→ CountAndSay(n) is the way you "Say" CountandSay(n-1)

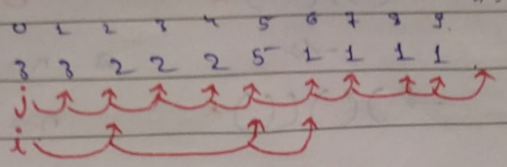
Cas(1) = 1

Cas(2) = 1 1

Case(3) = 2 1

Case(4) = 1 2 1 1.

Case(5) = 1 1 1 2 2 1



ans => 2 3 3 2 1 5 4 1

=> Code :-

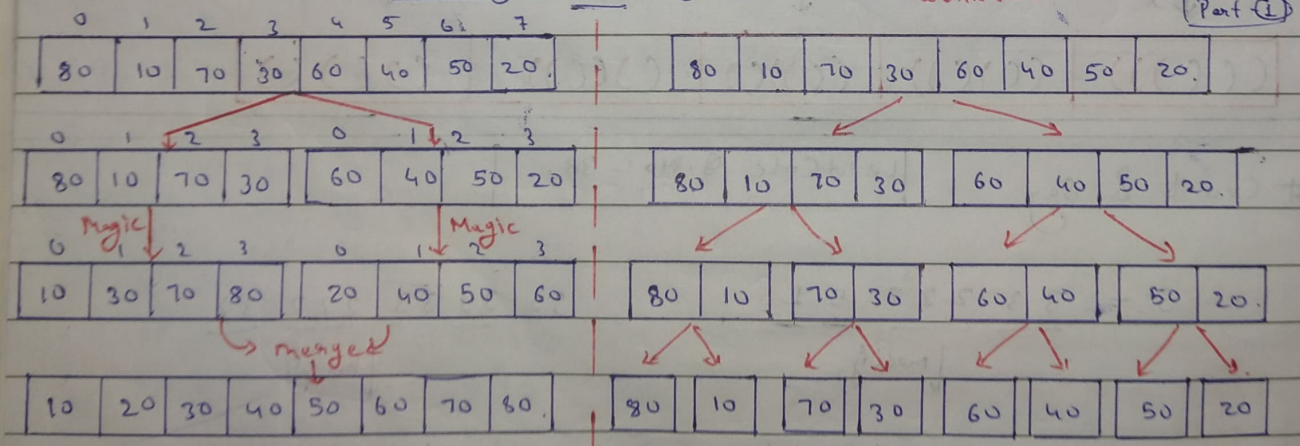
```
ps String countAndSay(int n){
    if(n==1) return "1";
    String s = countAndSay(n-1) + "@";
    // Now we have to modify s.
    String ans = "";
    int i=0, j=0;
    while(j < s.length()){
        if(s.charAt(i) == s.charAt(j)) j++;
        else {
            int len = j-i;
            ans += len;
            ans += s.charAt(i);
            i = j;
        }
    }
    return ans;
}
```

```
psvm {
    int n = 4;
    Sout(countAndSay(n));
}
```

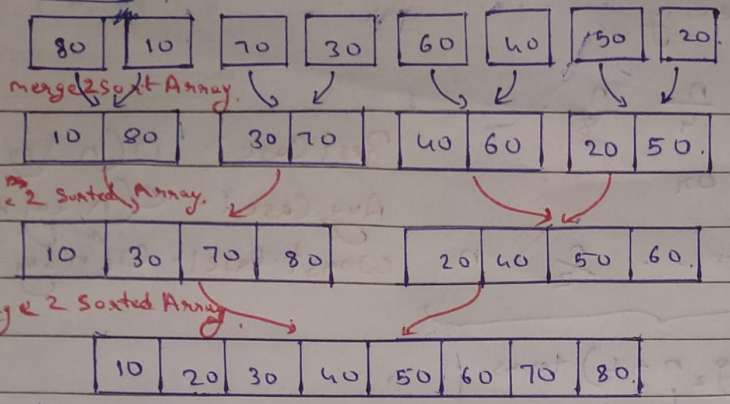
Merge Sort Algorithm.

Work flow :-

Part 1



Part (2)



Page No.: YOUVA
Date: Copy pasting.

