University Of Asia Pacific

Department of CSE

Course Code	**: CSE 208**
Course Title	**: Data Structures and Algorithms II Lab**
No Of Assignment	**: 01**
Date of Performance	**: 12.09.2024**
Date of Submission	**: 19.09.2024**

Submitted By:

Name : Nusrat Ahmmed Ekra

Student Id : 22201251

Section : E2

Semester : 2nd Year 2nd Semester

Submitted To:

Suri Dipannita Sayeed

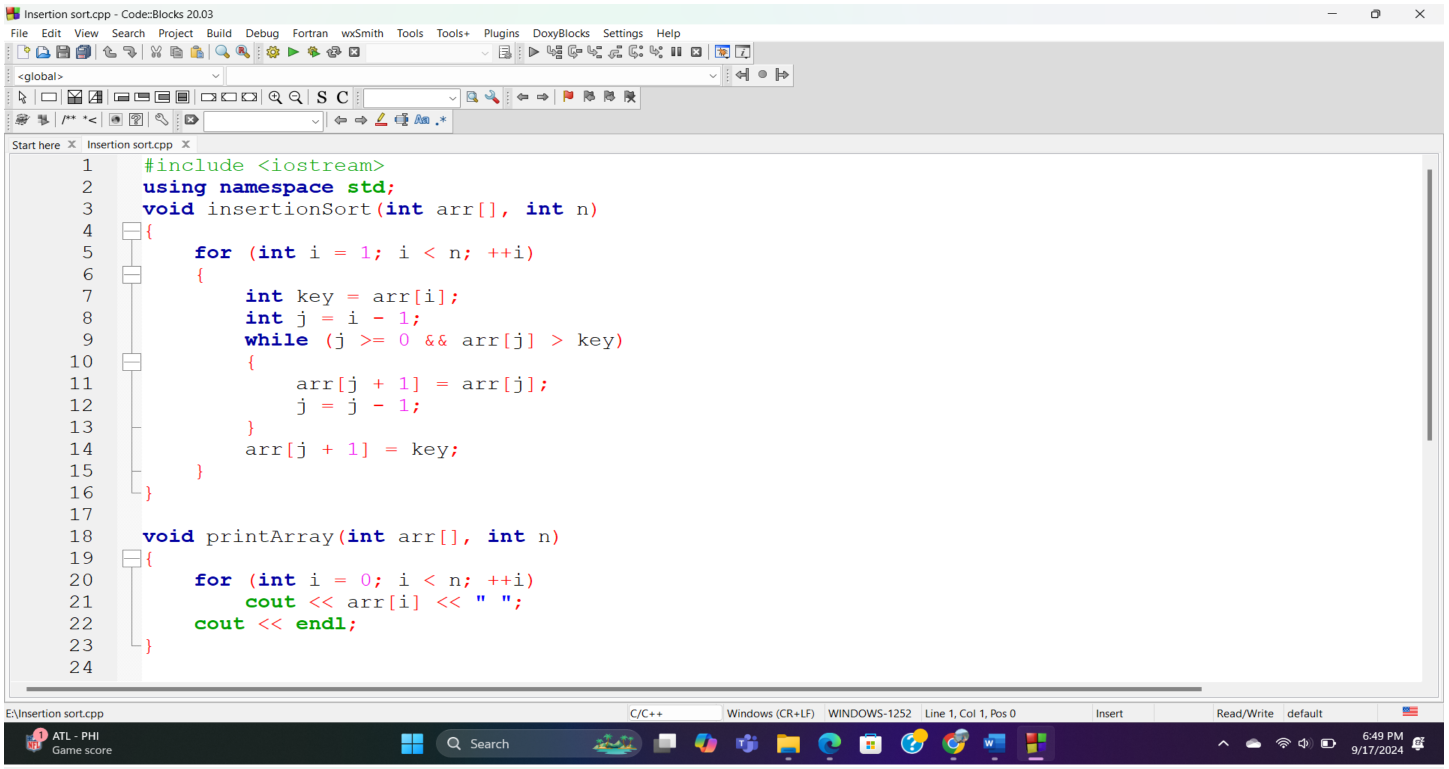
Lecturer,

Department of CSE, UAP

