Today's Content:

→ Merge 2 Sorted Arrays

→ Merge 2 Sorted Subarrays

→ Merge Sort

→ Comperator

Fri day → Sorting Completed

Saturday

Monday 6pm

Adv:
- Arrays - 3
- BPTs - 2
- Maths - 4
- Recur - 2
- Sorting - 3

Monday 9pm - Contest discussion
(E, M, AM) ↑ optional

Wednesday - {Binary Search}

Ques) Given 2 sorted arrays A[N], B[M], merge, create a and return a new sorted array.

a[3] : { -1 4 8 }

b[2] : { 2 9 }

c[5] : { -1 2 4 8 9 }

Ideal: A[N], B[M], C[N+M]

1) copy A[] → C[] : O(N)

2) copy B[] → C[] : O(M)

3) sort C[] : ≤ (N+M) log (N+M)

TC: N + M + (N+M) log (N+M)

a[7] = { -5 -1 3 7 10 12 15 }

0 1 2 3 4 5 6

P_1 P_1 P_1 P_1 P_1

copy remaining data into C[]

b[5] = { -4 0 2 8 9 }

0 1 2 3 4

P_2 P_2 P_2 P_2 P_2

5 → P_2 → out of bounds : no comparison

c[12] = { -5, -4, -1, 0, 2, 3, 7, 8, 9, 10, 12, 15 }

// Eg 1:

A[5] = { 3 8 9 11 14 20 }

0 1 2 3 4 5

P_1 P_1 P_1 P_1 P_1 P_1

6 → P_1 → out of bounds

B[3] = { 3 6 25 30 } → copy remaining elements to C[]

P_2 P_2 P_2

c[10] = { 3, 3, 6, 8, 9, 11, 14, 20, 25, 30 } Sorted

Pseudo Code

```
int[] merge(int A[], int N, int B[], int M) {

    int p1 = 0, p2 = 0, p3 = 0;

    int C[N+M];

    while( p1 < N  &&  p2 < M ) // out of bounds.        Loop breaks, when one of goes
    {
        if( A[p1] <= B[p2] ) { // A[p1] should come first
            C[p3] = A[p1]; p1 = p1+1; p3 = p3+1
        }
        else { // B[p2] should come first
            C[p3] = B[p2]; p2 = p2+1; p3 = p3+1
        }
    }

    // p1 is left out
    while( p1 < N ) { C[p3] = A[p1]; p1 = p1+1; p3 = p3+1 }

    // p2 is left out
    while( p2 < M ) { C[p3] = B[p2]; p2 = p2+1; p3 = p3+1 }

    return C;
}
```

Iterations → (N+M) SC: O(1) — It is given in question to create a new array to solve

TC: O(N+M)

28) Given N array elements & 3 input indices s, m, e

Also given { // Subarray [s m] is Sorted
// Subarray [m+1 e] is Sorted } → // Sort → Increasing

Q) Sort Subarray from [s e]

Eg: ar[12] =

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

s m e
2 6 9

e-s+1

tmp[8] =

0	1	2	3	4	5	6	7
-1	2	3	4	6	7	9	11

$P_1 = s$, $P_2 = m+1$

$P_3 = 0$

Copy tmp[] → ar[]

ar[12] =

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

Pseudocode

```
void merge(int A[], int s, int m, int e){
    int tmp[e-s+1];  // indexes: [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]; p1++; p3++
        }
        else{
            tmp[p3] = A[p2]; p2++; p3++
        }
    }
    while(p1 <= m){ tmp[p3] = A[p1]; p1++; p3++ }
    while(p2 <= e){ tmp[p3] = A[p2]; p2++; p3++ }
    // tmp[] ---copy---> ar[]
}
```

// indexes: [0, 1, e-s] [s, s+1, e]

// Can work on your own way

```
i = 0; i <= e-s; i++){
    A[i+s] = tmp[i]
}
```

TC: O(N)

SC: O(N)

