Todays Content

a) Re-arrange array
b) Quick Sort
c) Count Sort

Q1. Given ar[N] : re-arrange it such that

Bring last element to it's corrected sorted position

All values < last ele are continously on leftside of last ele

All value >= last ele are continously on rightside of last ele

Ex1: ar[8] =

0	1	2	3	4	5	6	7
9	8	1	6	5	11	4	7

All of them are correct we can return any one of them

	0	1	2	3	4	5	6	7
ans1:	1	6	5	4	7	9	8	11
ans1:	1	4	6	5	7	8	9	11
ans1:	1	4	5	6	7	8	9	11
ans1:	6	5	4	1	7	11	9	8

Ex2: ar[9] =

0	1	2	3	4	5	6	7	8
7	10	4	8	2	9	12	3	5

X = 5

	0	1	2	3	4	5	6	7	8
ans1:	2	3	4	5	7	8	9	10	12
ans1:	4	3	2	5	8	7	10	12	9
ans1:	7	3	2	5	4	10	12	9	8

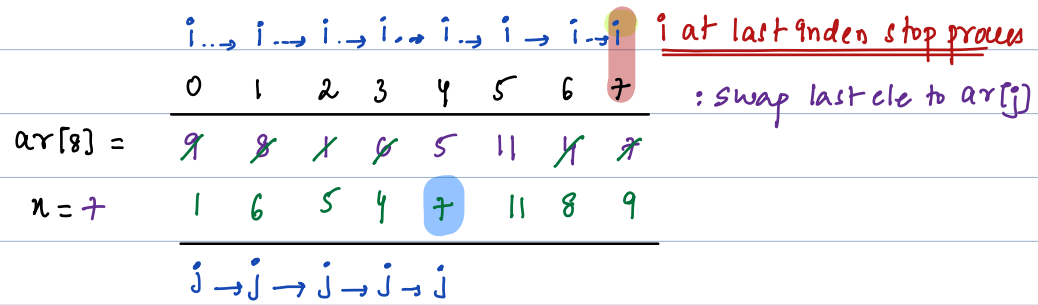
Idea1: Sort ar[] using mergeSort

TC: O(NlogN) SC: O(N)

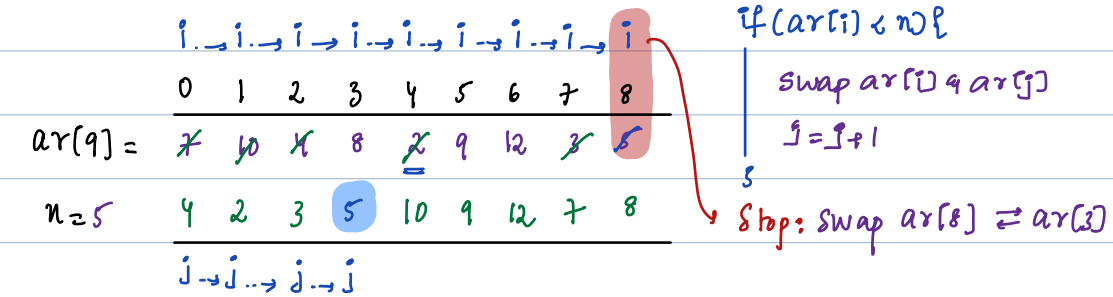
Idea: Take 2 indices i & j

i: Used to iterate on ar[]

