Count sort

A = [6 3 4 2 4 2 1]

if numbers are small [here 0 - 6]
create freq array

0	1	2	3	4	5	6
0	∅ 1	∅ ∅ 2	∅ 1	∅ ∅ 2	0	∅ 1

Now use this to generate sorted array

1 2 2 3 4 4 6

Does this work for large values like 10^9

No, because 10^9 size array I cannot make.

How to handle -ve numbers

A = [-3 -2 2 0 1]

min = -3

you want to represent min at idx 0

idx

-3 +3→ 0

-2 → 1

-1 → 2

0 → 3

1 → 4

2 → 5

elem ⟷ idx

-3 ←

+3 →

Now freq array

0	1	2	3	4	5
1	1		1	1	1

Now sorted arr. idx = 0 has 1

this means number = ? has 1

-3 -2 0 1 2

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

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

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

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

Brute: Simply append a after b & then sort.

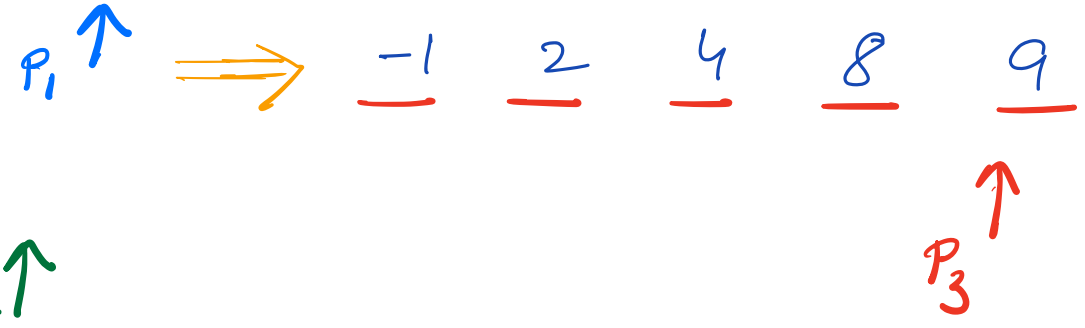
$\{-1, 4, 8, 2, 9\} \xrightarrow{\text{sort}} \{-1 \quad 2 \quad 4 \quad 8 \quad 9\}$

TC: $O(n \log n)$

Idea: Use two pointers to merge.

$-1 \quad 4 \quad 8$

$2 \quad 9$



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
}
```

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 funda 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++
    }
}
```

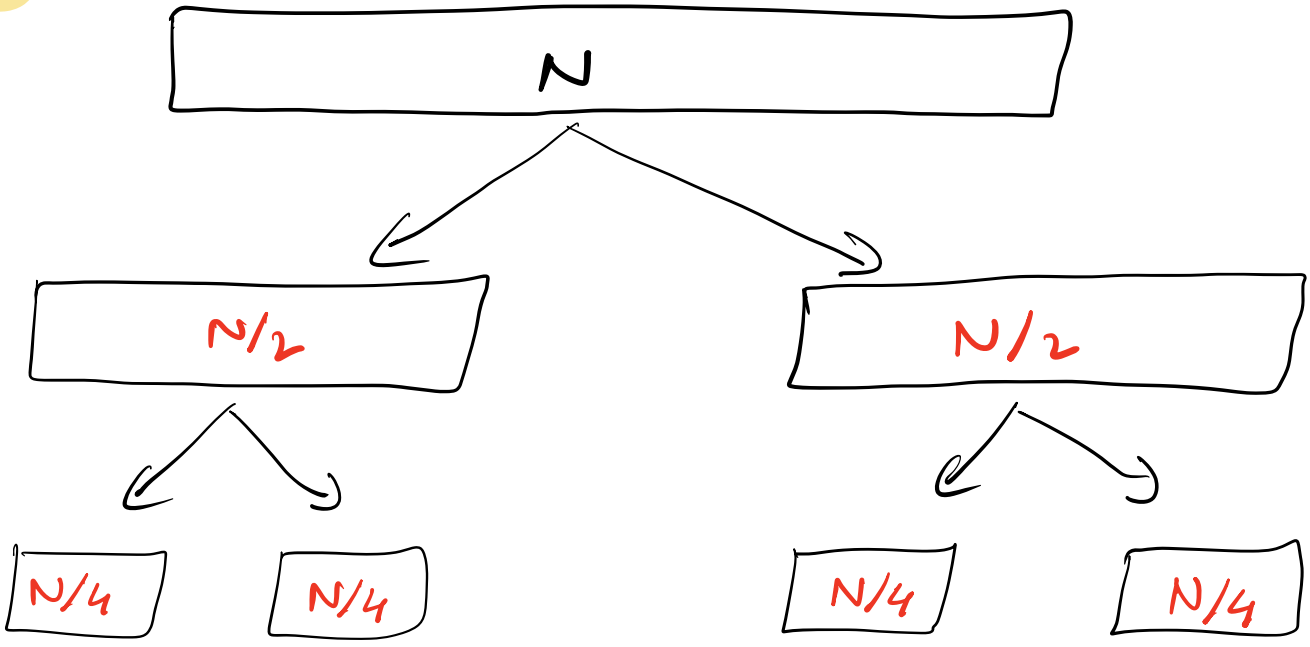
Now copy temp into the a.

