Today's Agenda :-

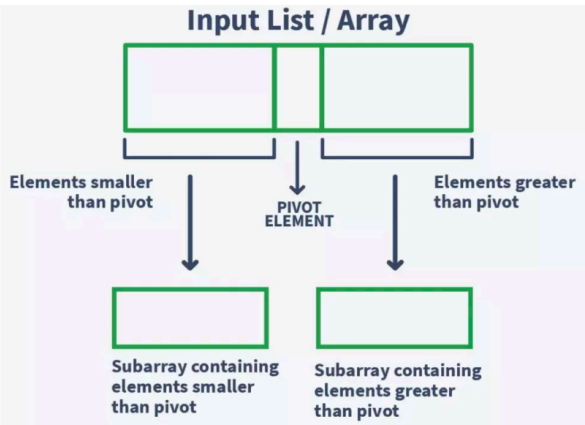
- Pivot Partition
- Quick Sort
- Comparator Problems

Ques Partition the array.

Given an integer array, consider first element as pivot, rearrange 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

Note: All elements are distinct



arr[]: 56, 26, 93, 17, 77, 31, 44, 55, 20.

after partitioning

20, 55, 44, 31, 17, 26, 56, 93, 77

<56 >56

sorted place

Soln :- Sort the array.

arr[] :- 54, 26, 93, 17, 77, 31, 44, 55, 20.

Sort

→ 17, 20, 26, 31, 44, 54, 55, 77, 93

17, 20, 26, 31, 44 : < 54 ; 55, 77, 93 : > 54

↳ nlogn.

Soln 2 :-

arr[] :- 54, 26, 93, 17, 77, 31, 44, 55, 20.

l r r l

swap pivot with r

31 26 20 17 44 54 77 55 93

< 54 ; > 54

we stop l >= r.

arr[] :- 10 3 8 15 6 12 2 18 8 15 4

l r

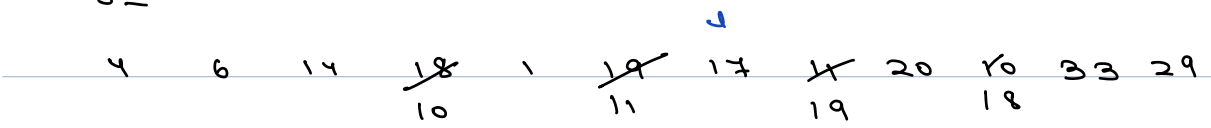
swap pivot with r

2 3 8 4 6 7 10 18 12 15 15

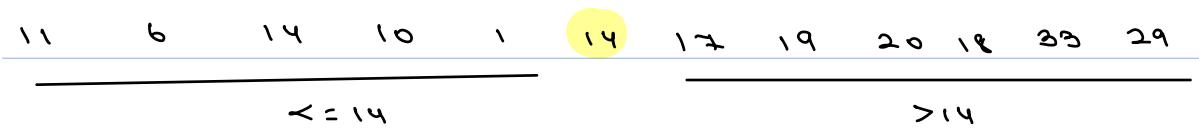
< 10 ; > 10

if l > r break.

e.g. arr[12] =



swap pivot with r.



- We begin by incrementing leftmark until we locate a value that is greater than the pivot value.
- We then decrement rightmark until we find a value that is less than the pivot value.
- At this point we have discovered two items that are out of place with respect to the eventual split point.

↓

swap these two items.

