Day 18: Merge Sort

Gati Goyal

”First, solve the problem. Then, write the code.”
— John Johnson

1 Introduction

Merge Sort is a divide-and-conquer algorithm that splits the input array into halves, recursively sorts each half, and then merges the two sorted halves back together. It is known for its efficiency and stability.

2 Problem Statement

Problem: Sort an array of integers using the merge sort algorithm. **Hint:** Divide the array into smaller subarrays, sort them recursively, and merge them in sorted order. **Edge Case:** Handle empty arrays and arrays with a single element.

3 Algorithm

1. Divide the array into two halves until each subarray contains a single element.
2. Recursively sort each half.
3. Merge the two sorted halves into a single sorted array.
4. Continue this process until the entire array is sorted.

4 Code

```c
#include <stdio.h>

// Function to merge two subarrays
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++;
    }
}

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

        // Recursively sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

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

```c

int main() {
    int n;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    mergeSort(arr, 0, n - 1);

    printf("Sorted array after merge sort: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

5 Complexity Analysis

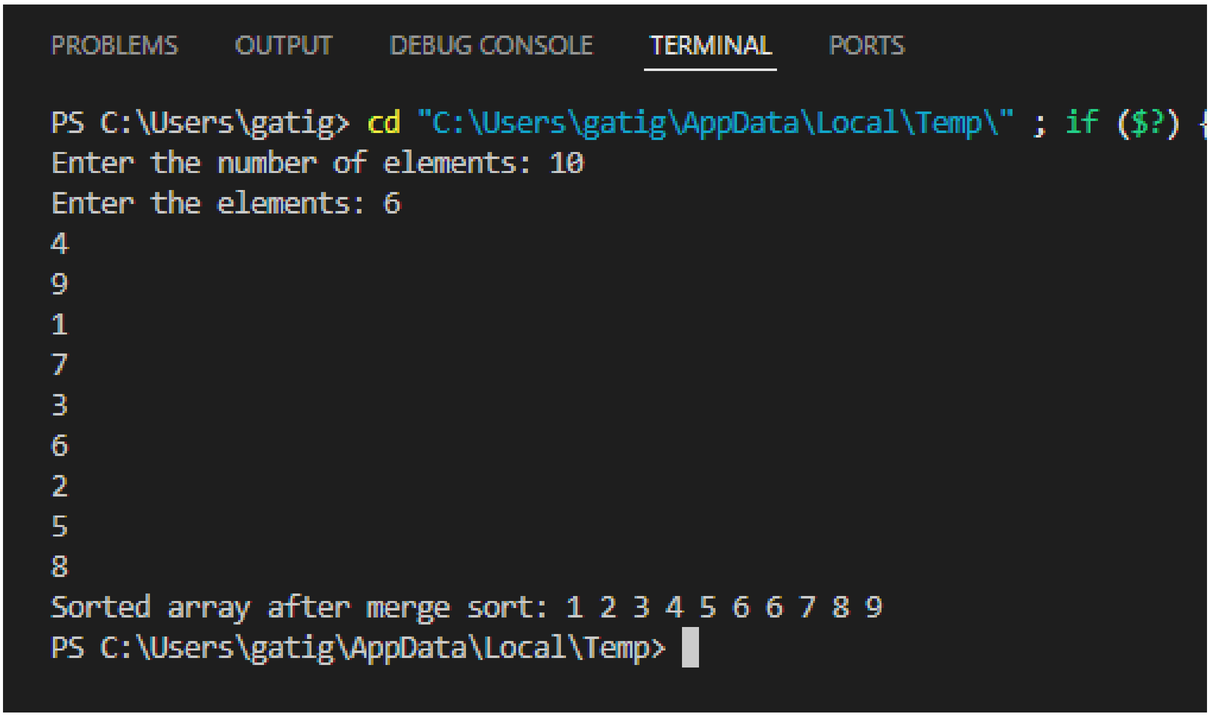
- **Time Complexity:**
  - Best Case: $O(n \log n)$.
  - Average Case: $O(n \log n)$.
  - Worst Case: $O(n \log n)$.
- **Space Complexity:** $O(n)$ (additional memory for temporary arrays).

6 Examples and Edge Cases

Input Array	Output Array	Steps Required
{64, 34, 25, 12, 22, 11, 90}	{11, 12, 22, 25, 34, 64, 90}	3 Splits, 6 Merges
{1, 2, 3, 4, 5}	{1, 2, 3, 4, 5}	3 Splits, 4 Merges
{5, 4, 3, 2, 1}	{1, 2, 3, 4, 5}	3 Splits, 6 Merges

7 Conclusion

Merge Sort is an efficient and stable sorting algorithm that performs well on large datasets due to its $O(n \log n)$ time complexity. However, it requires additional memory for temporary arrays, making it less suitable for memory-constrained systems.



```
PS C:\Users\gatig> cd "C:\Users\gatig\AppData\Local\Temp\" ; if ($?) {
Enter the number of elements: 10
Enter the elements: 6
4
9
1
7
3
6
2
5
8
Sorted array after merge sort: 1 2 3 4 5 6 6 7 8 9
PS C:\Users\gatig\AppData\Local\Temp>
```

Figure 1: Program Output Screenshot