Merge Sort

Merge Sort is a **Divide and Conquer** sorting algorithm.
It works by:

1. **Dividing** the array into two halves,
2. **Recursively sorting** each half, and then
3. **Merging** the two sorted halves back together.

Think of it like splitting a messy pile of papers into smaller piles, sorting each one, and then combining them neatly.

🔹 **Algorithmic Steps (pseudocode)**

Algorithm: MergeSort(A, left, right)

Input: Array A with indices from left to right

Output: Sorted array A in ascending order

1. if left < right then
2. mid ← (left + right) / 2
3. MergeSort(A, left, mid) // Sort left half
4. MergeSort(A, mid + 1, right) // Sort right half
5. Merge(A, left, mid, right) // Merge the two sorted halves
6. end if

Merge procedure (for step 5):

Algorithm: Merge(A, left, mid, right)

1. Create two temporary arrays:

    L[] ← A[left..mid]

    R[] ← A[mid+1..right]

2. i ← 0, j ← 0, k ← left

3. while i < length(L) and j < length(R) do

4. if L[i] ≤ R[j] then

5. A[k] ← L[i]

6. i ← i + 1

7. else

8. A[k] ← R[j]

9. j ← j + 1

10. k ← k + 1

11. end while

12. Copy any remaining elements of L[] (if any)

13. Copy any remaining elements of R[] (if any)

🔹 Implementation in C++

```
#include <iostream>

using namespace std;



// Function to merge two halves

void merge(int arr[], int left, int mid, int right) {

    int n1 = mid  left + 1;

    int n2 = right  mid;



    // Temporary arrays

    int L[n1], R[n2];



    // Copy data to temp arrays

    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];



    // Merge the temp arrays back into arr[left..right]

    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++;
}

// Copy remaining elements of L[], if any
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}

// Copy remaining elements of R[], if any
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
```

```
    }
}

// Recursive Merge Sort
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right  left) / 2;

        mergeSort(arr, left, mid);      // Sort first half
        mergeSort(arr, mid + 1, right); // Sort second half

        merge(arr, left, mid, right);   // Merge the sorted halves
    }
}

// Driver code
int main() {
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
    int n = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, n  1);
```

```
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

🔹 Time & Space Complexity

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

Space Complexity: O(n) - uses temporary arrays during merging.
Stable? ✅ Yes
Adaptive? ❌ No - still divides even if already sorted.