Best case : Bubble Sort :

$n = 5$

4 5 6 7 9

comparisons will be 4, while ~~no~~ there is no swapping cuz array is sorted.

→ You have to traverse only for once to check whether the array is sorted or not.

So therefore the equation will look like $n \rightarrow \underbrace{(n-1)}_{\text{comparisons}}$

so $O(n)$

Quick Sort : DAC

35 50 15 25 80 20 90 45.

Questions will be asked:

→ How it works?

→ Time complexities (Average, best, worst).

Rules:

→ first you have to choose a "pivot element" out of all the elements of an array.

→ 35 50 15 25 80 20 90 45

Pivot element (V)

→Next you have to choose two points as pointer p and q respectively.

35 50 15 25 80 20 90 45

p q.

⇒ p element will move to it's right side and stops when it will find an element greater than "pivot".

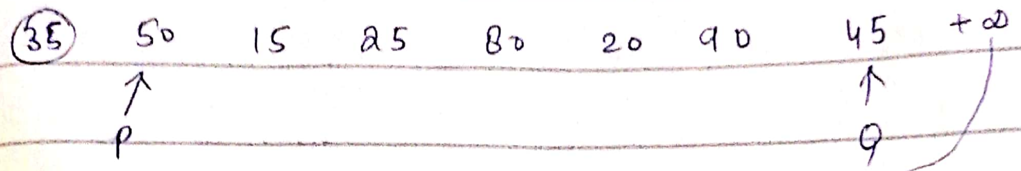
⇒ q pointer will move to it's LHS and stops when it will find the lesser element than "pivot value".

why we do it ?

Just to bring all the greater elements to the right of pivot value and the lesser value to it's left side.

Through this we can sort an array through quick sort method.

35 5 4 3 2 1 $\boxed{+\infty}$ → will stop here.

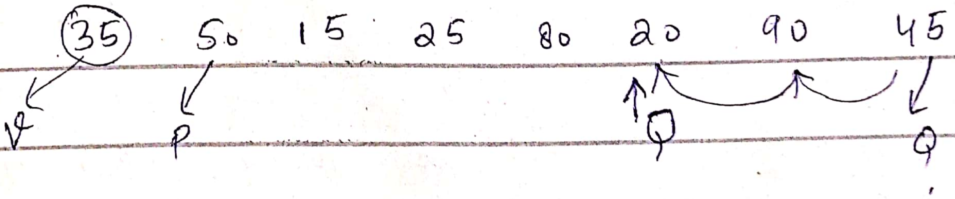


⇒ P will check ~~gr~~ >=

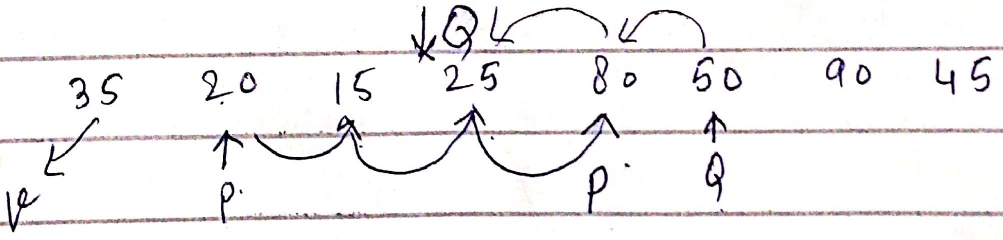
⇒ Q will check <=

This is just to stop the p traversing in case if it didn't find the greater element.

1st pass :



