Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Sorting Algorithms Using Divide and Conquer Approach	
Experiment No: 04	**Date:13/09/2025**	**Enrollment No: 92301733054**

Merge Sort

Code :-

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

void Print_Array(vector<int> Array) {
    for (int i = 0; i < Array.size(); i++) {
        cout << Array[i] << " ";
    }
    cout << endl;
}




void Merge(vector<int>& Array, int low, int mid, int high) {
    int lower_bound = mid - low + 1;
    int upper_bound = high - mid;
    vector<int> Left_Array(lower_bound);
    vector<int> Right_Array(upper_bound);
        for (int i = 0; i < lower_bound; i++) {
        Left_Array[i] = Array[low + i];
    }
    for (int i = 0; i < upper_bound; i++) {
        Right_Array[i] = Array[mid + 1 + i];
    }
    int i = 0;
    int j = 0;
    int k = low;
    while (i < lower_bound && j < upper_bound) {
        if (Left_Array[i] <= Right_Array[j]) {
            Array[k] = Left_Array[i];
            i++;
        }
        else {
            Array[k] = Right_Array[j];
            j++;
        }
        k++;
    }
    while (i < lower_bound) {
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Sorting Algorithms Using Divide and Conquer Approach	
Experiment No: 04	**Date:13/09/2025**	**Enrollment No: 92301733054**

```
            Array[k] = Left_Array[i];
            i++;
            k++;
        }
        while (j < upper_bound) {
            Array[k] = Right_Array[j];
            j++;
            k++;
        }
 }


void Merge_Sort(vector<int>& Array, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        Merge_Sort(Array, left, mid);
        Merge_Sort(Array, mid + 1, right);
        Merge(Array, left, mid, right);
    }
}

int main() {
    vector<int> Array = { 12, 45, 57, 78, 89, 62, 7, 49, 21, 23 };
    int size = Array.size();
    cout << "Array Before Sorting :- " << endl;
    Print_Array(Array);
    Merge_Sort(Array, 0, size - 1);
    cout << "Array After Sorting :- " << endl;
    Print_Array(Array);
    return 0;
}
```

Output :-

```
Array Before Sorting :-
12 45 57 78 89 62 7 49 21 23
Array After Sorting :-
7 12 21 23 45 49 57 62 78 89
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing the Sorting Algorithms Using Divide and Conquer Approach	
Experiment No: 04	**Date:13/09/2025**	**Enrollment No: 92301733054**

Quick Sort

Code :-

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

void Swap(int& x, int& y) {
   int temp = x;
   x = y;
   y = temp;
}

void Print_Array(vector<int> Array) {
   for (int i = 0; i < Array.size(); i++) {
      cout << Array[i] << " ";
   }
   cout << endl;
}

int Partion(vector<int>& Array, int left, int right) {
   int pivot = Array[left];
   while (true) {
      while (left < right && Array[left] < pivot) {
         left++;
      }
      while (left < right && Array[right] > pivot) {
         right--;
      }
      if (left >= right) {
         break;
      }

      Swap(Array[left], Array[right]);
   }


      Swap(Array[right], pivot);
   return right;
}

void Quick_Sort(vector<int>& Array, int left, int right) {
   if (left < right) {
      int Pivot_Index = Partion(Array, left, right);
      Quick_Sort(Array, left, Pivot_Index - 1);
```

```
        Quick_Sort(Array, Pivot_Index + 1, right);
    }
}

int main() {
    vector<int> Array = { 12, 45, 57, 78, 89, 62, 7, 49, 21, 23 };
    int size = Array.size();
    cout << "Array Before Sorting :- " << endl;
    Print_Array(Array);
    Quick_Sort(Array, 0, size - 1);
    cout << "Array After Sorting :- " << endl;
    Print_Array(Array);
    return 0;
}
```

Output :-

```
Array Before Sorting :-
12 45 57 78 89 62 7 49 21 23
Array After Sorting :-
7 12 21 23 45 49 57 62 78 89
```

Conclusion: We learnt in this experiment that both Merge Sort and Quick Sort use the Divide and Conquer technique to sort data efficiently. Merge Sort gives consistent performance with extra memory, while Quick Sort is faster on average but depends on pivot selection.