T.C → O(n), S.C → O(1).

```
Partition ( arr, first, last ) {
    pivotvalue = arr[first];
    l = first+1;
    r = last
    while( l <= r )
        if (arr[l] <= pivotvalue)
            l++;
        else if( arr[r] > pivotvalue){
            r--;
```

```
else
    swap(arr, i, j);
    i++;
    j--;
}
```

```
swap(arr, first, j);
return j;
```

e.g →

arr[4] = 8 4 1 7

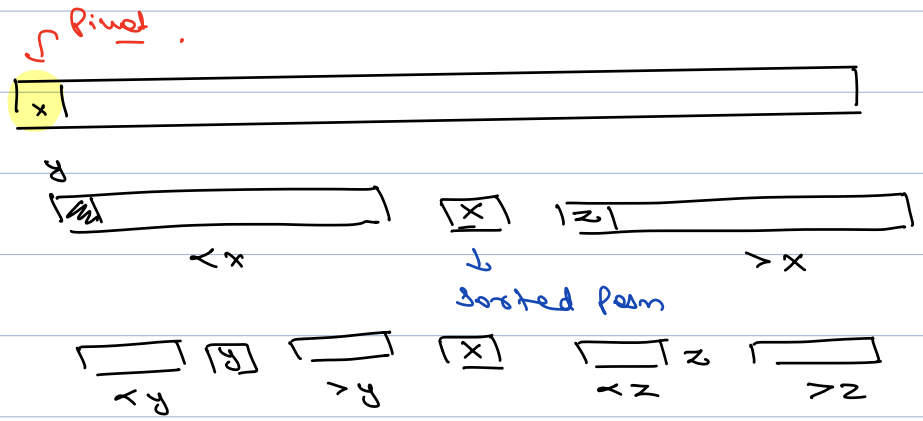
swap (pivot, j)

