
G
O
D
O
O
O
D

Order Statistics

The ith order statistics of a set of n elements is the ith smallest element

You're given n elements,

→ 5th order statistics = 5th smallest element

For example, the minimum of a set of elements is the first order statistics ($i=1$), and the maximum is the nth order statistics ($i=n$).

A median is the "halfway point" of the set.

n is odd ⇒ median at $i = (n+1)/2$

n is even ⇒ two medians at $i = \frac{n}{2}$, $\frac{n}{2}+1$, take avg.

We assume that the set contains distinct numbers.

We formally specify the selection problem as follows

I/p: A set A of n (distinct) numbers and an int i

$$1 \leq i \leq n$$

O/p: The element $x \in A$, that is larger than exactly $i-1$ other elements of A.

Sol^n:→ One possible sol^n is to sort the elements using heapsort / mergesort & simply return the i^{th} element in the sorted array

→ $O(n \log n)$

$i = 1$:- linear time algo for finding the minimum element

$i = 2$ → $n + n$ → $O(n)$

$i = k$ $O(kn)$

if $k < O(\log n)$ → Keep on finding smaller, second smaller.

$i = n$ find max → $O(n)$

$i = n-1$ second max → $O(2n)$

$n - i < O(\log n)$ → faster

1. Sort these & get ith element → $O(n \log n)$

2. Find ith smallest / $(n-i)$th largest element

→ $\min(i, (n-i))$

Worst case $O(n^2)$

3. Use Heaps : I can find Kth smallest element in $O(K \log n)$

* I need to create heap → $O(n)$

$O(n + K \log K)$

Worst case: $K = n/2$ → $O(n \log n)$

Can we do better?

→ Can I achieve a worst case $O(n)$ time algo for finding its smallest element?

- Finding max & min.
- General selection

max :→ $n-1$ comparisons

```
max = A[1]
for i = 2 → A.length
    if A[i] > max
        max = A[i]
return max
```

Think of it like a tournament

n players.

Find a winner

Game: comparison → Smaller element loses

* Only one element can lose in a game

for an element to win, $n-1$ elements have to loose

$\Rightarrow$ Atleast $n-1$ comparisons are required.

Same thing goes for finding the min. element

→ Max & Min. Simultaneously :- Can you do better?

→ $2n-2$ comparisons (Pairwise)

Better - $3\lfloor n/2 \rfloor$ comparisons

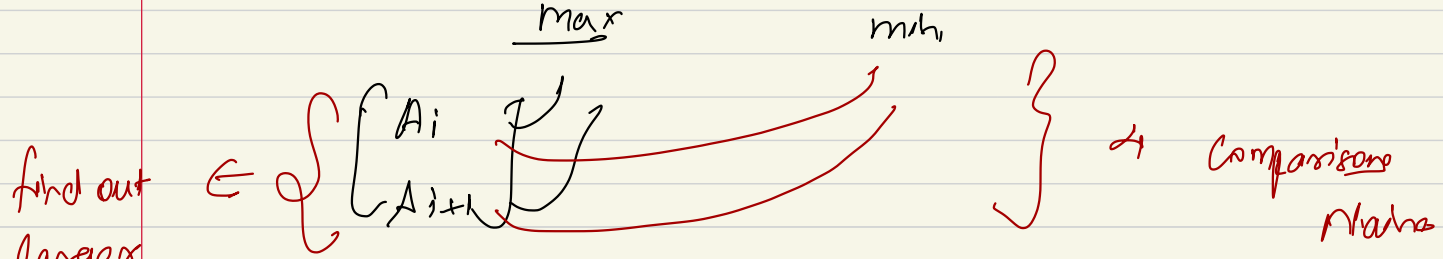
— Maintain max & min seen so far

For each element

— Rather than comparing it against the current min & max

$\Rightarrow$ 2 comp. / element

— we process the elements in pairs



find out larger element.

