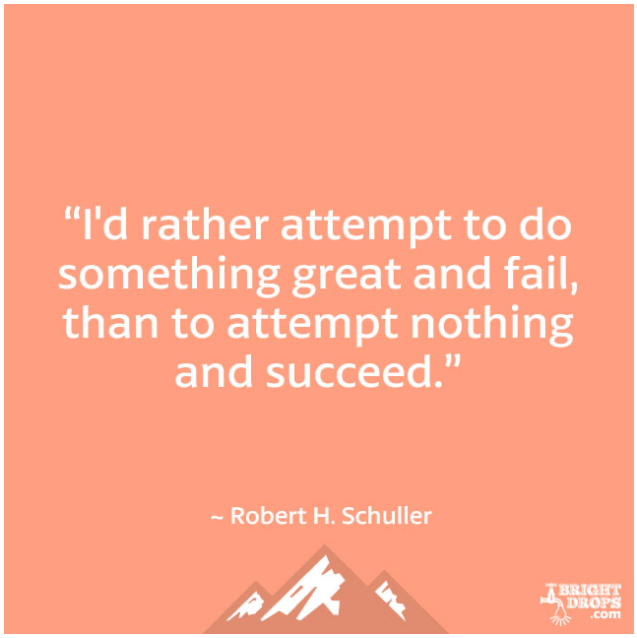
Sorting 1



Good Evening :)

Content for Today's class

- Basics of sorting → Stable / Unstable
  - → Inplace sorting

01. Find K^{th} smallest element
02. Selection sort
03. Bubble sort
04. Merge two sorted Arrays
05. Mergesort
06. Inversion count

Sorting: Arranging the data in specific order based on the parameter

1 2 3 5 7 → Increasing (parameter → values)

9 6 3 2 1 → Decreasing (parameter → values)

Quiz → 1 7 2 9 24 → Increasing (parameter → count of factors)

factors 1 2 2 3 8

Why sorting? → Searching of an ele becomes easy
→ accessing of ele is also easy

Stable / unstable → Two data points have the same parameter value & the order is preserved before & after the sorting, then this is stable sorting.

sort data in increasing order based on marks

Name	Marks
Mohit	0
Aman	100
Shreesh	45
Kajal	32
Aayush	45
Aditya	92

Mohit
Kajal
Shreesh
Aayush
Aditya
Aman
↑
Stable

Name	Marks
Mohit	0
Kajal	32
Aayush	45
Shreesh	45
Aditya	92
Aman	100

↑
unstable

arr[] = 1 5 2 6 2 → sort this array para=values

Sort 1 = 1 2 2 5 6 → stable

Sort 2 = 1 2 2 5 6 → unstable

Inplace sorting Algorithm → Sorting in the same array without extra space. SC = O(1)

Q1. Given ar[N] elements, find K^{th} smallest element

↳ in the sorted arr, return ar[K-1]

ar[8] =

2	8	4	-4	6	7	5	10	-1
0	1	2	3	4	5	6	7	8

K = 3

Idea 1 → Sort the array & return ar[K-1]

TC: O(nlogn)

Idea 2 → Iterate on arr, find min & keep it at its correct position. Repeat this process for K times

K=4

ar[8] =

2	8	4	-4	6	7	5	10	-1
0	1	2	3	4	5	6	7	8

	i	min	ind	swap
-4 8 4 2 6 7 5 10 -1	0	-4	3	ar[0] with ar[3]
-4 -1 4 2 6 7 5 10 8	1	-1	8	ar[1] ar[8]
-4 -1 2 4 6 7 5 10 8	2	2	3	ar[2] ar[3]
-4 -1 2 4 6 7 5 10 8	3	4	3	ar[3] ar[3]

ar[k-1]

Pseudocode

```
k^th smallest ele ( arr, k)
    for (i=0; i<k; i++) {
        min = ar[i]    minidx = i
        for (j=i; j<n; j++)
            if (ar[j] < min) {
                min = ar[j]
                minidx = j
            }
        swap(ar[i], ar[minidx]);
    }
    return ar[k-1];
}
```

TC: O(k*n)
SC: O(1)

