Assignment 3

Implement Merge Sort using divide and conquer technique.

Tasks:

1. **Recursive divide and merge steps**

   Step 1: If it is only one element in the list, consider it already sorted, so return.

   Step 2: Divide the unsorted array recursively into sub-array until it can no longer be divided, i.e., as long as it has more than one element.

   Step 3: Merge the smaller arrays into new array in sorted order, i.e. no sub-array left.

   Algorithm

```
BEGIN
   l1 ← low
   l2 ← mid + 1
   i ← low

   WHILE l1 ≤ mid AND l2 ≤ high DO
      IF arr[l1] ≤ arr[l2] THEN
         b[i] ← arr[l1]
         l1 ← l1 + 1
      ELSE
         b[i] ← arr[l2]
         l2 ← l2 + 1
      END IF
      i ← i + 1
   END WHILE

   WHILE l1 ≤ mid DO
      b[i] ← arr[l1]
      l1 ← l1 + 1
      i ← i + 1
   END WHILE
   WHILE l2 ≤ high DO
      b[i] ← arr[l2]
      l2 ← l2 + 1
      i ← i + 1
   END WHILE

   FOR i ← low TO high DO
      arr[i] ← b[i]
   END FOR
END
```

CODE

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

int b[10];

void merging(int arr[], int low, int mid, int high){
  int l1, l2, i;
  for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
    if(arr[l1] <= arr[l2])
      b[i] = arr[l1++];
    else
      b[i] = arr[l2++];
  }
  while(l1 <= mid)
    b[i++] = arr[l1++];
  while(l2 <= high)
    b[i++] = arr[l2++];
  for(i = low; i <= high; i++)
    arr[i] = b[i];
}
void sort(int arr[], int low, int high){
  int mid;
  if(low < high) {
    mid = (low + high) / 2;
    sort(arr, low, mid);
    sort(arr, mid + 1, high);
    merging(arr, low, mid, high);
  } else {
    return;
  }
}
int main(){
  int n;
  cout<<"Enter total number of elements :";
  cin>>n;
  cout<<"\nEnter "<<n<<" array elements : ";
  int arr[n];
  for(int i = 0; i < n; i++){
     cin>>arr[i];
  }
  cout << "\nArray before sorting\n";
  for(int i = 0; i < n; i++)
    cout<<arr[i]<<" ";
  sort(arr, 0, n - 1);
  cout<< "\nArray after sorting\n";
  for(int i = 0; i < n; i++)
    cout<<arr[i]<<" ";
```

```
}
```

2. Trace recursion tree

[12 16 11 8 23 5 17 10 21 4]

Level 1 :

Left : [12 16 11 8 23]

Right : [5 17 10 21 4]

Level 2 :

[12 16] [11 8 23] [5 17] [10 21 4]

Level 3 :

[12] [16] [11] [8 23] [5] [17] [10] [21 4]

Level 4 :

[12] [16] [11] [8] [23] [5] [17] [10] [21] [4]

Merge Sort

Step 1 : Merge Single Element

[12] + [16] → [12 16]

[8] + [23] → [8 23]

[5] + [17] → [5 17]

[21] + [4] → [4 21]

Step 2 : Merge next level

[11] + [8 23] → [8 11 23]

[10] + [4 21] → [4 10 21]

Step 3 : Merge larger subarrays

[12 16] + [8 11 23] → [8 11 12 16 23]

[5 17] + [4 10 21] → [4 5 10 17 21]

Step 4 : Merge Final

[8 11 12 16 23] + [4 5 10 17 21]→ [4 5 8 10 11 12 16 17 21 23]

3. Analyze time and space complexity

Time Complexity

Best Case	Worst Case	Average Case
O(n log n)	O(n log n)	O(n log n)

Space Complexity

Auxiliary Case	Recursion Case
O(n)	O(n log n)