Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Electronics & Telecommunication Engineering

Data Structures & Algorithms Lab (DJ19ECSBL1)

Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

Experiment no: 6 Date:10-11-2022

Aim: Write a program to implement Merge sort

Programming Language: C/C++/Java

Theory:

Merge sort is the sorting technique that follows the divide and conquer approach.It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. **merge()** function will be required to perform the merging.

The sub-lists are divided again and again into halves using **mergesort()** function until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs are merged into the four-element lists, and so on until we get the sorted list.

Merge sort algorithm steps

function **mergeSort(int A[], int l, int r)** sorts entire array A[] with left and right ends as input parameters.

- **Divide part:** We calculate the mid-index i.e. mid = l + (r - l)/2
- **Conquer part 1:** We call the same function with mid as the right end and recursively sort the left part of size n/2, i.e., mergeSort(A, l, mid).
- **Conquer part 2:** We call the same function with mid + 1 as the left end and recursively sort the right part of size n/2, i.e., mergeSort(A, mid + 1, r).
- **Combine part:** Inside the function mergeSort(), we use function merge(A, l, mid, r) to merge both smaller sorted halves into a final sorted array.
- **Base case:** If we find **l == r** during recursive calls, then sub-array has one element left, which is trivially sorted. So recursion will not go further and return from here. In other words, the sub-array of size one is the smallest version of sorting problem for which recursion directly returns the solution.

Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

Code:

```
#include <iostream>
using namespace std;

void merge(int arr[], int l, int mid, int r)
{
    int n1 = mid - l + 1;
    int n2 = r - mid;

    int a[n1];
    int b[n2];

    for (int i = 0; i < n1; i++)
    {
        a[i] = arr[l + i];
    }
    for (int i = 0; i < n2; i++)
    {
        b[i] = arr[mid + 1 + i];
    }

    int i = 0;
    int j = 0;
    int k = l;
    while (i < n1 && j < n2)
    {
        if (a[i] < b[j])
        {
            arr[k] = a[i];
            k++;
            i++;
        }
        else
        {
            arr[k] = b[j];
            k++;
            j++;
        }
    }
    while (i < n1)
    {
        arr[k] = a[i];
        k++;
```

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

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int mid = (l + r) / 2;
        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);

        merge(arr, l, mid, r);
    }
}

int main()
{
    int arr[] = {4, 1, 3, 9, 7};
    mergeSort(arr, 0, 4);
    for (int i = 0; i < 5; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

Output:

```
PS G:\programming\College_DSA> cd "g:\programming\College_DSA\sorting\" ; if ($?) { g++ mergeSort.cpp -o mergeSort } ; if ($?) { .\mergeSort }
Input Array: 4 1 3 9 7
Sorted Array: 1 3 4 7 9
PS G:\programming\College_DSA\sorting>
```

Result and Conclusion:

1) Implemented merge sort in c++.

2) Time complexity:
   a. Worst= nlogn
   b. Average= nlogn
   c. Best= nlogn

3) The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.