Selection sort

```
selection sort ( arr ) {
    for ( i=0 ; i<n ; i++) {
        min = ar[i]      minidx = i
        for ( j= i ; j<n ; j++)
            if (ar[j] < min) {
                min = ar[j]
                minidx = j
            }
        }
        swap (ar[i], ar[minidx]);
    }
}
```

TC : $O(n^2)$

SC : $O(1)$

↓

Inplace sorting algo

* 2 4 2 1 (0 1 2 3)

min	idx
1	3
2	2
2	3

1 4 2 2

1 2 4 2

1 2 2 4

unstable

No. of swaps = n-1 swaps

02. Given ar[N], we can only **swap adjacent** elements, sort the array in increasing order

	0	1	2	3	4	5	6	7	8
ar[i] =	2	8	4	-1	6	7	5	10	-1

2 -1 4 6 5 7 -1 **8** **10**

Repeat this process for (n-1) time

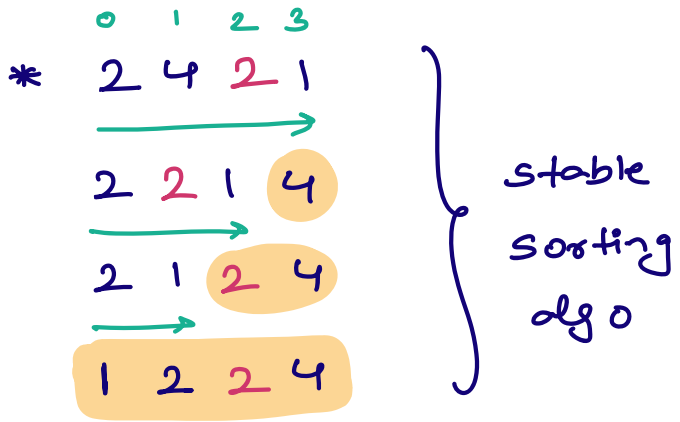
```
bubble sort ([] arr){

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

TC: $O(n^2)$

SC: $O(1)$

↳ inplace sorting algo

No. of swaps = $\frac{n * (n-1)}{2}$



Q3. Given 2 sorted arrays, a[N], b[M], create c[ ] which contains overall sorted data

Eg:- a[4] : {7 10 11 14} a[3] : {3 6 10}

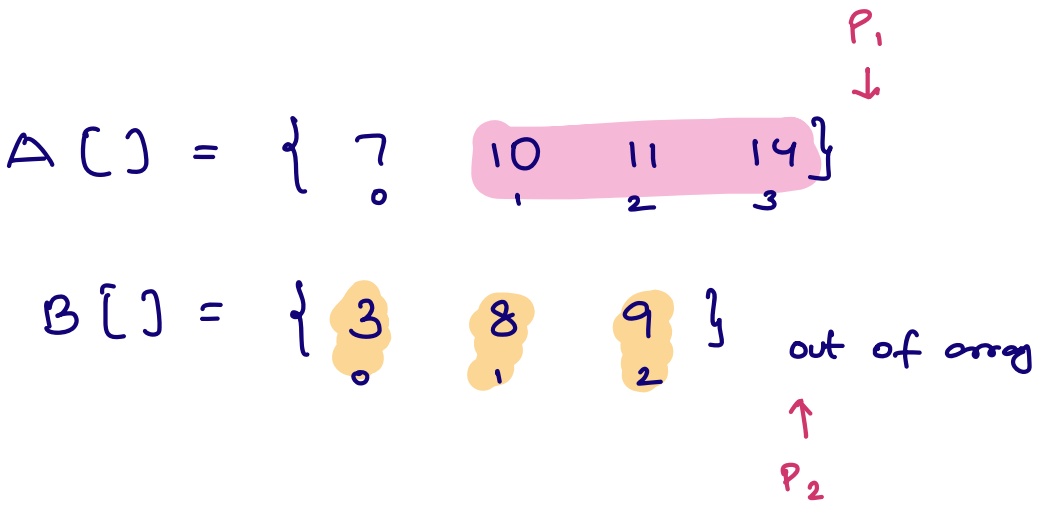
b[3] : {3 8 9} b[2] : {5 9}

c[4+3] = {3 7 8 9 10 11 14} c[3+2] = {3 5 6 9 10}

* Idea 1 → Copy both the arrays in C & then sort it

TC: $(n+m)\log(n+m)$

Idea 2 Two pointers to compare ele in both the arrays



c[] = { 3 7 8 9 10 11 14 }

Merge two sorted arrays

