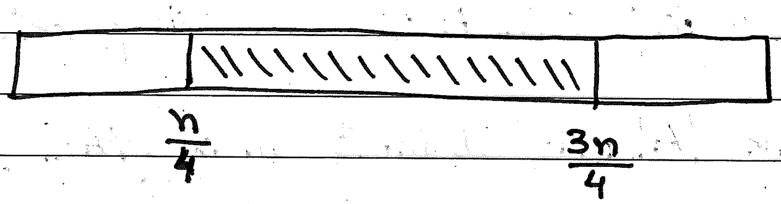
or

$T(n) = \Theta(n^2)$ } Also, very highly unlikely!

The average-case is b/w the best-case and worst-case, i.e. b/w $\Theta(n)$ and $\Theta(n^2)$.

Fortunately, it is close to the best-case!

Average-case Analysis of Quick Select



{ A sorted array Of n elements

Suppose, we pick the pivot element at random given an array of n elements.

Then half of the time, the pivot element is expected to be from the central-half (shaded) of the array.

Now, whenever the portition algorithm uses the pivot from between positions $n/4$ and $3n/4$, the largest remaining subarray contain at most $3n/4$ elements. (in $\underbrace{\text{case of a sorted}}_{\text{worst-case.}}$ array).

So, we'll have a good pivot if it lies between the 25th and 75th percentile.

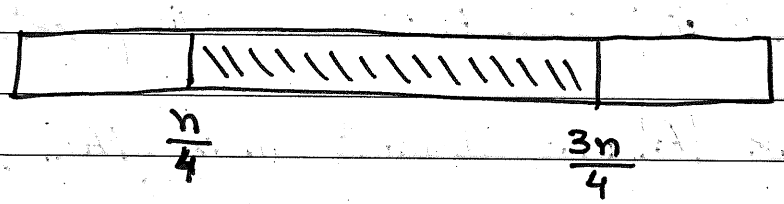
[**Lemma**: On average, a coin needs to be tossed 2 times before a head is seen.]

So, we have a 50% chance that our chosen pivot v is good (ie, b/w $\frac{n}{4}$ and $\frac{3n}{4}$ in an sorted array).

Then, in the first stage, the $\underbrace{\text{largest}}_{\text{possible}}$ subarray is $\frac{3n}{4}$.

In the second stage, it's expected to be $\frac{3}{4}\left(\frac{3n}{4}\right)$ $= \left(\frac{3}{4}\right)^2 n$. and so at the jth stage, the maximum possible size of the largest subarray

Average-case Analysis of Quick Select



A sorted array of n elements

Suppose, we pick the pivot element at random given an array of n elements.

Then half of the time, the pivot element is expected to be from the central-half (shaded) of the array.

Now, whenever the portition algorithm uses the pivot from between positions $n/4$ and $3n/4$, the largest remaining subarray contain at most $3n/4$ elements. (in $\underbrace{\text{case of a sorted}}_{\text{worst-case.}}$ array).

So, we'll have a good pivot if it lies between the 25th and 75th percentile.

[Lemma: On average, a coin needs to be tossed 2 times before a head is seen.]

So, we have a 50% chance that our chosen pivot v is good (ie, b/w $\frac{n}{4}$ and $\frac{3n}{4}$ in an sorted array).

Then, in the first stage, the $\underbrace{\text{largest}}_{\text{possible}}$ subarray is $\frac{3n}{4}$.

In the second stage, it's expected to be $\frac{3}{4}\left(\frac{3n}{4}\right)$ $= \left(\frac{3}{4}\right)^2 n.$ and so at the jth stage, the maximum possible size of the largest subarray

∴ # comparisons taken by the algorithm, $X = X_0 + X_1 + X_2 + \ldots$

Where,

X_j is the expected no. of comparisons taken by the algorithm in stage j (atmost $(3/4)^j n$ size).

ie

$$E[X] = \sum_j 2n\left(\frac{3}{4}\right)^j$$

$$\simeq O(n)$$

$T(n) = T(\frac{3}{4}n) + O(n)$

$\boxed{T(n) = \Theta(n)}$

We assume that when we pick a pivot randomly, on average twice, we can expect a decent pivot (half the time on average).

Lemma: If we repeatedly perform independent trials of an experiment, each of which succeeds with some probability $p > 0$, then the expected time of trials that we need to perform until the first success is $1/p$.

$$E[X] = \sum_j j \cdot Pr(X = j)$$

Recall,
$E[X] = \sum_x x . Pr(X = x)$

Let ~~X~~ X be a R.V. equal to the # of trials for $j > 0$, we have

$$Pr(X = j) = (1-p)^{j-1} . p \quad \{\text{Geometric dis b}\}$$

$$\Rightarrow E[X] = \sum_j j \cdot (1-p)^{j-1} . p = \frac{p}{(1-p)} . \underbrace{\sum_j j(1-p)^j}_{\sum k x^k = \frac{x}{(1-x)^2}}$$

$$= \frac{p}{(1-p)} \frac{(1-p)}{(1-1+p)^2} = \frac{p}{(1-p)} \frac{(1-p)}{p^2}$$

$$\therefore \boxed{E[X] = 1/p}$$

Page #