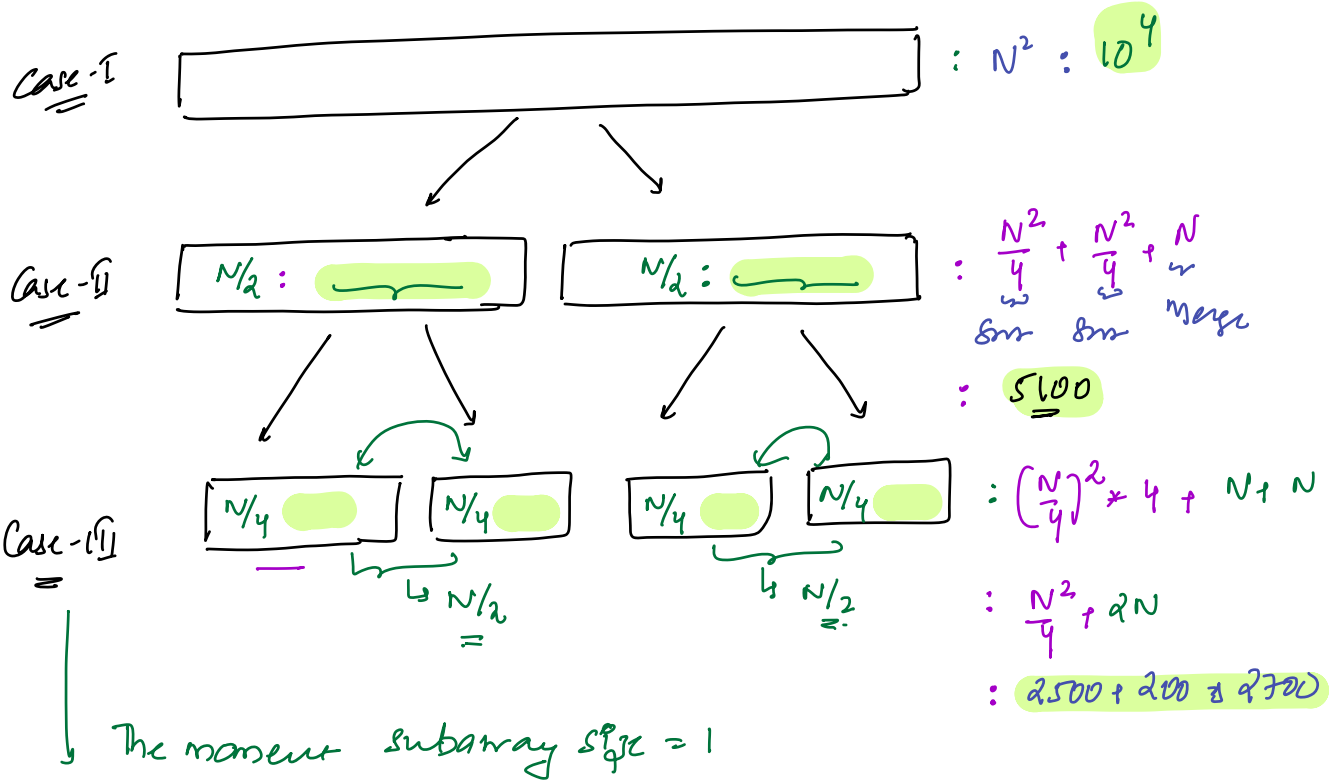
TODO: Merge 2 sorted subarrays TC: O(N) ?

& SC: O(1) Not possible ?