```
int [] merge ( int [n]a , int [b] m )
    int [ ] c = new int [n+m];
    P1 = 0
    P2 = 0
    K = 0   // fill ele in C array
    while (P1<n && P2<m)
        if (a[P1] ≤ b[P2]){
            c[k] = a[P1]
            P1++;
            k++;
        }
```

```
        else {
            c[k] = b[p2]
            p2++;
            k++;
        }
    }

    while (p1 < n){
        c[k] = a[p1]
        p1++;
        k++;
    }

    while (p2 < m){
        c[k] = b[p2]
        p2++;
        k++;
    }

    return c
}
```

TC: O(n+m)

SC: O(1)

* If the question asks you to return an array then SC: O(1)

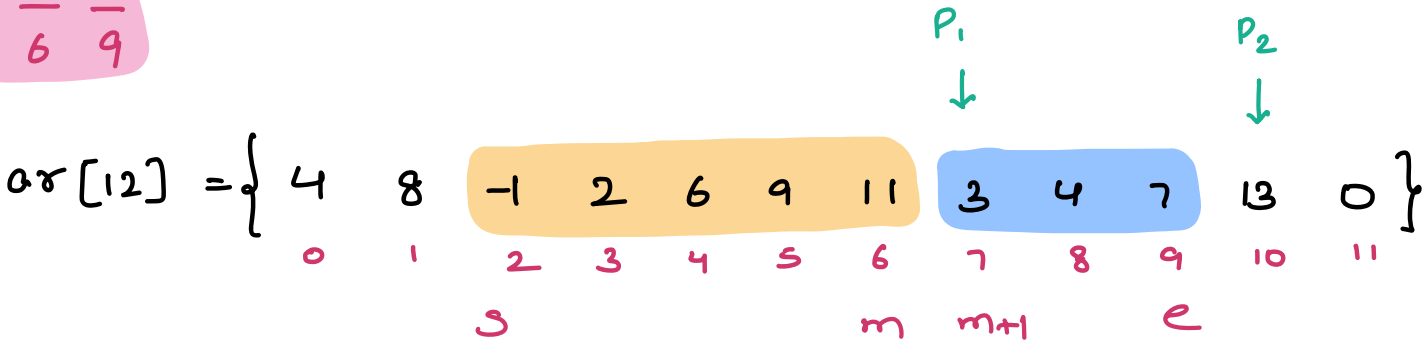
04 Given ar[N] elements & 3 indices s, m, e &

subarray [s m] is sorted

subarray [m+1 e] is sorted

Sort the subarray from [s e]

s m e
2 6 9



c[e-s+1] = {-1 2 3 4 6 7 9 11}

```
int [] merge ( int a[], s, m, e )
    int [ ] c = new int [e-s+1];
    P1 = s
    P2 = m+1
    k = 0   // fill ele in C array
```

```
while (p1 ≤ m && p2 ≤ e)
    if (a[p1] ≤ a[p2]){
        c[k] = a[p1]
        p1++;
        k++;
    }
    else {
        c[k] = a[p2]
        p2++;
        k++;
    }