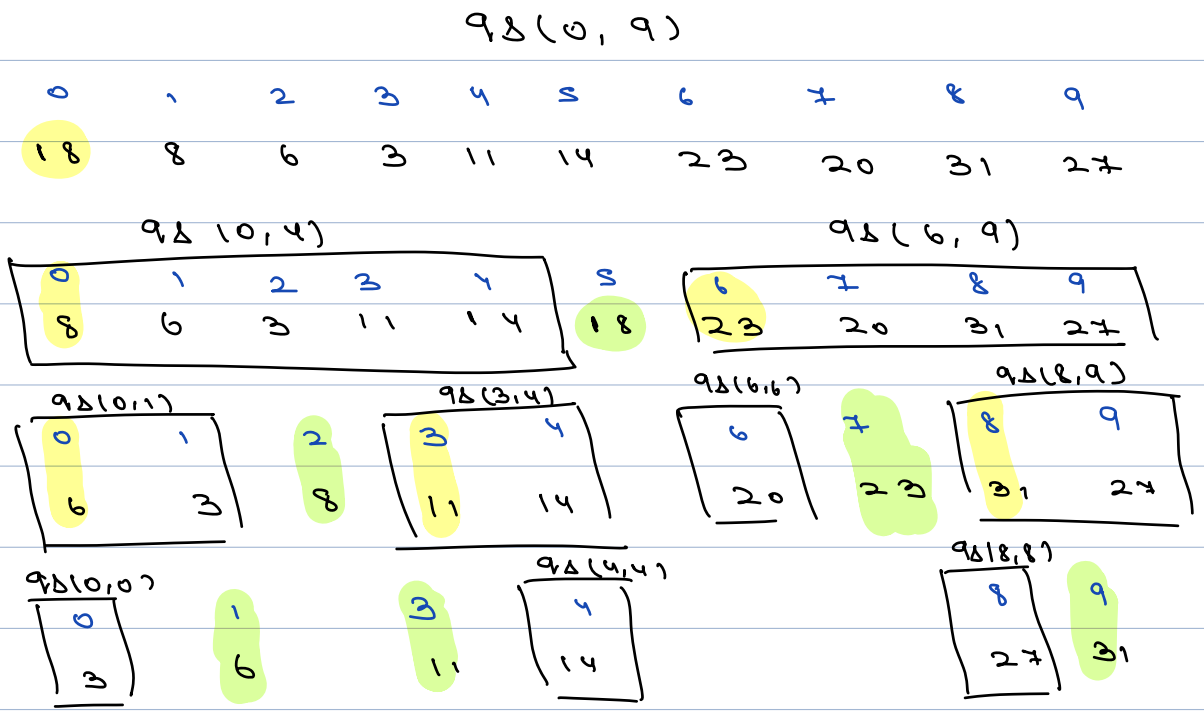
7, 4, 1, 8

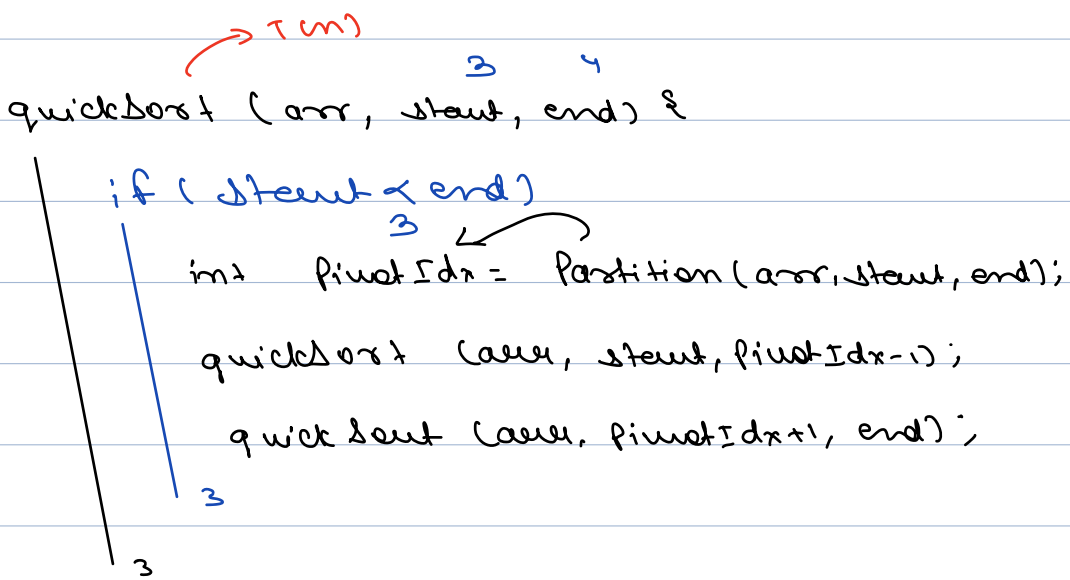
<= 8 | >= 8

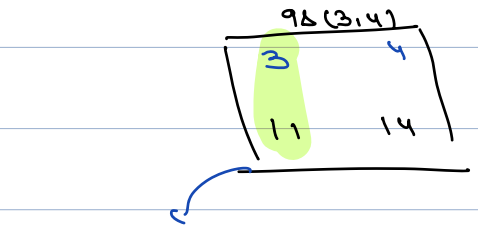
← Quick sort →

↳ Divide & Conquer strategy.

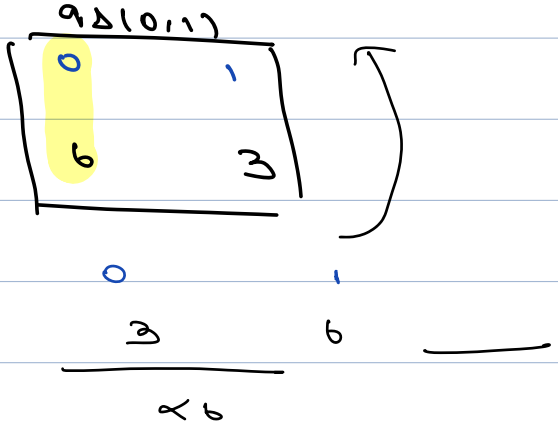



QS(0, 9)
0 1 2 3 4 5 6 7 8 9
18 8 6 3 11 14 23 20 31 27
QS(0,4)
0 1 2 3 4
8 6 3 11 14
5
18
QS(6, 9)
6 7 8 9
23 20 31 27
QS(0,1)
0 1
6 3
2
8
QS(3,4)
3 4
11 14
QS(6,6)
6
20
7
23
QS(8,9)
8 9
31 27
QS(0,0)
0
3
1
6
3
11
QS(4,4)
4
14
QS(8,8)
8
27
9
31