let Ai larger than Ai+1

$\Rightarrow$ Compare Ai with max

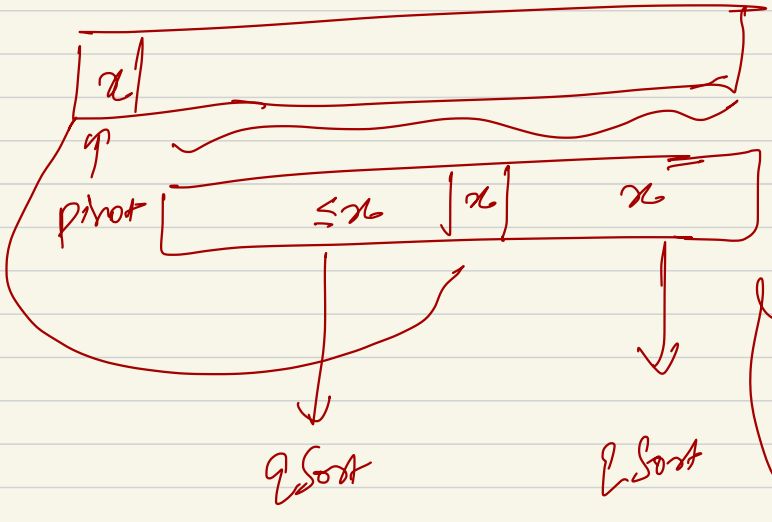
Ai+1 with min

$\Rightarrow$ 3 comparisons / 2 elements

$\Rightarrow \lfloor \frac{3n}{2} \rfloor$ / comparisons

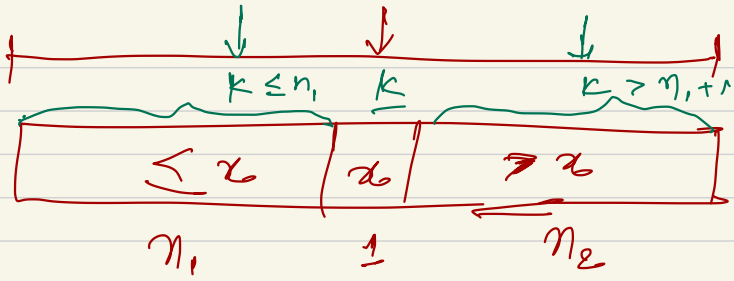
Selection in expected linear time?

Can we use a variation of quick sort



partition step of the quick sort, we choose a pivot & partition in two parts

As a result of partition x comes in its correct place for the final sorted array.



$n_1 + 1 + n_2 = n$

kth smallest element

if $k = n_1 + 1$ $\Rightarrow$ pivot itself is the kth smallest element of A

if $k \le n_1$ $\rightarrow$ we recursively find the kth smallest element in the left subarray

or if $k > n_1 + 1$ we recursively find the $(k - n_1 - 1)$th smallest element in the right subarray

The size of the subproblem depends on the sizes of left & right subarrays

Suppose $n_1 = n_2 \approx n/2$

The running time of the selection algo

$$T(n) = T(n/2) + O(n)$$

$$\rightarrow T(n) = O(n)$$

However, the worst case performance is —

If in each recursive call, the size decreases just by 1

$$T(n) = T(n-1) + O(n)$$

$$\Rightarrow T(n) = \Theta(n^2)$$

* Partitioning based on Quick Sort

→ Worst case $O(n^2)$

→ Average case → $O(n)$

* If we can find a constant (say 9/10) smaller than 1 s.t

$$T(n) \leq T\left(\frac{9}{10} n\right) + O(n)$$

$$\Rightarrow \boxed{T(n) = O(n)}$$

Worst case linear time algo

Target :→ Can we find a way to ensure that my subproblem reduces by a constant factor everytime !!

* Partitioning may be skew, but we'll be able to throw away a fraction of the array in each recursive call.