while (p1 ≤ m){
    c[k] = a[p1]
    p1++;
    k++;
}

while (p2 ≤ e){
    c[k] = a[p2]
    p2++;
    k++;
}

return c;
* Fill the original array.
```

* Given ar[N], sort it without inbuilt function

↳ N = 100

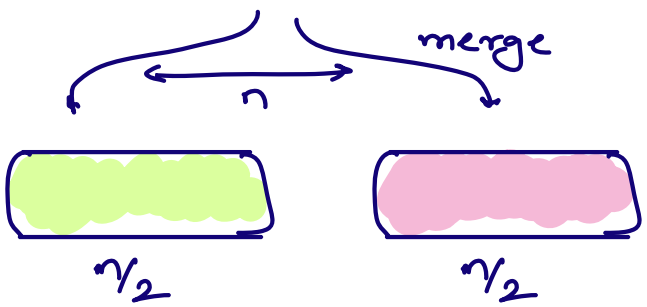
Case - I

TC | Iterations

ar[N] = 

$O(n^2)$ | 10^4

Case - II =

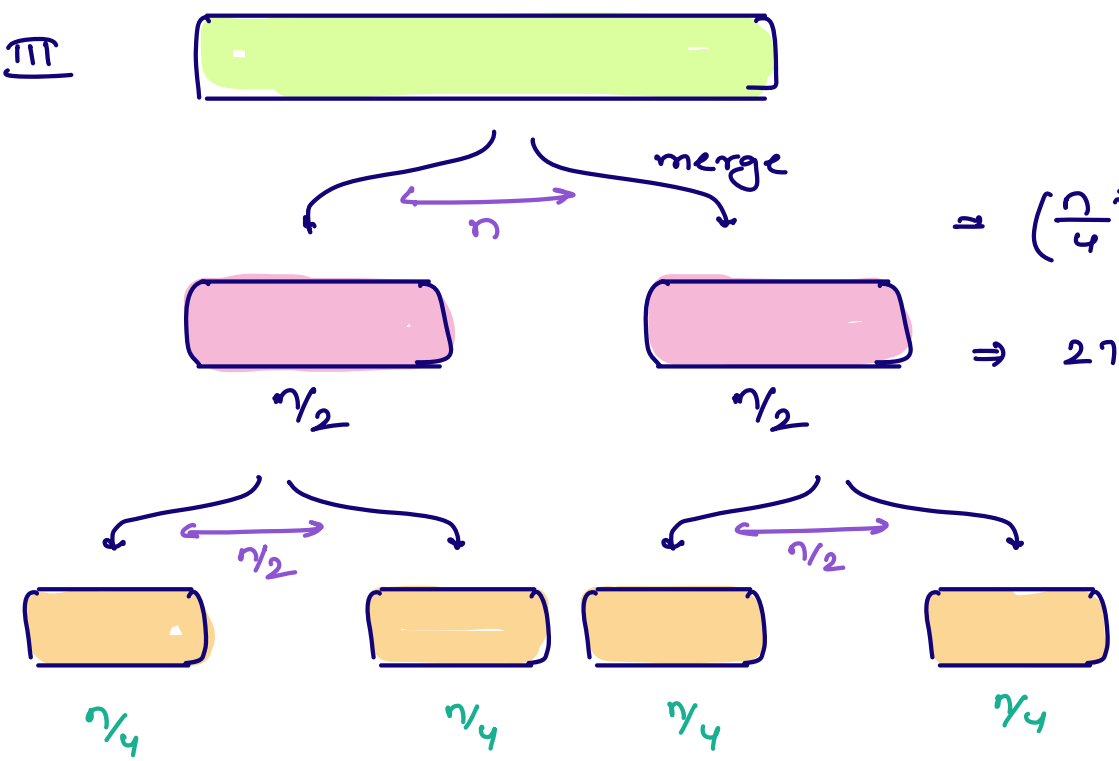


