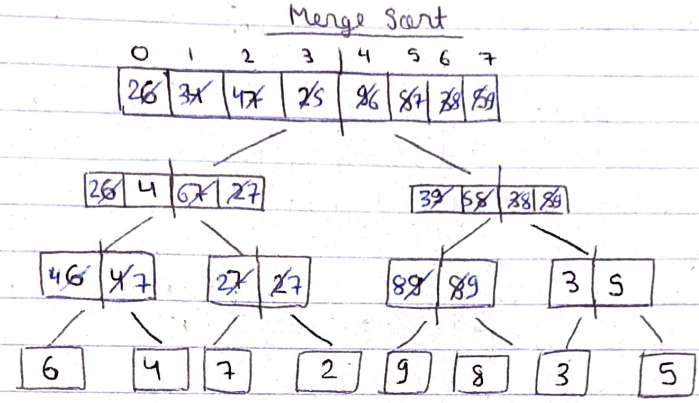
Day - 82

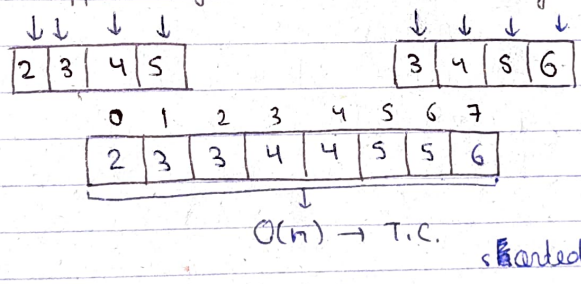
Merge Sort



⇒ In Merge Sort, we divide array upto single element, then we merge it.

=⇒ It is also known as Divide & merge.

⇒ Now, suppose, you have two arrays —



sorted

⇒ Now, we have merge these two arrays in one sorted array then, we can do by taking two pointers.

=⇒ First pointer on first array & second pointer on second array.

=⇒ And the smaller value comes in the new array & pointer value increases by 1.

Date: / /
Page:

⇒ This same approach, we will use at the time of merging in merge sort.

⇒ For dividing, we will first find mid i.e.

start → 0 1 2 3 4 5 6 7 ← end

2	3	4	5	6	7	8	9

So, mid = $\frac{0+7}{2}$ = 3

0	1	2	3
2	3	4	5

4	5	6	7
6	7	8	9

So, left (start, mid)
Right (mid+1, end)

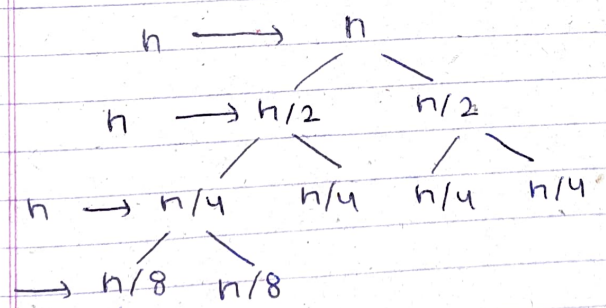
Code

```
void mergeSort( int arr[], int start, int end){
        if(start == end)
            return;
        int mid= start + (end- start) /2;
        mergeSort (arr, start ,mid);
        mergeSort (arr, mid+1, end);
        merge (arr, start, mid, end);
}

void merge ( int arr[] , int start, int mid, int end){
        vector <int> temp =(end- start +1) ;
        int left= start, right = mid +1, index=0;
        while( left <= mid && right <= end){
            if (arr[ left ] <= arr[ right ]){
                temp[ index ] = arr[ left ];
                index++, left++;
            }
```

```
            else{
                temp[index] = arr[right];
                index++, right++;
                }
        }
        while(left <= mid){
            temp[index++] = arr[left++];
        }
        while(right <= end){
            temp[index++] = arr[right++];
        }
        index = 0;
        while( start <= end){
            arr[start] = temp[index];
            start++, index++;
        }
}
```

* Time & Space Complexity:



n → ① n+n+n+n+... → O(n log₂n) → T.C.

n steps

↳ Height → log n

O(n) → S.C.