MERGE SORT – ALGORITHM IMPLEMENTATION AND PERFORMANCE ANALYSIS

Objective:

The objective of this task is to implement the **Merge Sort algorithm** using C++ and analyze its performance by measuring execution time for different input sizes. The experiment also aims to verify the theoretical time and space complexity of the algorithm

Algorithm Description:

Merge Sort is a **divide-and-conquer** sorting algorithm. It works by recursively dividing the array into two halves until each subarray contains only one element. Then, these subarrays are merged back together in a sorted order to form the final sorted array.

Steps involved:

1. Divide the array into two halves.
2. Recursively apply Merge Sort on each half.
3. Merge the two sorted halves into a single sorted array.

Time and Space Complexity:

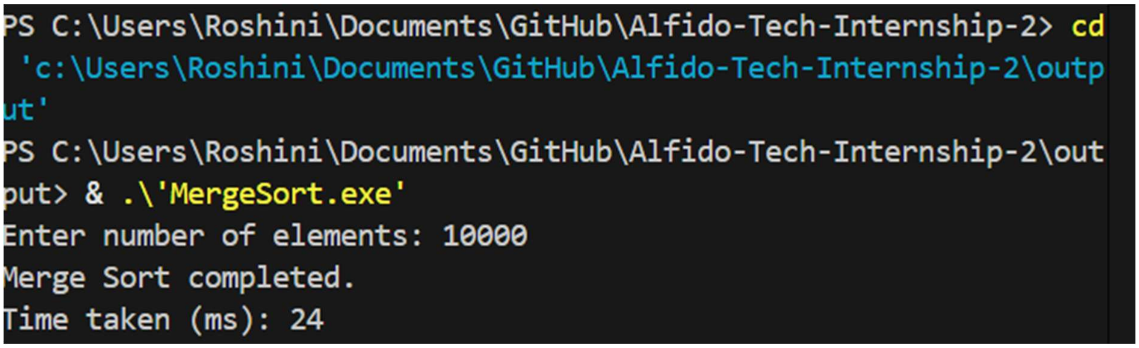
Case	Time Complexity
Best Case	O(n log n)
Average Case	O(n log n)
Worst Case	O(n log n)

Space Complexity:

- O(n) (additional memory is required for merging)

Where `n` is the number of elements in the array.

Sample Input and Output:



```
PS C:\Users\Roshini\Documents\GitHub\Alfido-Tech-Internship-2> cd
 'c:\Users\Roshini\Documents\GitHub\Alfido-Tech-Internship-2\outp
ut'
PS C:\Users\Roshini\Documents\GitHub\Alfido-Tech-Internship-2\out
put> & .\'MergeSort.exe'
Enter number of elements: 10000
Merge Sort completed.
Time taken (ms): 24
```

Runtime Table:

Input Size	Time Taken (ms)
100	0.02
1,000	0.30
10,000	4.5

Compile and Run Commands:

Compile (Using g++)

```
g++ mergesort.cpp -o mergesort
```

Run

```
./mergesort
```

Conclusion:

The Merge Sort algorithm was successfully implemented in C++. The measured runtime confirms that Merge Sort consistently performs in O(n log n) time for different input sizes, making it an efficient sorting algorithm for large datasets.