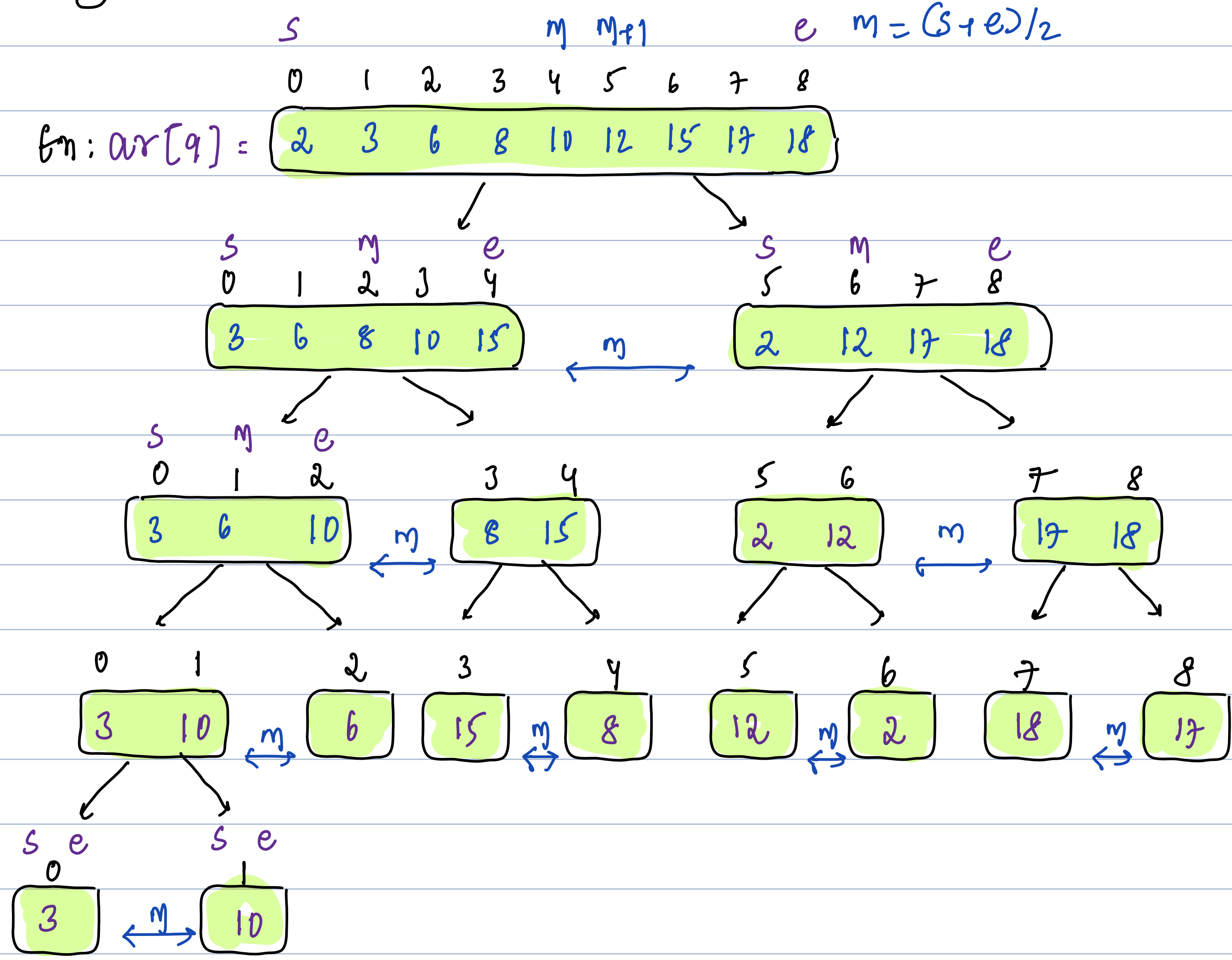
Todays Content

1. Merge Sort : TC & SC
2. Inversion Count

Merge Sort:

Keep dividing arr[] in 2 halfs, till it contains 1 element & Merge.

Tracing:



```
int[] sort(int[] arr, int N){
    mergeSort(arr, 0, N-1); # Sort arr[] from 0.. N-1;
    return arr; # return sorted array.
}
```

