Q1) Given 2 sorted arrays A & B, merge them & return new sorted array.

Eg $a[3] = \{-1, 4, 8\}$

$b[2] = \{2, 9\}$

$c[5] = \{-1\ 2\ 4\ 8\ 9\}$

Brute: Simply append a after b & then sort.

$\{-1, 4, 8, 2, 9\} \xrightarrow{\text{sort}} \{-1\ 2\ 4\ 8\ 9\}$

TC: $n \log n$

Idea: Use two pointers to merge.

-1 4 8

$p_1 \uparrow \Rightarrow$ -1 2 4 8 9 $p_3 \uparrow$

2 9

$p_2 \uparrow$

TC: $O(n+m)$

Code

```
int [] merge (int A[], int B[], int n, int m){
    int p1 = p2 = p3 = 0
    int c[N+M]
    while ( p1<n && p2<m) {
        if (A[P1] ≤ B[P2]) {
            c[p3] = A[p1]
            // update pointers
            p1++
            p3++
        }
        else {
            c[p3] = B[p2]
            // update pointers
            p2++
            p3++
        }
    }
// Now one array is finished, other
   should be copied as it is
```

```
while (P1 < N) {
    C[P3] = A[P1]
    P1++
    P3++
}

while (P2 < M) {
    C[P3] = B[P2]
    P2++
    P3++
}
return C
}
```

only 1 of them will work

TC: $O(n+m)$

SC: $O(n+m)$

Q2 Given an array & 3 input idx s, m, e

- Subarray [s:m] sorted
- Subarray [m+1:e] sorted

$s < m < e$

Sort the subarray [s:e]

	0	1	2	3	4	5	6	7	8	9	10	11
Eg: ar:	4	8	-1	2	6	9	11	3	4	7	13	0

s m e

2 6 9

How to use same fonda as above ?

	0	1	2	3	4	5	6	7	8	9	10	11
	4	8	-1	2	6	9	11	3	4	7	13	0

P_1 P_2

tmp[8] = -1 2 3 4 6 7 9 11

P_3

Now copy temp back to arr[s:e]

Code

```
void merge (int a[], int s, int m, int e){
    int tmp[e-s+1]
    int p1 = s   p2 = m+1   p3 = 0
    while (p1 ≤ m  && p2 ≤ e ){
        if ( a[p1] ≤ a[p2] ){
            tmp[p3] = a[p1]
            p1++   p3++
        }
        else {
            temp[p3] = a[p2]
            p2++   p3++
        }
    }
    while(p1 ≤ m) {
        tmp[p3] = a[p1]   p1++   p3++
    }
    while(p2 ≤ e) {
        tmp[p3] = a[p2]   p2++   p3++
    }
```

Now copy temp into the a.

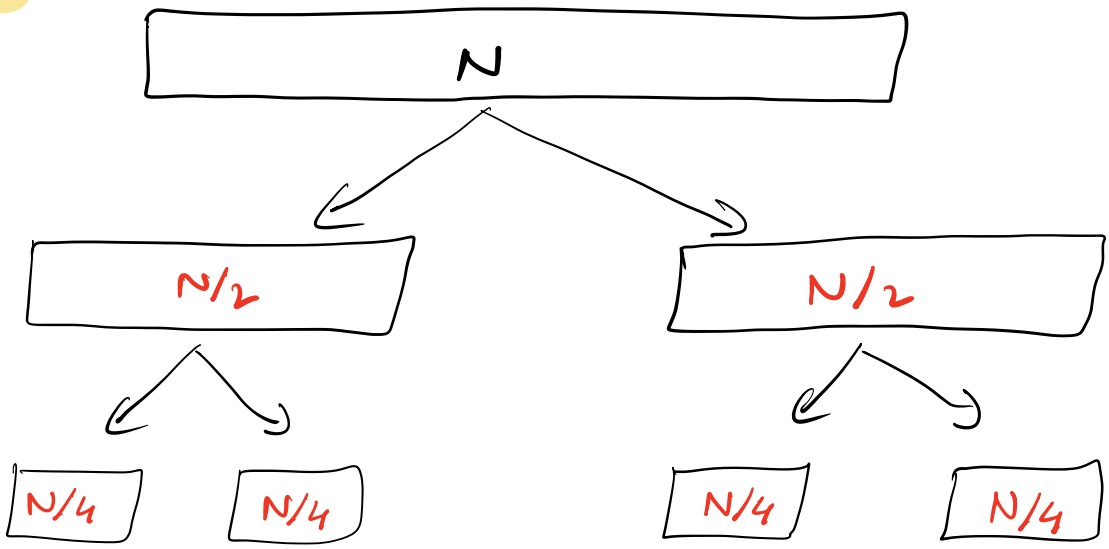
```
for (i=0; i<e-s+1; i++) {
    a[s+i] = tmp[i]
}
}
```