$\left(\frac{n}{2}\right)^2 * 2 + n$

$= \left(\frac{100}{2}\right)^2 * 2 + 100$

$= 5100$

Case - III

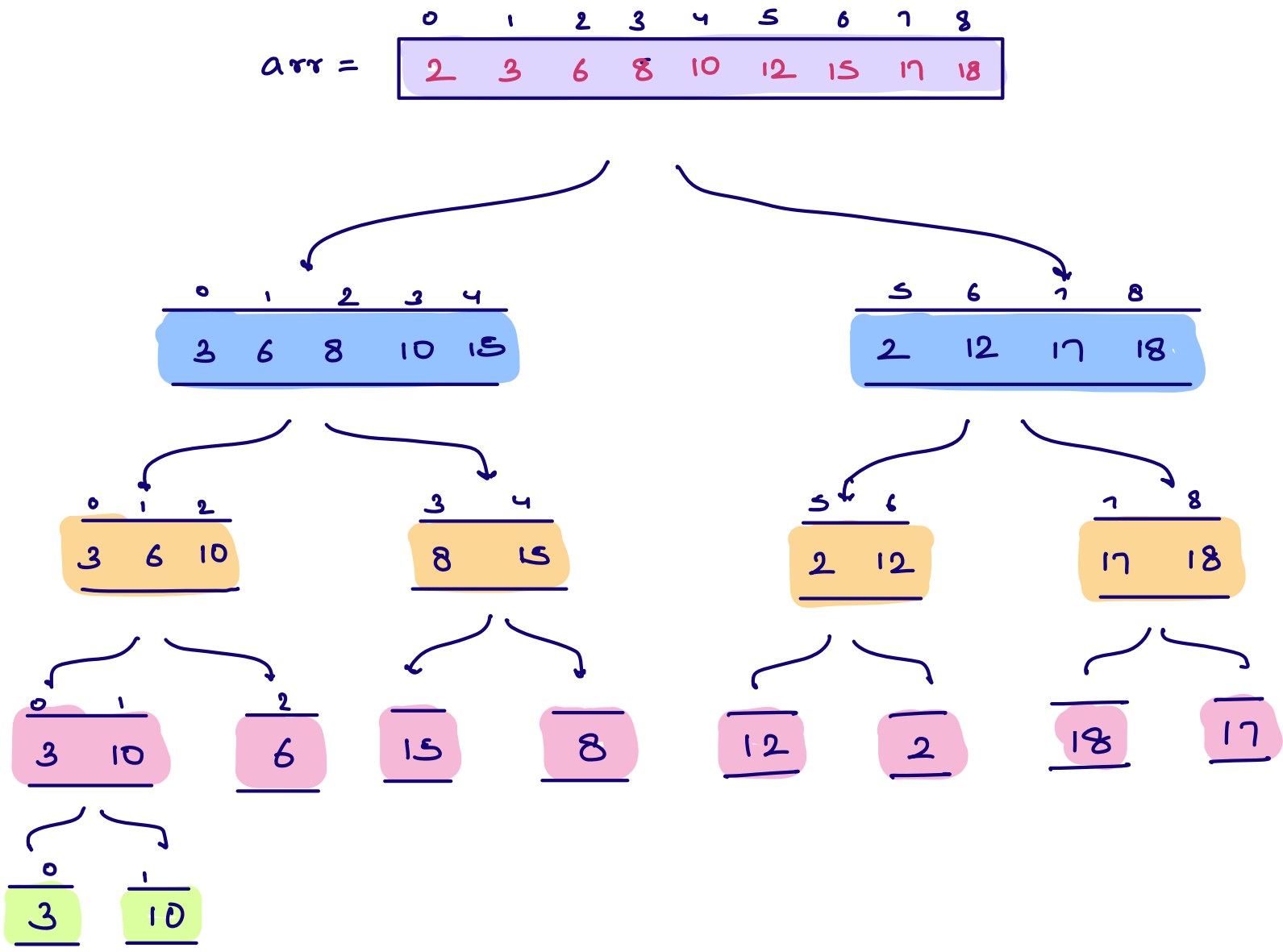


$= \left(\frac{n}{4}\right)^2 * 4 + 2n$

$\Rightarrow$ 2700

Idea for Merge Sort

↳ Split the array until it gets reduced to one ele & then start merging those arrays



* stable sorting algorithm