{10:30pm break?} → 7min in true

508) Given N array elements, sort them. } BS/SS →

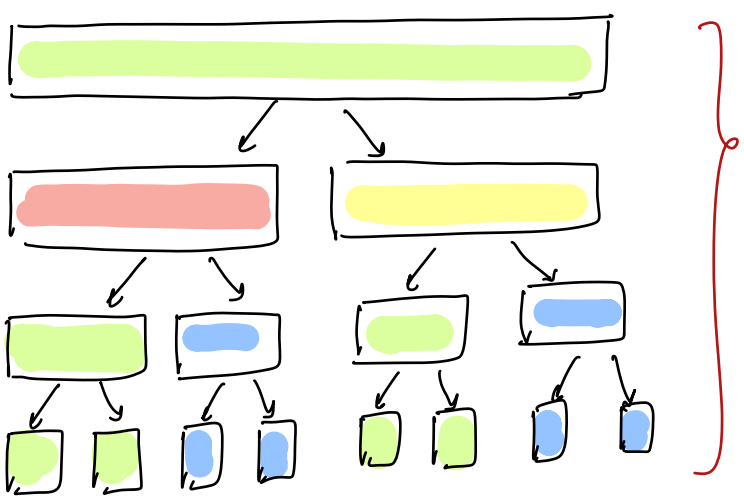
↳ N ≤ 100



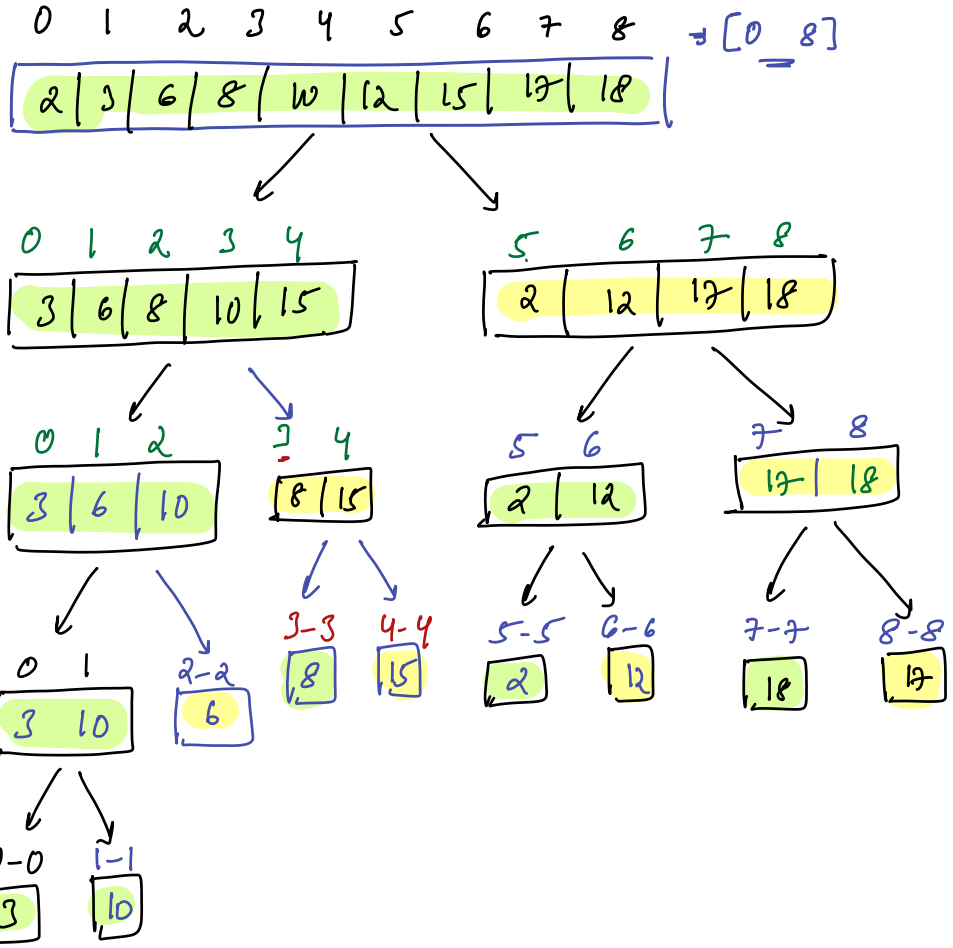
↳ The moment subarray size = 1

we cannot divide.

idea: MergeSort



Idea:



// code: main( )

Ass: given an subarray from [s e], array have to be sorted [s e]

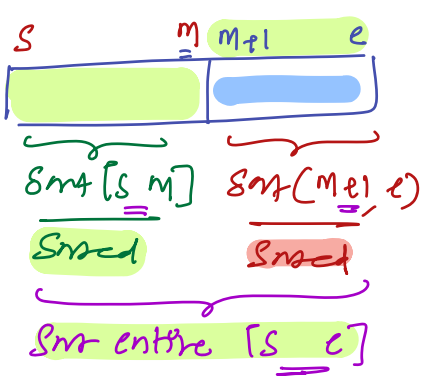
```
void MergeSort ( int A[], int s, int e) {
    if( s == e) { return }

    m = (s+e)/2
    MergeSort( A, s, m)
    MergeSort ( A, m+1, e)
    merge( A, s, m, e)
}
```

Main logic

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

$T(1) = 1$



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

    int tmp[e-s+1];  // index: [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]; p1++; p3++;
        }
        else{
            tmp[p3]=A[p2]; p2++; p3++;
        }
    }

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

    // tmp[] --copy--> ar[]
    // index: [0, 1, e-s]   [s, s+1, e]   // Can write in your own way
    for(i=0; i<=e-s; i++){
        A[i+s]=tmp[i];
    }
}
```

Comparator: → {Reverse}

{override: having our own order}

We can get our required order

TC: O(N)

SC: O(N)

TODO: Merge 2 sorted subarrays TC: O(N) ?

& SC: O(1) Not possible

{10:30pm break?} → 7min in mca

// code:

```
main( ) {
    // given ar[N] we need to sort
    MergeSort(ar, 0, N-1)
}
```

(s, e)

Recurrence Relation

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

$T(1) = 1$

TODO k

TC: SC:

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

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

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

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

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

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

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

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

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

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

// generalized expression :

$2^k T(N/2^k) + kN \quad T(1) = 1$

$\Rightarrow N/2^k = 1$

$N = 2^k$

$k = \log_2^N$

$a^{\log_a^N} = N$ formula

$\rightarrow 2^{\log_2^N} T\left(N/2^{\log_2^N}\right) + \left(\log_2^N\right)(N)$

$= (N)\,T(1) + N\log N$

$= O(N\log N)$ SC: $O(\log N + N)$

SC: $O(\log N)$ → recursion call stack

SC: $O(N)$ → space complexity for merging

Sorting : 1 → { Merge Sort }

Sorting : 2 → { All problems }

Sorting : 3 → { Algorithms }