T(n)
quickSort (arr, start, end) {
3 4
if (start < end)
int pivotIdx = Partition(arr, start, end);
3
quickSort (arr, start, pivotIdx-1);
quickSort (arr, pivotIdx+1, end);
3
3



qsort (3,2)



Merge Sort

Split the array

work

(post)

quick Sort

work

Split the array.

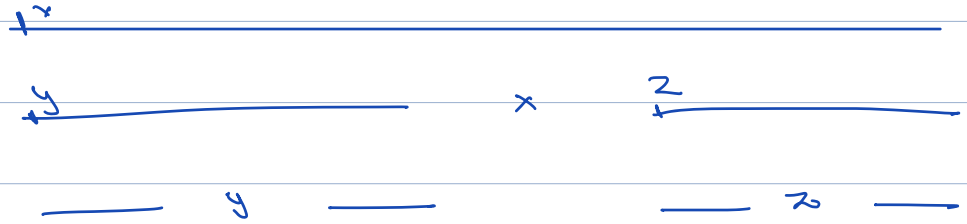
(Pre).

→ Best Case.

Case -1, (when split happens equally)

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

T.C → $O(n \log n)$.



Case-2 worst case :- 9Δ (0,4)

arr[5] = | 3 | 6 | 8 | 9 | 14 |
0 1 2 3 4

3 == 3 → 9Δ (1,4)

