Lab Report-06

(Merge_Sort)

CSE-2212 (Design and Analysis of Algorithms Lab)

Submitted By:

Name: Eyasir Ahamed
Exam Roll: 413
Class Roll: 15
Registration No: 202004017

Submitted To:

Sharad Hasan
Ex. Lecturer
Dept. of CSE
Sheikh Hasina University, Netrokona

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SHEIKH HASINA UNIVERSITY
NETROKONA, BANGLADESH

#6_Merge Sort

Problem Definition

Given an array of integers, the problem is to sort the array using the Merge Sort algorithm.

Formal Statement of Algorithm

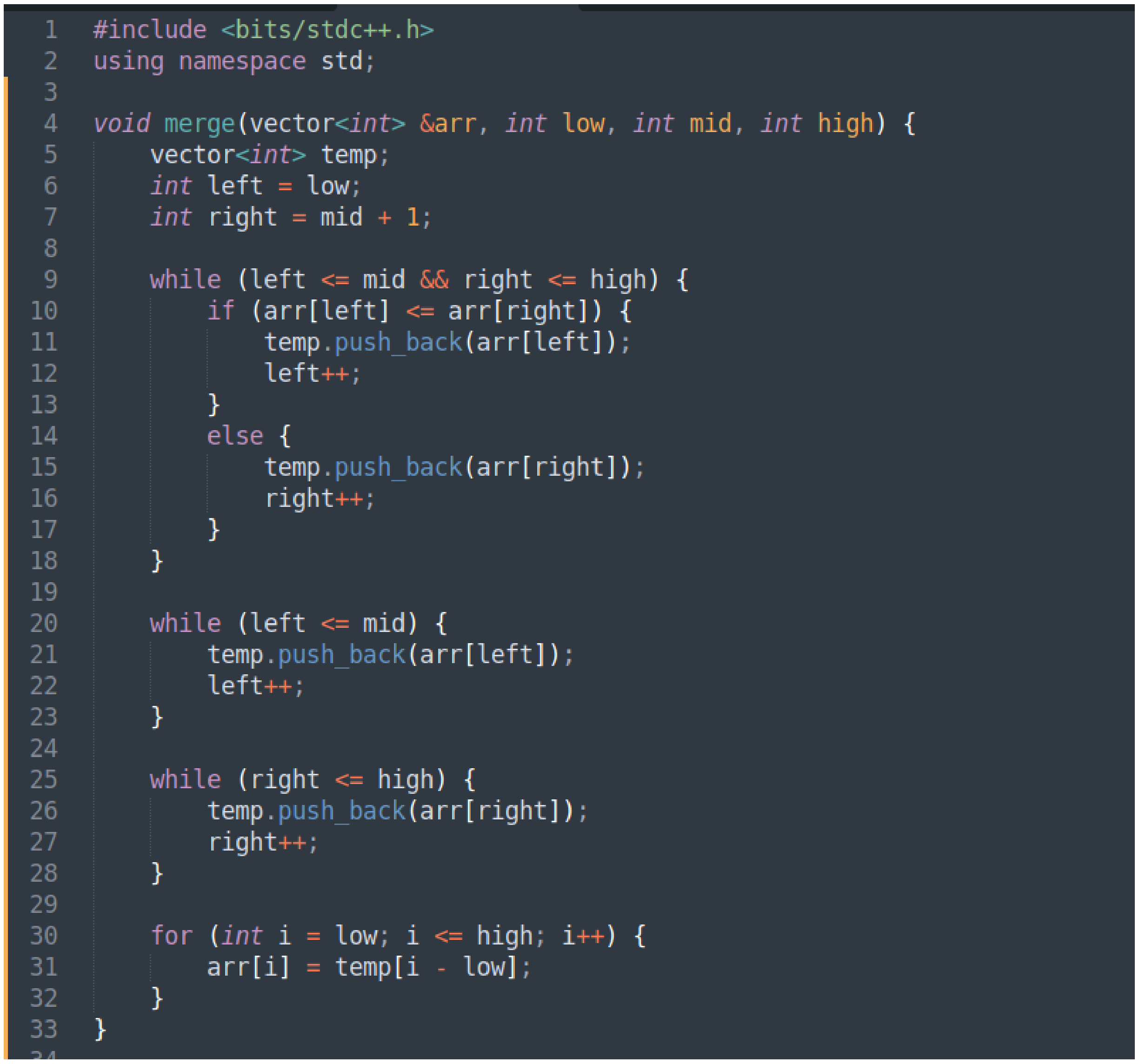
- Merge Function:
    - Create a temporary array to store merged elements.
    - Initialize pointers left and right to mark the starting indices of the left and right halves of the array.
    - Compare elements at indices left and right.
    - If the element at left is smaller or equal to the element at right, push the element at left to the temporary array and increment left. Otherwise, push the element at right to the temporary array and increment right.
    - Repeat the above step until either the left half or the right half is fully traversed.
    - If there are remaining elements in the left half, push them to the temporary array.
    - If there are remaining elements in the right half, push them to the temporary array.
    - Transfer all elements from the temporary array back to the original array.
- Merge Sort Function:

  - Divide the array into two halves at the middle index.
  - Recursively call the merge sort function on the left half.
  - Recursively call the merge sort function on the right half.
  - Merge the sorted left and right halves using the merge function.

