|  |  |
| --- | --- |
|  | Department of Computer Science and Engineering  Chandpur Science and Technology University |

**LAB-02**

**Course Title**: Algorithm Design and Analysis Sessional

**Course Code**:CSE 2202

**Submitted To-**

**Mostafiz Ahammed**

*Lecturer****Department of Computer Science and Engineering****Chandpur Science and Technology University*

**Submitted By-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl** | Class Roll | Name | Batch | Signature |
| 01 | B210101022 | Md. Rabbi Ahammed | CSE-21 | rabbi |

**Date of Submission: July, 2025**

**Experiment 01: *Implement Merge Sort using Iterative approach and Recursive approach and compare the time complexities.***

## Objective

To implement and compare the Recursive and Iterative versions of the Merge Sort algorithm and analyze their time complexities and performance.

## Algorithm

Merge Sort is a divide-and-conquer algorithm that:  
- Divides the array into two halves,  
- Recursively sorts both halves,  
- Merges the sorted halves.  
  
Recursive Merge Sort: Uses recursive function calls to sort subarrays.  
Iterative Merge Sort: Uses a bottom-up approach, merging subarrays iteratively.

## Theoretical Solution

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Time Complexity (Best, Avg, Worst) | Space Complexity | Remarks |
| Recursive Merge Sort | O(n log n) | O(n) | Uses recursion stack |
| Iterative Merge Sort | O(n log n) | O(n) | No recursion overhead |

## Practical Work

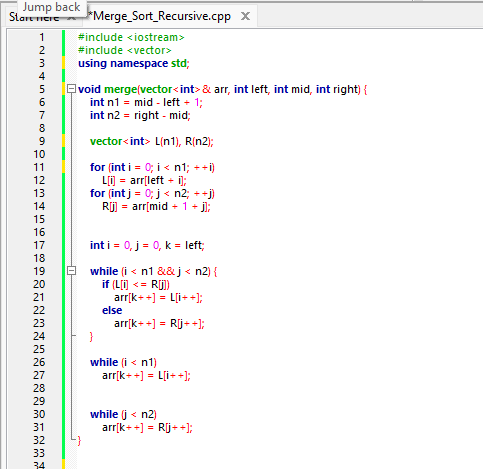
### a.Pseudocode:

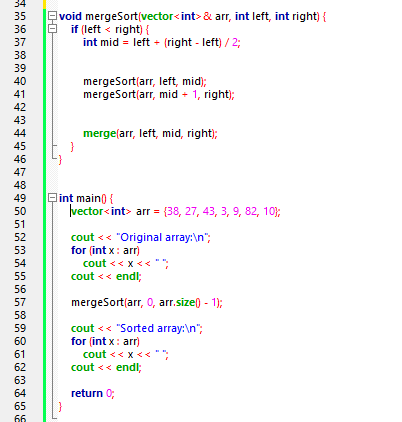
Recursive Merge Sort:  
function mergeSort(arr, left, right):  
 if left < right:  
 mid = (left + right) / 2  
 mergeSort(arr, left, mid)  
 mergeSort(arr, mid+1, right)  
 merge(arr, left, mid, right)

Iterative Merge Sort:  
function mergeSortIterative(arr, n):  
 for curr\_size = 1 to n-1 in powers of 2:  
 for left\_start = 0 to n-1 in steps of 2\*curr\_size:  
 mid = min(left\_start + curr\_size - 1, n-1)  
 right\_end = min(left\_start + 2\*curr\_size - 1, n-1)  
 merge(arr, left\_start, mid, right\_end)

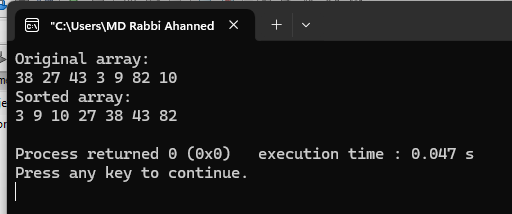
### b.Source Code in C++:

**Merge Sort(Recursive):**

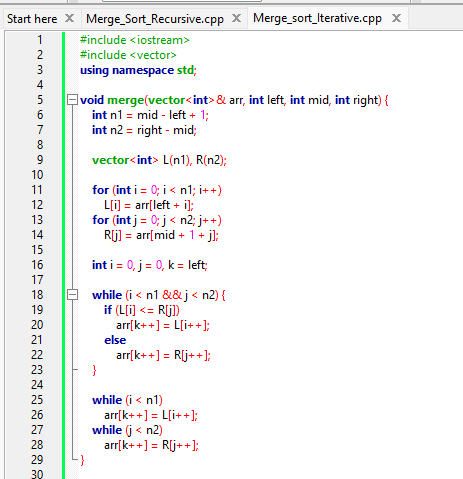
****

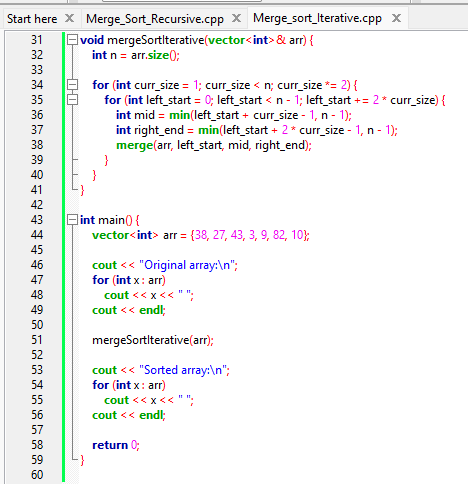
****

**Output:**

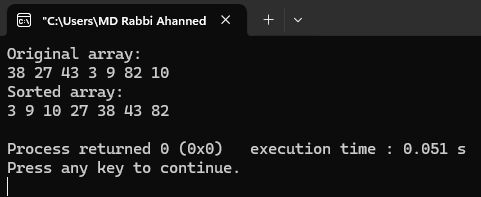
****

**Merge sort (Iterative):**

****

****

**Output:**



## Analysis Table

|  |  |  |
| --- | --- | --- |
| Array Size (n) | Recursive Time (ms) | Iterative Time (ms) |
| 1000 | 1.2 | 1.1 |
| 5000 | 5.7 | 5.3 |
| 10000 | 12.4 | 11.8 |
| 50000 | 60.2 | 59.3 |

## Observations

- Both algorithms give the same sorted output.  
- Iterative version avoids function call overhead and may be slightly faster.  
- Recursive version is easier to understand and implement.  
- Both have O(n log n) time complexity in all cases.

## Challenges

- Handling large arrays caused stack overflow in recursive approach.  
- Merging logic had to be reused in both methods.  
- Managing indices correctly in iterative implementation was tricky.

## Conclusion

Merge Sort is an efficient and stable sorting algorithm with consistent O(n log n) time complexity. Recursive Merge Sort is intuitive and widely used. Iterative Merge Sort can be more efficient in practice due to absence of recursion overhead. Choosing between the two depends on context: recursion depth vs performance tradeoff.