Divide & Conquer

⇒ Code :-

```
psv mergesort(int[] arr){
    int n = arr.length;
    if(n==1) return; //Base Case.
    //Create two arrays of n/2 size each...
    int[] a = new int[n/2];
    int[] b = new int[n-n/2];
    //Copy Pasting...
    for(int i=0; i<n/2; i++){
        a[i] = arr[i];
    }
    for(int i=0; i<n-n/2; i++){
        b[i] = arr[i+n/2];
    }
    //Magic...
    mergesort(a);
    mergesort(b);
    //Merge these 'a' & 'b'...
    merge(a, b, arr);
    //Delete a & b...
    a = null; b = null;
}
```

```
psv print(int[] arr){
    for(int ele : arr){
        sout(ele + " ");
    }
    sout();
}
```

$$T.n.o = \left[n + \left(\frac{n}{2}+\frac{n}{2}\right) + \left(\frac{n}{4}+\frac{n}{4}+\frac{n}{4}+\frac{n}{4}\right) + \left(\frac{n}{4}+\frac{n}{4}+\frac{n}{4}+\frac{n}{4}\right) + \left(\frac{n}{2}+\frac{n}{2}\right) + n \right]$$

merging

$$T.n.o = 2[n+n+n]$$

$$T.n.o = 2(l-1).n$$

```
psv main {
    int[] arr = {80, 30, 50, 20, 60, 10, 70, 40};
    print(arr);
    mergesort(arr);
    print(arr);
}
```

```
psv merge(int[] a, int[] b, int[] c){
    int i=0, j=0, k=0;
    while(i<a.length && j<b.length){
        if(a[i] <= b[j])
            c[k++] = a[i++];
        else
            c[k++] = b[j++];
    }
    while(j<b.length)
        c[k++] = b[j++];
    while(i<a.length)
        c[k++] = a[i++];
}
```

Time And Space Complexity :-

n
n/2
n/8
n/16
.
.
1

n n/2 n/4 n/8 . . . 1

Or

1 2 4 . . n/2 n.

$(\log_2 n + 1)$ terms.

levels.

Best Case :- $O(n \log n)$

Avg Case :- $O(n \log n)$

Worst Case :- $O(n \log n)$

$T.C = 2(1-1).n = 2.(\log_2 n + 1 - 1)\, n$

$= 2\, n \log n$

T.C = $O(n * \log n)$

In this basic code, the amount of space used $= (1-1).n$

$= n \log_2 n.$

→ S.C = $O(n \log n)$

↓

we can improve it by deletion of a & b after merging into arr.

→ Best case :- $O(n)$ Worst Case :- $O(n)$

Avg Case :- $O(n)$

Extra Space Used :- $n + \frac{n}{2} + \frac{n}{4} = n\left(1 + \frac{1}{2} + \frac{1}{4} \ldots \infty\right) < 2n$

S.C = $O(n)$

Maths :-

$1 + \frac{1}{2} + \frac{1}{4} \ldots < 2.$

$n\left(1 + \frac{1}{2} + \frac{1}{4} \ldots\right) < 2 * n$

Stability of Merge Sort :-

→ Yes. merge sort is stable.

arr = 5 1 2 5* 3

↓

Sorting.

1 2 3 5 5*.

→ Make sure to Code like this.

```
if(a[i] <= b[j])
    c[k++] = a[i++];
```

if I use '<' instead of '<=' then it won't be stable.

→ Applications of Merge Sort

① O(n log n) worst case time complexity. ② Custom Sorting.
③ Sorting Linked lists. ④ Inversion Count & Related Problems.

Page No.: YOUVA
Date:

[LeetCode 493].

Inversions Count Problem ⟶ You have to count the no. of inversions. $i < j$ and $arr[i] > arr[j]$

arr = { 8 , 2 , 5 , 3 , 1 , 4 }.

→ (8, 2) , (8, 5) , (8, 3) , (8, 1) , (8, 4)
→ (2, 1) , (3, 1)
→ (5, 3) , (5, 1) (5, 4).

[ans = 10.]

Note :- Just Need to modify the mergesort code.

Basic Approach [Brute Force].

⇒ Code :-

```
psv inversion(int[] a, int[] b){
    int i=0, j=0;
    while(i < a.length && j < b.length){
        if(a[i] > b[j]){
            count += (a.length - i);
            j++;
        }
        else i++;
    }
}
```

T.C ⇒ $O(n^2)$.
S.C ⇒ $O(1)$
Not the Best

arr = { 8 , 2 , 5 , 3 , 1 , 4 }

a = 2, 5, 8 (i) b = 1, 3, 4 (j)

count = ~~0~~ (+3) ~~3~~ (+2) ~~5~~ (+2) 7

arr[i] > arr[j]
↓ ele from a
→ ele from b.

Merge sort

→ Create two array a & b of n/2 size each & copy paste.
→ Sort (a).
→ Sort (b).
→ Count inversions in a & b. [Extra]. (add the inversion function & call it)
→ Merge(a, b, arr). (also create count global variable)

M-2 → [just modify the merge function in mergesort code.]