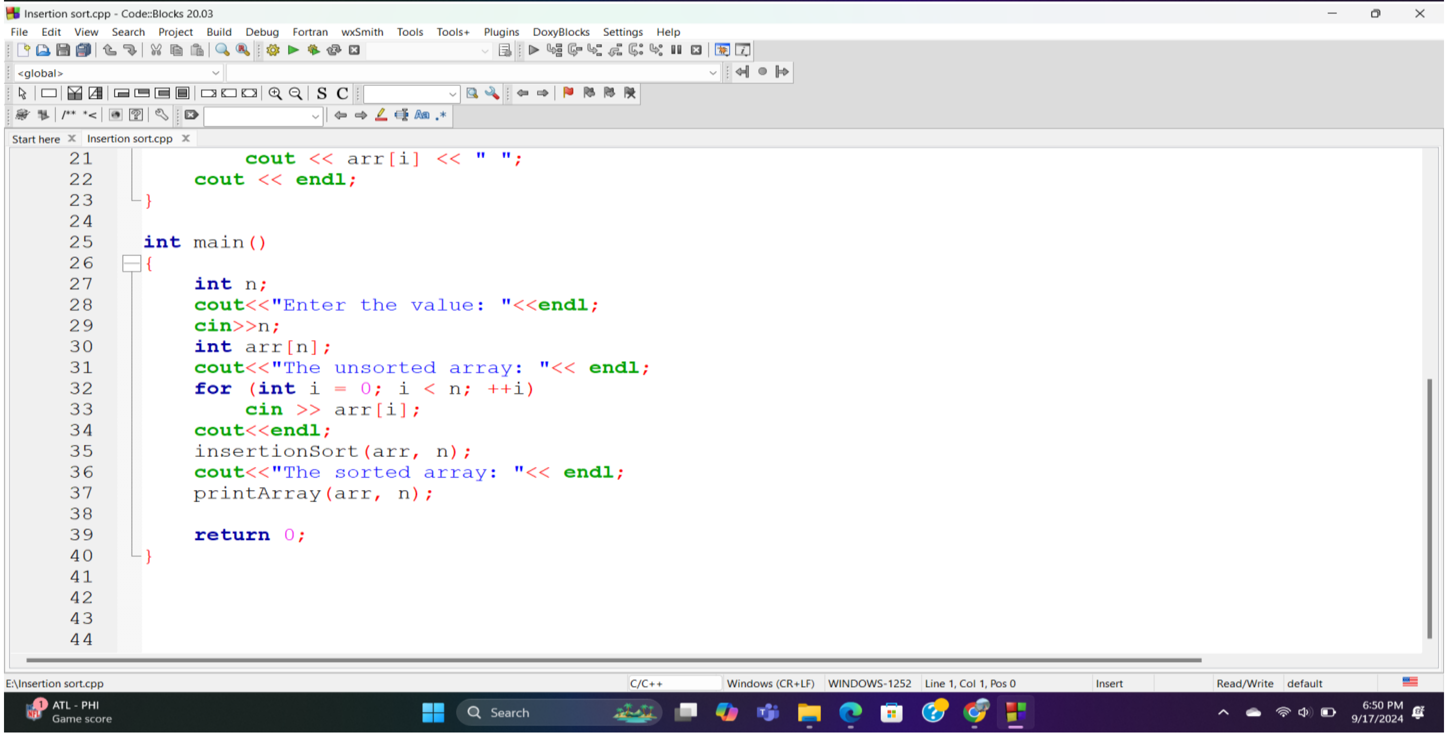
Problem no 03. Use Insertion sort, Bubble Sort and Selection sort to sort an unsorted array of Integers.

Insertion sort:

Code:

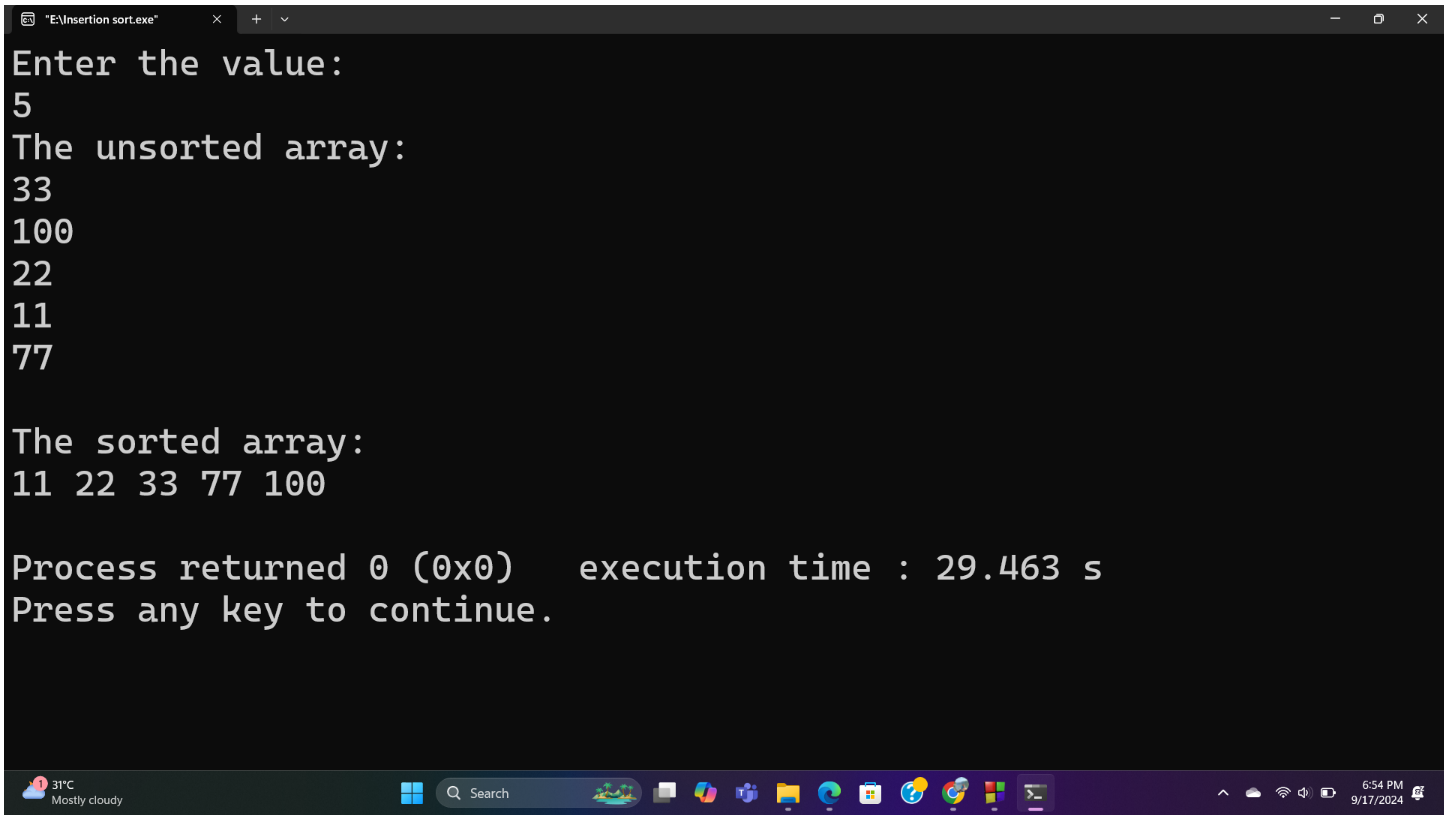


```cpp
#include <iostream>
using namespace std;
void insertionSort(int arr[], int n)
{
    for (int i = 1; i < n; ++i)
    {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

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

