ALGORITHMS LABORATORY
[CS-2098]
Individual Work

Lab. No.- 2 **Date.- 31/07/2023**

Topic-Fundamentals of Algorithmic Problem Solving

Roll Number:	21051577	Branch/Section:	CSE-3
Name in Capital:	MANAV MALHOTRA		

Program No: 2.1

Program Title:

Write a menu driven program as given below, to sort an array of n integers in ascending order by **insertion sort algorithm** and determine the **time required (in terms of step/frequency count)** to sort the elements. Repeat the experiment for different values of n and different nature of data (i.e.apply insertion sort algorithm on the data of array that are already sorted, reversely sorted and random data). Finally plot a graph of the time taken versus n for each type of data. The elements can be read from a file or can be generated using the random number generator. Assume the cost of each statement is 1.

INSERTION SORT MENU

0. Quit
1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using any sorting algorithm
5. Time Complexity (step count) to sort ascending of data for all Cases (Data Ascending, Data in Descending & Random Data) in tabular form for values n=5 to 9, step=1.
6. Time Complexity (step count) to sort ascending of data for all Cases (Data Ascending, Data in Descending & Random Data) in tabular form for values n=5000 to 50000, step=5000

Enter your choice:

Sample Input & Output:

In the output the above menu will be displayed.

Enter your choice: 1
Enter how many random numbers to store into an array: 10

Enter your choice: 2
The content of array is as follows: 10 30 34 56 70 36 90 88 72 38
(Note: Based on user choice as 1, each time this output may vary)

Enter your choice: 3
The content of array is as follows: 10 30 34 36 38 56 70 72 88 90

If option 5 is entered, then the output will be displayed as follows:

Sl.	Data Size	#Steps (Ascending data)	#Steps (Descending data)	#Steps (Random data)
1	5	19	39	29
2	6	23	53	29
3	7	27	69	49
4	8	31	87	55
5	9	35	107	8

Note: For a insertion sort function with specific data size, the number of steps required for that insertion sort function for ascending data and descending data will remain same always. For random data it will vary for each execution, but the value must come in between #Steps (Ascending data) and #Steps (Descending data).

Input/Output Screenshots:

RUN-1:

```
PS C:\5th Sem Notes\21051577Algo Lab> cd "c:\5th Sem Notes\21051577Algo Lab\Lab 3\" ; if ($?) { g++ lab3q1.cpp -o lab3q1 } ; if ($?) { .\lab3q1 }

------------------------------------------------------------------------
                      INSERTION SORT MENU
------------------------------------------------------------------------
0. Quit
1. n Random numbers => Array
2. Display the array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using Insertion Sort Algorithm
5. Time Complexity (step count) to sort ascending of data for all Cases
(Data Ascending, Data in Descending & Random Data) in tabular form for
values n=5 to 9, step=1
6. Time Complexity (step count) to sort ascending of data for all Cases
(Data Ascending, Data in Descending & Random Data) in tabular form for
values n=5000 to 500000, step=5000
------------------------------------------------------------------------

Enter your choice: 1
Enter how many random numbers to store into an array: 10

Enter your choice: 2

98 87 1 32 70 61 37 0 58 93

Enter your choice: 3

0 1 32 37 58 61 70 87 93 98

Enter your choice: 4

98 93 87 70 61 58 37 32 1 0

Enter your choice: 5

Sl. No.  Data Size       #Steps(Ascending data)          #Steps(Descending data)         #Steps(Random data)
-------  ---------       ----------------------          ----------------------         ------------------
1        5               20                              50                             26
2        6               25                              70                             46
3        7               30                              93                             51
4        8               35                              119                            65
5        9               40                              148                            103

Enter your choice: 6

Sl. No.  Data Size       #Steps(Ascending data)          #Steps(Descending data)         #Steps(Random data)
-------  ---------       ----------------------          ----------------------         ------------------
1        5000            24995                           37144028                       18508373
2        10000           49995                           148537044                      74327994
3        15000           74995                           334180555                      165922252
4        20000           99995                           594067289                      297420305
5        25000           124995                          928215060                      464437995
6        30000           149995                          1336613650                     662222527
7        35000           174995                          1819254017                     899695406
8        40000           199995                          2376145617                     1186301400
9        45000           224995                          3007273348                     1494864613
10       50000           249995                          3712679024                     1855357373

Enter your choice: 0

Quitting...
```

