ALGORITHMS LABORATORY
[CS-2098]
Individual Work

Lab. No.- 3 **Date.- 07/08/2023**

Topic-Fundamentals of Algorithmic Problem Solving

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

Program No: 3.1

Program Title:

Rewrite the program no-2.1 (Insertion Sort) with the following details.

I. Compare the best case, worst case and average case time complexity with the same data except time complexity will count the CPU clock cycle time.

II. Plot a graph showing the above comparison (n, the input data Vs. CPU times for best, worst & average case)

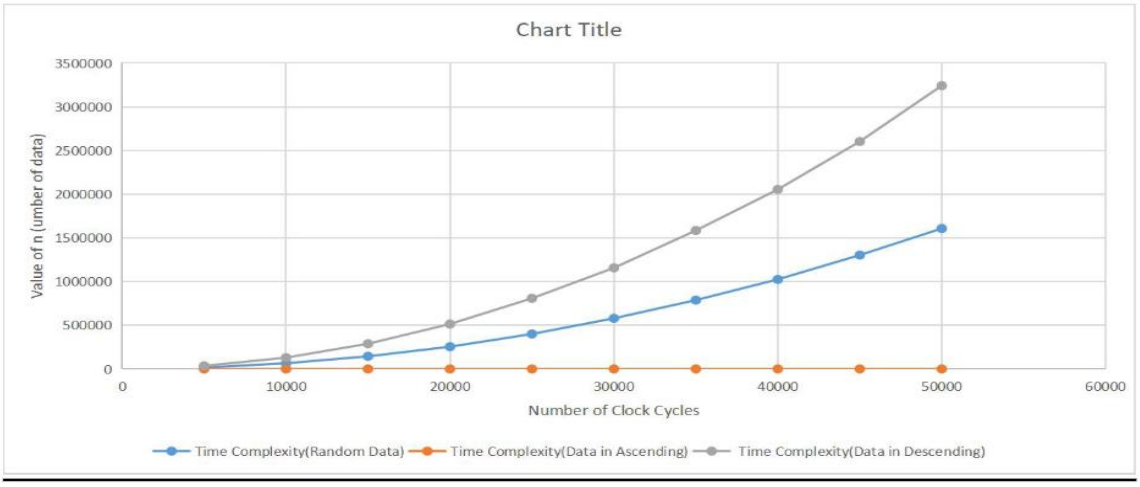
III. Compare manually program no-2.1 graph vs program no-3.1 graph and draw your conclusion.

Sample Input & Output:

If option 4 is entered, then the output may be displayed as follows:

Sl.	Data Size	#CPU Cycles (Ascending data)	#CPU Cycles (Descending data)	#CPU Cycles (Random data)
1	5000	18	32292	16351
2	10000	35	129217	64351
3	15000	53	293226	145881
4	20000	71	517780	257292

Note: The above data may vary each time program is executed. However the graph with the output data must be similar to the following graph. If data size increases, so also #CPU Cycles (Ascending data) and also #CPU Cycles (Descending data).



Input/Output Screenshots:

RUN-1:

```
------------------------------------------------------------------------
                        INSERTION SORT MENU
------------------------------------------------------------------------
0. Quit
1. n Random numbers
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: 15

Enter your choice: 2

79 28 24 18 55 7 35 47 82 47 90 66 33 88 37

Enter your choice: 3

7 18 24 28 33 35 37 47 47 55 66 79 82 88 90

Enter your choice: 4

90 88 82 79 66 55 47 47 37 35 33 28 24 18 7
Enter your choice: 5

Sl. No.  Data Size       CC(Ascending data)      CC(Descending data)           CC(Random data)
-------  ---------       ---------------------   ----------------------        ------------------
1        5               0                       1e-06                         2e-06
2        6               1e-06                   1e-06                         1e-06
3        7               1e-06                   1e-06                         0
4        8               0                       1e-06                         1e-06
5        9               1e-06                   1e-06                         1e-06

Enter your choice: 6

Sl. No.  Data Size       CC(Ascending data)      CC(Descending data)           CC(Random data)
-------  ---------       ---------------------   ----------------------        ------------------
1        5000            1.8e-05                 0.031086                      0.015405
2        10000           3.5e-05                 0.12568                       0.06128
3        15000           5.2e-05                 0.279651                      0.139076
4        20000           7.2e-05                 0.497688                      0.247219
5        25000           8.7e-05                 0.780257                      0.400338
6        30000           0.000104                1.12518                       0.559363
7        35000           0.000121                1.52501                       0.760922
8        40000           0.000142                1.99282                       0.996192
9        45000           0.000158                2.57034                       1.25373
10       50000           0.000172                3.10961                       1.5576

Enter your choice: 0

Quitting...
```

