Merge Sort Algorithm

Merge Sort is a divide-and-conquer algorithm that divides the array into halves, sorts them, and then merges the sorted halves.

Steps:

1. **Divide**: Divide the array into two halves.
2. **Conquer**: Recursively sort the two halves.
3. **Combine**: Merge the two sorted halves to produce the sorted array.

Implementation:

```
#include <stdio.h>
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    int i, j, k;
    int l, r, m;
    int temp[100];

    // Merge sort implementation
    for (int curr_size = 1; curr_size <= n - 1; curr_size = 2 * curr_size)
    {
        for (int left_start = 0; left_start < n - 1; left_start += 2 * curr_size)
        {
            l = left_start;
            m = left_start + curr_size - 1;
            r = ((left_start + 2 * curr_size - 1) < (n - 1)) ? (left_start + 2 *
curr_size - 1) : (n - 1);

            // Merge the subarrays
            i = l;
            j = m + 1;
            k = l;
            while (i <= m && j <= r) {
                if (arr[i] <= arr[j]) {
                    temp[k] = arr[i];
                    i++;
                } else {
                    temp[k] = arr[j];
                    j++;
                }
                k++;
            }

            // Copy the remaining elements of left subarray, if any
            while (i <= m) {
                temp[k] = arr[i];
                i++;
                k++;
            }

            // Copy the remaining elements of right subarray, if any
            while (j <= r) {
                temp[k] = arr[j];
                j++;
                k++;
            }

            // Copy the sorted subarray into Original array
            for (i = l; i <= r; i++) {
                arr[i] = temp[i];
            }
        }
    }

    // Print the sorted array
    printf("Sorted array: \n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Time Complexity:

- **Best Case**: O(n log n)
- **Average Case**: O(n log n)
- **Worst Case**: O(n log n)

Space Complexity:

- **O(n)**: Requires additional space for temporary arrays used during merging.

Merge Sort is highly efficient for large datasets due to its consistent O(n log n) time complexity. However, it does require additional memory for the temporary arrays.