TC: O(N)

SC: O(N)

Q3 Given an unsorted array, sort it (Merge Sort)

Idea



when subarray size = 1, we cannot divide further

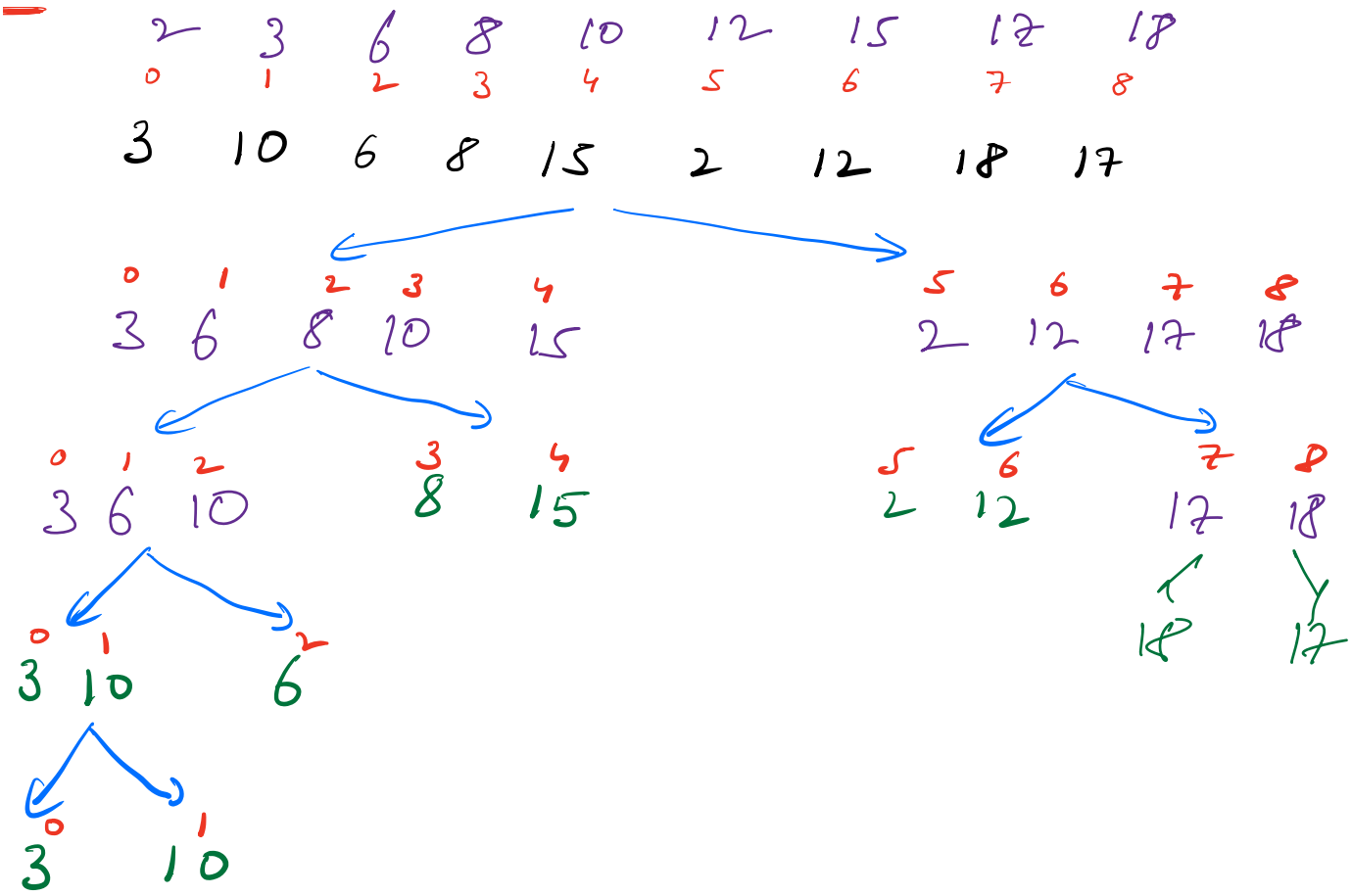
4 16

↓

10

sum

2


2 3 6 8 10 12 15 17 18
0 1 2 3 4 5 6 7 8
3 10 6 8 15 2 12 18 17
3 6 8 10 15
2 12 17 18
3 6 10
8 15
2 12
17 18
3 10
6
3
10

Code

```
void mergesort(int A[], int s, int e){
    if(s==e) return;

    // Use recursion to sort the
          two halves
    int m = (s+e)/2
    mergesort(a, s, m)
    mergesort(a, m+1, e)
    // Now merge the two sorted
          halves
    // Have we written code to merge
    merge(a, s, m, e)
}

main(){
    mergesort(A, 0, n-1)
}
```

$$T(N) = T(N/2) + T(N/2) + O(N)$$
$$= 2T(N/2) + O(N)$$

using master theorem

TC: $O(n \log n)$

Q. Given N elements, find no of pairs i, j st

$i < j$ && $ar[i] > ar[j]$ } inversion pair

Eg

0	1	2
3	1	2

0,1 0,2

ans = 2

Eg

0	1	2	3
8	4	2	1

0,1 0,2 0,3

1,2 1,3

2,3

ans = 6

Brute: 2 nested loops TC: $O(n^2)$

Want to solve in nlogn

Idea: Mergesort is helpful

How?

10 3 8 15 6 | 12 2 18 7 1

A B

IC means Inversion Count

IC(ars) = IC(A) + IC(B) +

IC(i lies in A, j lies in B)

Obs Changing the order of A & B will not affect IC(A&B).