RUN-2

```
------------------------------------------------------------------------
                        INSERTION SORT MENU
------------------------------------------------------------------------
0. Quit
1. n Random numbers
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

70 98 56 94 13 0 26 45 74 72 98 14 76 61 58 72 63 20 11 76

Enter your choice: 3

0 11 13 14 20 26 45 56 58 61 63 70 72 72 74 76 76 94 98 98

Enter your choice: 4

98 98 94 76 76 74 72 72 70 63 61 58 56 45 26 20 14 13 11 0

Enter your choice: 5

Sl. No.  Data Size       CC(Ascending data)      CC(Descending data)            CC(Random data)
-------  ---------       ----------------------  ----------------------         ------------------
1        5               0                       0                              0
2        6               0                       0                              0
3        7               0                       1e-06                          0
4        8               1e-06                   1e-06                          1e-06
5        9               0                       0                              0

Enter your choice: 6

Sl. No.  Data Size       CC(Ascending data)      CC(Descending data)            CC(Random data)
-------  ---------       ----------------------  ----------------------         ------------------
1        5000            1.8e-05                 0.032079                       0.016084
2        10000           3.5e-05                 0.137288                       0.06453
3        15000           5.2e-05                 0.285572                       0.142679
4        20000           6.9e-05                 0.503195                       0.249207
5        25000           8.6e-05                 0.783046                       0.387799
6        30000           0.000103                1.12905                        0.564286
7        35000           0.000121                1.54786                        0.766349
8        40000           0.000138                2.01499                        1.00457
9        45000           0.000155                2.54023                        1.27896
10       50000           0.000173                3.22845                        1.56699

Enter your choice: 0

Quitting...
```