Ass: Given arr[ ], s, e : Sort arr[] from s.. e & return nothing.

```
void mergeSort(int arr[], int s, int e){   TC: O(N log N)   SC: O(N)
    if(s==e){return;}
    int m = (s+e)/2;
    mergeSort(arr, s, m)
    mergeSort(arr, m+1, e);          arr[] : {.. [s ✓ —— m] [m+1 ✓ —— e] }
    merge(arr, s, m, e);
}
```

```
void merge(int arr[], int s, int m, int e){
    #Merge 2 consecutive sorted subarray [s..m]  [m+1..e]
    int tmp[e-s+1]; # 0 ... e-s
    int p1=s, p2=m+1, p3=0;

    while(p1<=m && p2<=e){
        if(A[p1] < A[p2]){tmp[p3]=A[p1]; p3++; p1++;}
        else{tmp[p3]= A[p2]; p3++; p2++;}
    }

    while(p1<=m){                               while(p2<=e){
        tmp[p3]=A[p1]; p3++; p1++;                  tmp[p3]= A[p2]; p3++; p2++;
    }                                           }

    for(int i=s; i<=e; i++){
        arr[i]= tmp[i-s]
    }
}
```

Recursive Relation:

$T(N) = 2T(N/2) + O(N) \longrightarrow 2^1 T(N/2^1) + 1*N$

$T(N/2) = 2T(N/4) + O(N/2)$

$= 2[2T(N/4) + N/2] + N$

$= 4T(N/4) + 2N \longrightarrow 2^2 T(N/2^2) + 2*N$

$T(N/4) = 2T(N/8) + O(N/4)$

$4[2T(N/8) + N/4] + 2N$

$8T(N/8) + N + 2N$

$= 8T(N/8) + 3N \longrightarrow 2^3 T(N/2^3) + 3*N$

$T(N/8) = 2T(N/16) + O(N/8)$

$8[2T(N/16) + N/8] + 3N$

$16T(N/16) + N + 3N$

$= 16T(N/16) + 4N \longrightarrow 2^4 T(N/2^4) + 4*N$

Generalized

$T(N) = 2^k T(N/2^k) + k*N \qquad T(1) = 1$

$N/2^k = 1, \quad N = 2^k, \quad k = \log_2 N$

$= N\,T(N/N) + \log_2 N * N$

$= N*1 + N\log_2 N$

$\boxed{T(N) = O(N\log_2 N)}$

Simpliefied Master's Therom:

$T(N) = aT(N/b) + F(N)$

$F(N) = N^c$ {c is highest degree in F(N)} $t = \log_b^a$

if $t < c$: $O(N^c)$

if $t == c$: $O(N^t \log N)$

if $t > c$: $O(N^t)$

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

$a = 2$ $b = 2$ # $F(N) = O(N)$ $c =$ degree in $F(N) = 1$

$t = \log_b^a = \log_2^2 = 1$, $c = 1$.

↳ highest power in exp

if $(t == c)$: $O(N^1 \log^N) = O(N \log N)$

Inversion Count:

Given an ar[N] calculate no: of pairs (i, j) such that i < j && ar[i] > ar[j]

	0	1	2	3	4
Ex1: ar[5] :	10	3	8	15	6
	i	j			

0 < 1 : ar[0] > ar[1] ✓

0 < 2 : ar[0] > ar[2] ✓

0 < 4 : ar[0] > ar[4] ✓

2 < 4 : ar[2] > ar[4] ✓

3 < 4 : ar[3] > ar[4] ✓

Ans = 5

	0	1	2	3	4	5	6	7	8	9	
Ex2: ar[10]: {	10	3	8	15	6	12	2	18	8	1	}
	6	2	3	5	2	3	1	2	1	0	= 25

Idea1:

Generate all pairs & count no: of valid pairs (i, j) are there such i < j && ar[i] > ar[j]

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

Idea2: Uses idea mergeSort

	0	1	2	3	4	5	6	7	8	9
arr[10]:	10	3	8	15	6	12	2	18	8	1

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

Sort   Sort

s $\boxed{p_1 \quad m}$

0	1	2	3	4		5	6	7	8	9
~~3~~	~~6~~	~~8~~	~~10~~	~~15~~		~~1~~	~~2~~	~~8~~	~~12~~	18

p_1 p_2

	0	1	2	3	4	5	6	7	8	9	
tmp[] =	1	2	3	6	8	8	10	12	15	18	
	+5	+5	+0	+0	+0	+2	+0	+1	+0	0	= 13

