Sorting 2 : Quick Sort & Comparator Problems

AGENDA

- Pivot Partition
- Quick Sort
- Comparator problems

Question :- Partition the array based on the first element

Given an integer array, consider first element as pivot, re-arrange the elements such that for all `i` :

→ if `A[i] < p` then it should be present on left side.

→ if `A[i] > p` then it should be present on right side.

Ex :- [54] 26 93 17 77 31 44 55 20

P

↓ After Partition

26 17 31 44 20 [54] 93 77 55

Observation

→ Can we say pivot reached at its sorted place?

↳ yes

Brute force

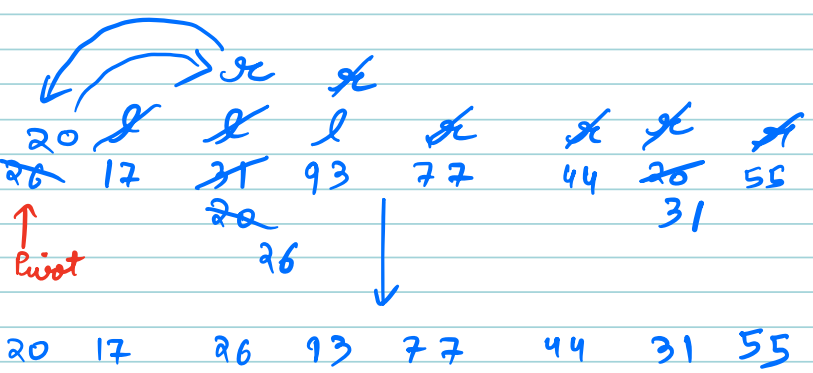
↳ sort the array because pivot will reach at its correct place.

TC → O(N log N)

↳ OPTIMIZATION

Problem above → We don't want to sort Left & Right side of pivot.

IDEA :-



→ Left pointer (l) happy when pointing to something smaller than pivot, then l++

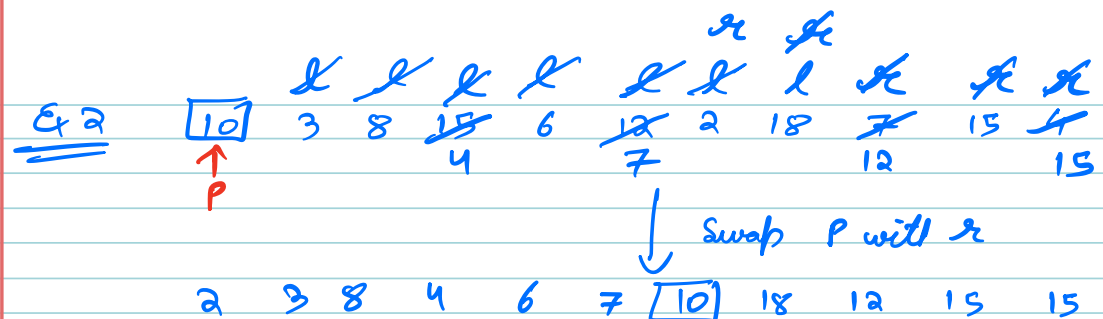
→ Right pointer (r) happy when pointing to something greater than pivot, then r--

