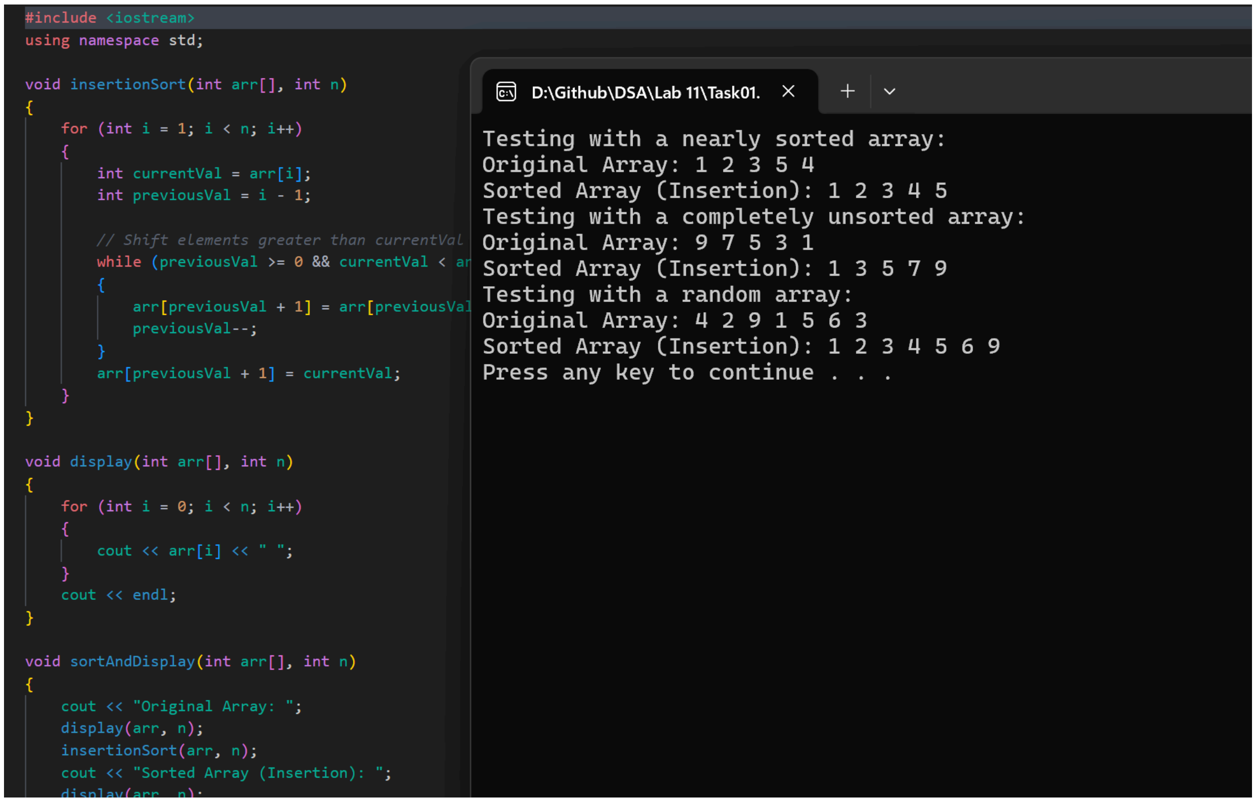
RIPHAH INTERNATIONAL UNIVERSITY

Name: Sufyan Ahmed Mughal

SAP ID: 55766

Data Structure & Algorithms
Lab 11

Task 01

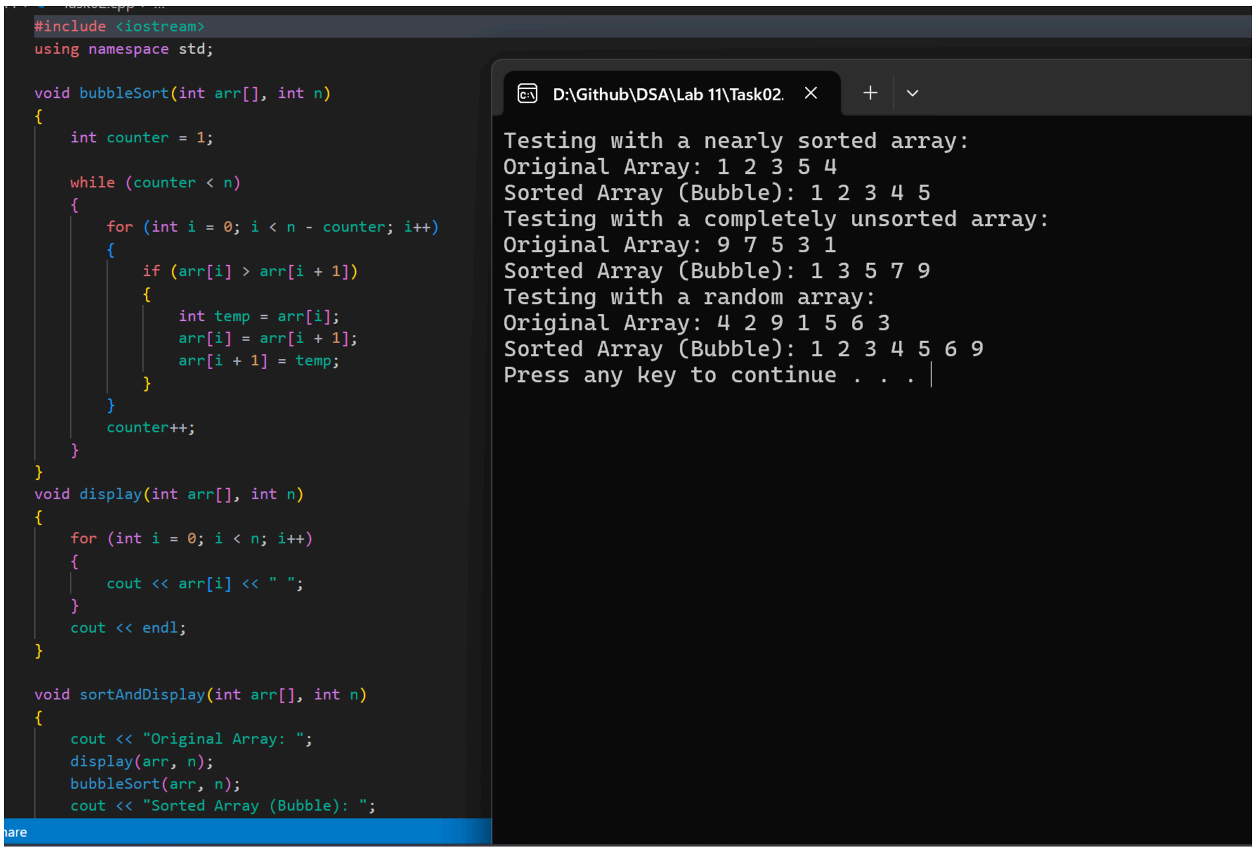


```cpp
#include <iostream>
using namespace std;

void insertionSort(int arr[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int currentVal = arr[i];
        int previousVal = i - 1;

        // Shift elements greater than currentVal
        while (previousVal >= 0 && currentVal < ar
        {
            arr[previousVal + 1] = arr[previousVal
            previousVal--;
        }
        arr[previousVal + 1] = currentVal;
    }
}

void display(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

void sortAndDisplay(int arr[], int n)
{
    cout << "Original Array: ";
    display(arr, n);
    insertionSort(arr, n);
    cout << "Sorted Array (Insertion): ";
    display(arr, n);
```

D:\Github\DSA\Lab 11\Task01.

```
Testing with a nearly sorted array:
Original Array: 1 2 3 5 4
Sorted Array (Insertion): 1 2 3 4 5
Testing with a completely unsorted array:
Original Array: 9 7 5 3 1
Sorted Array (Insertion): 1 3 5 7 9
Testing with a random array:
Original Array: 4 2 9 1 5 6 3
Sorted Array (Insertion): 1 2 3 4 5 6 9
Press any key to continue . . .
```

Task 02



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

void bubbleSort(int arr[], int n)
{
    int counter = 1;

    while (counter < n)
    {
        for (int i = 0; i < n - counter; i++)
        {
            if (arr[i] > arr[i + 1])
            {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }
        counter++;
    }
}
