Today's Content:

- → Quick Sort
- → Count Sort

18] Given N array elements, re-arrange the array such that

- → ar[0] should go to it's sorted position
- → All Elements <= ar[0] goto left side of ar[0]
- → All Elements > ar[0] goto right side of ar[0]

	0	1	2	3	4	5	6	7	8	9	10
ar[11] = {	10	3	8	15	6	12	2	18	7	15	14 }

<= 10 | 10 | > 10

Idea1: Sort the array:

	0	1	2	3	4	5	6	7	8	9	10
ar[11] = {	2	3	6	7	8	10	12	14	15	15	18 }

<= 10 ✓ | > 10 ✓

TC: O(N log N)

Idea2:

	0	1	2	3	4	5	6	7	8	9	10
ar[11] = {	10	3	8	15	6	12	2	18	7	15	14 }

	0	1	2	3	4	5	6	7	8	9	10
tmp[11] = {	3	8	6	2	7	10	14	15	18	12	15 }

<= 10 | 10 | > 10

P1 P2

TC: O(N)

SC: O(N)

Copy tmp[] → ar[]

Pseudo code :

```
void re-arrange( int ar[], N) {

    // given ar[N]
    tmp[N];

    p1 = 0,  p2 = N-1;

    i = 1; i < N; i++) {

        if (ar[i] <= ar[0]) {
            //ar[i] to left
            tmp[p1] = ar[i]
            p1++;
        }
        else { // ar[i] to right
            tmp[p2] = ar[i]
            p2--
        }
    }

    tmp[p1] = ar[0]

    // copy tmp[] → ar[]
}
```

TC: O(N)

SC: O(N)

optimize space Complexity

SC: O(1)

re-arrange in same array

Idea 2: re-arrange without space

If $p_1 > p_2$ break:

	0	1	2	3	4	5	6	7	8	9	10
ar[11] = {	10	3	8	15	6	12	2	18	7	15	14 }

Swap ar[0] & ar[p_2]

p_2 = 5, p_1 = 6

2 3 8 7 6 10 12 18 15 15 14

if ($p_1 > p_2$) break

Swap (ar[0] ar(p_2))

	0	1	2	3	4	5	6	7	8	9	10	12
ar[12] = {	14	6	8	18	1	19	17	11	20	10	33	29 }

p_2 = 5, p_1 = 7

ar[12]: 11 6 8 10 1 14 17 19 20 18 33 29

	0	1	2	3	4
ar[4] =	8	4	1	7	4

p_2 = 3, p_1 = 4

$p_1 > p_2$ break

Swap (ar[0], ar[p_2])

Swap (ar[0], ar[3])

ar[4] = 7 4 1 8

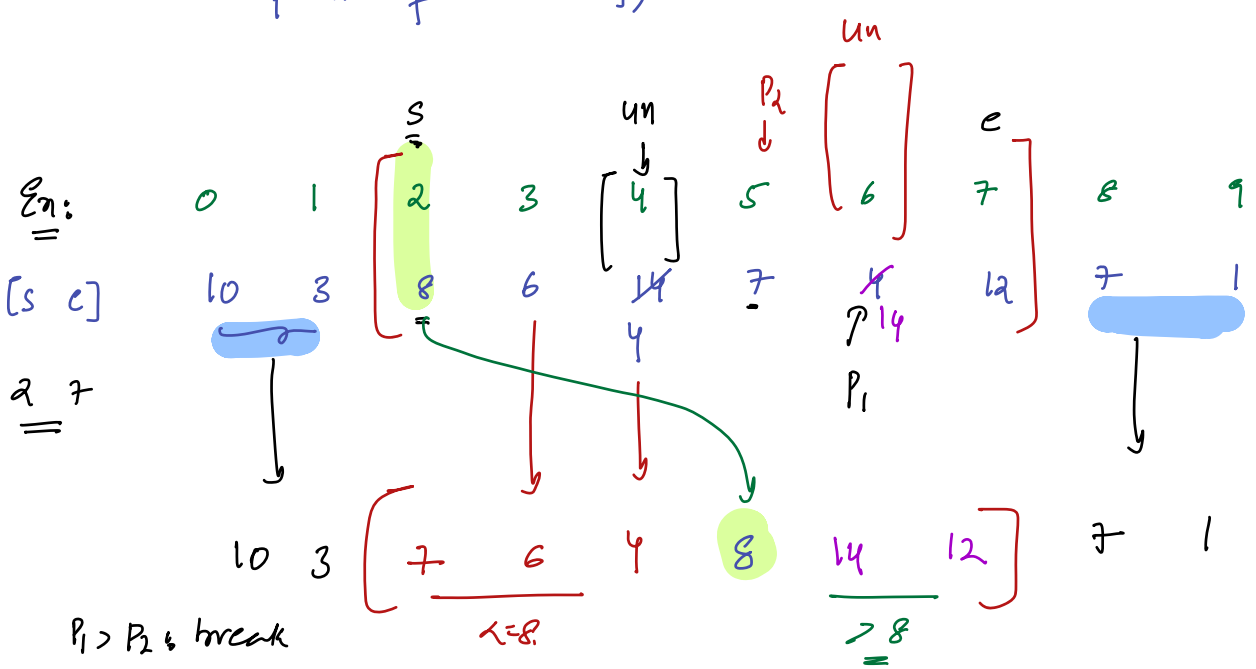
```
void re-arrange (int ar[], int N){
    p1 = 1, p2 = N-1;
    while( p1 <= p2 ){
        if ( ar[0] > = ar[p1] ) { // p1 is in left
            p1++
        }
        else if (ar[0] < ar[p2]) { // p2 in in right
            p2--
        }
        else { // swap ar[p1] & ar[p2]
            p1++, p2--
        }
    }
    // swap ar[0] & ar[p2]
}
```

