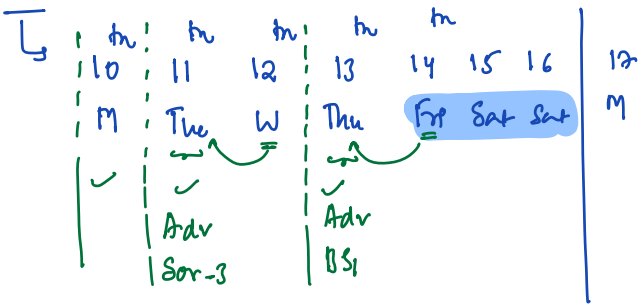
Todays Content: →



- → Sum of max of all subsequences ✓
- → Insertion sort ✓
- → Inversion Count ✓

Q1) Given ar[N] calulate sum of max of every subsequence ?

Ex: ar[3] = { 3 1 -4 } : TODO: sum of min of every subseq

All subsequence	max
{ }	0
{3}	3
{1}	1
{-4}	-4
{3 1}	3
{3 -4}	3
{1 -4}	1
{3 1 -4}	3

// sum of all max = 10

3 * 4 + 1 * 2 + -4 * 1 = 10

Idea1: Generate all Sub, for every subseq get max & add all

TC: $2^N * N$

Idea2: Contribution technique :

= Calulate contribution of each ar[] element to final sum

```
ans = 0;
for(i=0; i<N; i++){
    ans = ans + ar[i] * freq
}
```

// In how subsequence is ar[i] max

Note: an ele will contribute, if it's max of subsequence

Ex: ar[] = { 4 7 2 5 8 10 }

In how many subseq 7 is max?

4 2 5 * *

{ 7 }

{ 4 7 }

{ 7 2 }

{ 7 5 }

{ 4 7 2 }

{ 4 7 5 }

{ 7 2 5 }

{ 4 7 2 5 }

Obs: For every ar[i] we need to no: of elements smaller than that, to calculate no: of subseq in which ar[i] is max?

↳ To do the above thing we can simply sort ar[]

Ex: ar[] = { 4 7 2 5 8 10 }

↳

	0	1	2	3	4	5
Sort ar[] = {	2	4	5	7	8	10 }
# ele < ar[i] =	0	1	2	3	4	5
# subseq ar[i] max =	2^0	2^1	2^2	2^3	2^4	2^5

Contrib = 2 + 8 + 20 + 56 + 128 + 320 = 534

```
int summanSub(int ar[N]) {        TC: O(NlogN + N)
    sort(ar[])
    ans = 0
    i=0; i<N; i++) {
        // no: of sub in which ar[i] is max = 2^i
        ans = ans + ar[i] * (1<<i)
    }
    return ans;
}
```

// Doubts? Even if data repeats above logic work

	0	1	2
ar[3] = {	3	5	3 }

All sub	max
{ }	0
{3}	3
{5}	5
{3}	3
{3 5}	5
{5 3}	5
{3 3}	3
{3 5 3}	5

Sum = 29

	0	1	2
ar[3] = {	3	5	3 }
[sort			
ar[3] = {	3	3	5 }
#sub =	2^0	2^1	2^2
cont =	3	6	20 = 29

20)

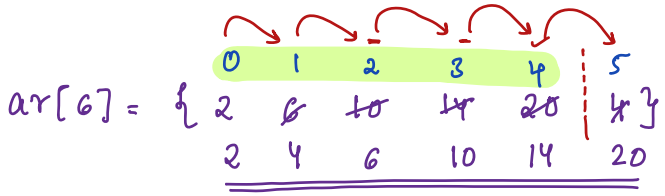
Given ar[N], first n-1 elements are sorted, sort entire ar[]

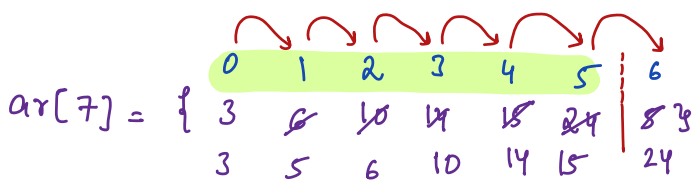
Expected SC: O(1)

	0	1	2	3	4	5
ar[6] = {	2	6	10	14	20	4 }

→ Output ar[6]: { 2 4 6 10 14 20 }

Idea: Sort entire arr[]: TC: O(N log N)





j :	ar[j] > ar[j+1]
5	ar[5] > ar[6]: swap
↓	
4	ar[4] > ar[5]: swap
↓	
3	ar[3] > ar[4]: swap
↓	
2	ar[2] > ar[3]: swap
↓	
1	ar[1] > ar[2]: swap
↓	
0	ar[0] > ar[1]: *

→ no swap

→ break it

Idea: Iterate from back & compare adj elements, if they are not in correct order we swap.