→ when both unhappy we can just swap both of them & do l++ & r--

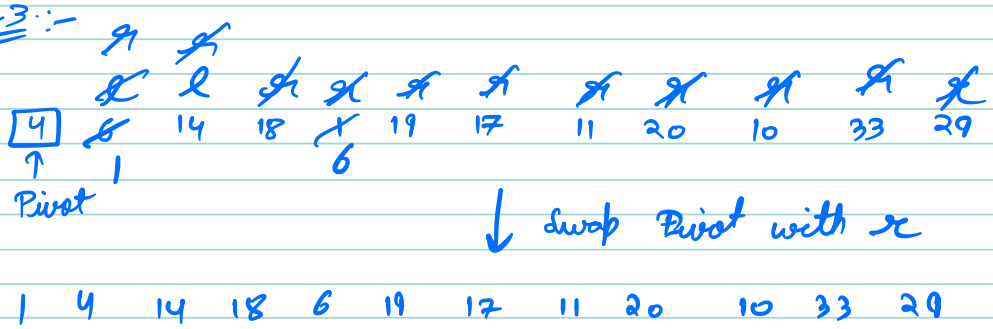
→ we will stop when r < l

→ At last swap pivot with r

Ex 2



Ex 3 :-



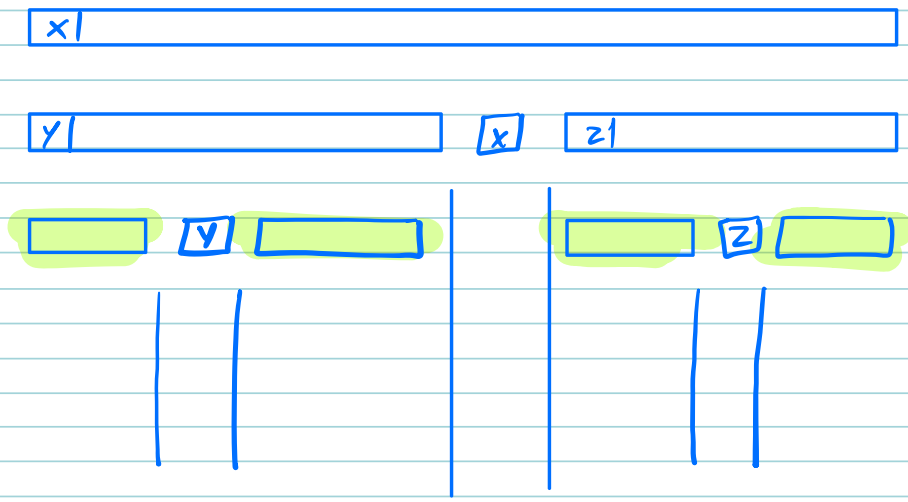
PSEUDO CODE

```
Partition ( A , first, last) {
    Pivot = A [ first ];
    l = first + 1 ;
    r = last ;
    while ( l <= r ) {
        if ( A[l] <= pivot ) {
            l++;
        } else if ( A[r] > pivot) {
            r--;
        } else {
            Swap ( A, l, r);
            l++ ;
            r-- ;
        }
    }
    Swap ( A, first , r);
}
```

TC → O(N)

SC → O(1)

Quick Sort



★ This is an idea of quick Sort.

Eg

0	1	2	3	4	5	6	7	8	9
18	8	6	3	11	14	23	20	31	27

0	1	2	3	4	5	6	7	8	9
14	8	6	3	11	18	23	20	31	27

0	1	2	3	4	6	7	8	9
11	8	6	3	14	20	23	31	27

0	1	2	3	8	9
3	8	6	11	27	31

0	1	2
3	6	8

→ when L & R is same then its base Case

→ while writing code we need pivot point for nested calls, so we can rely on for returning Partition point which will be x^{th} index

PSEUDO CODE

```
quick Sort ( A , start , end ) {
    if ( start < end )
        P = Partition ( A, start , end);
        quick Sort ( A, start , P-1);
        quick Sort ( A, P+1 , end);
}
}
```