| 6 | 8 | 9 | 14 |

6 > 3

6 | 8 | 9 | 14

8 | 9, 14

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

↓

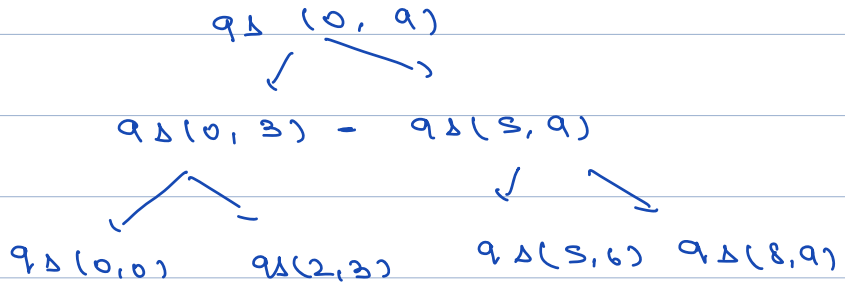
T.C → $O(n^2)$

arr[5] = | 10 | 5 | 4 | 3 | 2 | 1 |

| 5 | 4 | 3 | 2 | 1 | 10 |

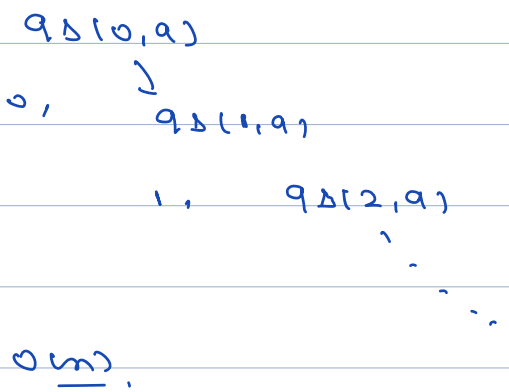
| 4 | 3 | 2 | 1 | 5 |

Best case

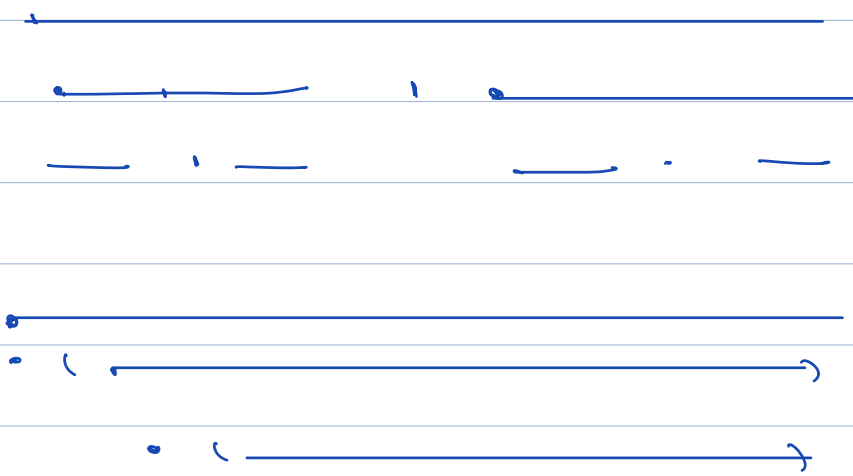


T.C → O(log n)

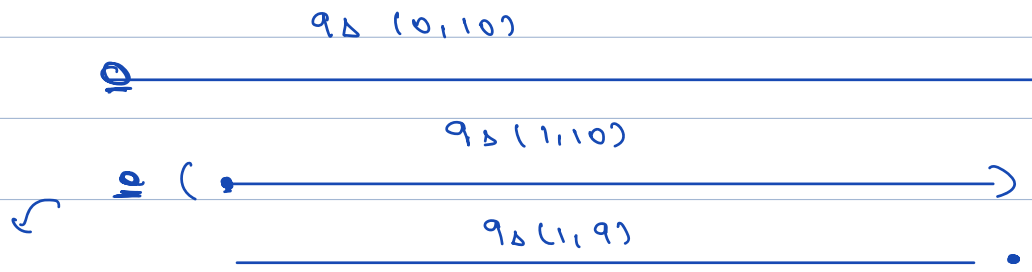
worst case :-



T.C → O(n).



merge sort	quick sort	
T.C → O(nlogn)	Best	worst case
S.C → O(n)	T.C → O(nlogn)	T.C → O(n²)
	S.C → O(logn)	S.C → O(n)



$$\text{first element} = \frac{1}{n}$$

Avg. case T.C :- quick sort.

T.C → O(nlogn).

Randomized Quick Sort

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

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

avg. T.C → O(nlogn).

0 1 2 3 4
3 6 8 9 14

∴ Comparators :- (1, 100, 2, 3, 9, 101)

Arrays.sort (arr);

- In programming, a **comparator** is a function that compares two values and returns a result indicating whether the values are equal, less than, or greater than each other.
- The **comparator** is typically used in sorting algorithms to compare elements in a data structure and 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.
```

Ques Sort Based on factors

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.

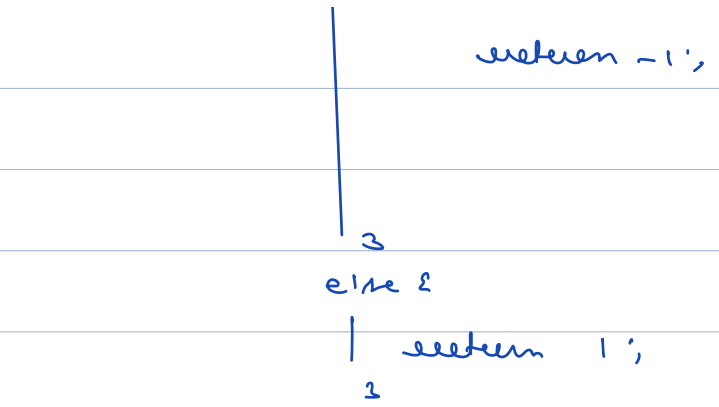
arr[] = { 9, 3, 10, 6, 4 }

3 2 4 4 3

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

ArrayList