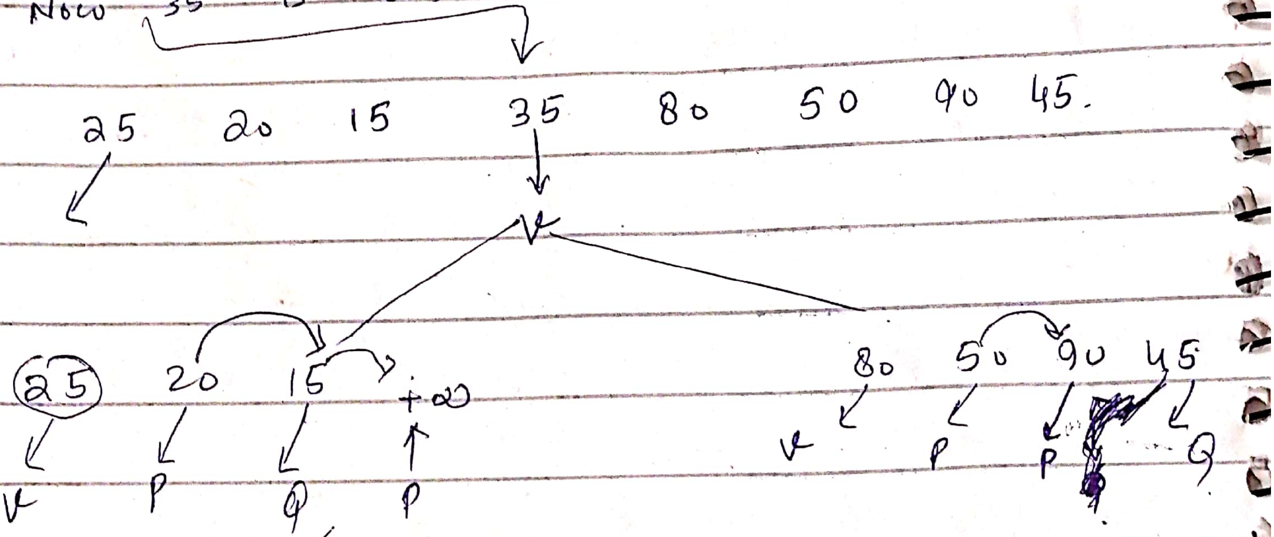
→ Now p is at 50 and Q is at 20, check that they both crossed each other or not, if not then swap both the values.



⇒ Now check whether p and Q are have crossed each other or not, if crossed then replace "pivot value" with Q.

⇒ This condition also holds when both pe and q will at same position.

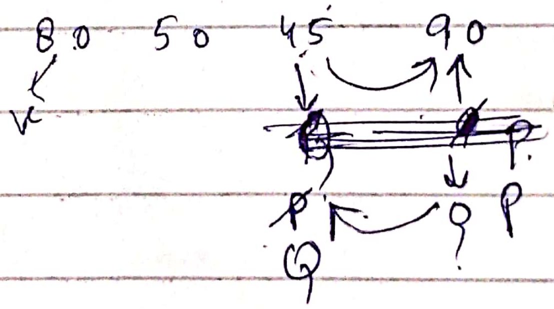
Now 35 is it's on right position.



Q and P have crossed each other, now replace pivot with q.

15 20 25

Since P and Q have not crossed each other, so swap P and Q.



Now both p and Q have crossed each other, so replace v with Q.

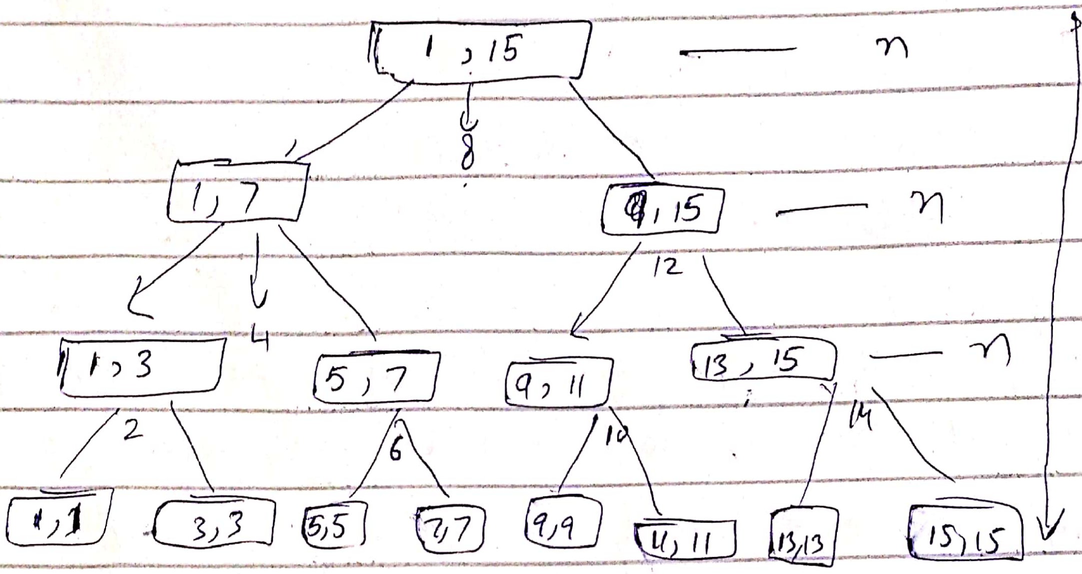
15 20 25 35 45 50 80 90.