It prevents execution when,

$S == e$

$S > e$

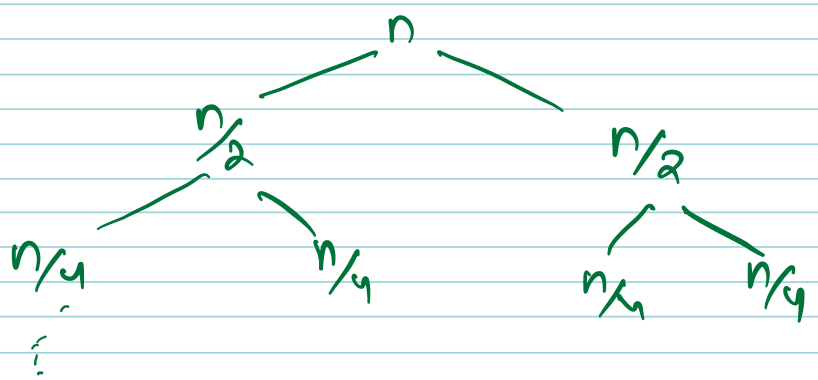
TC

① Best Case

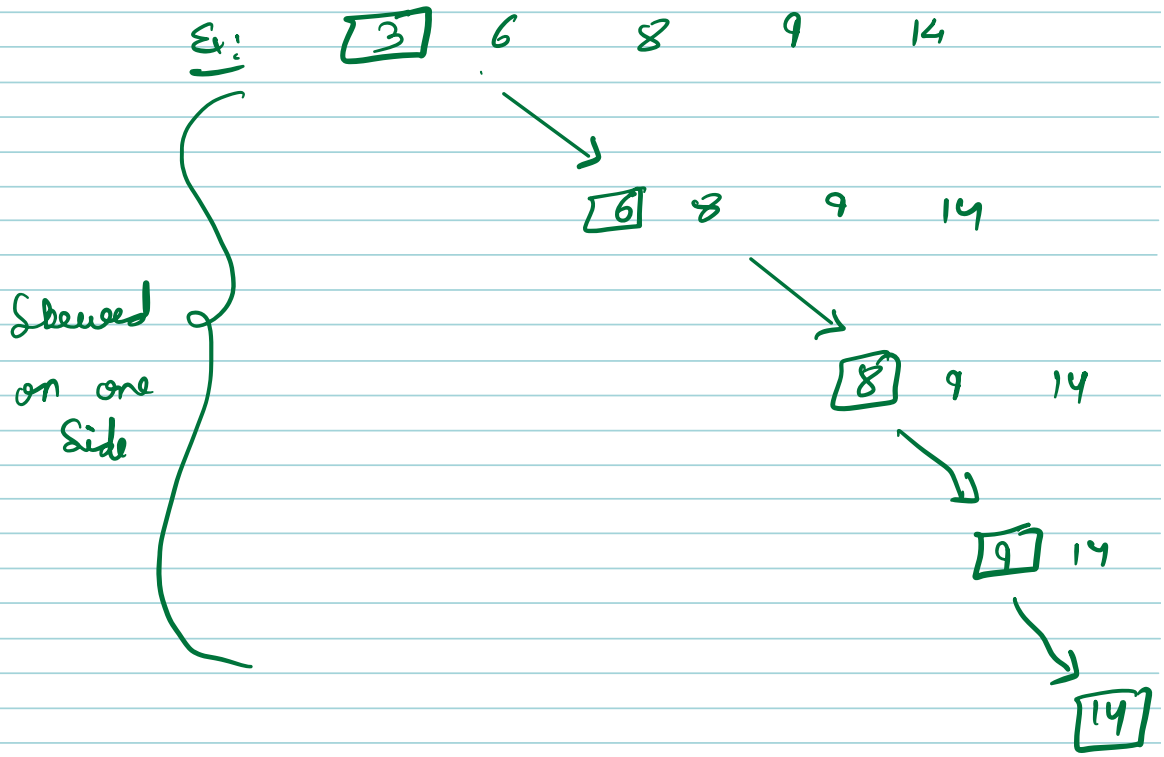
↳ when x divides equally

Recurrence Relation:- $T(n) = 2T(\frac{n}{2}) + n.$

TC	$\rightarrow$ $N \log N$
SC	$\rightarrow$ $\log N$



② Worst Case

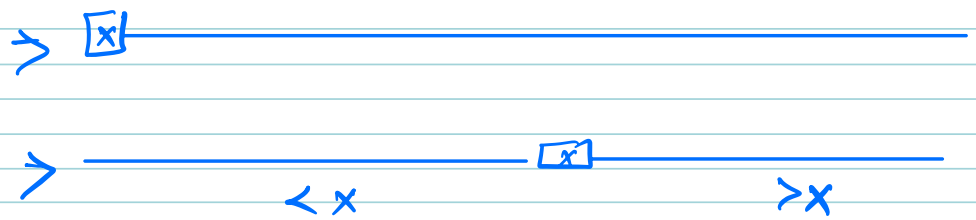


Recurrence relation ⇒ $T(n) = T(n-1) + n$

TC → $O(N^2)$

SC → $O(N)$ ↓ Recursion stack.

