**DAA ASSIGNMENT-1**

****

**SUBMITTED TO:**

**PROF. SUBHASH TATALE**

**VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE**

**COMPUTER ENGINEERING DEPARTMENT**

**BY:**

**NAME: ABHISHEK MORE**

**G.R No.: 21810033**

**ROLL NO.: 323036**

**CLASS: T.Y COMP**

**BATCH: COMP C2**

**Assignment-1**

**Aim:**

Implement Merge Sort Algorithm using Divide and Conquer technique.

**Objective:**

Our objective is to learn the implementation of Divide and Conquer technique by performing Merge Sort algorithm to sort the given array of elements.

**What is Merge Sort?**

Merge Sort is one of the sorting algorithms which is implemented using Divide & Conquer technique. The idea behind Merge Sort is to divide the given array of elements into ‘n’ sub-arrays (where, ‘n’ is number of elements in the original array). This is where the Dividing is done i.e. divide strategy is applied. Then, after that again the sub-arrays are sorted and merged together to get the sorted array of elements.

**Time Complexity:**

Time complexity of Merge Sort in all three cases i.e. worst, average and best is O(nLogn) as merge sort in each step divide the array in two halves (for each recursive call that takes ‘Logn’ time) and again the merging takes linear time (i.e. that takes ‘n’ time).

**Source code:**

**package** assignment1\_MergeSort;

**import** java.util.Scanner;

**class** MergeSortAlgorithm

{

**void** merge(**int** array[], **int** left, **int** mid, **int** right)

{

**int** sizeLSubArr = (mid-left)+1;

**int** sizeRSubArr = right-mid;

**int** lSubArr[] = **new** **int**[sizeLSubArr];

**int** rSubArr[] = **new** **int**[sizeRSubArr];

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

{

lSubArr[i] = array[left+i];

}

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

{

rSubArr[j] = array[(mid+1) + j];

}

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

**while** (i<sizeLSubArr && j<sizeRSubArr)

{

**if** (lSubArr[i] <= rSubArr[j])

{

array[k] = lSubArr[i];

i++;

}

**else**

{

array[k] = rSubArr[j];

j++;

}

k++;

}

**while** (i < sizeLSubArr)

{

array[k] = lSubArr[i];

i++;

k++;

}

**while** (j < sizeRSubArr)

{

array[k] = rSubArr[j];

j++;

k++;

}

}

**void** sort(**int** array[], **int** left, **int** right)

{

**if** (left < right) {

**int** mid = (left+right) / 2;

sort(array, left, mid);

sort(array, mid + 1, right);

merge(array, left, mid, right);

}

}

**void** printArray(**int** array[])

{

**int** n = array.length;

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

{

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

}

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

}

@SuppressWarnings({ "null", "resource" })

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

{

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter size of array to be sorted: ");

**int** n=0;

n = sc.nextInt();

**int** array[] = **new** **int**[n];

System.***out***.println("\nEnter array elements: ");

**for**(**int** i=0; i<array.length; i++)

{

array[i] = sc.nextInt();

}

MergeSortAlgorithm obj = **new** MergeSortAlgorithm();

System.***out***.println("\nYour unsorted array is: ");

obj.printArray(array);

obj.sort(array, 0, array.length - 1);

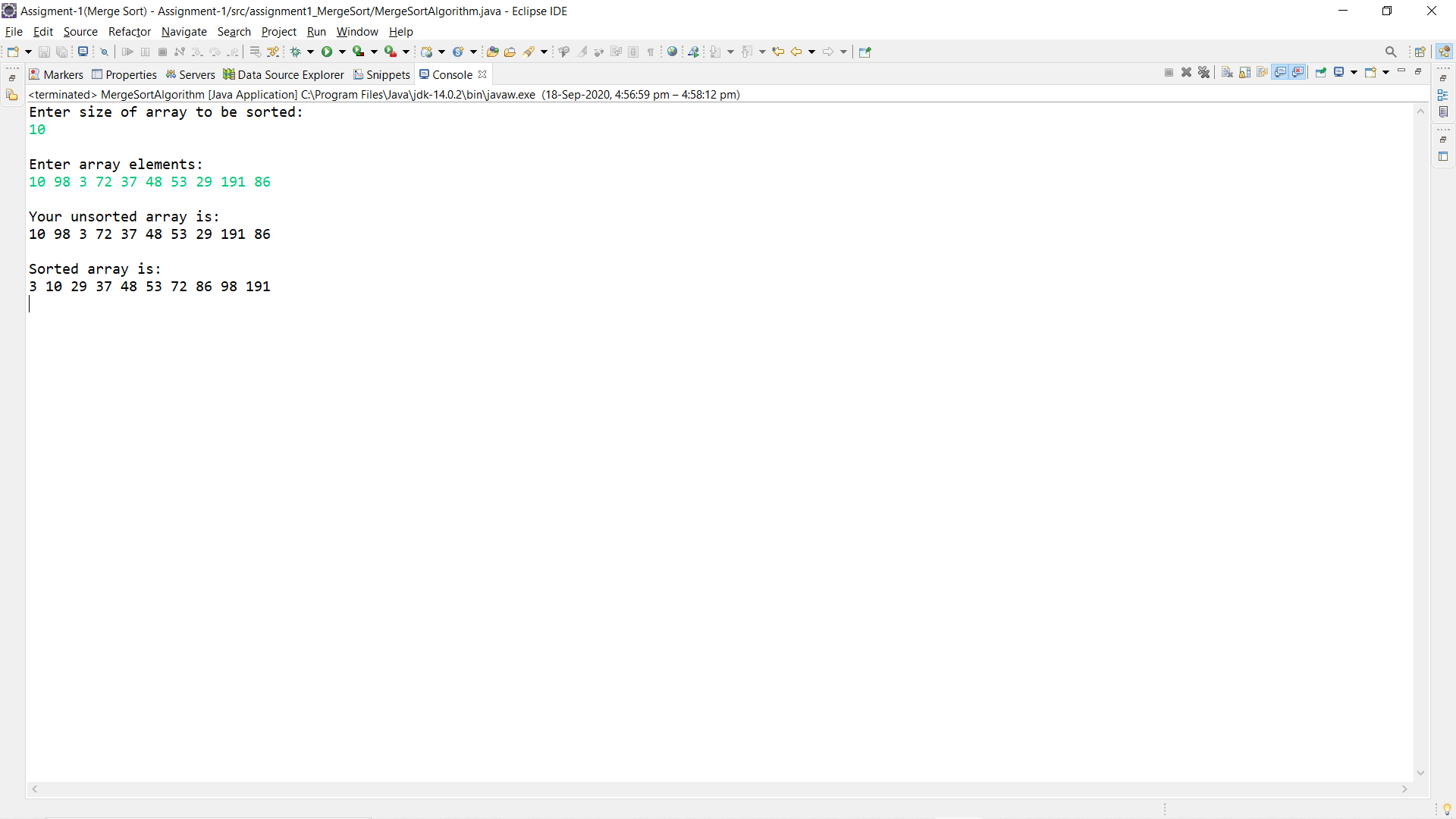
System.***out***.println("\nSorted array is:");

obj.printArray(array);

}

}

**Output:**

****

**Conclusion:**

Learnt how Divide and Conquer strategy optimizes the solution by successfully implementing the Merge Sort algorithm.