**MERGE SORT   
Merge Sort** is a **Divide and Conquer** sorting algorithm that divides the given array into smaller subarrays, sorts them, and then merges them back together in sorted order.It was invented by **John von Neumann in 1945**.

**Concept – Divide and Conquer**

Merge Sort works in **three main steps:**

1. **Divide:**  
   Split the array into two halves recursively until each subarray has one element.  
   (A single element is already sorted.)
2. **Conquer:**  
   Recursively sort both halves.
3. **Combine (Merge):**  
   Merge the two sorted halves into a single sorted array.

**Working Process (Step-by-Step)**

Array: [38, 27, 43, 3, 9, 82, 10]

**Step 1: Divide**

Divide array into halves recursively:

[38, 27, 43, 3, 9, 82, 10]

/ \

[38, 27, 43, 3] [9, 82, 10]

Further divide until one element remains:

[38, 27, 43, 3] → [38, 27] and [43, 3]

[9, 82, 10] → [9, 82] and [10]

Keep dividing:

[38, 27] → [38] and [27]

[43, 3] → [43] and [3]

[9, 82] → [9] and [82]

**Step 2: Conquer (Sort subarrays)**

Now start merging **sorted** subarrays:

Merge [38] and [27] → [27, 38]

Merge [43] and [3] → [3, 43]

Merge [9] and [82] → [9, 82]

**Step 3: Combine (Merge)**

Merge sorted halves:

Merge [27, 38] and [3, 43] → [3, 27, 38, 43]

Merge [9, 82] and [10] → [9, 10, 82]

Finally merge both:

Merge [3, 27, 38, 43] and [9, 10, 82] → [3, 9, 10, 27, 38, 43, 82]

**Algorithm (Recursive)**

MERGE\_SORT(arr, left, right):

1. If left < right:

a. mid = (left + right) / 2

b. MERGE\_SORT(arr, left, mid)

c. MERGE\_SORT(arr, mid + 1, right)

d. MERGE(arr, left, mid, right)

**MERGE(arr, left, mid, right):**

1. Create two temporary arrays L[] and R[]

where L[] = arr[left...mid]

R[] = arr[mid+1...right]

2. Initialize i, j, k = left

3. While both arrays have elements:

if L[i] <= R[j]:

arr[k] = L[i]; i++

else:

arr[k] = R[j]; j++

k++

4. Copy remaining elements (if any)

while i < size(L): arr[k++] = L[i++]

while j < size(R): arr[k++] = R[j++]

**Program**

#include <stdio.h>

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

int n1 = mid - left + 1;

int n2 = right - mid;

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];

int i = 0, j = 0, k = left;

// Merge the temp arrays back into arr

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[]

while(i < n1) {

arr[k] = L[i];

i++; k++;

}

// Copy remaining elements of R[]

while(j < n2) {

arr[k] = R[j];

j++; k++;

}

}

void mergeSort(int arr[], int left, int right) {

if(left < right) {

int mid = (left + right) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

int main() {

int arr[] = {38, 27, 43, 3, 9, 82, 10};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original array: ");

for(int i = 0; i < n; i++)

printf("%d ", arr[i]);

mergeSort(arr, 0, n - 1);

printf("\nSorted array: ");

for(int i = 0; i < n; i++)

printf("%d ", arr[i]);

return 0;

}

**7. Output**

Original array: 38 27 43 3 9 82 10

Sorted array: 3 9 10 27 38 43 82

**Time and Space Complexity**

| **Case** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| **Best Case** | O(n log n) | Even if already sorted, still divides and merges |
| **Average Case** | O(n log n) | Always divides array into halves |
| **Worst Case** | O(n log n) | Same for all cases |
| **Space Complexity** | O(n) | Needs temporary arrays for merging |
| **Stable?** | Yes | Equal elements maintain order |