③ Merge Sort.

i) → Divide & Conquer Algorithm.

ii) O(nlogn) time & O(n) Auxil space.

iii) Stable Algorithm.

iv) Best for Lists, but outperformed by quicksort in case of arrays.

→ Eg: Merge 2 Sorted Arrays

→ Naive method → create a new array, copy elem of both array in it & sort it.

→ 2 pointer method. O(m+n) time.

→ Algo of 2 pointer.

```
Void 2pointMethod (int a[], int b[])
{ int i=0; j=0;          m → length of a
                         n → length of b.
  while (i<m && j<n)
  { if (a[i] <= b[j])
    { print (a[i]);
      i++;
    }
    else { print (b[j]);
           j++;
         }
  }
  while (i<m)
  { print (a[i]); i++;
  }
  while (j<n)
  { print (b[j]); j++;
  }
}
```

V) Merge fn of Merge sort

From previous problem :- We can derive merge fn.

```
Void merge (int a[], int low, int mid, int high)
{
    int n1 = mid - low + 1, n2 = high - mid;
    int left[n1], right[n2];

    for(int i=0; i<n1; i++) { left[i] = a[low + i]; }
    for(int i=0; i<n2; i++) { right[i] = a[mid + i + 1]; }

    int i=0, j=0, k=0;
    while (i<n1 && j<n2)
    { if(left[i] <= right[i]) { a[k] = left[i];
        i++; k++; }
      else { a[k] = right[i];
        k++; j++; }
    }
    while (i<n1) {a[k] = left[i]; i++; k++; }
    while (i<n2) {a[k] = right[j]; j++; k++; }
}
```

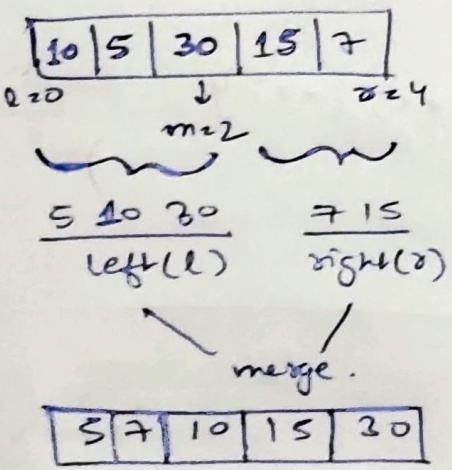
Like our last ques but low, mid & high given

low to mid → arr1 i.e left

mid to high → arr2 i.e right

vi) Merge sorting Algo

```
void MergeSort (int arr[], int l, int r)
{   if (r>l) {  → (check if atleast 2 elem)
      int m = l + (r-l)/2  → (same as (r+l)/2, but our formula avoides overflow.)
      mergeSort (arr, l, m);
      mergeSort (arr, m+1, r);
      merge (arr, l, m, r);  → (in next page).
    }
}
```



Example(s).

O(n) work at each level.
×
O(logn) level
↳ O(nlogn).