THE UNIVERSITY OF AZAD JAMMU AND KASHMIR, MUZAFFARABAD

Department of Software Engineering

Submitted to:	Engr. Sidra Rafique
Course Title:	Data Structure and Algorithm
Course Code:	CS-2101
Session:	2024-2028
Lab No:	06
Roll No:	2024-SE-15
Submitted By:	Shahzad Ahmed Awan
Submission date:	December 20, 2025

Contents

Table of Figures

Lab 06 – Sorting and Searching Algorithms Implementation

1. Objective

To implement Linear Search, Binary Search, Bubble Sort, and Merge Sort in C++ for searching and sorting arrays, demonstrating algorithm logic, array manipulation, and user interactivity.

2. Background

- **Linear Search:** Checks each element sequentially until the key is found.
- **Binary Search:** Efficient search on a **sorted array** using divide-and-conquer.
- **Bubble Sort:** Simple sorting by repeatedly swapping adjacent elements.
- **Merge Sort:** Efficient recursive divide-and-conquer sorting algorithm.

Advantages: Efficient searching and sorting; demonstrates algorithmic logic.

Disadvantages: Linear search and bubble sort are less efficient for large datasets.

3. Algorithms and Implementation

3.1 Linear Search

Algorithm:

1. Traverse the array from start to end.
2. Compare each element with the key.
3. Return index if found; otherwise, indicate not found.

Sample Code Screenshot:

```cpp
#include <iostream>
using namespace std;

// Linear Search Function with step-by-step simulation
int linearSearch(int arr[], int n, int key) {
    cout << "\nStarting Linear Search for " << key << "...\n\n";

    for (int i = 0; i < n; i++) {
        cout << "Checking index " << i << ": " << arr[i] << " ";
        if (arr[i] == key) {
            cout << "--> Found!\n";
            return i;  // element found
        } else {
            cout << "--> Not this one.\n";
        }
    }

    return -1; // element not found
}

int main() {
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;

    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    int key;
    cout << "Enter the element to search for: ";
    cin >> key;

    int result = linearSearch(arr, n, key);

    if (result != -1)
        cout << "\nResult: Element " << key << " found at index " << result << ".\n";
    else
        cout << "\nResult: Element " << key << " not found in the array.\n";

    return 0;
}
```

Figure 1: Code Implementation Linear Search

Sample Output Screenshot:

```
Enter the number of elements in the array: 4
Enter 4 elements: 23 45 78 98
Enter the element to search for: 2

Starting Linear Search for 2...

Checking index 0: 23 --> Not this one.
Checking index 1: 45 --> Not this one.
Checking index 2: 78 --> Not this one.
Checking index 3: 98 --> Not this one.

Result: Element 2 not found in the array.

--------------------------------
Process exited after 17.28 seconds with return value 0
Press any key to continue . . .
```

Figure 2: Output of the code of linear Search

3.2 Binary Search

Algorithm:

1. Sort array automatically if not sorted.
2. Compare key with middle element.
3. Narrow search to left or right half until key is found.

Sample Code Screenshot:

```cpp
#include <iostream>
using namespace std;

void sortArray(int arr[], int n) {
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n-1;
    while (low <= high) {
        int mid = low + (high - low)/2;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

Figure 4: Code Implementation of Binary Search -1

```cpp
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int original[n], arr[n];
    cout << "Enter elements: ()we will sort and search):";
    for (int i = 0; i < n; i++) {
        cin >> original[i];
        arr[i] = original[i]; // copy for sorting
    }

    // Sort the copy
    sortArray(arr, n);

    int key;
    cout << "Enter element to search: ";
    cin >> key;

    int resultSorted = binarySearch(arr, n, key);

    if (resultSorted != -1) {
        // Find index in original array
        int originalIndex = -1;
        for (int i = 0; i < n; i++) {
            if (original[i] == key) {
                originalIndex = i;
                break;
            }
        }
        cout << "Element " << key << " found at index " << originalIndex << " in the original array.\n";
    } else {
        cout << "Element " << key << " not found.\n";
    }

    return 0;
}
```

Figure 3:Code implementation of Binary Search-2

Sample Output Screenshot:

```
Enter number of elements: 4
Enter elements: ()we will sort and search):5
56 60 65
Enter element to search: 60
Element 60 found at index 2 in the original array.