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

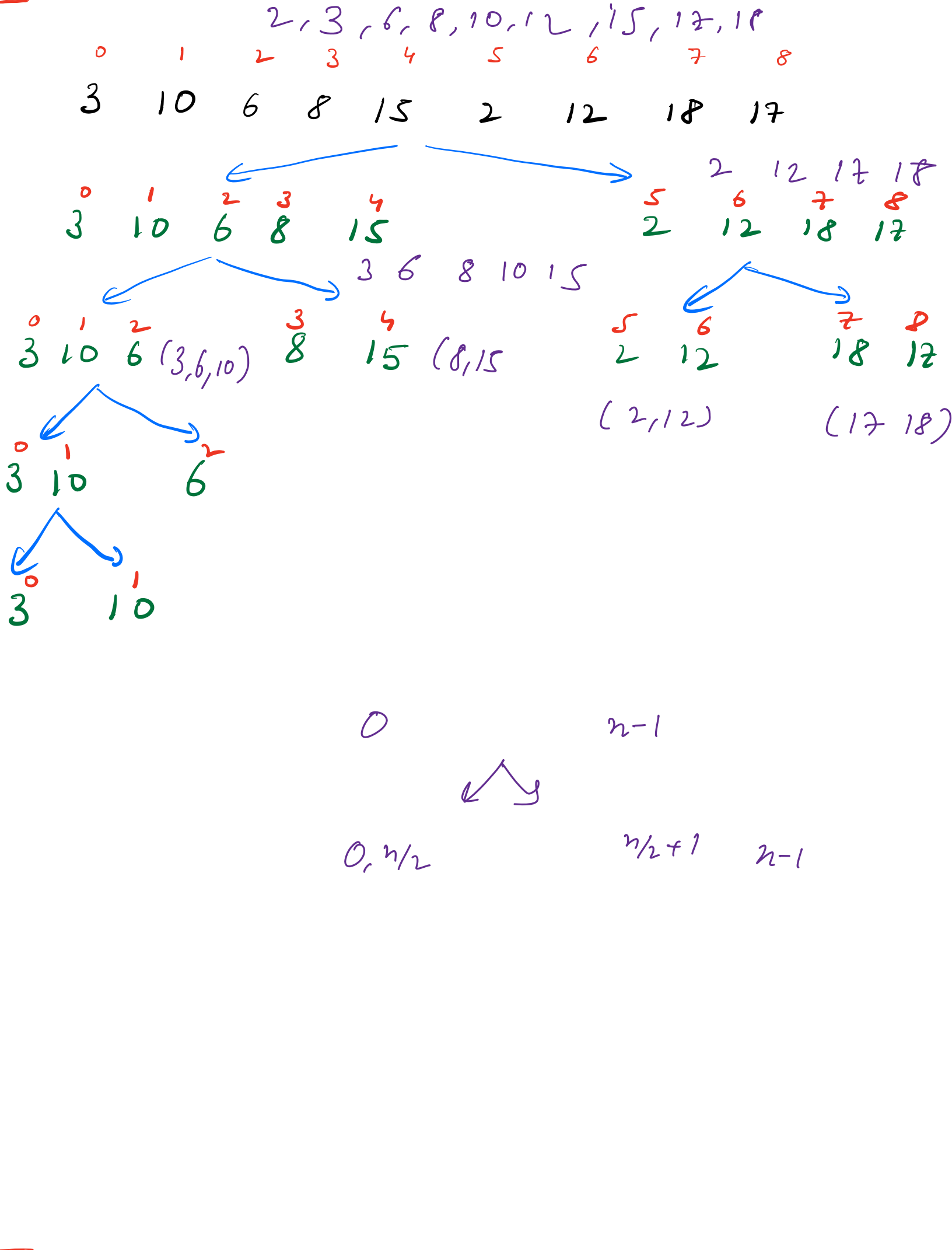
TC:
SC: } linear

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

Idea



when subarray size = 1, we cannot divide further


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
2 12 17 18
3 6 8 10 15
(3,6,10)
(8,15
(2,12)
(17 18)
0 n-1
0, n/2
n/2+1 n-1

