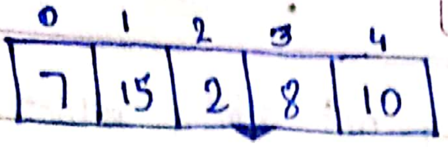
Merge Sort Algorithm:

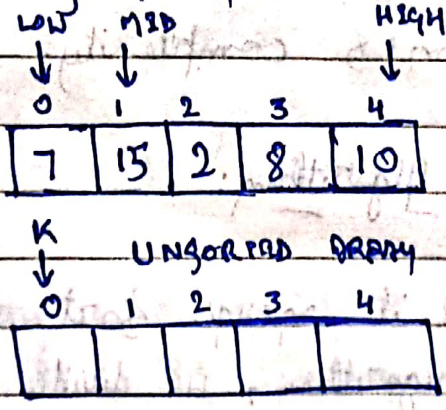
-> Why we call it merge sort algorithm?
-> In this algorithm, we divide the arrays into subarrays and subarrays into more subarrays until the size of each subarray become 1. Since arrays with a single element are always considered sorted, this is where we merge. Merging two sorted subarrays creates another sorted subarray.

-> Suppose there is an array A of 5 elements and contains two sorted subarrays of length 2 & 3 in itself.



UNSORTED ARRAY A

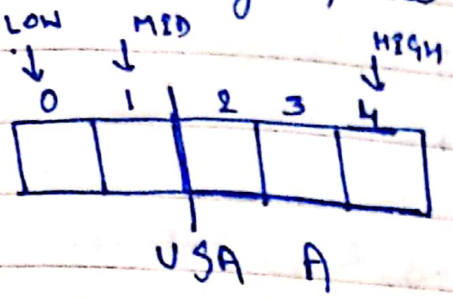
→ To merge both the sorted subarrays and produce a sorted array of length 5, we will create an auxillary array B of size 5. Now the process would be more or less the same, and the only change we would need to make is to consider the first index of the first subarray as low and the last index of the second subarray as high. And mark the index prior to the first index of the second subarray as mid.

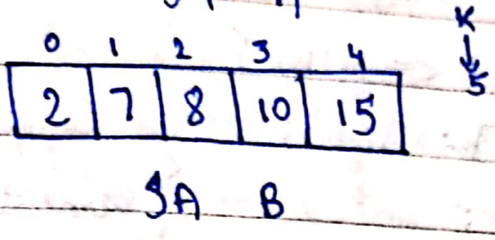


UNSORTED ARRAY A

SORTED ARRAY B

→ Previously we had index variables i, j & K initialised with 0 of their respective arrays, But here, i gets initialised with low, j gets initialised with mid+1, and K gets initialized with low only. And similar to what we did earlier, i runs from low to mid, j runs from mid+1 to high, and until & unless they both get all their elements merged, we continue filling in array B.





-> After all the elements get filled in array B, we revert back to our original array A and fill the sorted elements again from low to high, making our array merge-sorted.

-> PSEUDO-CODE :

```
VOID MERGE(INT A[], MID, LOW, HIGH)
{
    INT I, J, K, B[HIGH + 1];
    I = LOW;
    J = MID + 1;
    K = LOW;
    WHILE (i <= MID && j <= HIGH) {
        IF(A[i] < A[j]) {
            B[K] = A[i];
            i++;
            K++;
        }
        ELSE{
```

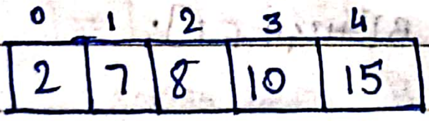
Page No.
Date

```
            B[K] = A[j];
            j++;
            K++;
        }
    }

    WHILE (i <= MID) {
        B[K] = A[i];
        K++;
        j++;
    }
```
Copying all remaining elements from A to C.

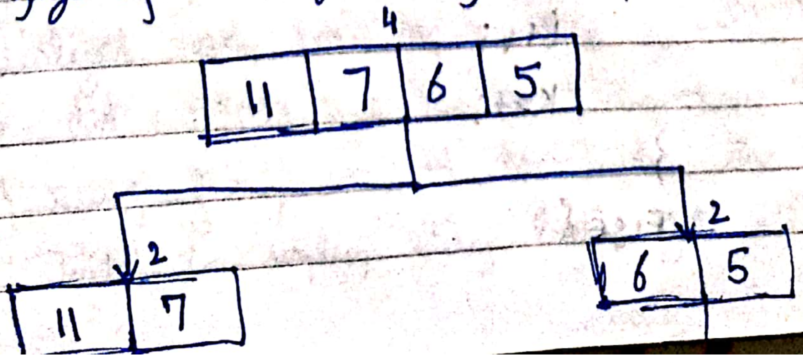
```
    WHILE (j <= HIGH) {
        B[K] = A[j];
        K++;
        j++;
    }
```
Copying all remaining.