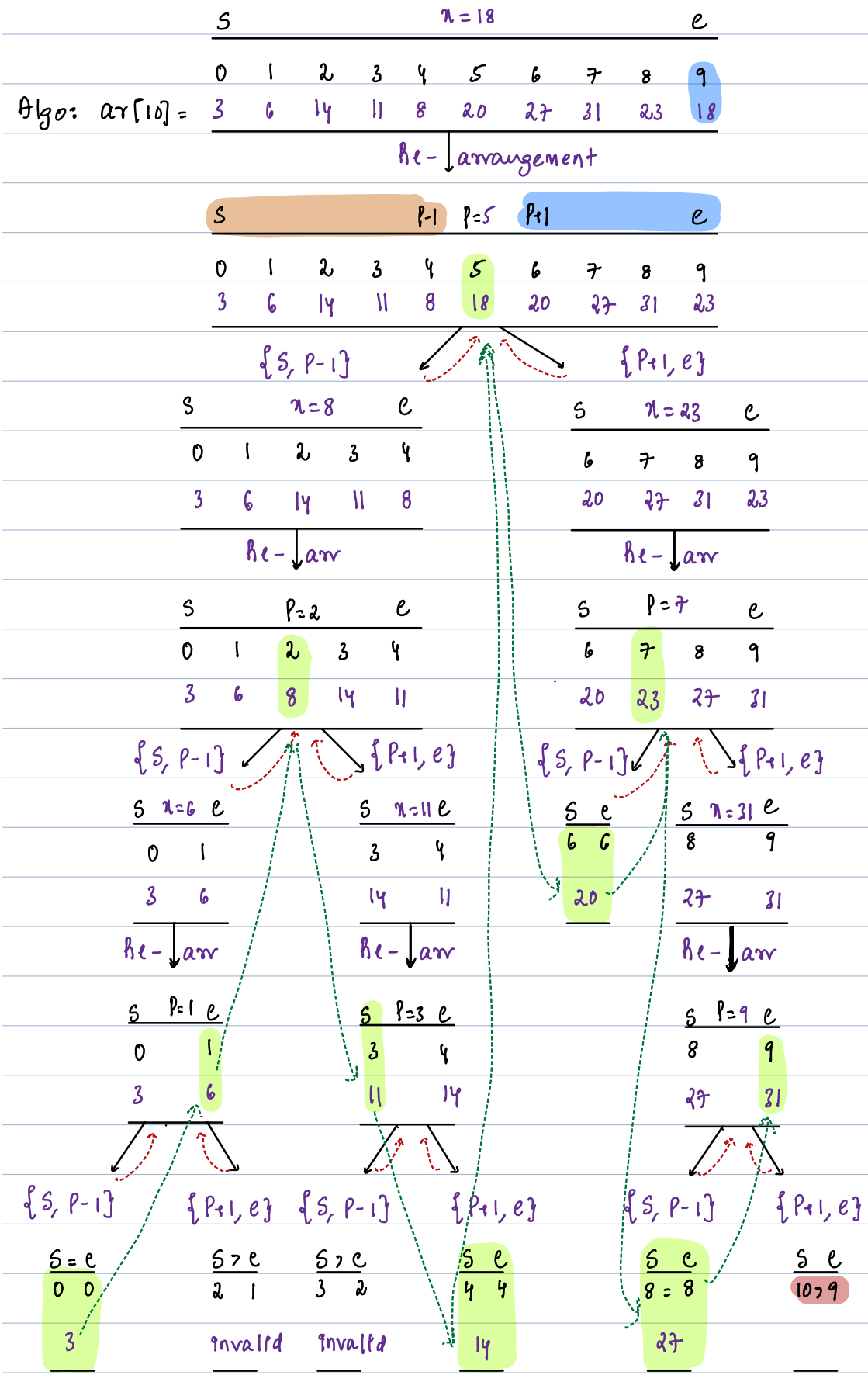
j: Used to keep an ele on left



Note: Only ele < n should come on left




```
int[] rearrange(int ar[]) { TC: O(N)  SC: O(1)
    int N = ar.length;
    int n = ar[N-1]
    int j = 0;
    for(int i = 0; i < N-1; i++){
        if(ar[i] < n) { // ar[i] is relevant, at left
            int t = ar[i]; ar[i] = ar[j]; ar[j] = t;
            j++;
        }
    }
    swap ar[N-1] & ar[j]
    return ar
}
```


S x=18 e
0 1 2 3 4 5 6 7 8 9
Algo: ar[10] = 3 6 14 11 8 20 27 31 23 18
Re-arrangement
S P-1 P=5 P+1 e
3 6 14 11 8 18 20 27 31 23
{S, P-1}
{P+1, e}
S x=8 e
3 6 14 11 8
Re-arr
S P=2 e
3 6 8 14 11
S x=23 e
20 27 31 23
S P=7 e
20 23 27 31
S x=6 e
3 6
S x=11 e
14 11
S e
6 6
20
S x=31 e
27 31
S P=1 e
3 6
S P=3 e
11 14
S P=9 e
27 31
S=e
0 0
3
S>e
2 1
Invalid
S>e
3 2
Invalid
S e
4 4
14
S e
8=8
27
S e
10>9