$\Rightarrow$ linear-time (worst-case) algo

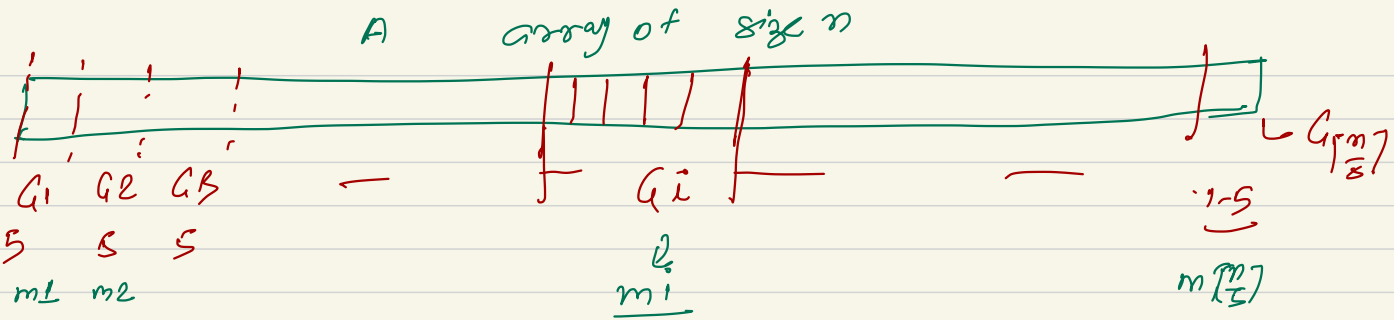
Blum, Floyd, Pratt, Rivest, Tarjan

* Interesting part is how they select the pivot

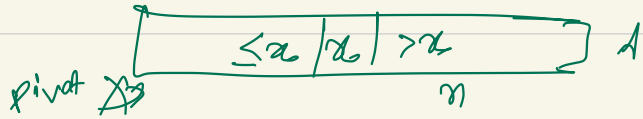
* Every recursive call reduces the size of the array at least by $3/10$.

$$x \Rightarrow \frac{7}{10}x$$

* But firstly the pivot requires another recursive call on an array of size $\lceil n/5 \rceil$.



- Break A in $\lceil \frac{n}{5} \rceil$ groups of 5 elements each (except the last may have 1-4 elements)
- Compute median in each group → all the $\lceil \frac{n}{5} \rceil$ medians can be computed in $O(n)$ time
- Recursively call the algo on this array of medians & obtain the median of medians. Call this element x.
- Take x as the pivot for the partitioning algo



Is x a good choice?

Can it help throw away a fraction of elements?

x

array of $\lceil \frac{n}{5} \rceil$ medians

atleast $\frac{\lceil \lceil \frac{n}{5} \rceil / 2 \rceil - 2}{\geq \frac{n}{10}}$ are $< x$

For each of these medians $m < x$, the corresponding

$\{ \underbrace{x_1 \; x_2}_{<m} \; m \; \underbrace{x_4 \; x_5}_{>m} \}$

group contains 2 elements less than m

$\Rightarrow$ less than x.

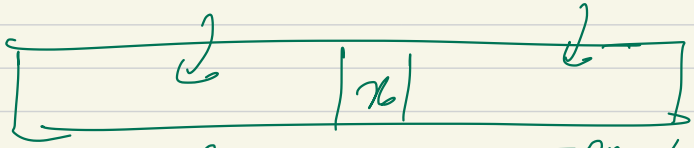
$\Rightarrow$ The number of elements of A that are less than x

$$\geq 3\left(\left\lfloor \left\lfloor \frac{n}{5} \right\rfloor / 2 \right\rfloor - 2\right)$$

$$\geq \frac{3n}{10} - 6$$

By asymmetric argument, number of elements larger than x

$$\geq \frac{3n}{10} - 6.$$



⇒ The next recursive call is made on a subarray of size $\leq \frac{7n}{10} + 5$

* You required a recursive call to find median of $\lceil \frac{n}{5} \rceil$ medians

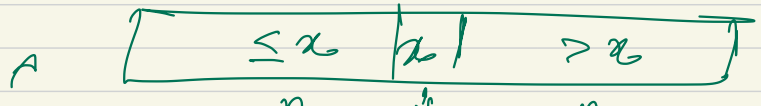
$$\leq \frac{n}{5} + 1$$

$$T(n) = \underbrace{T\left(\frac{n}{5} + 1\right)}_{\text{find pivot}} + T\left(\frac{7n}{10} + 5\right) + O(n)$$

$$T(n) = O(n)$$

⇒ Solve kth order statistics in worst case linear time.

* Use quick sort partitioning based algo



choose a pivot x

$|n_1 - n_2| \leq \frac{2n}{10} - 6$

$\leq \frac{7n}{10} + 6$

$k = n_1 + 1 \rightarrow$ ans is x

$k < n_1 + 1 \rightarrow$ recursively go to left

$> n_1 + 1$ ——— right

median of medians

$\lceil \frac{n}{5} \rceil$ groups

- get median
- take their median (recursively)

$T(n) \leq T\left(\frac{n}{5} + 1\right) + T\left(\frac{7n}{10} + 6\right) + O(n)$

$\Rightarrow T(n) = O(n)$