Performance Analysis:

Best case:

Assumption: Every time list will be partinoed into half exactly

List | 1 ——— 15 |



Now we have seen, every time a list is divided by two, it is converted into half.

e.g $\frac{n/2/2}{2/2} = 1$

$\Rightarrow \frac{n}{2K} = 1 \Rightarrow n = 2^K$

$\Rightarrow K = \log_2 n$

So there are logn levels, and partition algorith will take n time

So $O(n \log n)$

but best case is not possible every time cuz we can not partioned the list into half every time.

e.g 1 2 3 $\underset{\text{median}}{\underset{\downarrow}{\textcircled{4}}}$ 5 6 7

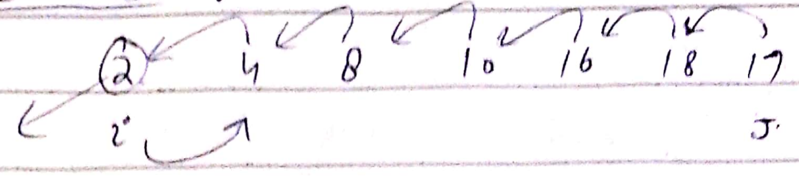
median is the middle element of a ~~list~~ ~~of~~ sorted list.

In the best case, that the partioned should be in the middle, it means the element that is selected as pivot should be a median.

→ So we can not control it, it may happen randomly.

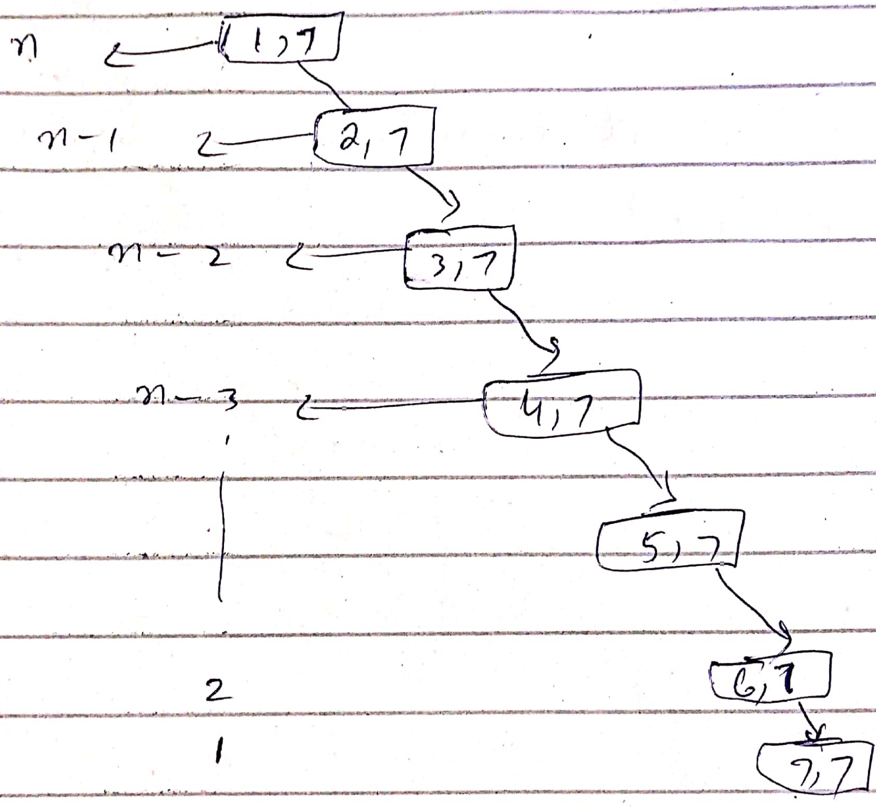
⇒ Achieving best case is not possible everytime. it may achieve randomly.

worst case :



this will become J as list is already sorted.

→ if you choose 4 as pivot, then list will be partioned at 4.



$$= \frac{n(n+1)}{2}.$$

$$= O(n^2)$$

Problem we are sorting a list which is already sorted.

Q# 14

ii) 1548

two ~~reasons~~ ways to ~~remove~~ convert worst case into best case.

i- select middle element as pivot. if list is already sorted, it will become best case as O(n logn).

ii- select Random pivot element.

~~it may take n logn or~~ n^2.

worst time taken ~~by~~ will be $O(n^2)$.