Day 7: Merge Two Sorted Arrays

Gati Goyal

"C teaches you to think clearly – it makes you focus on the essence of a problem."
— Kernighan and Ritchie

---

1 Introduction

Merging two sorted arrays into a single sorted array is a fundamental operation in computer science, commonly used in divide-and-conquer algorithms like merge sort. This problem can be solved efficiently using the two-pointer technique, which leverages the sorted nature of the input arrays to minimize comparisons.

2 Problem Statement

Problem: Merge two sorted arrays into a single sorted array without sorting explicitly.
Hint: Use two pointers to traverse both arrays, adding the smaller element to the result.
Edge Case: Handle cases where one array is empty.

3 Algorithm

3.1 Steps to Solve the Problem

1. Initialize:
   - Two pointers, `i` and `j`, to traverse the input arrays.
   - An index `k` for the merged array.
2. Traverse both arrays:
   - Compare the elements pointed to by `i` and `j`.
   - Add the smaller element to the merged array and move the corresponding pointer.
3. Append remaining elements:
   - If any elements remain in one of the arrays, add them directly to the merged array.

4 Code

```
#include <stdio.h>

// Function to merge two sorted arrays
void mergeSortedArrays(int arr1[], int n1, int arr2[], int n2, int merged[]
    int i = 0, j = 0, k = 0;

    // Traverse both arrays
    while (i < n1 && j < n2) {
        if (arr1[i] <= arr2[j]) {
            merged[k++] = arr1[i++];
        } else {
            merged[k++] = arr2[j++];
        }
    }

    // Add remaining elements from arr1
    while (i < n1) {
        merged[k++] = arr1[i++];
    }

    // Add remaining elements from arr2
    while (j < n2) {
        merged[k++] = arr2[j++];
    }
}

int main() {
    int n1, n2;

    printf("Enter the size of the first array: ");
    scanf("%d", &n1);
    int arr1[n1];
    printf("Enter the elements of the first sorted array:\n");
    for (int i = 0; i < n1; i++) {
        scanf("%d", &arr1[i]);
    }

    printf("Enter the size of the second array: ");
    scanf("%d", &n2);
    int arr2[n2];
    printf("Enter the elements of the second sorted array:\n");
    for (int i = 0; i < n2; i++) {
        scanf("%d", &arr2[i]);
    }

    int merged[n1 + n2];
```

```
    mergeSortedArrays(arr1, n1, arr2, n2, merged);

    printf("Merged Sorted Array:\n");
    for (int i = 0; i < n1 + n2; i++) {
        printf("%d ", merged[i]);
    }
    printf("\n");

    return 0;
}
```

5 Step-by-Step Explanation

1. Initialize three indices: `i` and `j` for input arrays, `k` for the merged array.
2. Traverse both arrays:
   - Compare `arr1[i]` and `arr2[j]`.
   - Append the smaller value to `merged` and increment the corresponding index.
3. Append any remaining elements in `arr1` or `arr2`.

6 Complexity Analysis

6.1 Brute-Force Approach

- Combine the arrays and sort them, which has a time complexity of $O((m+n)\log(m+n))$.

6.2 Optimized Approach

- Using the two-pointer technique ensures a linear time complexity of $O(m+n)$.
- Space complexity is $O(m+n)$ for the merged array.

7 Examples and Edge Cases

Input Arrays	Merged Array	Remarks
{1, 3, 5}, {2, 4, 6}	{1, 2, 3, 4, 5, 6}	Both arrays sorted.
{1, 2, 3}, {}	{1, 2, 3}	Second array empty.
{}, {4, 5, 6}	{4, 5, 6}	First array empty.
{1, 1, 1}, {1, 1}	{1, 1, 1, 1, 1}	Duplicate elements.

8 Conclusion

The two-pointer technique efficiently merges two sorted arrays in $O(m+n)$ time without requiring extra sorting. This approach is optimal for merging in-place during algorithms like merge sort.

9 Output

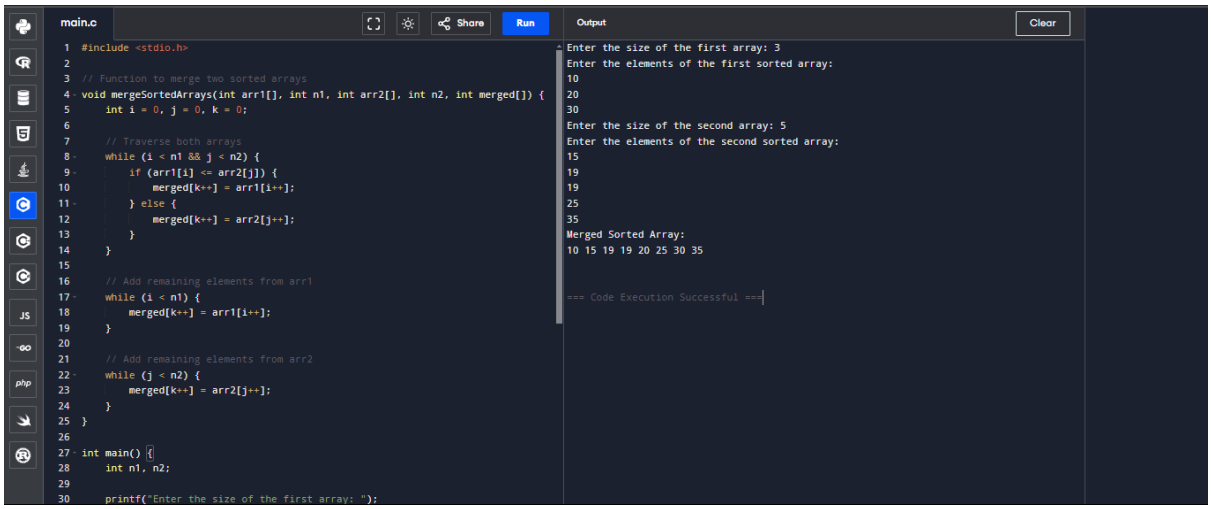

Figure 1: Output in online compiler