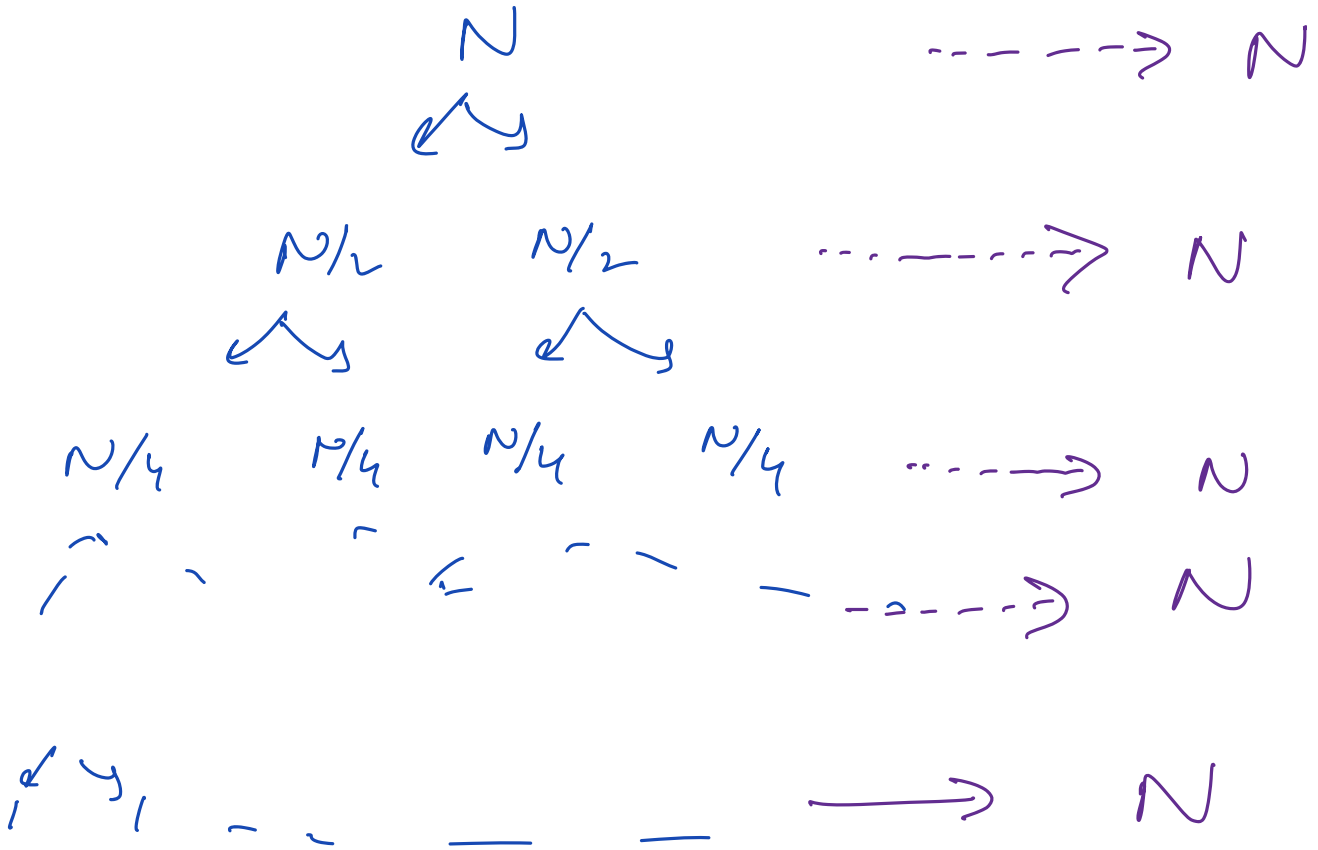
Code

```
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)
}
```



How many levels = $\log N$

$\underbrace{N + N + N + \cdots N}_{\log N \text{ times}} = N \log N$

SC: $O(N)$

5 3 1 4 2

5 3 5 1 5 4 5 2 3 1 3 2 4 2

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

ans = 2

Eg

0	1	2	3
5	2	6	1

ans = 5,2 5,1 6,1 2,1

Brute: Consider all pairs 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(arr) = IC(A) + IC(B) + one guy on A one guy on 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)= 4

Now sort both parts & perform merge

s m m+1 e

3 6 8 10 15 | 1 2 7 12 18

p_1 p_2

1 2 3 6 7 8 10 12 15

$b_2 - m - 1$

ans = 2 + 2 + 3 + 3 + 4

= 14

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

```
while(k2<=e){
    temp[k3] = a[k2]      k3++    k2++
}

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

return cnt
}
```

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

TC: $O(N \log N)$

SC: $O(N)$

Stable sort ⇒ relative order doesn't change

Inplace sort → SC O(1)

```
sum (x, y) {
|   let x+y
}
```

23	21	24	24	22
A	B	C	D	E

21 22 23 24 24

Unstable ⇒ B E A D C

21 22 23 24 24

B E A C D ⇐ Stable

{done}