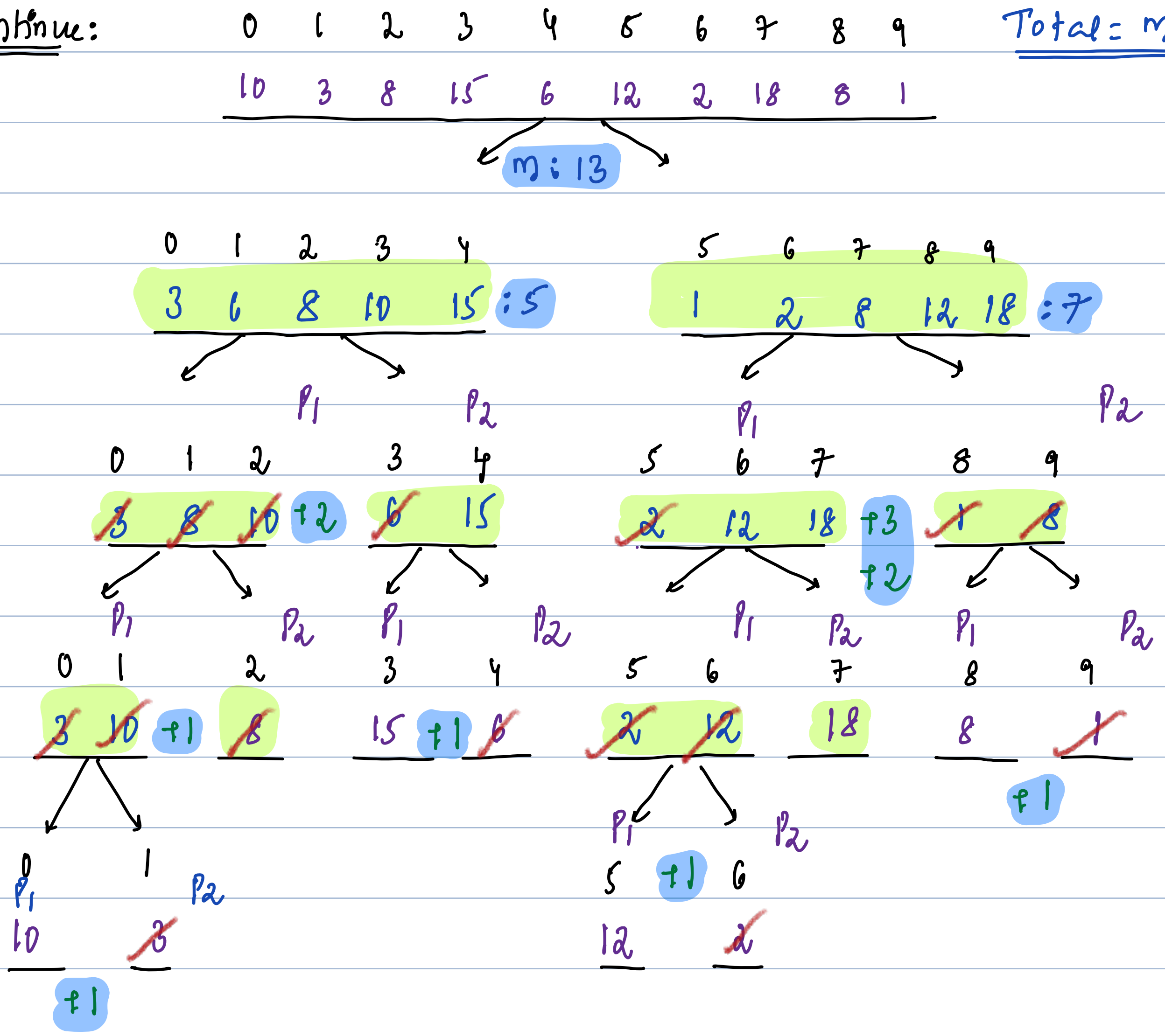
#obs1: if(arr[p_1] <= arr[p_2]) {
  # keep arr[p_1], It won't effect count
}
else # arr[p_1] > arr[p_2]
{
  # keep arr[p_2];
  c = c + # Element in p_1 side
    # m - p_1 + 1;
}

s [p_1 m]

#obs2: We are missing pairs
Only in leftpart & Only in right part
Total pairs = leftpairs + rightpairs + Mergepairs.

Continue:

Total = M + L + R = 25



Final Conclusion:

In merge code:

If we select element from right side.

Inc count by no: of elements in left side.

```
long c=0;
long pairs (int[] arr, int N){
    c=0; # For global variable initialize before function call
    mergeSort(arr, 0, N-1); # Sort arr[] from 0.. N-1;
}   return c; # We are adding c for all merge functions.

void mergeSort(int arr[], int s, int e){ TC: O(NlogN)  SC: O(N)
    if(s==e){return;}
    int m = (s+e)/2;
    mergeSort(arr, s, m)
    mergeSort(arr, m+1, e);
    merge(arr, s, m, e);
}

void merge(int arr[], int s, int m, int e){
    #Merge 2 consecutive sorted subarray [s..m]  [m+1..e]
    int tmp[e-s+1]; # 0... e-s
    int p1=s, p2=m+1, p3=0;

    while(p1<=m && p2<=e){
        if( A[p1] <= A[p2]){tmp[p3]=A[p1]; p3++; p1++;}
        else{tmp[p3]= A[p2]; c=c+[M-p1+1]; p3++; p2++;}
    }

    while(p1<=m){                               while(p2<=e){
        tmp[p3]=A[p1]; p3++; p1++;                  tmp[p3]= A[p2]; p3++; p2++;
    }                                           }

    for(int i=s; i<=e; i++){
        arr[i]= tmp[i-s]
    }
}
```