Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Lab Report NO 03

Course Title: Data Structures Lab

Course Code: CSE 206 Section: 232_D10

Student Details

	Name	ID
1.	Md.Rabby Khan	213902037

Lab Date : 18-09-2024

Submission Date : 23-09-2024

Course Teacher's Name : Fatema Akter

Lab Report Status

Marks: …… **Signature:.....**

Comments:..... **Date:.....**

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of Bubble Sort, Insertion Sort, and Selection Sort Algorithms Using Arrays

2. OBJECTIVES/AIM

- Implement Bubble Sort, Insertion Sort, and Selection Sort using arrays.
- Understand the basic working of each sorting algorithm.
- Compare the performance and behavior of each sorting algorithm.

3. PROCEDURE / ANALYSIS / DESIGN

Bubble Sort Algorithm:

1. Set a variable n to the size of the array.
2. Iterate through the array n-1 times.
3. For each iteration, compare adjacent elements.
4. If the current element is greater than the next element, swap them.
5. Continue this process until no more swaps are needed.

Insertion Sort Algorithm:

1. Start with the second element in the array.
2. Compare it with the elements before it and shift all larger elements to the right.
3. Insert the current element into its correct position.
4. Repeat the process for the remaining elements.

Selection Sort Algorithm:

1. Divide the array into a sorted and an unsorted part.
2. Find the smallest element in the unsorted part.
3. Swap it with the first element of the unsorted part.
4. Move the boundary of the sorted part one step to the right.

4. IMPLEMENTATION

4.1 Bubble short

```
// Bubble short
#include <iostream>
using namespace std;

int main()
{
   int i, j, value, size, s_item;

   cout << "Enter the array size: ";
   cin >> size; // Input array size

   int arr[size]; // Declare array

   cout << "Enter the array values: ";
   for (i = 0; i < size; i++)
      cin >> arr[i]; // Input array elements

   cout << "Array values: ";
   for (i = 0; i < size; i++)
      cout << arr[i] << " "; // Display original array
   cout << endl;

   for (i = 1; i < size; i++)
      for (j = 0; j < size - i; j++)
         if (arr[j] > arr[j + 1])
            swap(arr[j], arr[j + 1]);

   cout << "Array values after Bubble sorted: ";
   for (i = 0; i < size; i++)
      cout << arr[i] << " "; // Display original array
   cout << endl;

   return 0;
}
```

5. TEST RESULT / OUTPUT

5.1 Bubble short

```
Enter the array size: 5
Enter the array values: 8 4 2 6 1
Array values: 8 4 2 6 1
Array values after Bubble sorted: 1 2 4 6 8
PS C:\Users\DELL\Desktop\Data Stucture Lab>
```

4. IMPLEMENTATION

4.2 Insertion Sort

```
// insertion sort
#include <iostream>
   using namespace std;

int main()
{
   int i, j, value, size;

   cout << "Enter the array size: ";
   cin >> size; // Input array size

   int arr[size]; // Declare array

   cout << "Enter the array values: ";
   for (i = 0; i < size; i++)
      cin >> arr[i]; // Input array elements

   cout << "Array values: ";
   for (i = 0; i < size; i++)
      cout << arr[i] << " "; // Display original array
   cout << endl;

   for (i = 1; i < size; i++)
   {
```

```
        value = arr[i];
        j = i - 1;

              while (j >= 0 && arr[j] > value)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = value;
    }

    cout << "Array values after  Insertion sorted : ";
    for (i = 0; i < size; i++)
        cout << arr[i] << " "; // Display sorted array
    cout << endl;

    return 0;
}
```

5. TEST RESULT / OUTPUT

5.2 Insertion Sort

```
Enter the array size: 6
Enter the array values: 5 7 3 4 1 9
Array values: 5 7 3 4 1 9
Array values after  Insertion sorted: 1 3 4 5 7 9
PS C:\Users\DELL\Desktop\Data Stucture Lab>
```

4. IMPLEMENTATION

4.3 Selection Sort

```
// Selection sort
#include <iostream>
using namespace std;

int main()
{
   int i, j, value, size, position, s_item, smallest;

   cout << "Enter the array size: ";
   cin >> size; // Input array size

   int arr[size]; // Declare array

   cout << "Enter the array values: ";
   for (i = 0; i < size; i++)
      cin >> arr[i]; // Input array elements

   cout << "Array values: ";
   for (i = 0; i < size; i++)
      cout << arr[i] << " "; // Display original array
   cout << endl;

   for (position = 0; position < size - 1; position++)
   {
      smallest = position;
      for (i = position + 1; i < size; i++)
         if (arr[smallest] > arr[i])
            smallest = i;

      swap(arr[position], arr[smallest]);
   }
```

```
    cout << "Array values after Selection sorted: ";
    for (i = 0; i < size; i++)
        cout << arr[i] << " "; // Display original array
    cout << endl;

    return 0;
}
```

5. TEST RESULT / OUTPUT

5.3 Selection Sort

```
Enter the array size: 5
Enter the array values: 4 6 7 3 1
Array values: 4 6 7 3 1
Array values after Selection sorted: 1 3 4 6 7
PS C:\Users\DELL\Desktop\Data Stucture Lab>
```

6. ANALYSIS AND DISCUSSION

From the implemented algorithms, we can observe that each sorting method operates differently in terms of comparisons and swaps:

Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order. This process continues for each element, leading to a time complexity of $\mathbf{O(n^2)}$ in the worst case.

Insertion Sort builds the sorted array one element at a time by repeatedly picking the next element and inserting it into its correct position. It generally performs better than Bubble Sort on smaller or partially sorted arrays, but still has a worst-case time complexity of $\mathbf{O(n^2)}$.

Selection Sort works by repeatedly selecting the smallest element from the unsorted portion and swapping it with the first element of the unsorted portion. While its number of swaps is lower compared to Bubble Sort, it still has a time complexity of $\mathbf{O(n^2)}$ due to the nested loop structure.

7. SUMMARY

The objective of this lab was to implement and compare three sorting algorithms using arrays. **Bubble Sort**, **Insertion Sort**, and **Selection Sort** were successfully implemented and tested. Each algorithm has its strengths and weaknesses based on the number of comparisons and swaps. This lab provided a foundational understanding of sorting algorithms, which is crucial for optimizing data arrangement tasks.