merge sort is of complexity O(nlogn),, in all of its scenarios ( avg, best,worst), it is also stable but requires O(n) space in terms of memory

**package** mergesort;

**class** mergesort

{

**void** merge(**int** arr[], **int** l, **int** m, **int** r)

{

// Find sizes of two subarrays to be merged

**int** n1 = m - l + 1;

**int** n2 = r - m;

/\* Create temp arrays \*/

**int** L[] = **new** **int** [n1];

**int** R[] = **new** **int** [n2];

/\*Copy data to temp arrays\*/

**for** (**int** i=0; i<n1; ++i)

L[i] = arr[l + i];

**for** (**int** j=0; j<n2; ++j)

R[j] = arr[m + 1+ j];

/\* Merge the temp arrays \*/

// Initial indexes of first and second subarrays

**int** i = 0, j = 0;

// Initial index of merged subarry array

**int** k = l;

**while** (i < n1 && j < n2)

{

**if** (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

**else**

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy remaining elements of L[] if any \*/

**while** (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy remaining elements of R[] if any \*/

**while** (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

**void** sort(**int** arr[], **int** l, **int** r)

{

**if** (l < r)

{

// Find the middle point

**int** m = (l+r)/2;

// Sort first and second halves // bottleneck if duplicates exist

sort(arr, l, m);

sort(arr , m+1, r);

// Merge the sorted halves

merge(arr, l, m, r);

}

}

**static** **void** printArray(**int** arr[])

{

**int** n = arr.length;

**for** (**int** i=0; i<n; ++i)

System.***out***.print(arr[i] + " ");

System.***out***.println();

}

**public** **static** **void** main(String args[])

{

**int** arr[] = {10, 13, 11, 9, 8, 14, 20};

System.***out***.println("Given Array");

*printArray*(arr);

mergesort ob = **new** mergesort();

ob.sort(arr, 0, arr.length-1);

System.***out***.println("\nSorted array");

*printArray*(arr);

}

}