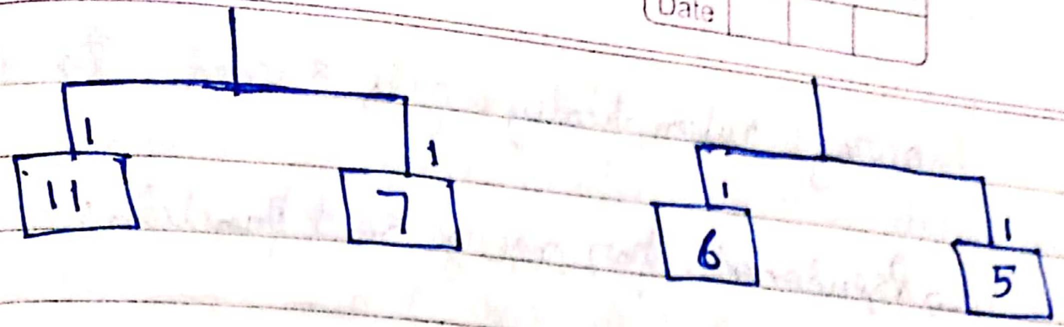
```
    FOR (INT I = LOW; i <= HIGH; i++)
        A[i] = B[i];
```
copying elements back to A from B



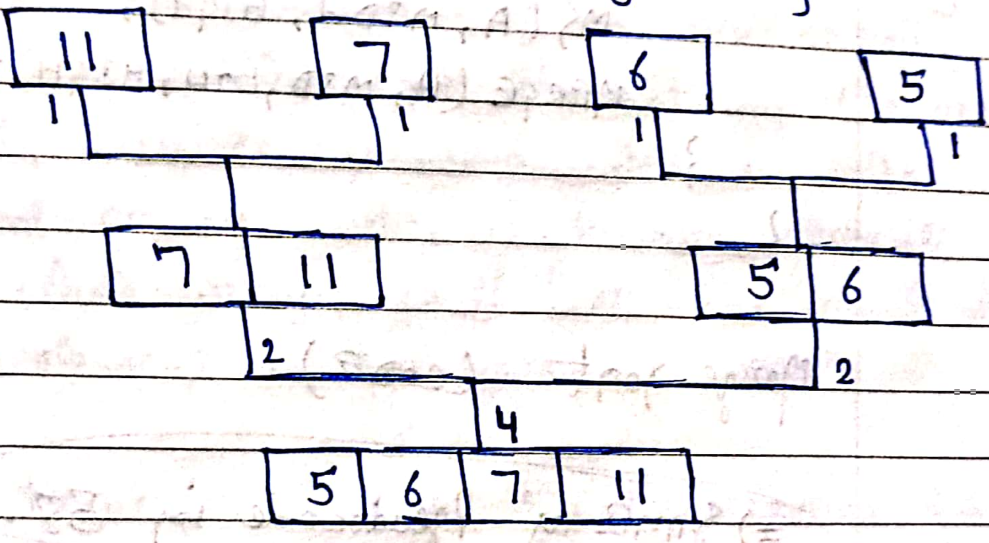
MERGE SORTED ARRAY A.

→ Whenever you receive an unsorted ~~a~~ array, you break the array into fragments till the size of each subarray becomes 1. ~~Let this~~





→ So, we divided the array until there are all subarrays of just length ~~5~~ 1. Since any array/ subarray of length 1 is always sorted, we just ~~to~~ need to merge all these singly-sized subarrays into a single entity.



→ And this is how our array got merge sorted.
To ~~achieve~~ achieve this divided merging and sorting, we create a recursive function merge sort and pass our array and the low and high index into it. This function divides the array into two parts; ~~one from~~ one from low to mid and another from mid+1 to high.
Then, it recursively calls itself passing these divided subarrays. And the resultant subarrays are sorted. ~~It~~ In the next step, it just merges them. And that's it. Our

Page No.
Date

array automatically gets sorted. Bs

→ Pseudocode for merge sort function:

```
VOID MS (A[], LOW, HIGH){
    INT MID;
    IF(LOW <HIGH){
        MID = (LOW + HIGH)/2;
        MS (A, LOW, MID);
        MS (A, MID+1, HIGH);
        MERGE (A, MID, LOW, HIGH);
    }
}
```