TC: O(N)

SC: O(1)

28) Given N array elements & subarray [s e]

Re-arrange subarray [s e] such that, ar[s] should come to correct position of subarray,



Swap (ar[s], ar[p2])

Pseudocode : QuickSort

```
int rearrange (int ar[], int s, int e) {
    p1 = s+1, p2 = e
    while( p1 <= p2 ) {
        if ( ar[s] >= ar[p1] ) { // p1 is in left
            p1++
        else if (ar[s] < ar[p2]) { // p2 in in right
            p2--
        else { // swap ar[p1] & ar[p2]
            p1++, p2--
    }
    // swap ar[s] & ar[p2]
    // return corr position of ar[s]
    return p2
}
```

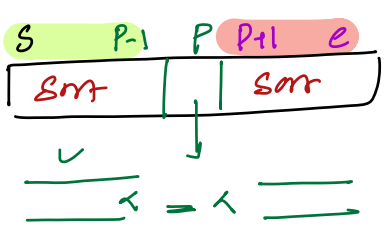
TC: O(N)
SC: O(1)

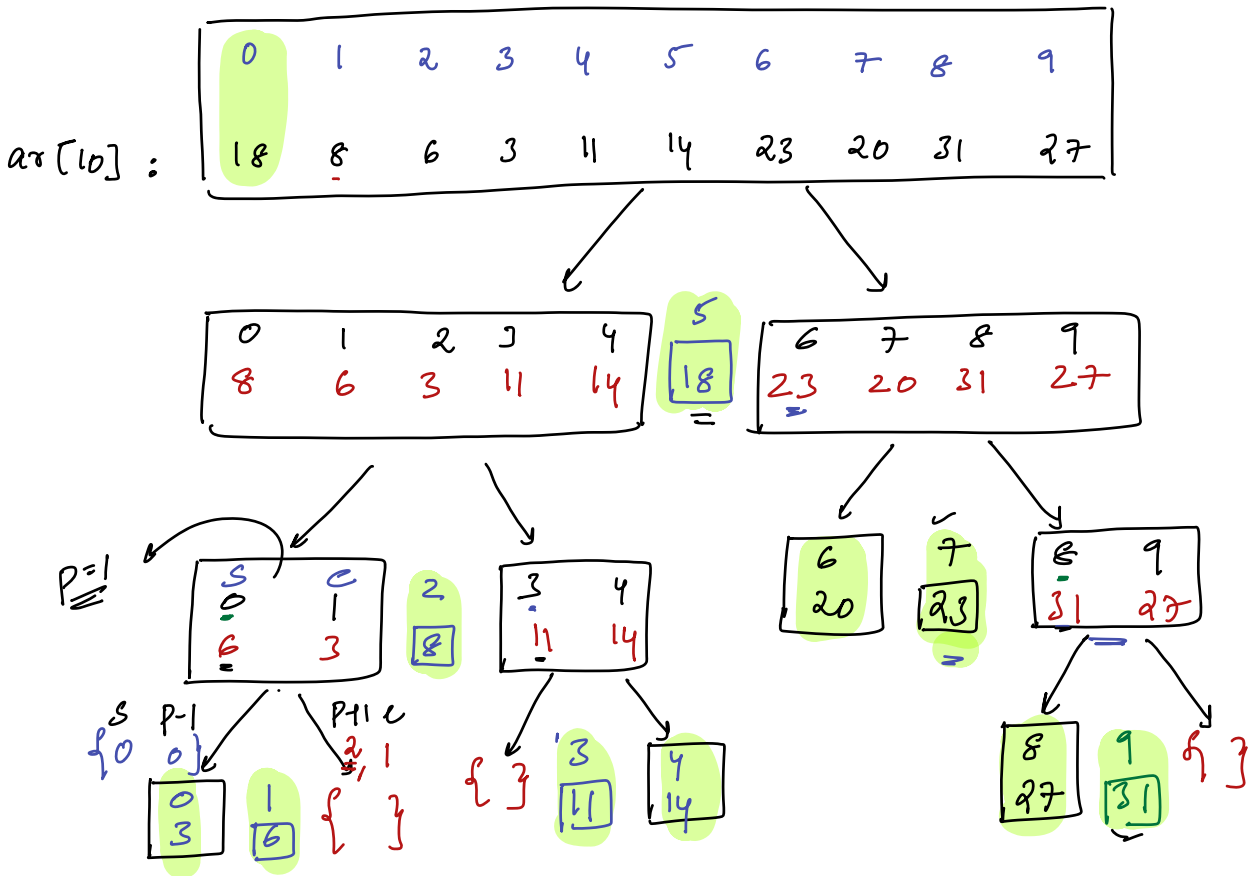
// Assum: It should sort subarray from [s → e]

```
// void QSort( int ar[], int s, int e) {
    if(s >= e) { return } → Doubt?
    // ar[s] is in corr pos + We left 1 Elem
    p = rearrange (ar, s, e)
    QSort(ar, s, p-1) // N/2
    QSort (ar, p+1, e) // N/2
}
```