So can I sort both the parts ? yes

10 3 8 15 6 | 12 2 18 7 1

IC(A) = 5 IC(B) = 7

Now sort both parts & perform merge

0 1 2 3 4 5 6 7 8 9

3 6 8 10 15 | 1 2 7 12 18

p_1 p_2

5 + 5 + 3 + 1

$[p_1, m]$

$m - p_1 + 1$

1 2 3 6 7 8 10 12 15 18

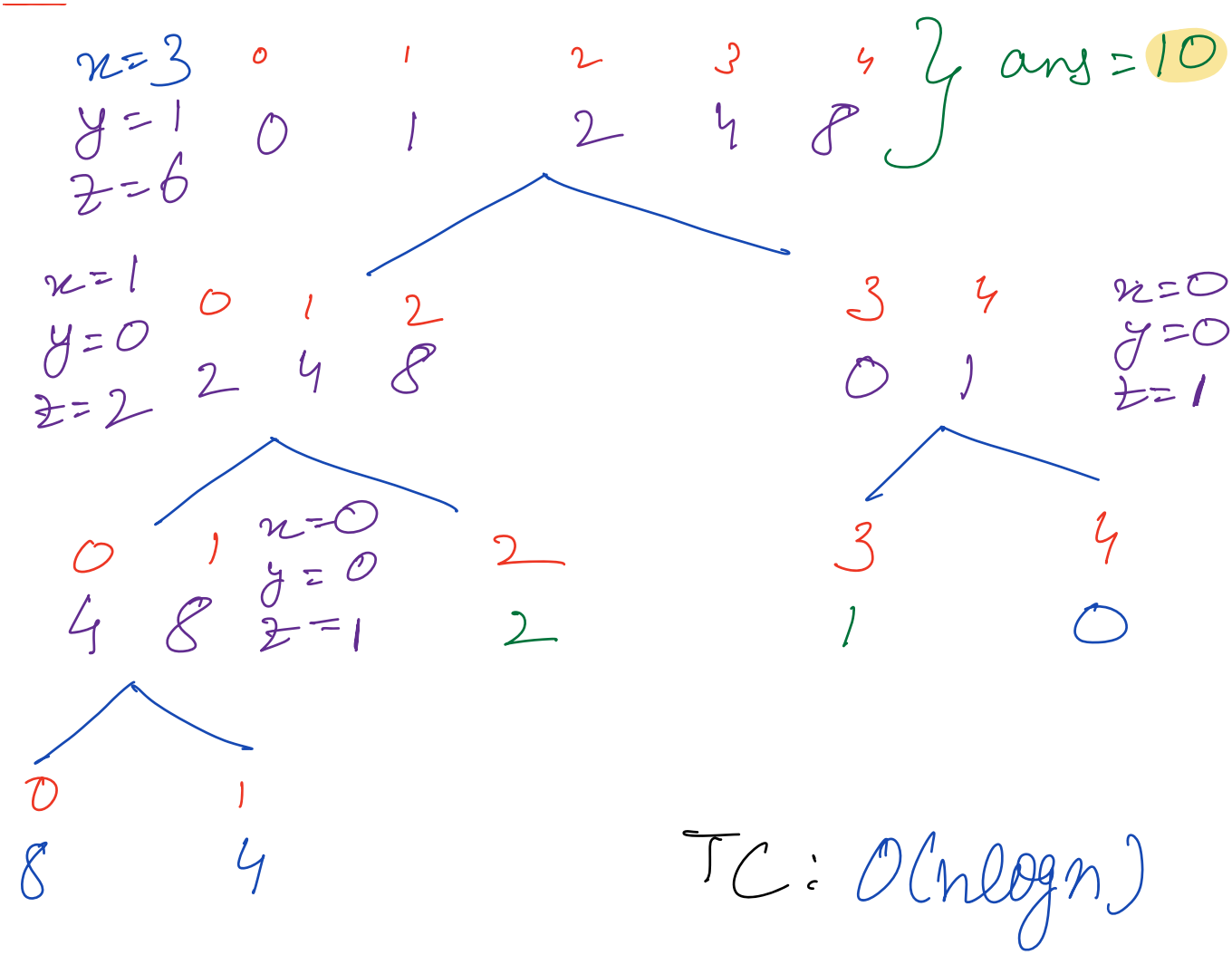
```
int merge (int a[], int s, int m, int e) {
    int cnt = 0
    int p1 = s    p2 = m+1    p3 = 0
    int tmp [ e-s+1]
    while ( p1 ≤ m && p2 ≤ e) {
        if (a[p1] ≤ a[p2]) {

            temp[p3] = a[p1]
            p3++    p1++
        }
        else { cnt++ = m - p1 + 1
            temp[p3] = a[p2]
            p3++    p2++
    }
    while(p1 ≤ m) {

            temp[p3] = a[p1]
            p3++    p1++
    }
```

```
while(p2<=e){
    temp[p3] = a[p2]        p3++    p2++
}

for(i=0; i<e-s+1; i++)
    a[i+l] = temp[i]

return cnt
}
```