Insertion Step: To insert a single ele in sorted data to make overall data sorted

Pseudocode: Given ar[N]

```
// first N-1 are sorted [0, N-2]
for(j = n-2; j >= 0; j--){   TC: O(N) SC: O(1)
    if (ar[j] > ar[j+1])
        swap ar[j] & ar[j+1]
    else
        break;
}
```

3Q) Given ar[N], sort it using Insertion Step → Insertion Sort

ar[6] : { 10 | 3 6 8 2 5 }
(indices: 0 | 1 2 3 4 5)

Insertion Step

Insert 1 : { 3 10 | 6 8 2 5 }

Insert 2 : { 3 6 10 | 8 2 5 }

Insert 3 : { 3 6 8 10 | 2 5 }

Insert 4 : { 2 3 6 8 10 | 5 }

Insert 5 : 2 3 5 6 8 10

```
void Sort( int ar[N]){   TC: O(N^2)  SC: O(1)  ↳ inplace
    i = 1; i < n; i++){
        // insert ar[i] is sorted arr[]      0 1 2 ... i-1 | i   (Sorted)
        j = i-1; j >= 0; j--){
            if ( ar[j] > ar[j+1]) {
                swap ar[j] & ar[j+1]
            }
            else { break }
        }
    }
}
```

Note: ar[j] == ar[j+1] we won't swap, hence relative order won't change

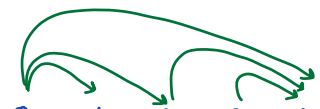
Q8) Given ar[N] calculate no: of pairs [i, j] such that

$i < j$ & $ar[i] > ar[j]$

Inversion count?

$i < j \Rightarrow ar[i] < ar[j]$

$i < j \Rightarrow ar[i] > ar[j]$



Ex1: 10 3 8 15 6 } pairs = 5

Ex2:

	0	1	2	3	4	5	6	7	8	9
	10	3	8	15	6	12	2	18	7	1
#pairs:	6	2	4	5	2	3	1	2	1	0

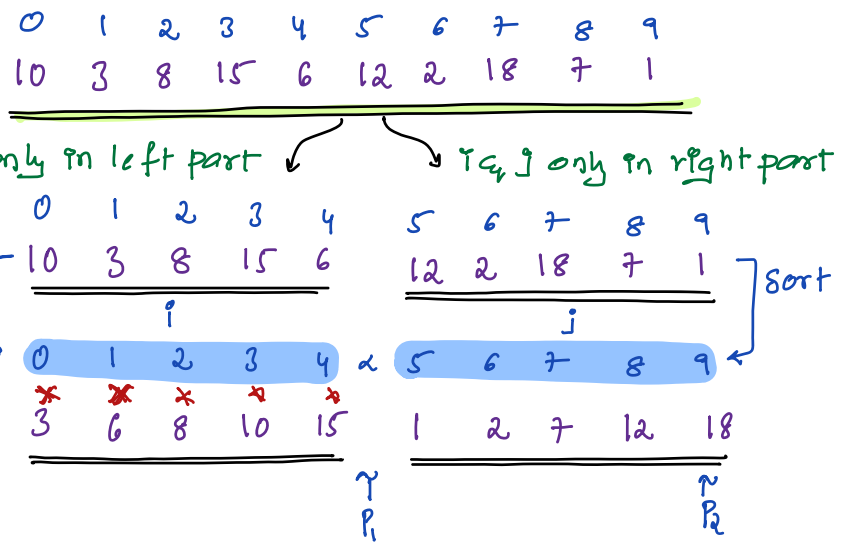
} pairs = 26

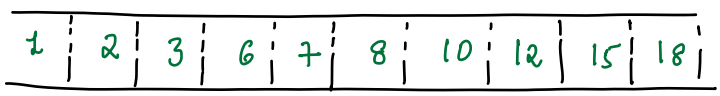
Idea: Compare all pairs (i, j)

```
int pairs( int ar[n] ) {   TC: O(N^2)   SC: O(1)
    c = 0
    for(i = 0; i < n; i++) {
        for(j = i+1; j < n; j++) {
            if( ar[i] > ar[j] ) {
                c = c + 1
            }
        }
    }
    return c;
}
```

Pairs $i<j$ and $ar[i] > ar[j]$

Idea:



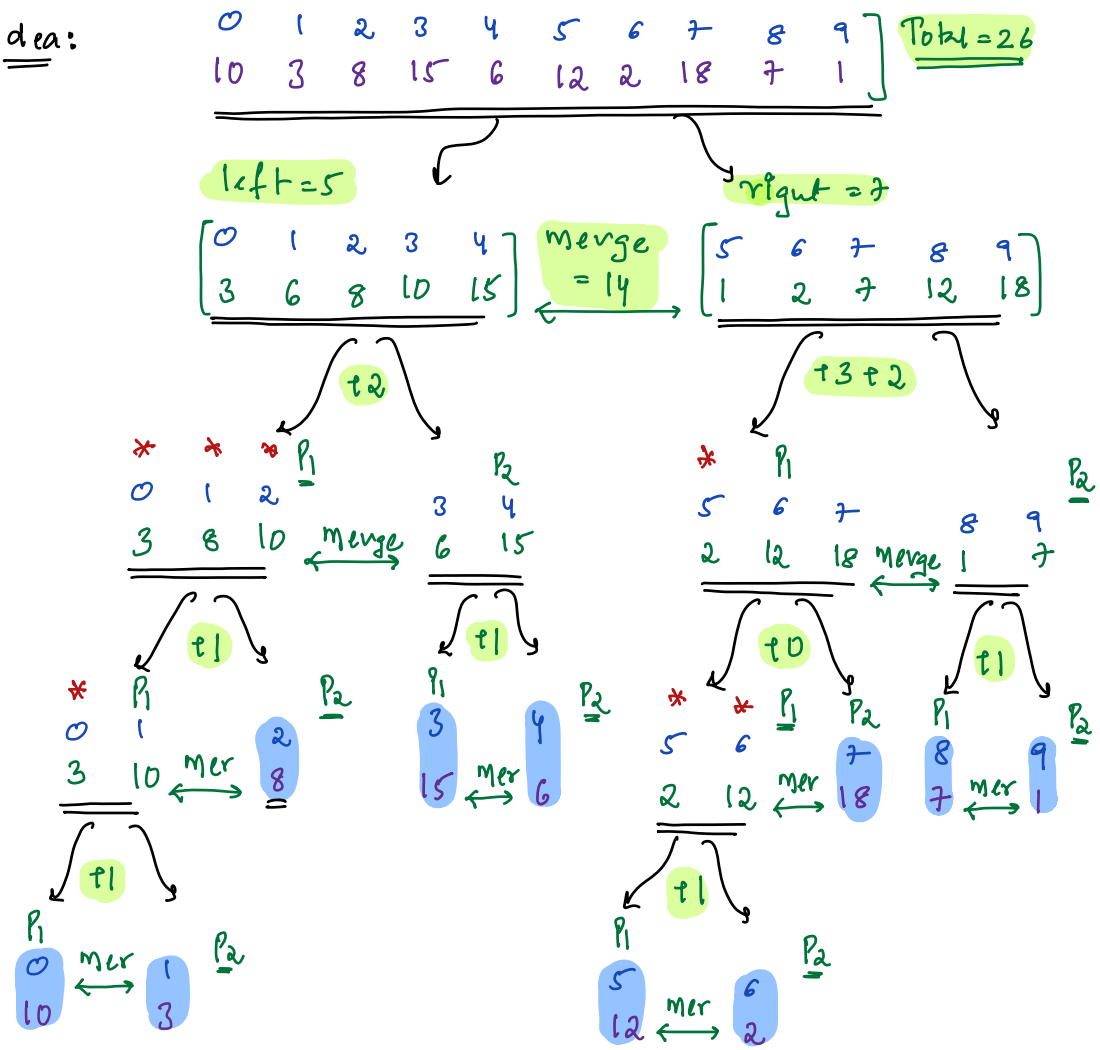


Inversion pairs $c = 5 + 5 + 0 + 0 + 3 + 0 + 0 + 1 + 0 + 0 = 14$

pairs only in left part + pairs only in right part

Obs: during merge process, when we take an element from right $c = c +$ no. of elements in left side

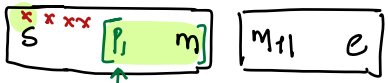
Idea:


0 1 2 3 4 5 6 7 8 9
10 3 8 15 6 12 2 18 7 1
Total = 26
left = 5
right = 7
0 1 2 3 4
3 6 8 10 15
merge = 14
5 6 7 8 9
1 2 7 12 18
t2
t3 t2
P1
P2
merge
t1
t0
mer

```
int c = 0 // inversion pair count        T: O(N log N)  SC: O(N)

void merge(int A[], int s, int m, int e){

    tmp[e-s+1];
    p1 = s, p2 = m+1, p3 = 0
    while( p1<=m && p2<=e){
        if( A[p1] <= A[p2] ){
            tmp[p3] = A[p1]; p3++, p1++
        }
        else{// A[p2] comes first
            tmp[p3] = A[p2], p3++, p2++,   c = c + m - p1 + 1
        }
    }
    while( p1<=m) { tmp[p3] = A[p1]; p3++, p1++ }
    while( p2<=e) { tmp[p3] = A[p2], p3++, p2++ }
    //copy tmp[ ] -> ar[s e]
    i = s, j=0; i<=e; i++, j++ ){
        ar[i] = tmp[j]
    }
}
void mergeSort( int ar[], int s, int e){
    if(s==e){return}
    int m = (s+e)/2
    mergesort(ar, s, m) -> f(N/2)
    mergeSort(ar, m+1, e) -> f(N/2)
    merge(ar, s, m, e) -> N
}
return c; // contains inversion pair
```



#elements [p1 m] = m - p1 + 1

#No. of elements in the left side