⇒ Code :-

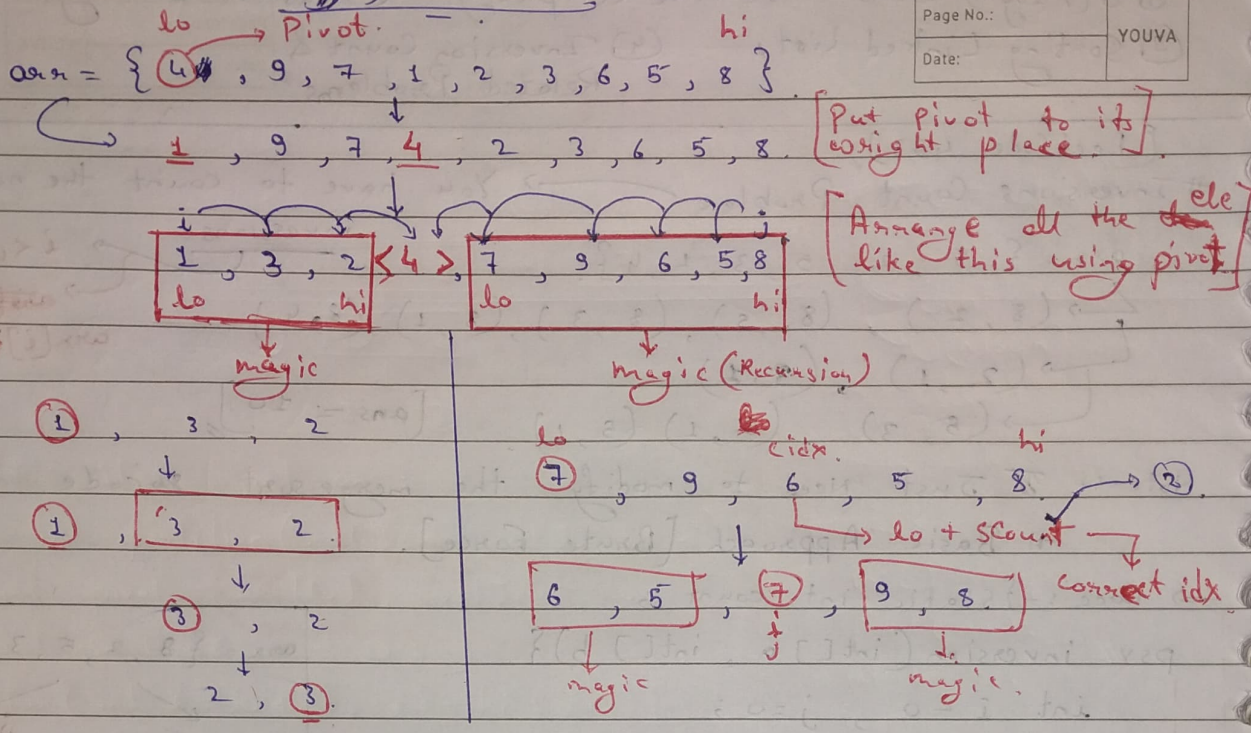
```
while(i < a.length && j < b.length){
    if(a[i] <= b[j])
        c[k++] = a[i++];
    else { // a[i] > b[j]
        count += (a.length - i); // Extra
        c[k++] = b[j++];
    }
}
```

[Only thing need to modify.]

T.C ⇒ $O(n * \log n)$

No need to create extra function

#8 Quick Sort.



⇒ Code :-

```
pubic static int partition(int[] arr, int lo, int hi){
    int pivot = arr[lo], pivotIdx = lo;
    int smallerCount = 0;
    for (int i = lo + 1 ; i <= hi ; i++){
        if (arr[i] <= pivot) smallerCount++;
    }
    int correctIdx = pivotIdx + smallerCount;
    swap(arr, pivotIdx, correctIdx);
    // Partition
    int i = lo , j = hi ;
    while(i < correctIdx && j > correctIdx){
        if(arr[i] <= pivot) i++ ;
        else if (arr[j] > pivot) j-- ;
        else if(arr[i] > pivot && arr[j] <= pivot){
            swap(arr, i, j);
        }
    }
    return correctIdx;
}
```

```
psvm {
    int[] arr = {3,2,1,5,6,4};
    print(arr);
    int n = arr.length;
    quickSort(arr, 0, n-1);
    print(arr);
}
```

```
psv quicksort(int[] arr, int lo, int hi){
    if(lo >= hi) return ;
    int idx = partition(arr, lo, hi);
    quickSort(arr, lo, idx-1);
    quickSort(arr, idx+1, hi);
}
```

At max itni hi calls lagti

M T W S
Page No.:
Date:
YOUVA

⇒ Time Space Complexity :-

↳ Avg. Case = $O(n * \log n)$

→ Space Complexity = Recursion Call Stack space → $O(\log n)$

⇒ Worst Case T.C. → arr = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 }

↳ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9

→ the no. of levels here are not "log n" they are "n".