```



```cpp
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int n;
    cout<<"Enter the value: "<<endl;
    cin>>n;
    int arr[n];
    cout<<"The unsorted array: "<< endl;
    for (int i = 0; i < n; ++i)
        cin >> arr[i];
    cout<<endl;
    insertionSort(arr, n);
    cout<<"The sorted array: "<< endl;
    printArray(arr, n);

    return 0;
}
```

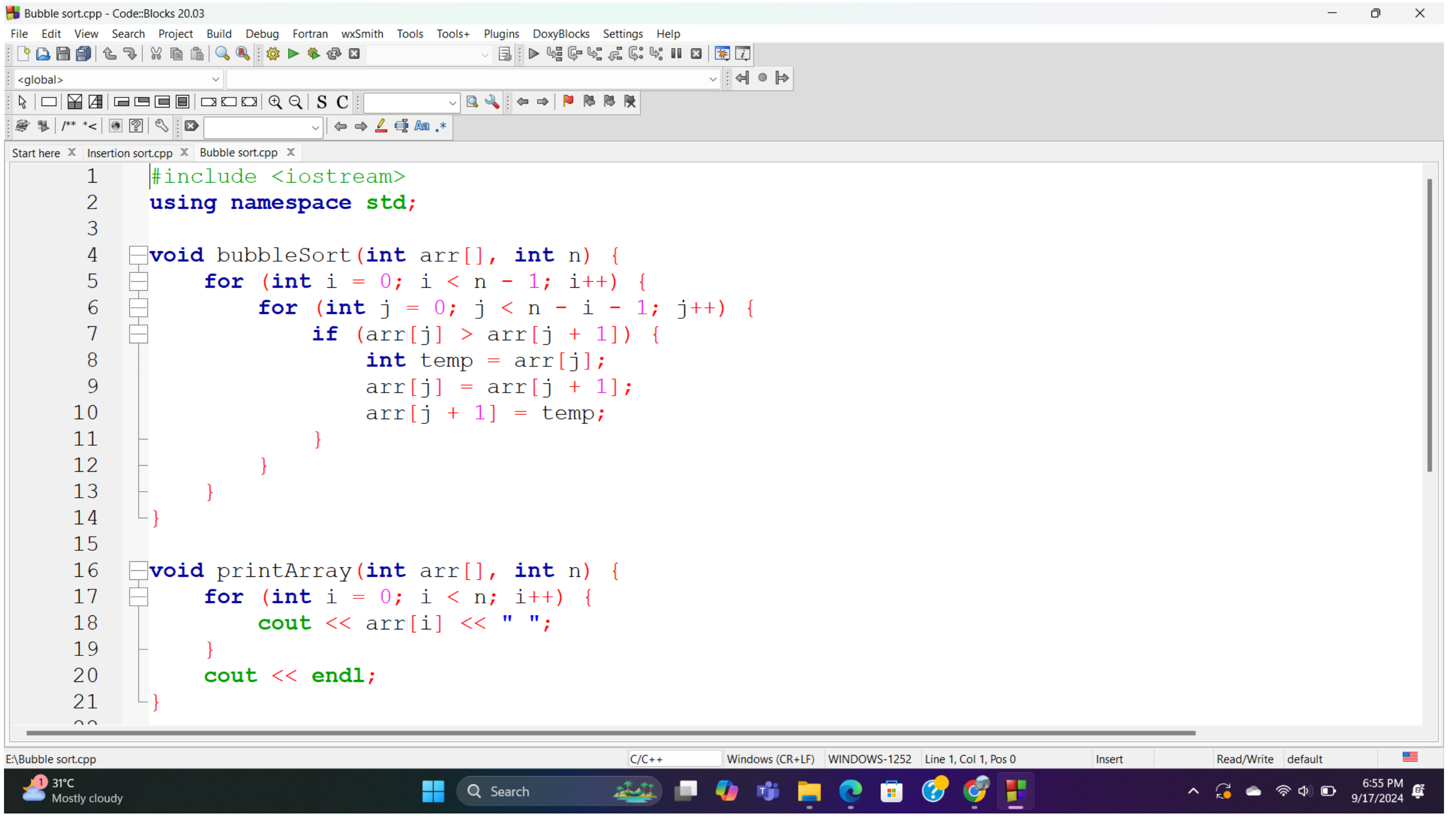
Output:



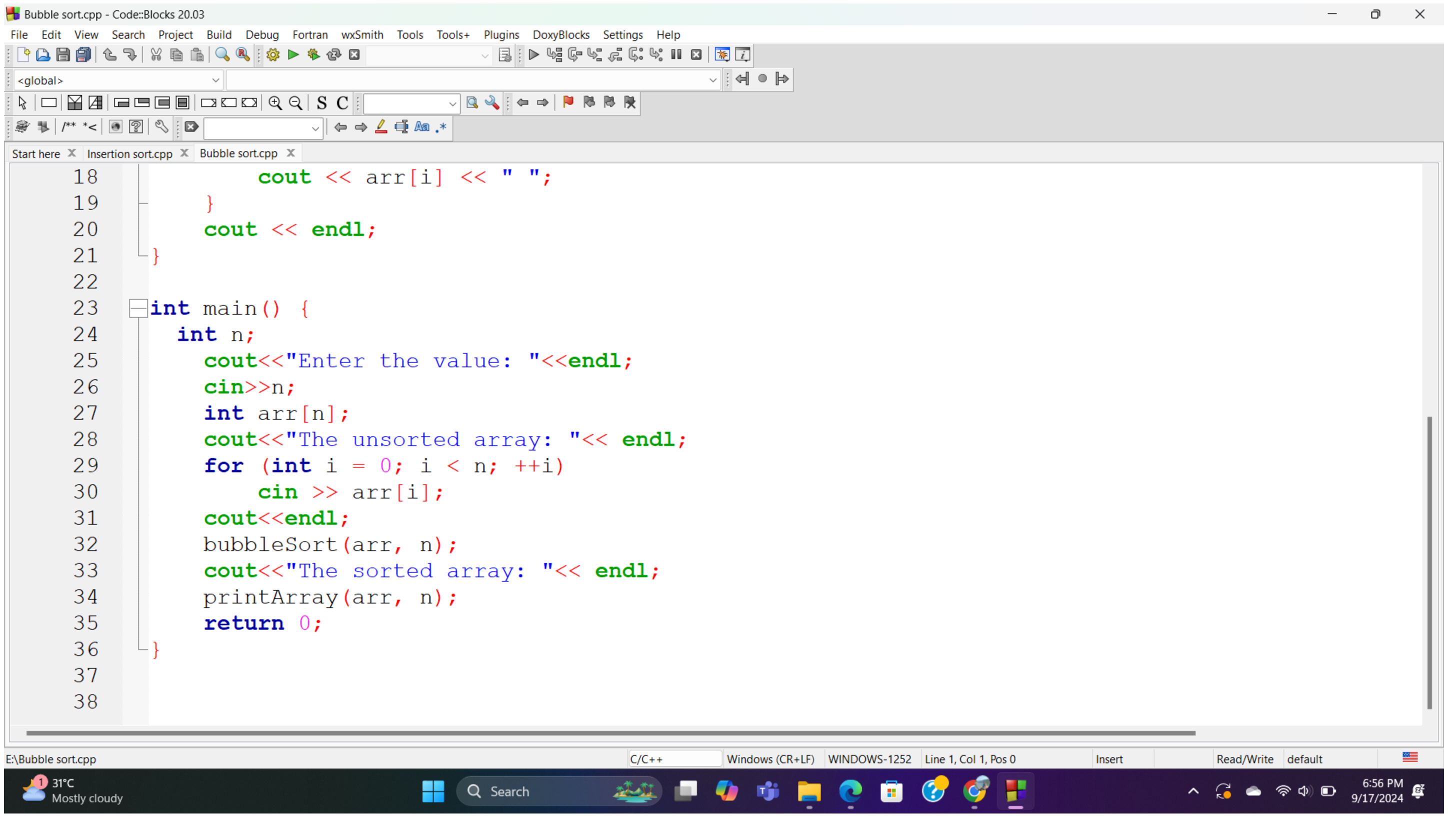
```
Enter the value:
5
The unsorted array:
33
100
22
11
77

The sorted array:
11 22 33 77 100

Process returned 0 (0x0)   execution time : 29.463 s
Press any key to continue.
```

Bubble Sort:

Code:

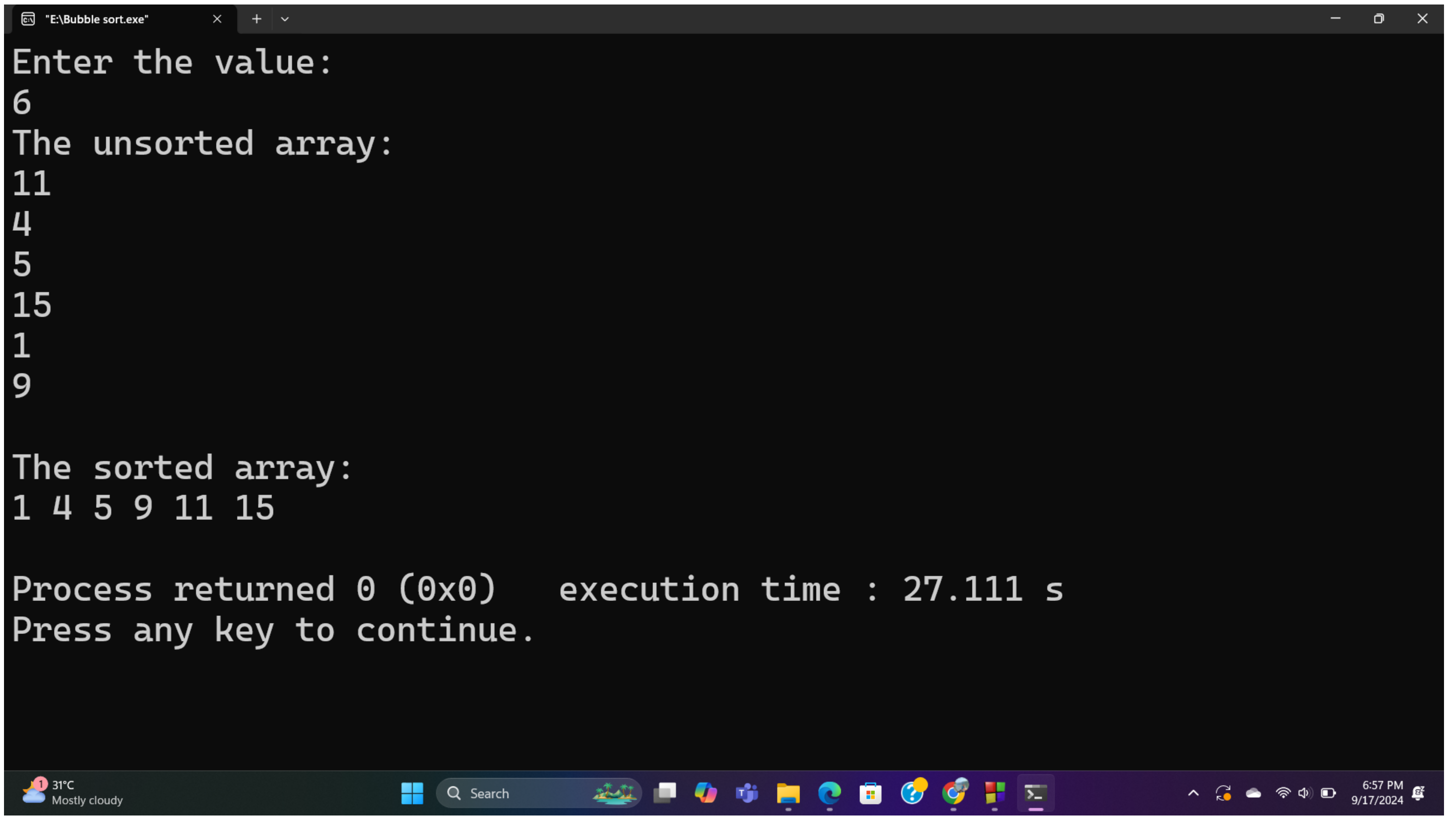


```cpp
#include <iostream>
using namespace std;

void bubbleSort(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]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

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



```
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
  int n;
    cout<<"Enter the value: "<<endl;
    cin>>n;
    int arr[n];
    cout<<"The unsorted array: "<< endl;
    for (int i = 0; i < n; ++i)
        cin >> arr[i];
    cout<<endl;
    bubbleSort(arr, n);
    cout<<"The sorted array: "<< endl;
    printArray(arr, n);
    return 0;
}
```

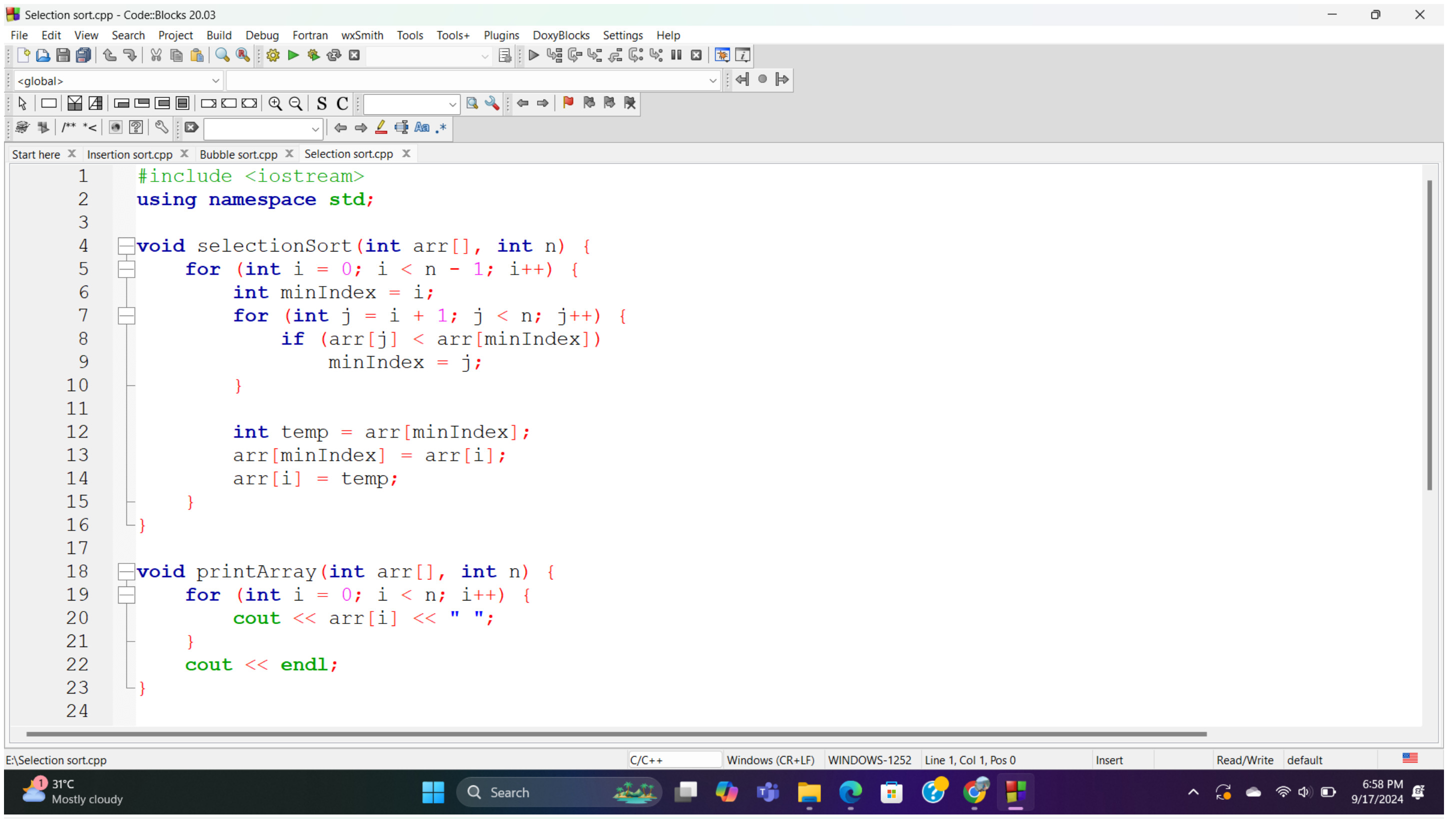
Output:



```
Enter the value:
6
The unsorted array:
11
4
5
15
1
9

The sorted array:
1 4 5 9 11 15

Process returned 0 (0x0)   execution time : 27.111 s
Press any key to continue.
```

Selection sort:

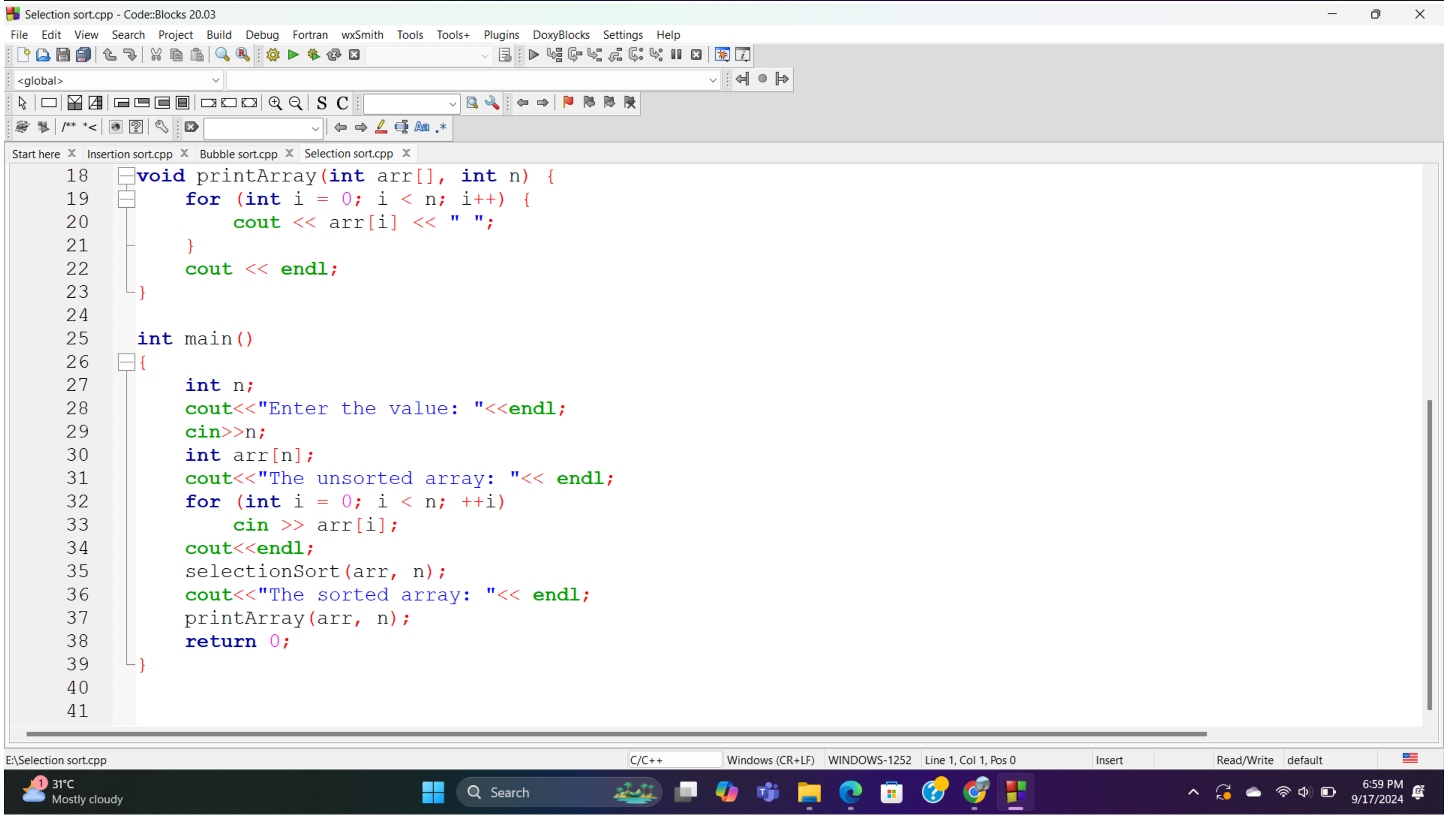
Code:



Selection sort.cpp - Code::Blocks 20.03

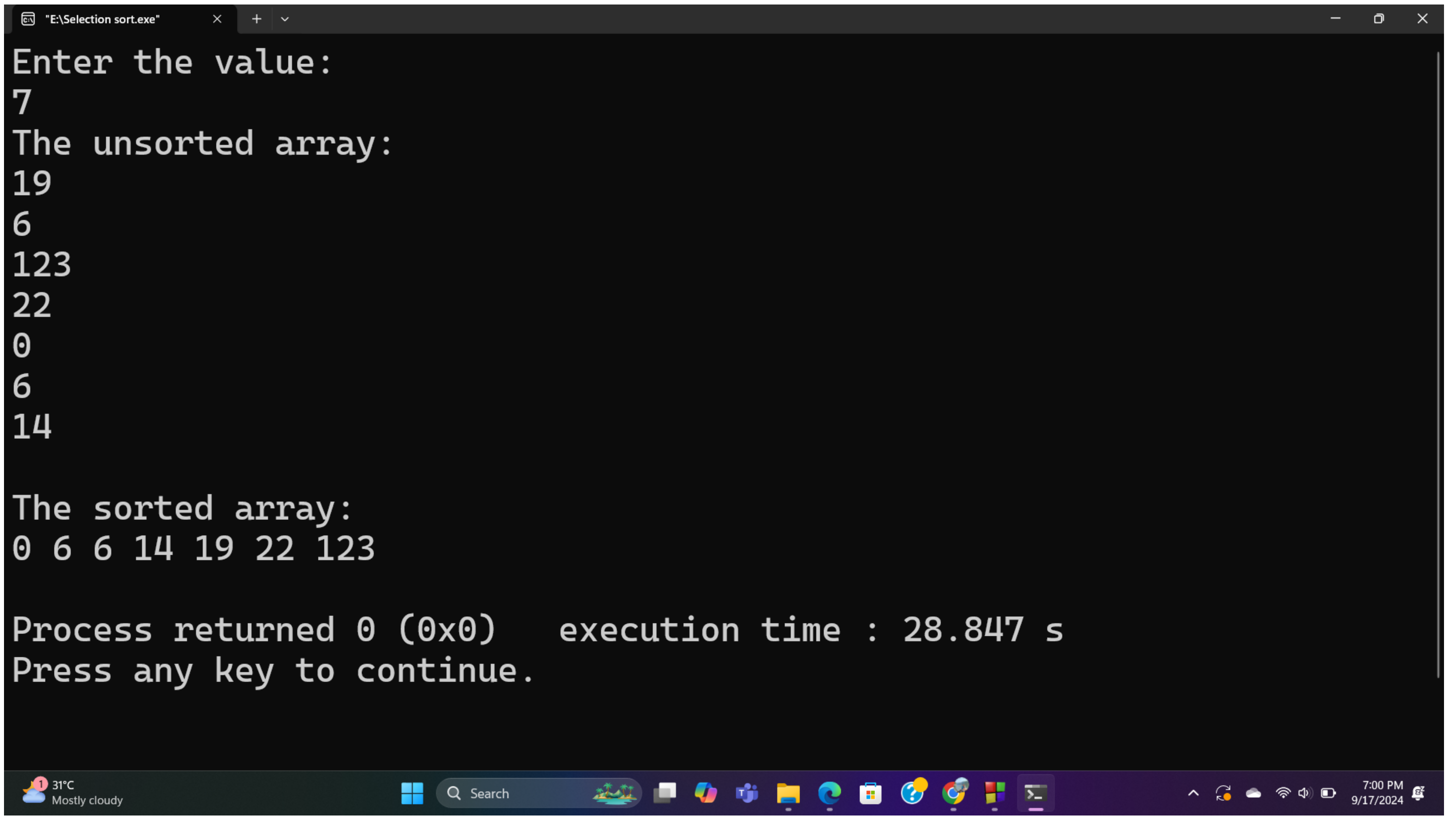
E:\Selection sort.cpp

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

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

        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

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

```



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

int main()
{
    int n;
    cout<<"Enter the value: "<<endl;
    cin>>n;
    int arr[n];
    cout<<"The unsorted array: "<< endl;
    for (int i = 0; i < n; ++i)
        cin >> arr[i];
    cout<<endl;
    selectionSort(arr, n);
    cout<<"The sorted array: "<< endl;
    printArray(arr, n);
    return 0;
}

```

Output:



