# Lab 7

# Implement Merge sort Algorithm and specify Time complexity of merge sort.

# Code:

#include <iostream>

#include <vector>

using namespace std;

void merge(vector<int>& arr, int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)

        L[i] = arr[left + i];

    for (int j = 0; j < n2; j++)

        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

void mergeSort(vector<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);

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    vector<int> arr(n);

    cout << "Enter " << n << " elements: ";

    for (int i = 0; i < n; i++)

        cin >> arr[i];

    mergeSort(arr, 0, n - 1);

    cout << "Sorted array is: ";

    for (int i = 0; i < n; i++)

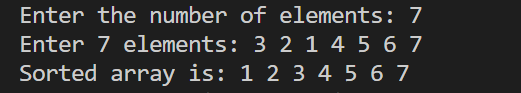
        cout << arr[i] << " ";

    cout << endl;

    return 0;

}

# Results



# Time Complexity Analysis:

* Divide Step: In the divide step, the array is recursively divided into halves until each sub-array contains only one element. This step has a time complexity of O(log n) because the array is halved at each recursion level until it reaches single elements.
* Conquer Step (Merge): In the conquer step, the divided sub-arrays are merged back together. Merging two sub-arrays of size n/2 each takes O(n) time. Since there are log n levels of recursion (due to the divide step), each level requires O(n) time to merge. Therefore, the total time complexity of the conquer step is O (n log n).
* Combining both the divide and conquer steps, we get a total time complexity of O(n log n) for Merge Sort. This time complexity is consistent regardless of the initial order of elements in the array, making Merge Sort a stable and efficient sorting algorithm for large datasets.