```
Collections.sort ( A, new Comparator<Integer>(){
    @Override
    public int comp ( Integer v1, Integer v2){
        if (factors(v1) == factors(v2)){
            if (v1 < v2) {
                return -1;
            }
            else if (v1 > v2) {
                return +1
            }else{
                return 0;
            }
        } else if (factors(v1) < factors(v2){
```




```
import functools

//please write the code for finding factors by yourself

def compare(v1, v2):
    if(factors(v1) == factors(v2)):
        if(v1<v2):
            return -1;
        if(v2<v1):
            return 1;
        else
            return 0;
    elif (factors(v1)<factors(v2)):
        return -1;
    else
        return 1;


class Solution:
    def solve(self, A):
        A = sorted(A, key = functools.cmp_to_key(compare))
        return A
```

```
bool compare(int val1, int val2)
{
    int cnt_x = count_factors(x);
    int cnt_y = count_factors(y);

    if(factors(val1) == factors(val2))
    {
        if(val1<val2)
        {
            return true;
        }
        return false;
    }
    else if(factors(val1)<factors(val2))
    {
        return true;
    }
    return false;
}

vector<int> solve(vector<int> A) {
    sort(A.begin() , A.end() , compare);
```

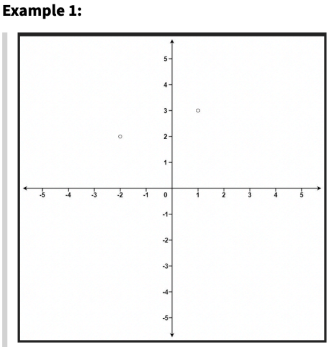
Ques 2 :-

Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the B closest points to the origin (0, 0).

The distance between two points on the X-Y plane is the Euclidean distance (i.e., $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$).

You may return the answer in any order.

Example 1:



(x_1, y_1) — (x_2, y_2)

$(0,0)$, $(2,3)$

$\sqrt{(2-0)^2 + (3-0)^2}$

$\Rightarrow \sqrt{13}$.

(a,b) from $(0,0)$

$\sqrt{a^2+b^2}$

e.g. ([1,3], [-2,2]) B = 1.

$\sqrt{1^2+3^2}$ → $\sqrt{10}$; $\sqrt{2^2+2^2}$ → $\sqrt{8}$

e.g 2 ([3,3], [5,-1], [-2,4]) B = 2.

$\sqrt{3^2+3^2}$ → $\sqrt{18}$; $\sqrt{25+1}$ → $\sqrt{26}$; $\sqrt{4+16}$ → $\sqrt{20}$

array of points → x_1, y_1 ; x_2, y_2

$dist_1 = x_1^2 + y_1^2$

$dist_2 = x_2^2 + y_2^2$

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

Ques Largest Number.

Given a list of non-negative integers nums, arrange them such that they form the largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

e.g1 → [10, 2] → [2, 10] → 210.

e.g2) [3, 30, 34, 5, 9] → [9, 5, 34, 3, 30] → 9534330

[3, 30] → 303

↓

330

While it might be tempting to simply sort the numbers in descending order, but this doesn't work.

[3, 30] → 303

↓

330

[a, b] → ba

↓

ab

[3, 30, 34, 5, 9]

[9, 5, 34, 3, 30]

We shall use **custom sorting**.
Say we pick two numbers **X** and **Y**. Now, we can check if `X (appends) Y` > `Y (appends) X`, then it means that X should come before Y.
For example, let X and Y be 542 and 60. To compare X and Y, we compare 54260 and 60542. Since 60542 is greater than 54260, we put Y first.
Once the array is sorted, the most "signficant" number will be at the front.

T.C → O(n log n) * (Complexity of your Comparator)