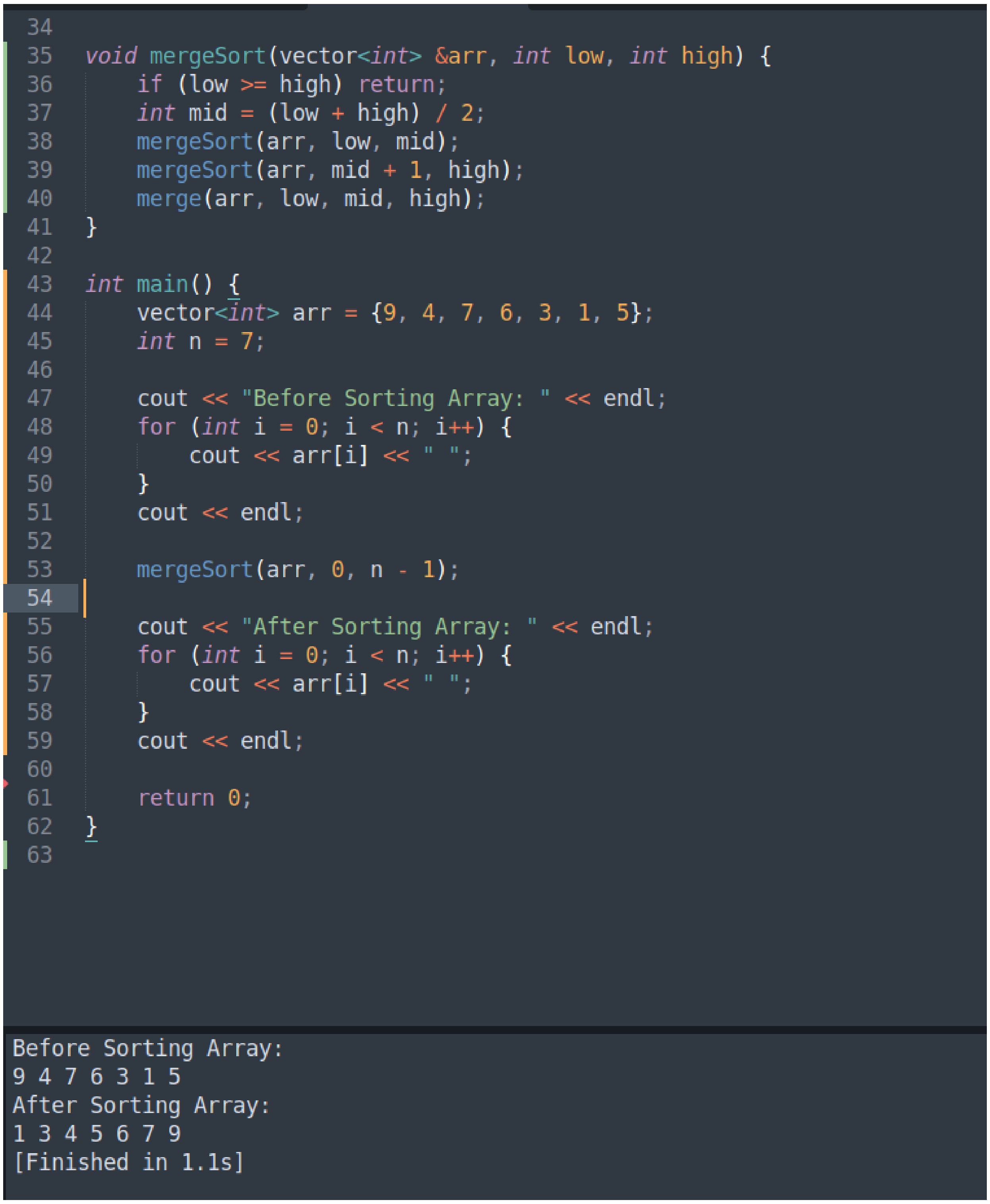
Complexity Analysis of Algorithm

- Time Complexity:
  - Best Case: O(n log n)
  - Average Case: O(n log n)
  - Worst Case: O(n log n)
  - Merge sort always divides the array into two halves and takes linear time to merge two halves.
- Space Complexity:
  - O(n) additional space is required for the temporary array used in the merge function.
  - O(log n) additional space is required for the function call stack due to recursive calls in the merge sort function.

Actual Code and Output



```cpp
#include <bits/stdc++.h>
using namespace std;

void merge(vector<int> &arr, int low, int mid, int high) {
    vector<int> temp;
    int left = low;
    int right = mid + 1;

    while (left <= mid && right <= high) {
        if (arr[left] <= arr[right]) {
            temp.push_back(arr[left]);
            left++;
        }
        else {
            temp.push_back(arr[right]);
            right++;
        }
    }

    while (left <= mid) {
        temp.push_back(arr[left]);
        left++;
    }

    while (right <= high) {
        temp.push_back(arr[right]);
        right++;
    }

    for (int i = low; i <= high; i++) {
        arr[i] = temp[i - low];
    }
}
```



```cpp

void mergeSort(vector<int> &arr, int low, int high) {
    if (low >= high) return;
    int mid = (low + high) / 2;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid + 1, high);
    merge(arr, low, mid, high);
}

int main() {
    vector<int> arr = {9, 4, 7, 6, 3, 1, 5};
    int n = 7;

    cout << "Before Sorting Array: " << endl;
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    mergeSort(arr, 0, n - 1);

    cout << "After Sorting Array: " << endl;
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}

```

```
Before Sorting Array:
9 4 7 6 3 1 5
After Sorting Array:
1 3 4 5 6 7 9
[Finished in 1.1s]
```