↳ remaining N-1

// keep ar[s] in its corr position





s == e : 1

s > e : Empty subarray

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

Recurrence Relation :

Best Case

$T(N) = N + 2T(N/2)$: $T(N) = O(N \log N)$ $S_C : O(\log_2 N)$

$k = \log_2 N$

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

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

$T(N-1) = T(N-2) + N-1$

$= T(N-2) + N-1 + N$

$T(N-2) = T(N-3) + N-2$

$= T(N-3) + N-2 + N-1 + N$

$T(N-3) = T(N-4) + N-3$

$= T(N-4) + N-3 + N-2 + N-1 + N$

generalize

$= T(N-k) + N + N-1 + N-2 \cdots$

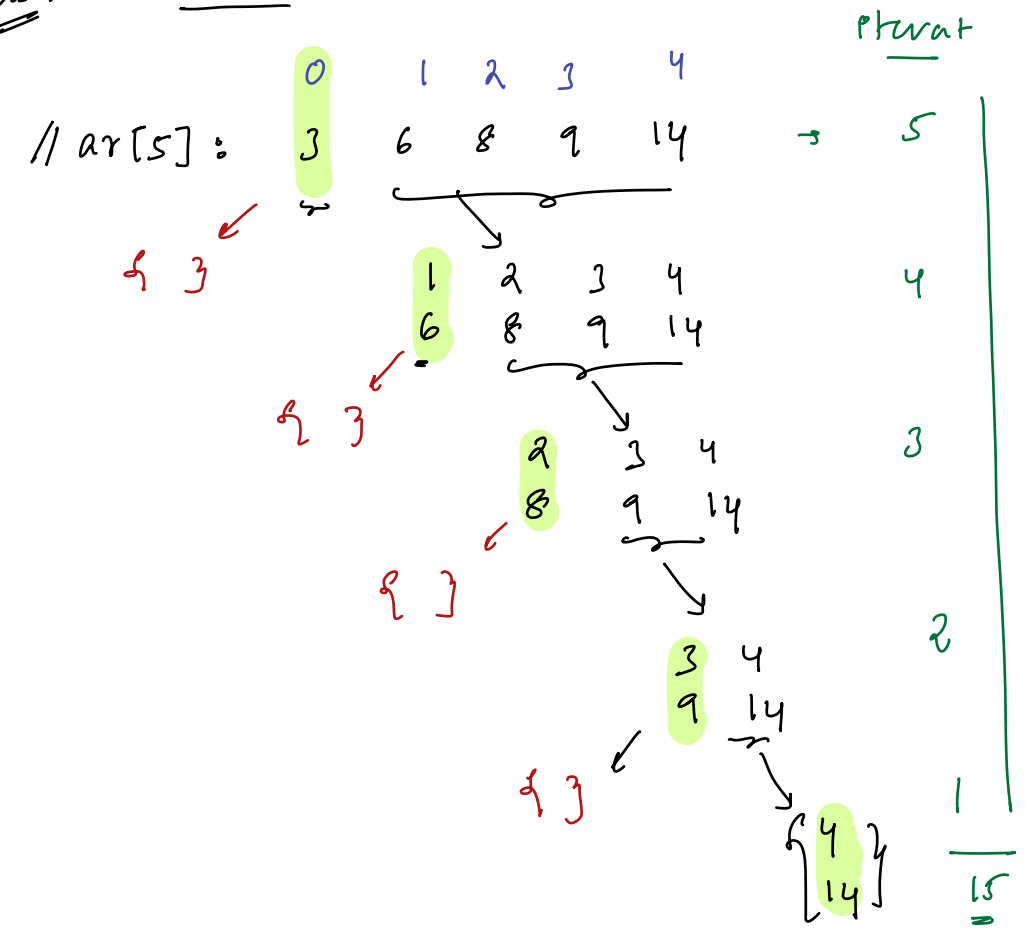
$N-k = 0 \Rightarrow k = N$

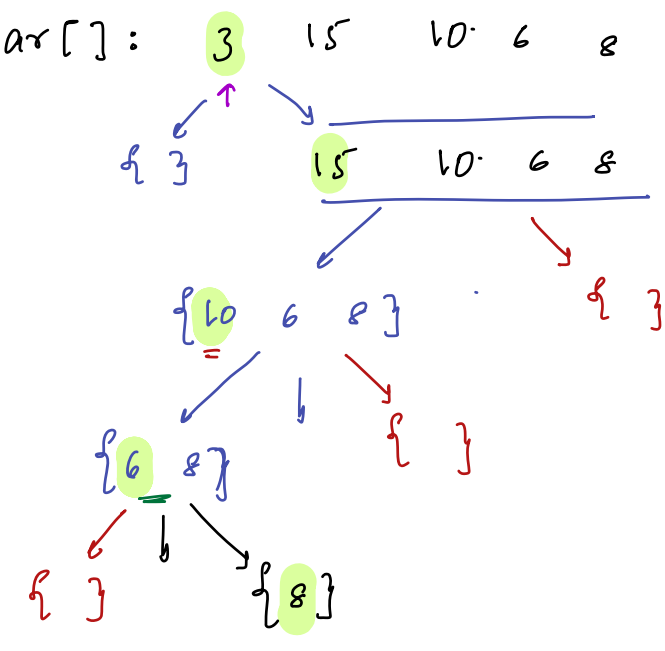
$= T(0) + N + N-1 + N-2 \cdots 1$

Because of Call Stack.

$TC = \frac{(N)(N+1)}{2} \Rightarrow TC: O(N^2)$ $\quad SC: O(N)$

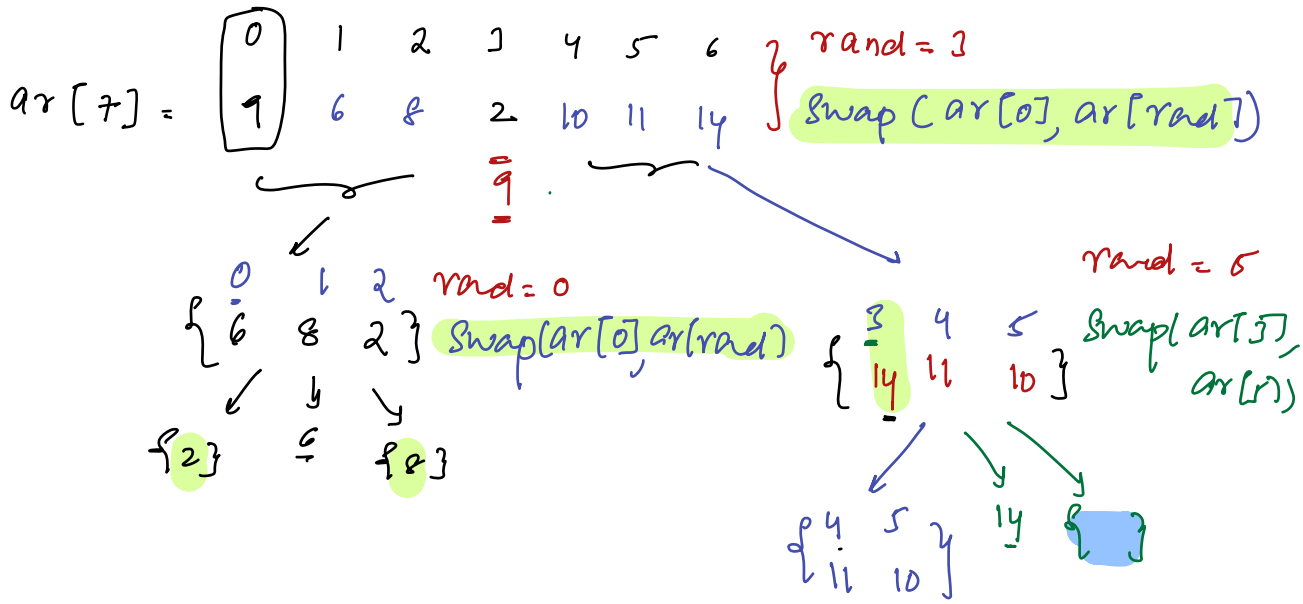
Cases :→ worst case :