RUN-2

```
PS C:\5th Sem Notes\21051577Algo Lab> cd "c:\5th Sem Notes\21051577Algo Lab\Lab 3\" ; if ($?) { g++ lab3q1.cpp -o lab3q1 } ; if ($?) { .\lab3q1 }

------------------------------------------------------------------------
                        INSERTION SORT MENU
------------------------------------------------------------------------
0. Quit
1. n Random numbers => Array
2. Display the array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using Insertion Sort Algorithm
5. Time Complexity (step count) to sort ascending of data for all Cases
(Data Ascending, Data in Descending & Random Data) in tabular form for
values n=5 to 9, step=1
6. Time Complexity (step count) to sort ascending of data for all Cases
(Data Ascending, Data in Descending & Random Data) in tabular form for
values n=5000 to 500000, step=5000
------------------------------------------------------------------------

Enter your choice: 1
Enter how many random numbers to store into an array: 20

Enter your choice: 2

62 98 93 66 48 1 56 23 68 72 53 89 75 41 86 44 44 33 62 52

Enter your choice: 3

1 23 33 41 44 44 48 52 53 56 62 62 66 68 72 75 86 89 93 98

Enter your choice: 4

98 93 89 86 75 72 68 66 62 62 56 53 52 48 44 44 41 33 23 1
```

```
Enter your choice: 5

Sl. No.  Data Size       #Steps(Ascending data)          #Steps(Descending data)         #Steps(Random data)
-------  ---------       ----------------------          ----------------------         -------------------
1        5               20                              47                             41
2        6               25                              67                             55
3        7               30                              90                             75
4        8               35                              116                            80
5        9               40                              145                            103

Enter your choice: 6

Sl. No.  Data Size       #Steps(Ascending data)          #Steps(Descending data)         #Steps(Random data)
-------  ---------       ----------------------          ----------------------         -------------------
1        5000            24995                           37143431                       18712625
2        10000           49995                           148535523                      74734794
3        15000           74995                           334178464                      166304005
4        20000           99995                           594067379                      296127188
5        25000           124995                          928215840                      467340420
6        30000           149995                          1336609720                     672779698
7        35000           174995                          1819255646                     910737290
8        40000           199995                          2376150471                     1192626177
9        45000           224995                          3007288483                     1510959466
10       50000           249995                          3712665719                     1845966335

Enter your choice: 0

Quitting...
```