```
Enter the value:
7
The unsorted array:
19
6
123
22
0
6
14

The sorted array:
0 6 6 14 19 22 123

Process returned 0 (0x0)   execution time : 28.847 s
Press any key to continue.
```

Problem no 04: Write the relative comparison of these three algorithms. Write main idea of the three approach and point out the difference of one from another. Write the best case, average case and worst-case complexity of the three approaches.

Solution:

Main Idea of Each Approach:

Selection Sort:

Find the minimum element from the array's unsorted section repeatedly, then replace it with the initial unsorted element. It keeps both the sorted and unsorted portions of the array intact. The smallest element from the unsorted area is chosen at each iteration, and it is then positioned correctly in the sorted section.

Selection Sort Iteration:

Suppose ,the Array is [3, 1, 4, 5, 2]

Pass 1:Unsorted portion: [3, 1, 4, 5, 2].Find the minimum element: ‘1’.Swap ‘1’ with ‘3’; the array is, 1, 3, 4, 5, 2

Pass 2:Unsorted portion: [3, 4, 5, 2].Find the minimum element: ‘2’.Swap ‘2’ with ‘3’ ; the array is,

1, 2, 4, 5, 3

Pass 3:Unsorted portion: [4, 5, 3].Find the minimum element: ‘3’ .Swap ‘3’ with ‘4’ ; the array is,

1, 2, 3, 5, 4

Pass 4:Unsorted portion: [5, 4].Find the minimum element: ‘4’. Swap ‘4’ with ‘5’;the array is,

1, 2, 3, 4, 5

Bubble Sort:

examines surrounding elements repeatedly and switches them if the order is incorrect. In every iteration, this technique "bubbles" the largest member to the end of the list. The largest unsorted member is in its proper location after each pass, hence reducing the amount of the array that is unsorted.

Bubble Sort Iteration:

Suppose ,the Array is [3, 1, 4, 5, 2]

Pass 1:Compare ‘3’ and ‘1’: Swap ; the array is,

1, 3, 4, 5, 2

Compare ‘3’ and ‘4’: No swap; the array is,

1, 3, 4, 5, 2

Compare ‘4’ and ‘5’: No swap; the array is,

1, 3, 4, 5, 2

Compare ‘5’ and ‘2’: Swap; the array is,

1, 3, 4, 2, 5

Pass 2: Compare ‘1’ and ‘3’: No swap ; the array is,

1, 3, 4, 2, 5

Compare ‘3’ and ‘4’: No swap ; the array is,

1, 3, 4, 2, 5

Compare ‘4’ and ‘2’: Swap ; the array is,

1, 3, 2, 4, 5

Pass 3:Compare ‘1’ and ‘3’: No swap; the array is,

1, 3, 2, 4, 5

Compare ‘3’ and ‘2’: Swap; the array is,

1, 2, 3, 4, 5

Pass 4:Compare ‘1’ and ‘2’: No swap; the array is,

1, 2, 3, 4, 5

Insertion Sort:

takes elements from the unsorted portion and places them in the appropriate spot in the sorted portion to build the sorted array one element at a time.Works similarly like sorting playing cards in your hands by placing each card in the appropriate location in relation to the previously sorted cards.

Insertion sort iteration:

Suppose, the array is [3, 1, 4, 5, 2])

Pass 1:There is two part sorted and unsorted; 3 is sorted and others are unsorted.

'1'will compare with '3'; here '3' is greater than '1' so it will swap.

1,3,4,5,2

Pass 2: '4' will compare with '3' and '1' but '4' is greater so the array will;

1,3,4,5,2

Pass 3: Same for '5'; the array will,

1,3,4,5,2

Pass 4: '2' is key; it will compare with sorted part of array and swap. Then the array will be,

1,2,3,4,5

Differences Between the Algorithms:

Selection Sort vs. Bubble Sort:

1.Selection Sort focuses on minimizing the number of swaps by finding the minimum element in each pass, but it requires comparisons regardless of the input's initial order. It performs exactly swaps.

2.Bubble Sort, on the other hand, frequently swaps adjacent elements to bubble the largest unsorted element to its correct position. However, if the array is nearly sorted, it can stop early, which gives it a potential performance boost in those scenarios. The worst-case time complexity remains , but its best-case complexity can be if the array is already sorted.

Bubble Sort vs. Insertion Sort:

1.Bubble Sort makes multiple passes and compares adjacent elements. In contrast, Insertion Sort moves elements only when necessary, and is generally more efficient for nearly sorted arrays.

2.Insertion Sort performs fewer comparisons than Bubble Sort and is more efficient in the average case, especially for partially sorted data.

Selection Sort vs. Insertion Sort:

1.Selection Sort scans through the unsorted portion of the array during each pass to find the minimum element, leading to comparisons regardless of the array's order. This results in many unnecessary comparisons, especially when the array is already sorted or nearly sorted.

2.Insertion Sort, on the other hand, becomes much more efficient in nearly sorted arrays because it compares only until the correct spot for the current element is found and then shifts elements as needed. Its best-case time complexity is (when the array is already sorted), and its worst-case time complexity is (when the array is sorted in reverse order). In practice, Insertion Sort generally outperforms Selection Sort for small or partially sorted arrays because it avoids many unnecessary comparisons and shifts only when necessary.

Time Complexity:

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$