No. of ops ~ n + n−1 + n−2 1

$= n\left(\frac{n+1}{2}\right) \sim O(n^2)$ T.C.

$O(n)$ S.C.

⇒ Randomized Pivot Point :-

↳ Instead of Choosing arr[lo] as pivot, we can choose arr$\left[\frac{lo + hi}{2}\right]$ as pivot.

arr = { 4 , 9 , 7 , 1 , 2 , 3 , 6 , 5 , 8 }.

1 , 2 , 7 , 4 , 9 , 3 , 6 , 5 , 8

3 | 4 , 9 , 7 , 6 , 5 , 8

4 , 5 , 6 , 7 , 9 , 8 .

Worst Case

T.C ⇒ $O(n \log n)$

S.C ⇒ $O(\log n)$.

⇒ modified Code :-

```
ps int partition(int[] arr, int lo, int hi){
    int mid = (lo+hi)/2 ;   // modified
    int pivot = arr[mid] , pivotIdx = mid; // modified
    int smallerCount = 0 ;
    for(int i = lo ; i <= hi ; i++){
        if(i == mid) continue ;   // modified
        if(arr[i] <= pivot)
            smallerCount ++ ;
    }
    int correctIdx = lo + smallerCount;   // modified
    swap(arr, pivotIdx, correctIdx);
    // Partition
    :
}
```

⇒ Just Need to modify partition function Little bit

Quick Sort
↳ Unstable.

⇒ Merge Sort v/s Quick Sort

Merge Sort:
→ worst case is Better.
→ Stable

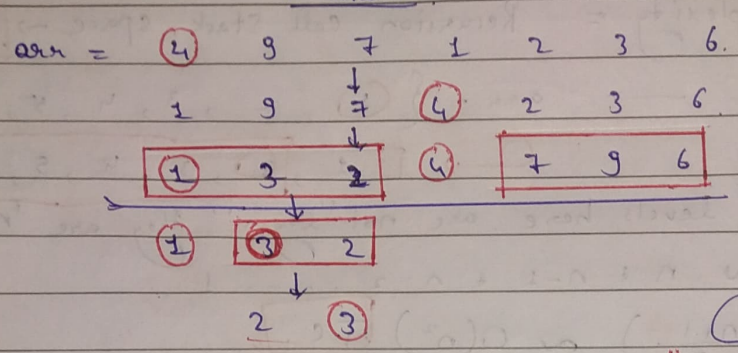
Quick Sort:
→ More space Optimized

LeetCode Qno. 215

Q. kth Largest/Smallest Element in an Array

k^{th} Largest = $(n-k+1)^{th}$ Smallest.

Smallest



Using QuickSort.

T.C = O(n) [Best & Avg. Case.]

→ $O(n^2)$ ~~worst Case~~

→ $O(n \log n)$ Worst Case.

Largest.

↳ Just need to modify the main method little bit.

⇒ Code :- static int ans;

```
ps int partition(int[] arr, int lo, int hi){
    int pivot = arr[lo], pivotIdx = lo;
    int smallerCount = 0;
    for(int i = lo+1; i<=hi; i++){
        if(arr[i] <= pivot) smallerCount++;
    }
    int correctIdx = pivotIdx + smallerCount;
    swap(arr, pivotIdx, correctIdx);
    // Partition.
    int i = lo, j = hi;
    while(i < correctIdx && j > correctIdx){
        if(arr[i] <= pivot) i++;
        else if(arr[j] > pivot) j--;
        else if(arr[i] > pivot && arr[j] <= pivot){
            swap(arr, i, j);
        }
    }
    return correctIdx;
}

psvm{
    int[] arr = {4,9,1,2,6,5,8};
    print(arr);
    int k = 4;
    ans = -1;
    int n = arr.length;
    quickSelect(arr, 0, n-1, k);
    print(arr);
    Sout(ans);
}
```

```
psvm{
    int[] arr = {4,9,1,2,6,5,8};
    print(arr);
    int k = 2;
    ans = -1;
    int n = arr.length;
    quickSelect(arr, 0, n-1, n-k+1);
    print(arr);
    Sout(ans);
}
```

Change.

```
psv quickSelect(int[] arr, int lo, int hi, int k){
    if(lo > hi) return;
    if(lo == hi){ // Not Really Required
        if(lo == k-1) ans = arr[lo]; // NRR
        return; // NRR
    } // NRR
    int idx = partition(arr, lo, hi);
    if(idx == k-1){
        ans = arr[idx];
        return;
    }
    if(k-1 < idx){
        quickSelect(arr, lo, idx-1, k);
    }
    else
        quickSelect(arr, idx+1, hi, k);
}
```