Source code

```
#include <iostream>
#include <cstdlib>
#include <time.h>

using namespace std;

// Function to count steps in insertion sort for ascending order
long long int is_ascend_count(int arr[], int n, int s)
{
	long long int steps = 0;
	for (int i = 1; i < n; i++)
	{
		steps++;
		int key = arr[i];
		steps++;
		int j = i - 1;
		steps++;
		steps++;
		while (j >= 0 && arr[j] > key)
		{
			steps++;
			arr[j + 1] = arr[j];
			steps++;
			 j--;
			steps++;
		}
		arr[j + 1] = key;
		steps++;
	}
	return steps;
}

// Function to perform insertion sort
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--;
		}
		arr[j + 1] = key;
	}
}
```

```
// Function to sort in descending order
void descendingSort(int arr[], int n)
{
	int temp, j = n - 1;
	for (int i = 0; i < n / 2; i++)
	{
		temp = arr[i];
		arr[i] = arr[j];
		arr[j] = temp;
		j--;
	}
}

// Function to print an array
void printArray(int arr[], int n)
{
	cout << endl;
	for (int i = 0; i < n; i++)
	{
		cout << arr[i] << " ";
	}
	cout << endl;
}

// Function to generate a random array of given size
int* generate_random_array(int n)
{
	int* array = new int[n];
	srand(time(NULL));
	for (int i = 0; i < n; i++)
	{
		array[i] = rand() % 100 + 1;
	}
	return array;
}

// Function to copy contents from one array to another
void copyArray(int arr1[], int arr2[], int n)
{
	for (int i = 0; i < n; i++)
	{
		arr2[i] = arr1[i];
	}
}
```

```
int main()
{
	int choice, n, steps, *arr, m; // Declare variables

	// Display the menu
	cout << endl;
	cout << "--------------------------------------------------------------------------" << endl;
	cout << "\t\t\tINSERTION SORT MENU" << endl;
	cout << "--------------------------------------------------------------------------" << endl;
	cout << "0. Quit" << endl;
	cout << "1. n Random numbers => Array" << endl;
	cout << "2. Display the array" << endl;
	cout << "3. Sort the Array in Ascending Order by using Insertion Sort Algorithm" << endl;
	cout << "4. Sort the Array in Descending Order by using Insertion Sort Algorithm" << endl;
	cout << "5. Time Complexity (step count)..." << endl;
	cout << "6. Time Complexity (step count)..." << endl;
	cout << "--------------------------------------------------------------------------" << endl;

	do
	{
		cout << endl << "Enter your choice: ";
		cin >> choice; // Get user's choice

		switch (choice)
		{
			case 0: cout << endl << "Quitting..." << endl;
				break;

			case 1: cout << "Enter how many random numbers to store into an array: ";
				cin >> n; // Get array size
				arr = new int[n]; // Create array
				srand(time(NULL)); // Seed random number generator
				for (int i = 0; i < n; i++)
				{
					arr[i] = rand() % 100; // Fill array with random numbers
				}
				break;

			case 2: printArray(arr, n); // Display array
				break;

			case 3: insertionSort(arr, n); // Sort array in ascending order
				printArray(arr, n); // Display sorted array
				break;

			case 4: insertionSort(arr, n); // Sort array in ascending order
				descendingSort(arr, n); // Sort array in descending order
				printArray(arr, n); // Display sorted array
				break;
```

```
case 5: cout << endl;
        // Display header for table
        cout << "Sl. No.\t Data Size \t #Steps(Ascending data) \t
        #Steps(Descending data) \t #Steps(Random data)" << endl;
        cout << "-------\t --------- \t ---------------------- \t ----------------------- \t ------
        -------------" << endl;

        m = 1; // Initialize serial number counter

        // Loop through different array sizes
        for (int n = 5; n <= 9; n++)
        {
                // Initialize arrays
                int *ascending_array = new int[n];
                int *descending_array = new int[n];
                int *random_array;
                // Generate random array
                random_array = generate_random_array(n);

                // Calculate step counts for different sorting methods
                int random_steps = is_ascend_count(random_array, n, 1);

                // Copy arrays and perform sorting
                copyArray(random_array, ascending_array, n);
                copyArray(random_array, descending_array, n);
                insertionSort(ascending_array, n);
                descendingSort(descending_array, n);

                // Calculate step counts for sorted arrays
                int ascending_steps = is_ascend_count(ascending_array, n, 1);
                int descending_steps = is_ascend_count(descending_array, n, 1);

                // Display results in a table format
                cout << m << " \t " << n << "\t\t " << ascending_steps << "\t\t\t\t "
                << descending_steps << "\t\t\t\t " << random_steps << endl;
                m++;
        }
        break;

case 6: cout << endl;
        // Display header for table
        cout << "Sl. No.\t Data Size \t #Steps(Ascending data) \t
        #Steps(Descending data) \t #Steps(Random data)" << endl;
        cout << "-------\t --------- \t ---------------------- \t ----------------------- \t ------
        -------------" << endl;

        m = 1; // Initialize serial number counter

        // Loop through different array sizes
```