void display(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

void sortAndDisplay(int arr[], int n)
{
    cout << "Original Array: ";
    display(arr, n);
    bubbleSort(arr, n);
    cout << "Sorted Array (Bubble): ";
```

```
Testing with a nearly sorted array:
Original Array: 1 2 3 5 4
Sorted Array (Bubble): 1 2 3 4 5
Testing with a completely unsorted array:
Original Array: 9 7 5 3 1
Sorted Array (Bubble): 1 3 5 7 9
Testing with a random array:
Original Array: 4 2 9 1 5 6 3
Sorted Array (Bubble): 1 2 3 4 5 6 9
Press any key to continue . . .
```

Task 03

1. Comparisons and Swaps Calculation

For each test case and algorithm:

- ➔ **Nearly Sorted Array** { 1, 2, 3 ,5 ,4 }
  - ◆ **Insertion Sort:** Comparisons = 4, Swaps = 1
  - ◆ **Bubble Sort:** Comparisons = 10, Swaps = 1
- ➔ **Completely Unsorted Array** {9,7,5,3,1}
  - ◆ **Insertion Sort:** Comparisons = 10, Swaps = 10
  - ◆ **Bubble Sort:** Comparisons = 10, Swaps = 10
- ➔ **Random Array** {4, 2, 9 ,1, 5 ,6, 3}
  - ◆ **Insertion Sort:** Comparisons = 12, Swaps = 8

        - **Bubble Sort:** Comparisons = 21, Swaps = 14

2. Algorithm Performance Summary

- **Nearly Sorted Array**: Insertion Sort performs better with minimal swaps.
- **Completely Unsorted Array**: Both algorithms perform similarly, but Bubble Sort takes more passes.
- **Random Array**: Insertion Sort is more efficient due to fewer swaps and comparisons.

3. Report Summary

- **Insertion Sort** is faster for nearly sorted data due to fewer required operations.
- **Bubble Sort** is less efficient across test cases due to its repetitive comparisons, especially on random and unsorted arrays.