At any re-arrange if we pick min or max as our reference, in that case it will goto worst case

// concept : Instead of picking reference as start, make a random choice



// Because of this random probability to pick min element or max element as reference it is very less

```
int re-arrange (int ar[], int s, int e) {
    int r = {rand(s, e)};   // will return a random number in range [s e]
    Swap( ar[s], ar[r])

    p1 = s+1, p2 = e

    while( p1 <= p2 ) {
        if ( ar[s] >= ar[p1] ) { // p1 is in left
            p1++
        }
        else if (ar[s] < ar[p2]) { // p2 in in right
            p2--
        }
        else { // swap ar[p1] & ar[p2]
            p1++, p2--
        }
    }
    // swap ar[s] & ar[p2]
    // return corr position of ar[s]
    return p2
}
```

TC: O(1)

{ Randomized Quick Sort }

Avg Case:

TC: O(N log N)

// Given N array elements, when every element is in range [1 - 4]

Sort array:

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	3	1	4	4	2	1	3	3	2	1 }

Ideal:
→ MergeSort
TC: NlogN

Hashmap: <ele, freq> (key, val)

<3:3>
<1:3>
<4:2>
<2:2>

We know all keys [1 - 4]

```
k=0
for(i=1; i<=4; i++){
    int c = hm[i]
    no: of occurs of i
    for(j=1; j<=c; j++){
        print(i)
        { ar[k] = i
          k++ }
    }
}
// ar[] is sorted
```

Trace:

	hm[i]	print
i = 1 :	3	1 1 1
i = 2 :	2	2 2
i = 3 :	3	3 3 3
i = 4 :	4	4 4

1 1 1 2 2 3 3 3 4 4

→ Count Sorting: Sorting Based on frequencies of data

// Given N array elements, where every element is in range [a - b]

Idea: Even if range is not given

a = min(ar[]) b = max(ar[])

hashmap <int, int> hm

Iterate & insert all elements in hm

```
k=0
for(i=a; i<=b; i++){
    int c = hm[i]
    do: of occurence of i
    for(j=1; j<=c; j++){
        print(i)
        ar[k] = i
        k++
    }
}
// ar[] is sorted
```

Spc: O(N)

Spc: O(N)

TC: O(N) → ?

R →

Total outer loop iterations = [b-a+1]

Total inner loop iterations = [N]

Total iterations

TC: (R + N)

TC: O(R+N)

SC: O(N)

Ex:
ar[4] -> {1, 100, 2, 10}
max = 100
min = 1
Range = 100 >> 4

if R = N

TC: O(N+N)

TC: O(N): Linear

SC: O(N)

Same for

if R > NlogN

TC: O(R+N)

TC: O(R) > NlogN

```
for(i=1; i<=N; i=i+1){
    print(i) -> N times
    for(j=i; j<=N; j=j+1){
        print(j)
    }
}
```

i	j : [i N]	Total inner
1	[1 N]	N
2	[2 N]	N-1
3	[3 N]	N-2
4	[4 N]	N-3
⋮		
N	[N N]	1

Total outer loop → N

$\frac{(N)(N+1)}{2}$

→ Contest: → 3hrs

↳ Contest discussion Monday

↳ 25% Advanced ✓

- Arrays → 3
- Bits → 2
- Maths → 4
- Recur + Sort → 5

14 Sessions

Recursion → 2

Dp → 7

graph → 5

14

Binary Search + 2 Pointer → 4

Strings + Hashmap → 4

Stack Qu Deque → 4

Linked List → 3

Next 25%

Tree → 6

Heaps → 4

Trie → 4

greedy → 2

12

Doubts:

```
Class points {
    int x
    int y
    int dist
    points(int a, int b) {
        x = a, y = b
        dist = x^2 + y^2
    }
}
```

We have N point objects

```
points obj[] = new point();
=
sort(obj, comp)
```

```
comp( points ob1, points ob2) {
    if(ob1.dist < ob2.dist)
        return 1
    else return 2
}
```

// Reverse pair