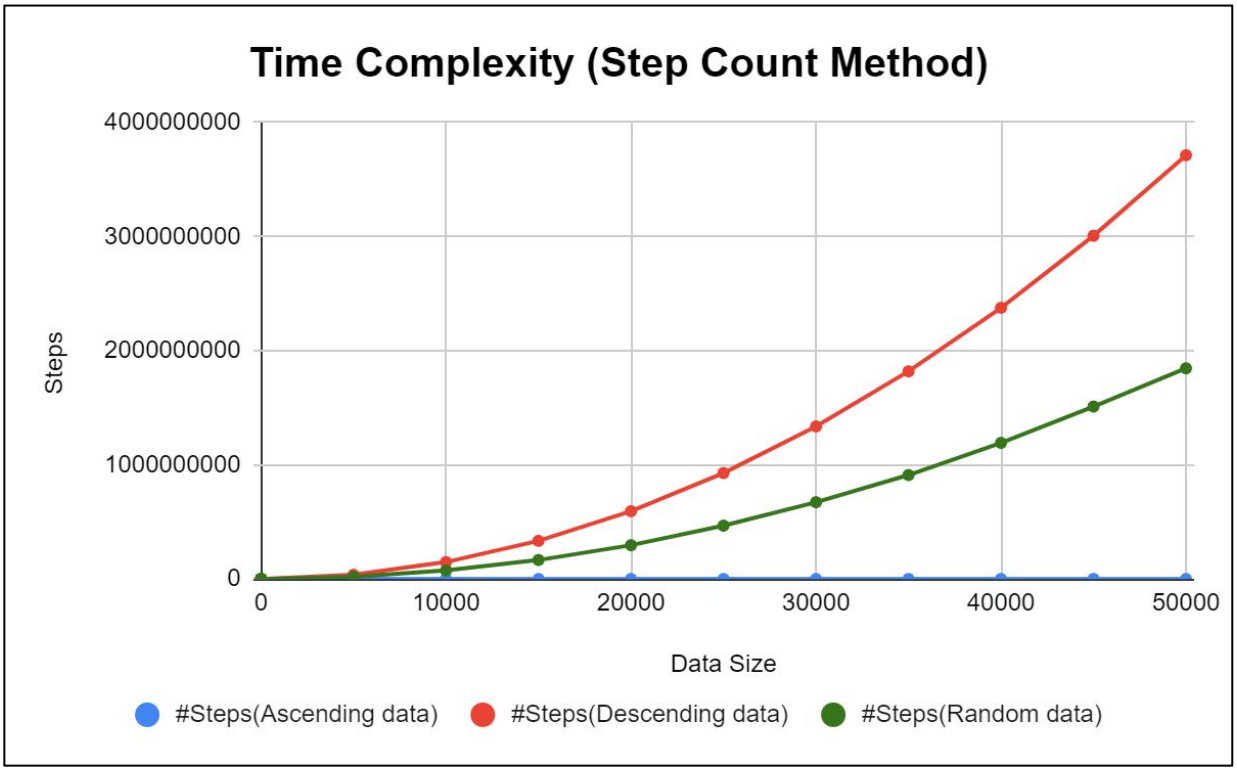
```
                        for (int n = 5000; n <= 50000; n += 5000)
                        {
                                // Initialize arrays
                                int *ascending_array = new int[n];
                                int *descending_array = new int[n];
                                int *random_array;

                                // Generate random array
                                 random_array = generate_random_array(n);

                                // Calculate step counts for different sorting methods
                                long long int random_steps = is_ascend_count(random_array, n,
                                5000);

                                // Copy arrays and perform sorting
                                copyArray(random_array, ascending_array, n);
                                copyArray(random_array, descending_array, n);
                                insertionSort(ascending_array, n);
                                descendingSort(descending_array, n);

                                // Calculate step counts for sorted arrays
                                long long int ascending_steps = is_ascend_count(ascending_array,
                                n, 5000);
                                long long int descending_steps =
                                is_ascend_count(descending_array, n, 5000);

                                // Display results in a table format
                                cout << m << " \t " << n << "\t\t " << ascending_steps << "\t\t\t\t "
                                << descending_steps << "\t\t\t " << random_steps << endl;
                                m++;
                        }
                        break;

                default: cout << "Invalid choice!" << endl;
                         break;
            }
        } while (choice != 0);

        return 0;
}
```

Analysis of Insertion Sort Algorithm:

Sl. No.	Data Size	#Steps(Ascending data)	#Steps(Descending data)	#Steps(Random data)
1	5	20	47	41
2	6	25	67	55
3	7	30	90	75
4	8	35	116	80
5	9	40	145	103

Sl. No.	Data Size	#Steps(Ascending data)	#Steps(Descending data)	#Steps(Random data)
1	5000	24995	37143431	18712625
2	10000	49995	148535523	74734794
3	15000	74995	334178464	166304005
4	20000	99995	594067379	296127188
5	25000	124995	928215840	467340420
6	30000	149995	1336609720	672779698
7	35000	174995	1819255646	910737290
8	40000	199995	2376150471	1192626177
9	45000	224995	3007288483	1510959466
10	50000	249995	3712665719	1845966335

Time Complexity Graph Plot:



Conclusion/Observation:

From the graph, we can observe the following trends:

Ascending Data: The ratio of steps to data size increases linearly but at a slower pace.

Descending Data: The ratio of steps to data size increases significantly faster than ascending data. It indicates a quadratic growth pattern.

Random Data: The ratio of steps to data size shows an intermediate growth between ascending and descending data. It's also closer to quadratic growth, suggesting that insertion sort's time complexity for random data is closer to its worst-case scenario.

In summary, the analysis of the time complexities of the insertion sort algorithm for different data types reveals that it performs best on ascending data (closer to linear time complexity) and worst on descending data (closer to quadratic time complexity). Random data falls somewhere in between, also exhibiting a growth pattern closer to quadratic. This aligns with the theoretical understanding of insertion sort's performance characteristics.