Ass: Given arr[], sort arr[] from [s..e]

```
void QuickSort(int ar[], int s, int e){          ----> T(N)
    if(s>=e){return}
    // Sort ar[]: [s s+1... e-2 e-1 e]   -> lastindex
    //Step1: Re-arrage last ele to its corrected sorted pos
    int x = ar[e];
    int j = s;
    for(int i = s; i<e ; i++){
        if (ar[i] < x){
            swap ar[i] & ar[j]; // write code
            j++;
        }
    }
    swap ar[j] & ar[e] //correct index of ar[e] = j
    int p = j; // no need
    QuickSort(ar, s, p-1)   0 : T(0)=1
    QuickSort(ar, p+1, e)   N-1 : T(N-1)
}
```

sort left: [s s+1... p-1]   p   sort right: [p+1.. e-2 e-1 e]

Recursive Relation:

To sort N elements assume it take = $T(N)$

Best Case:

$T(N) = 2T(N/2) + N \quad T(1) = O(1)$

$T(N) = O(N\log N)$

Worst Case:

$T(N) = T(N-1) + N \quad T(1) = O(1)$

$T(N) = O(N^2)$

Research:

90% of time, for most examples TC: $O(N\log N)$

Con: On a avg TC: $O(N\log N)$

TODO: Inbuilt Sort uses which sorting algo?

CountSort / FreqSort

Given ar[N], all ele in range [2-6] sort arr[]

	0	1	2	3	4	5	6	7	8	9
ar[] =	3	2	4	6	4	2	3	4	3	6

Idea1: Sort it with mergeSort / QuickSort : TC = O(N log N)

Idea: Store freq of elements hashmap:

3: 3
6: 2
2: 2
4: 3

} using freq, sort arr[]?

i: [2 6] ...

2: freq(2) = 2
3: freq(3) = 3
4: freq(4) = 3
5: freq(5) = 0
6: freq(6) = 2

0	1	2	3	4	5	6	7	8	9
~~2~~	~~2~~	~~4~~	~~6~~	~~4~~	~~2~~	~~3~~	~~4~~	~~3~~	~~6~~
2	2	3	3	3	4	4	4	6	6
j→	j→	j→	j→	j→	j→	j→	j→	j→	j→

Total iterations = N+5 iterations

Total i iterations = 5 : i = [2 6) = 6-2+1

Total j iterations = N : j = [0 N-1] = N

Q) Given ar[N], sort it, using above idea:

Step 1: Iterate on arr[] & get min of arr[] = a, max of arr[] = b

Step 2: Insert all arr[] ele in hashmap<Integer, Integer> hm;

Step 3: Sort arr[] using hashmap;

```
int j=0;
for(int i = a; i<=b; i++){
    int f=0;
    if(hm.containsKey(i)) { f = hm.get(i)}
    for(int l=1; l<=f; l++){ // looping f times
    }   ar[j] = i; j++;
}
```

SC: O(N)

TC: Total i iterations = [a..b] = b-a+1 Total j iterations = [0..N-1] = N TC: N+[b-a+1]

CountSort TC:

TC: N + b-a+1 // b-a+1 is max-min+1 of arr[] say R

Assume R = b-a+1,

TC: N+R

Constraints:	CountSort: N+R	MergeSort
$N \approx 10^5$ $1 <= arr[i] <= 10^3$	iterations $\approx 10^5 + 10^3$ ✓	✓
$N \approx 10^5$ $1 <= arr[i] <= 10^6$	iterations $\approx 10^5 + 10^6$ ✓	✓
$N \approx 10^5$ $1 <= arr[i] <= 10^9$	$\approx 10^5 + 10^9$ *	✓

When to use, which sorting algo?

1. Given arr[N] get 3rd largest elements:
   a) MergeSort & get last 3/4 elements TC: NlogN
   b) In bubble Sort, iterate on arr[] 3rd times & get required elements TC: 3N SC: O(1)
2. Sorted stream of data, insert 1 ele to make entire sort
   a) Insertion idea
3. Given arr[N] sort it

   Constraints:

   $1 <= N <= 10^6$
   $1 <= arr[i] <= 10^2$
   } : CountSort
4. Given arr[N], sort it as fast as possible, without help of any temp[]
   : QuickSort
5. Sort arr[] ele : MergeSort

1. Next monday : Searching1

1. Next tuesday : Searching2