Government College of Engineering, Jalgaon (An Autonomous Institute of Government of Maharashtra)		
Name :	**Semester :** VI	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25	
Course Teacher : Mr. Vinit Kakde	**Subject :** CO356U DAA Lab	
Date of Performance :	**Date of Completion :**	

Practical no. 3

Aim : Implement algorithms using divide and conquer approach (mergesort) .

Mergesort :-

Merge sort is a sorting algorithm that follows the divide and conquere approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

How does Merge Sort work?

Merge sort is a popular sorting algorithm known for its efficiency and stability. It follows the **divide-and-conquer** approach to sort a given array of elements. Here's a step-by-step explanation of how merge sort works:

1. ***Divide:*** *Divide the list or array recursively into two halves until it can no more be divided.*
2. ***Conquer:*** *Each subarray is sorted individually using the merge sort algorithm.*
3. ***Merge:*** *The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.*

Complexity Analysis of Merge Sort

- **Time Complexity:**
  - **Best Case:** O(n log n), When the array is already sorted or nearly sorted.
  - **Average Case:** O(n log n), When the array is randomly ordered.
  - **Worst Case:** O(n log n), When the array is sorted in reverse order.
- **Auxiliary Space:** O(n), Additional space is required for the temporary array used during merging.

Mergesort Algorithm :-

Step 1: Create two pointers, one for each sorted half.

Step 2: Initialize an empty temporary array to hold the merged result.

Step 3: Compare the elements at the pointers of the two halves:

Copy the smaller element into the temporary array.

Move the pointer of the sublist with the smaller element forward.

Step 4: Repeat step 3 until one of the sublists is empty.

Step 5: Copy the remaining elements from the non-empty sublist to the temporary array.

Step 6: Copy the elements back from the temporary array to the original list.

Program:-

```
import java.util.Scanner;

public class MergeSort {

   static void merge(int arr[], int left, int mid, int right) {
      int n1 = mid - left + 1;
      int n2 = right - mid;

      int leftArr[] = new int[n1];
      int rightArr[] = new int[n2];

      for (int i = 0; i < n1; i++) {
         leftArr[i] = arr[left + i];
      }
      for (int j = 0; j < n2; j++) {
         rightArr[j] = arr[mid + 1 + j];
      }

      int i = 0, j = 0, k = left;
      while (i < n1 && j < n2) {
         if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
         } else {
            arr[k] = rightArr[j];
            j++;
         }
         k++;
      }

      while (i < n1) {
         arr[k] = leftArr[i];
         i++;
         k++;
      }

      while (j < n2) {
         arr[k] = rightArr[j];
         j++;
         k++;
      }
```

```
    }

    static void mergeSort(int arr[], int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    static void printArray(int arr[]) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("\nGiven array is: ");
        printArray(arr);

        mergeSort(arr, 0, n - 1);

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

        scanner.close();
    }
}
```

Output :-

```
DELL@DESKTOP-9QQ0ROL MINGW64 /f/Java
$ javac MergeSort.java

DELL@DESKTOP-9QQ0ROL MINGW64 /f/Java
$ java MergeSort
Enter the number of elements: 5
Enter the elements:
4
8
9
2
3

Given array is:
4 8 9 2 3

Sorted array is:
2 3 4 8 9
```

Question :-

1. What is the time complexity of Merge Sort in the worst case?
Answer:
The time complexity of Merge Sort in the worst case is O(n log n).

2. Is Merge Sort a stable sorting algorithm?
Answer:
Yes, Merge Sort is a stable algorithm because it preserves the relative order of equal elements.

3. What is the space complexity of Merge Sort?
Answer:
The space complexity of Merge Sort is O(n) because it requires additional space for the temporary arrays during the merge process.

4. Can Merge Sort be implemented iteratively?
Answer:
Yes, Merge Sort can be implemented iteratively, but it is most commonly implemented recursively.

5. How does Merge Sort divide the array for sorting?
Answer:
Merge Sort recursively divides the array into two halves until each sub-array contains one element.

Name and sign of teacher
Mr. Vinit Kakde