assumption ⇒ calc inv_cont & sort

```
int invcount(int arr, int s, int e){
    if(s==e) return 0
    mid = (s+e)/2
    int x = invcount(arr, s, mid)
    int y = invcount(arr, mid+1, e)
    int z = merge(arr, s, mid, e)
    return x+y+z
}
```

Sorted


x=3
y=1
z=6
0 1 2 3 4
0 1 2 4 8
ans = 10
x=1
y=0
z=2
0 1 2
2 4 8
3 4
0 1
x=0
y=0
z=1
0 1
4 8
x=0
y=0
z=1
2
2
3 4
1 0
0 1
8 4

TC: O(nlogn)

Q Find K^{th} max [Selection sort]

If first max $\Rightarrow$ $O(N)$

For second max $\Rightarrow$ Ignore first max
again find max
$N + N$

[5, 8, 1, 15, 7, 6, 2]

5, 8, 1, 2, 7, 6, 15
5, 6, 1, 2, 7, 8, 15
5, 2, 1, 6, 7, 8, 15
1, 2, 5, 6, 7, 8, 15

Code

```
for (i = n-1 ; i >= 0 ; i--) {
    max_id = 0
    for (j = 0; j <= i ; j++) {
        if (a[j] > a[max_id])
            max_id = j
    }
    swap (a[i], a[max_id])
}
```

TC: $O(n^2)$

Done

Kth max $i >= n-k$