=> Randomize Quick Sort



→ The randomised quicksort is a technique where we randomly pick the pivot element, not necessarily the first & last.

→ There is a random function available in all the languages, to which we can pass array & get random index. Now, we can swap random index element with first element & execute our algorithm as it is.

→ Probability of Picking 2 minimums randomly $= \frac{1}{10} \times \frac{1}{9} = \frac{1}{90}$

→ Hence, using randomised quick sort, we can achieve average case of $O(N \log N)$ most of the time.

COMPARATOR

→ In programming, a comparator is a function that compares two values & returns a result indicating whether the values are equal, less than or greater than each other.

→ The comparator is typically used in sorting algorithm to compare elements in a data structure & arrange them in a specified order.

For languages – Java, Python, JS, C#, Ruby, the following logic is followed.

① In sorted form, if first argument should come before second, -ve value is returned.

② In sorted form, if second argument should come before first, +ve value is returned.

③ If both are same, 0 is returned.

For C++, following logic is followed.

① In sorted form, if first argument should come before second, true is returned.

② otherwise, false is returned.

question :- Given an array of size n, sort the data in ascending order of count of factors, if count of factors are equal then sort the elements on the basis of their magnitude.

Ex arr[] = {9 3 10 6 4}

factor → 3 2 4 4 3

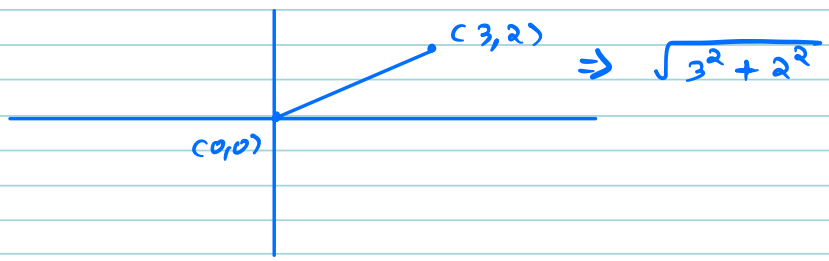
Ans = 3 4 9 6 10

PSEUDO CODE

```
collection.sort(A, new Comparator<Integer>(){
    @override
    Public int compare(Integer V1, Integer V2){
        if(factor(V1) == factor(V2)){
            if(V1 < V2){
                return -1;
            }else if(V2 < V1){
                return 1;
            }else{
                return 0;
            }
        }else if(factors(V1) < factors(V2)){
            return -1;
        }else{
            return 1;
        }
    }
});
```

Question 2 :- Given an array of points where point[i] = [x_i, y_i] represents a point on the x-y plane & integer k, return k closest point to the origin (0,0).
The distance b/w two points on the x-y plane is the Euclidean distance

i.e. $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$



$\Rightarrow \sqrt{3^2 + 2^2}$

Ex 1 :- Points = [[1,3], [-2,2]] | k = 1

$\sqrt{1^2+3^2}$ → $\sqrt{10}$

$\sqrt{(-2)^2+2^2}$ → $\sqrt{8}$ ✓

Ex 2 :- Points = [[3,3] [5,-1] [-2,4]] | k = 2

$\sqrt{3^2+3^2}$ → $\sqrt{18}$ ✓

$\sqrt{5^2+(-1)^2}$ → $\sqrt{26}$

$\sqrt{(-2)^2+4^2}$ → $\sqrt{20}$ ✓

IDEA :- array of points

(x_1, y_1) (x_2, y_2)

dist 1 = $x_1^2 + y_1^2$
dist 2 = $x_2^2 + y_2^2$

} can ignore root

```
if ( dist1 < dist2) {
        return -1;
} else {
        return 1;
}
```

Quiz 1 :- Best Case TC of Quick Sort ?

↳ $N \log N$

Quiz 2 :- Worst Case TC of Quick Sort ?

↳ N^2

Quiz 3 :- Worst Case SC of Quick Sort?

↳ N

Quiz 4 :- Best Case SC of Quick Sort ?

↳ $\log N$