Source code

```
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;

// function for insertion sort and step count
void is_ascend_count(int arr[], int n)
{
	// Implementation of insertion sort algorithm
	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 for insertion sort
void insertionSort(int arr[], int n)
{
	// Implementation of insertion sort algorithm
	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 for descending sort
void descendingSort(int arr[], int n)
{
	// Implementation of descending sort using array reversal
	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 for displaying the array
void printArray(int arr[], int n)
{
        cout << endl;
        for (int i = 0; i < n; i++)
        {
                cout << arr[i] << " ";
        }
        cout << endl;
}

// function to generate array with random numbers
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 array
void copyArray(int arr1[], int arr2[], int n)
{
        // Copy elements from arr1 to arr2
        for (int i = 0; i < n; i++)
        {
                arr2[i] = arr1[i];
        }
}

// driver code
int main()
{
        int choice, n, steps, *arr, m;

        // Display menu and get user input
        cout << endl;
        cout << "--------------------------------------------------------------------------" << endl;
        cout << "\t\t\tINSERTION SORT MENU" << endl;
        cout << "--------------------------------------------------------------------------" << endl;
```

```
 cout << "0. Quit" << endl;
cout << "1. n Random numbers" << 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) to sort ascending of data for all Cases\n(Data
Ascending, Data in Descending & Random Data) in tabular form for\nvalues n=5 to 9, step=1"
<< endl;
cout << "6. Time Complexity (step count) to sort ascending of data for all Cases\n(Data
Ascending, Data in Descending & Random Data) in tabular form for\nvalues n=5000 to 500000,
step=5000" << endl;
cout << "--------------------------------------------------------------------------" << endl;

do
{
        cout << endl << "Enter your choice: ";
        cin >> 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;
                        arr = new int[n];
                        srand(time(NULL));
                        for (int i = 0; i < n; i++)
                        {
                                arr[i] = rand() % 100;
                        }
                        break;

                 case 2: printArray(arr, n);
                        break;

                case 3: insertionSort(arr, n);
                        printArray(arr, n);
                        break;

                case 4: insertionSort(arr, n);
                        descendingSort(arr, n);
                        printArray(arr, n);
                        break;

                case 5: cout << endl;
                        cout << "Sl. No.\t Data Size \t CC(Ascending data) \t CC(Descending data)
                        \t\t CC(Random data)" << endl;
                        cout << "-------\t --------- \t ---------------------- \t ----------------------- \t --------
                        ----------" << endl;
```

```
m = 1;

// Calculate and display time complexity for different data cases and sizes
for (int n = 5; n <= 9; n++)
{
        // Time measurement variables
        clock_t start_t1, end_t1, start_t2, end_t2, start_t3, end_t3;
        double random_steps, ascending_steps, descending_steps;
        int *ascending_array = new int[n];
        int *descending_array = new int[n];
        int *random_array;

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

        // Measure time for counting steps in ascending case
        start_t1 = clock();
        is_ascend_count(random_array, n);
        end_t1 = clock();
        random_steps = (double)(end_t1 - start_t1) / CLOCKS_PER_SEC;

        // Create copies of arrays for sorting
        copyArray(random_array, ascending_array, n);
        copyArray(random_array, descending_array, n);

        // Perform sorting and measure steps
        insertionSort(ascending_array, n);
        descendingSort(descending_array, n);

        // Measure steps for sorted arrays
        start_t2 = clock();
        is_ascend_count(ascending_array, n);
        end_t2 = clock();
        ascending_steps = (double)(end_t2 - start_t2) /
        CLOCKS_PER_SEC;

        start_t3 = clock();
        is_ascend_count(descending_array, n);
        end_t3 = clock();
        descending_steps = (double)(end_t3 - start_t3) /
        CLOCKS_PER_SEC;

        // Display results in tabular form
        cout << m << " \t " << n << "\t\t " << ascending_steps << "\t\t\t "
        << descending_steps << "\t\t\t\t " << random_steps << endl;
        m++;
}
break;
```

```
case 6: cout << endl;
        cout << "Sl. No.\t Data Size \t CC(Ascending data) \t CC(Descending data)
        \t\t CC(Random data)" << endl;
        cout << "-------\t --------- \t ---------- \t ---------- \t --------
        ----------" << endl;
        m = 1;

        // Calculate and display time complexity for larger data sizes
        for (int n = 5000; n <= 50000; n += 5000)
        {
                // Time measurement variables
                clock_t start_t1, end_t1, start_t2, end_t2, start_t3, end_t3;
                double random_steps, ascending_steps, descending_steps;
                int *ascending_array = new int[n];
                int *descending_array = new int[n];
                int *random_array;

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

                // Measure time for counting steps in ascending case
                start_t1 = clock();
                is_ascend_count(random_array, n);
                end_t1 = clock();
                random_steps = (double)(end_t1 - start_t1) / CLOCKS_PER_SEC;

                // Create copies of arrays for sorting
                copyArray(random_array, ascending_array, n);
                copyArray(random_array, descending_array, n);

                 // Perform sorting and measure steps
                insertionSort(ascending_array, n);
                descendingSort(descending_array, n);

                // Measure steps for sorted arrays
                start_t2 = clock();
                is_ascend_count(ascending_array, n);
                end_t2 = clock();
                ascending_steps = (double)(end_t2 - start_t2) /
                CLOCKS_PER_SEC;

                start_t3 = clock();
                is_ascend_count(descending_array, n);
                end_t3 = clock();
                descending_steps = (double)(end_t3 - start_t3) /
                CLOCKS_PER_SEC;

                // Display results in tabular form
                cout << m << " \t " << n << "\t\t " << ascending_steps << "\t\t " <<
                descending_steps << "\t\t\t " << random_steps << endl;
```

