DELUXE
PAGE NO.:
DATE:

Order Statistics

⇒ Order statistics is a name given to the "k-th" element in an N-element sorted sequence in ascending order.

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

For example :- Minimum → 1^{st} order statistic
Maximum → n^{th} order statistic
Median → $\left(\frac{n+1}{2}\right)^{th}$ order statistic (n is odd)
$\left(\frac{n}{2}+1\right)^{th}$ order statistic (n is even)

In general, medians occurs at
$i = \left\lfloor \frac{(n+1)}{2} \right\rfloor$ (lower median) and
$i = \left\lceil \frac{n+1}{2} \right\rceil$ (upper median).

Here, we will assume the median to be the lower median i.e. $\left\lfloor \frac{n+1}{2} \right\rfloor$.

⇒ The goal is to find the i^{th} order statistic from a set of n distinct elements.

⇒ We can formally specify the selection problem as follows :-

DELUXE
PAGE NO.:
DATE:

Input: A set of n (distinct) elements and an integer i, with $1 \leq i \leq n$.

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

⇒ The most straightforward approach to select the i^{th}-order statistic is to simply sort the array and then pick up the i^{th} number. But the time complexity would be $O(n \log n)$. [using heap sort or merge sort]

Can we do better than $O(n \log n)$?

Let us examine the running time for different cases.

I) Minimum and Maximum

(1^{st}-order statistic and n^{th}-order statistic)

```
MINIMUM(A)
1) min = A[1]
2) for i=2 to n
3)     if min > A[i]
4)         min = A[i]
5) return min
```

(n-1) comparisons would be needed to select the minimum element in an array.

Similarly, (n-1) comparisons are needed to select the maximum element in an array.

DELUXE
PAGE NO.:
DATE:

So, we can conclude that find the 1^{st}-order statistic or the n^{th}-order statistic can be performed in $O(n)$ (linear time).

II) Simultaneous Minimum and Maximum

⇒ In many applications, we must find both the minimum and the maximum of a set of n elements. For example, a graphics program may need to scale a set of (x,y) data to fit onto a rectangular display screen or other graphical output device. To do so, the program must first determine the minimum and maximum value of each coordinate.

⇒ Finding minimum and maximum independently, would take $(n-1)+(n-1) = (2n-2)$ comparisons. Is it optimal? Can we do better than this?

⇒ Yes, we can find the minimum and maximum in an array of n distinct elements, which require less than $(2n-2)$ comparisons. If we read the array only once, then this task can be performed in less than $(2n-2)$ comparisons.

DELUXE
PAGE NO.:
DATE:

⇒ Pair-wise comparison

(i) n is even

7 3 8 1 4 5

(7 3) (8 1) (4 5) [forming pairs]

gmax gmin lmax lmin

gmax = global max.
gmin = global min.
lmax = local max.
lmin = local min.

Comparing lmax with gmax and lmin with gmin we get,

gmax gmin

(7 3) (8 1) (4 5)

(7 3) (8 1) (4 5)

gmax gmin lmin lmax

Again comparing lmax with gmax, and lmin with gmin we get,

gmax gmin

(7 3) (8 1) (4 5)

* No. of comparisons = 1st comparison (initial comp. between first two elements) + for each pair we need 3 comparisons. (one local and two global)

$= 1 + 3\left(\frac{n-2}{2}\right)$

(b/w first two elements) (∵ there are $\frac{n-2}{2}$ pairs)

$= 1 + 3\left(\frac{n-2}{2}\right)$

$= 1 + \frac{3n}{2} - \frac{6}{2}$

No. of comparisons $= \frac{3n}{2} - 2 = O(n)$

$\leq (2n-2)$

(ii) n is odd

5 6 1 0 2

gmin → 5 (6 1) (0 2)

gmax

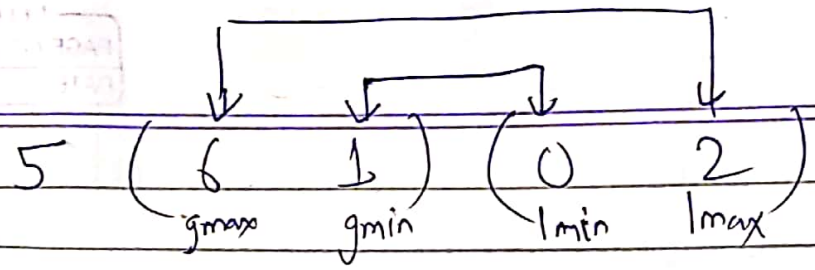
(Here, the first element is considered as gmin as well as gmax)

gmin → 5 (6 1) (0 2)

gmax lmax lmin

5 (6 1) (0 2)

gmax gmin



5 (6 1) (0 2)
gmax gmin

* No. of comparisons = for each pair we need 3 comparisons (one local and two global)

$$= 3\left(\frac{n-1}{2}\right)$$

$$= \frac{3n}{2} - \frac{3}{2} = O(n)$$

$$< (2n-2)$$

Thus, in either case (n is even or odd), the total number of comparisons is at most $3\lfloor \frac{n}{2} \rfloor$.

III) General Selection (i^{th}-minimum)

⇒ We have observed that finding the minimum/maximum and simultaneous minimum-maximum can be performed in $O(n)$ (linear time).

DELUXE
PAGE NO.:
DATE:

⇒ Can we perform the general selection i.e. finding the k^{th}-minimum in linear time? Which approach should be used to perform this task in linear time?

⇒ We apply "Divide-and-Conquer" strategy as follows:-

RANDOMIZED-SELECT (A, p, r, i)

```
1) if p == r
2)     return A[p]
3) q = RANDOMIZED-PARTITION (A, p, r)
4) k = q - p + 1
5) if i == k        // pivot value is the answer
6)     return A[q]
7) else if i < k
8)     return RANDOMIZED-SELECT (A, p, q-1, i)
9) else return RANDOMIZED-SELECT (A, q+1, r, i-k)
```