```
mergesort (arr, s, e)
    if (s == e) return
    int m = (s+e)/2
    mergesort (arr, s, m)
    mergesort (arr, m+1, e)
    merge (arr, s, m, e)
}
```

TC: O(nlogn)
SC: O(n)

Recurrence relation

$f(n) = f(n/2) * 2 + n$ ⇒ Expand by yourself

Inversion count

Given an integer array, count the number of inversion pairs in the array

Inversion pair → (i, j) where $(i<j \;\&\&\; A[i] > A[j])$

$A = [4\ 5\ 1\ 2\ 6\ 3]$

$0\ 1\ 2\ 3\ 4\ 5$

No. of inversion pair = 7

(0,2) (0,3) (0,5)

(1,2) (1,3) (1,5)

(4,5)

$A = [8, 3, 4]$

$0\ 1\ 2$

No. of inversion pair = 2

(0,1) (0,2)

Brute force → Check for all pairs (i, j) & $i<j$ and $a[i] > a[j]$, increase the count

TC: $O(n^2)$

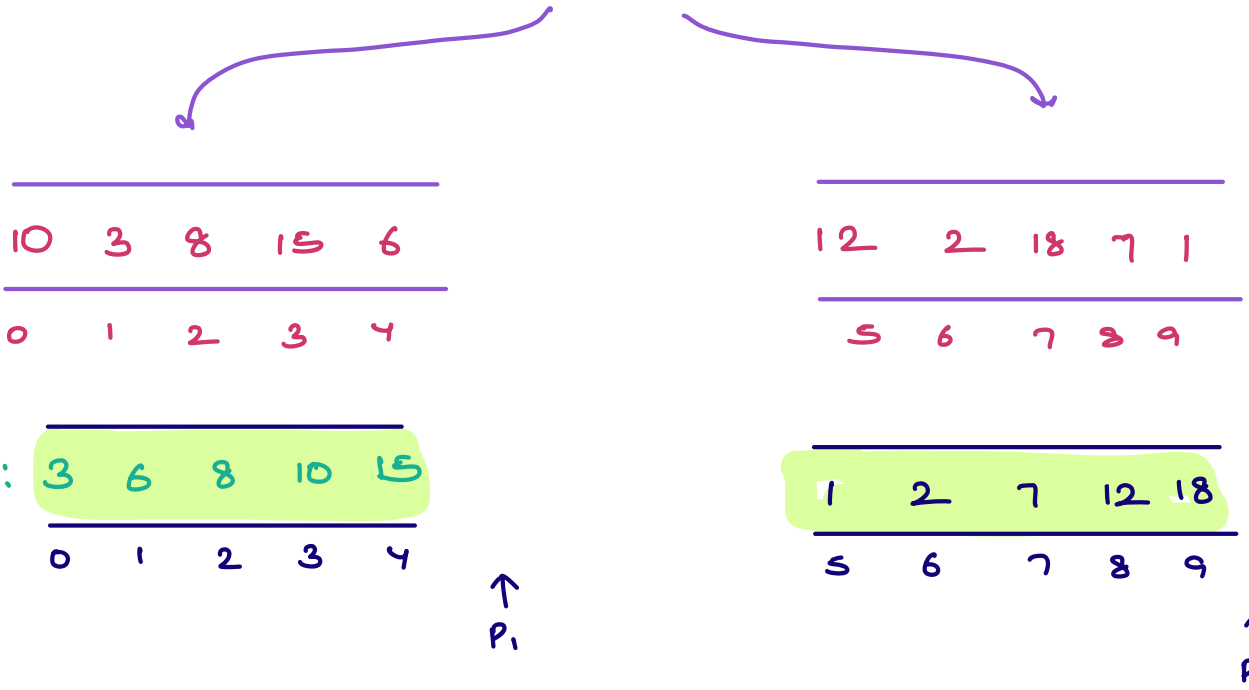
SC: $O(1)$

Idea 2 ⇒ Use Merge sort

ans = 6 + 2 + 4 + 5 + 2 + 3 + 1 + 2 + 1 = 26

arr[] = { 10 3 8 15 6 12 2 18 7 1 }

0 1 2 3 4 5 6 7 8 9



Merge = 1 2 3 6 7 8 10 12 15 18

arr[P1] < arr[P2] → no inversion

arr[P1] > arr[P2] → inversion

Inversion = 5 + 5 + 3 + 1 = 14

(3,1) (3,2) (8,7) (15,12)

(6,1) (6,2) (10,7)

(8,1) (8,2) (15,7)

(10,1) (10,2)

(15,1) (15,2)

Inversion count = $m - P_1 + 1$

Code → { TODO }