--------------------------------
Process exited after 31.22 seconds with return value 0
Press any key to continue . . .
```

Figure 5: Output of Binary Search

3.3 Bubble Sort

Algorithm:

1. Compare adjacent elements and swap if necessary.
2. Repeat passes until the array is sorted.

Sample Code Screenshot:

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;

    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Bubble Sort algorithm
    for (int i = 0; i < n - 1; i++) {        // Number of passes
        for (int j = 0; j < n - i - 1; j++) { // Adjacent comparisons
            if (arr[j] > arr[j + 1]) {
                // Swap elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    // Display sorted array
    cout << "\nSorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Figure 6: Code implementation of Bubble sort

Sample Output Screenshot:

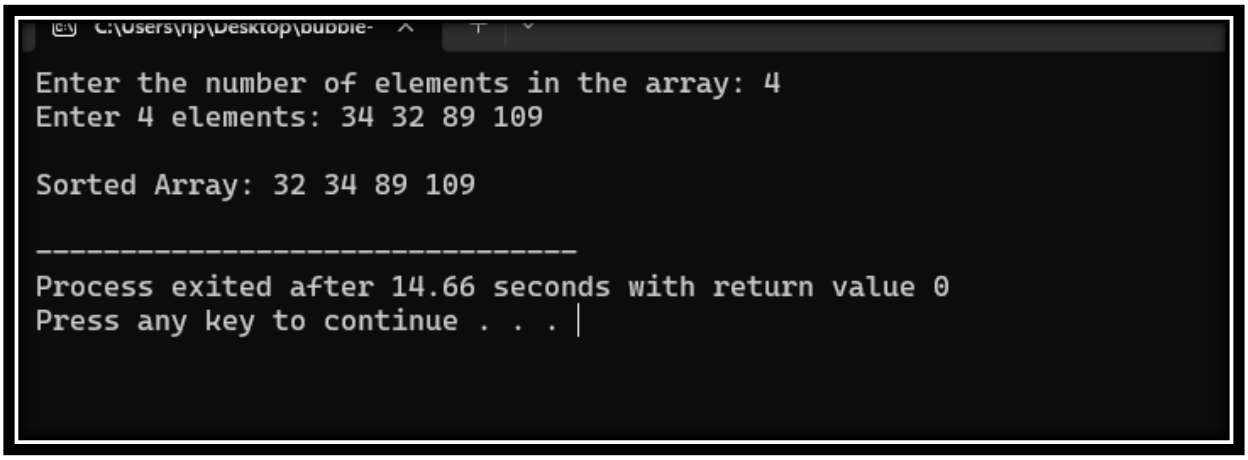


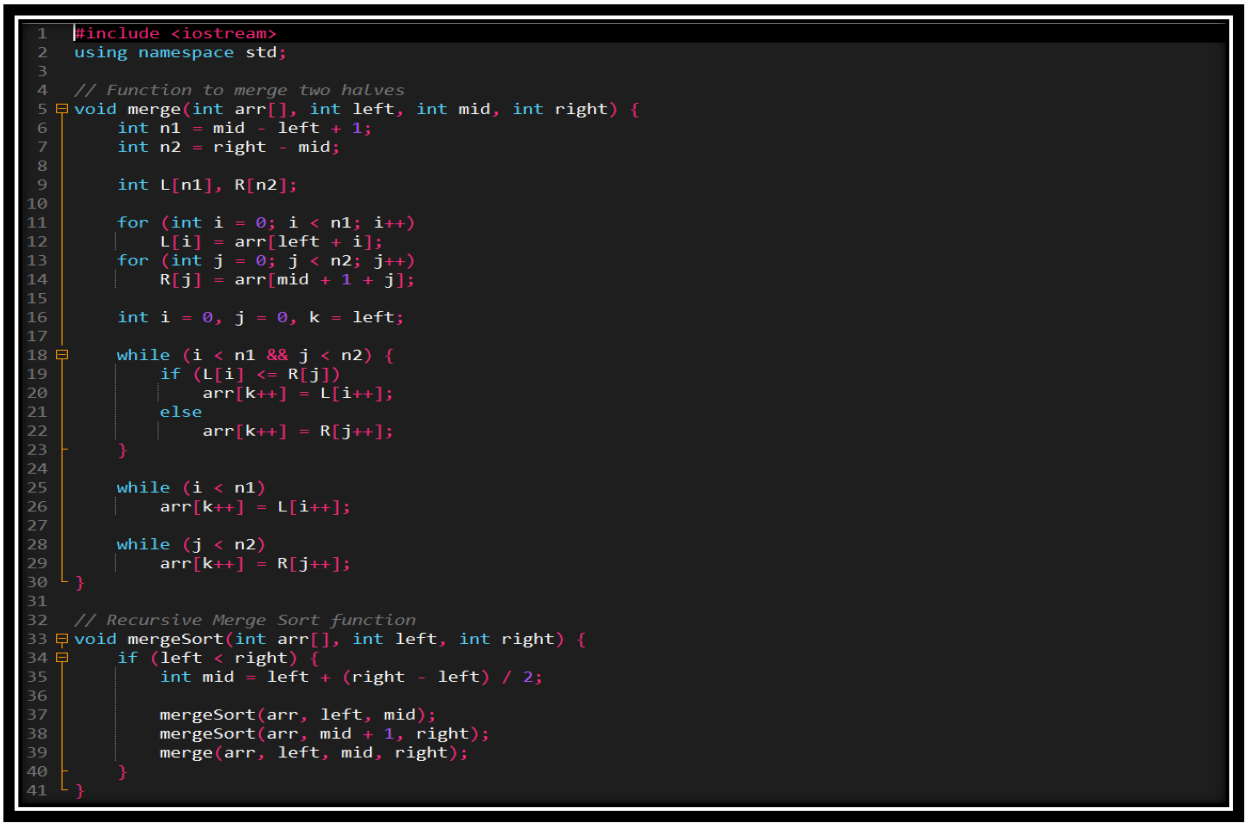
Figure 7: Output Bubble Sort

3.4 Merge Sort

Algorithm:

1. Recursively divide the array into halves.
2. Merge halves in sorted order.

Sample Code Screenshot:



```cpp
#include <iostream>
using namespace std;

// Function to merge two halves
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    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];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }

    while (i < n1)
        arr[k++] = L[i++];

    while (j < n2)
        arr[k++] = R[j++];
}

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

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

Figure 8: Code implementation of merge Sort -1

```cpp
int main() {
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;

    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << "\nOriginal array: ";
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    cout << endl;

    // Call mergeSort
    mergeSort(arr, 0, n - 1);

    cout << "Sorted array:   ";
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

Figure 9: Code implementation merge sort

Sample Output Screenshot:

```
Enter the number of elements in the array: 5
Enter 5 elements: 45 56 22 45 89

Original array: 45 56 22 45 89
Sorted array:   22 45 45 56 89

--------------------------------
Process exited after 19.57 seconds with return value 0
Press any key to continue . . .
```

Figure 10: Output merge sort

---

4. Conclusion

- Successfully implemented searching and sorting algorithms in C++.
- Demonstrated array manipulation, automatic sorting, and efficient searching.
- Code is interactive and user-friendly, allowing dynamic input.