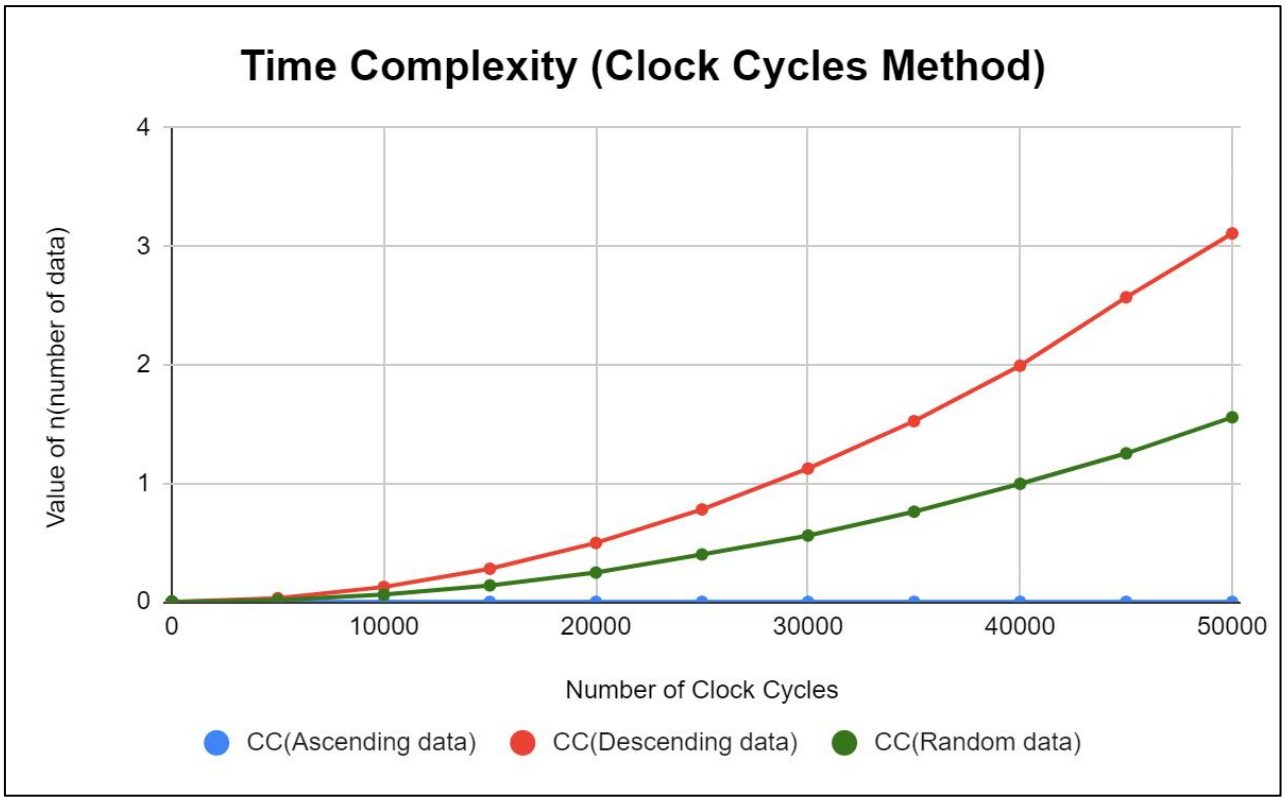
```
                    m++;
                }
                break;

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

    return 0;
}
```

Analysis of Insertion Sort Algorithm:

Sl. No.	Data Size	CC(Ascending data)	CC(Descending data)	CC(Random data)
1	5	0	1e-06	2e-06
2	6	1e-06	1e-06	1e-06
3	7	1e-06	1e-06	0
4	8	0	1e-06	1e-06
5	9	1e-06	1e-06	1e-06

Sl. No.	Data Size	CC(Ascending data)	CC(Descending data)	CC(Random data)
1	5000	1.8e-05	0.031086	0.015405
2	10000	3.5e-05	0.12568	0.06128
3	15000	5.2e-05	0.279651	0.139076
4	20000	7.2e-05	0.497688	0.247219
5	25000	8.7e-05	0.780257	0.400338
6	30000	0.000104	1.12518	0.559363
7	35000	0.000121	1.52501	0.760922
8	40000	0.000142	1.99282	0.996192
9	45000	0.000158	2.57034	1.25373
10	50000	0.000172	3.10961	1.5576

Time Complexity Graph Plot:



Conclusion/Observation:

Ascending Data:

- As the data is already sorted in ascending order, insertion sort's best-case time complexity applies.
- The number of CPU cycles is consistently the lowest among the three cases.
- This aligns with the expected O(n) time complexity for insertion sort on sorted data.

Descending Data:

- This scenario represents insertion sort's worst-case time complexity since each element needs to be inserted at the beginning of the sorted portion.
- The number of clock cycles is significantly higher compared to ascending data, indicating the expected O(n^2) time complexity for worst-case insertion sort.

Random Data:

- Random data has an element of randomness, so it's closer to the average-case scenario.
- The number of clock cycles falls between ascending and descending data, suggesting an intermediate time complexity.
- The measurements are also higher than those for ascending data but lower than those for descending data, which aligns with the O(n^2) average-case time complexity of insertion sort.