**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Sem V

**Course: Design and Analysis of Algorithms**

w.e.f. 1st Jul 2020

**Faculty:** Abhay Kolhe.

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.01**

**A.1 Aim:**

Implementation of Insertion Sort.

**A.2 Prerequisite:**

1 Concepts of C/C++/Python Programming

2. Knowledge of Array Handling.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Design & develop a program sorting given data collection.
2. Identify the applications of insertion sort technique.
3. Analyze insertion sort technique.

**A.4 Theory:**

**A.4.1.**

Insertion sort is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, **insertion sort**.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets.

**Working of Insertion Sort:**

We take an unsorted array for our example.

Unsorted Array

Insertion sort compares the first two elements.

Insertion Sort

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion Sort

Insertion sort moves ahead and compares 33 with 27.

Insertion Sort

And finds that 33 is not in the correct position.

Insertion Sort

It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

Insertion Sort

By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.

Insertion Sort

These values are not in a sorted order.

Insertion Sort

So we swap them.

Insertion Sort

However, swapping makes 27 and 10 unsorted.

Insertion Sort

Hence, we swap them too.

Insertion Sort

Again we find 14 and 10 in an unsorted order.

Insertion Sort

We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.

Insertion Sort

This process goes on until all the unsorted values are covered in a sorted sub-list. Now we shall see some programming aspects of insertion sort.

**A.5 Procedure/Algorithm:**

**A.5.1:**

Step 1 – If it is the first element, it is already sorted. Return -1;

Step 2 – Pick next element

Step 3 – Compare with all elements in the sorted sub-list.

Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted.

Step 5 – Insert the value

Step 6 – Repeat until list is sorted

Step 7 – Save your file as **EXP1\_DAA\_your Roll no.py**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. B032 | Name: Naman Garg |
| Class : Btech CS B | Batch : B2 |
| Date of Experiment: 14-7-2020 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

# naman garg B032

# lab 1 DAA

# AIM: Implementation of Insertion Sort.

# driver code.

if \_\_name\_\_ == "\_\_main\_\_":

    # taking input, storing as a list

    print("enter your numbers")

    l1 = list(map(int, input().split()))

# initalizing the no. of total swaps and comparisons

    ctr1, ctr2 = 0, 0

# pritning the inputed array as it is

    print("Initial Array: ", l1)

# the for loop works for the length of the list i.e the no. of

    for i in range(1, len(l1)):

        # initalizing variables for current iteration's swaps and comparisons

        temp1, temp2 = 0, 0

        x = i

        for j in range(i-1, -1, -1):

            # increasing the no of comparisons by 1

            temp2 += 1

            # swaping inside this if

            if l1[j] > l1[x]:

                l1[x], l1[j] = l1[j], l1[x]

            # increasing the no of swaps by 1

                temp1 += 1

                x -= 1

            else:

                break

        ctr1 += temp1

        ctr2 += temp2

        # pritning info

        print(f"\nPass {i}: ", l1)

        print("No of Swaps Done: ", temp1)

        print("No of Comparisions: ", temp2)

    # pritning final info

    print("\n Final Sorted Array: ", l1)

    print(" Total No of Swaps: ", ctr1)

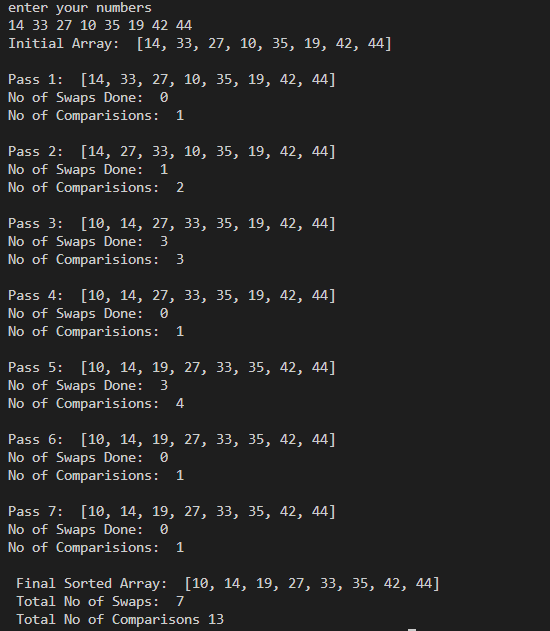
    print(" Total No of Comparisons", ctr2)

**B.2 Input and Output:**

**Input Data: 14 33 27 10 35 19 42 44**

**1 2 3 4 5 6 7 8**

**Output Data:**

****

**B.3 Observations and learning:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Q.1 Identify the applications of Insertion sort Technique.

Q.2 Analyze the algorithm for insertion sort – Best case, Worst Case and Average case.

Q.3 Comment on the performance of Insertion Sort, after filling up the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Array Size(n)** | **Data** | **No. of Comparisons** | **No. of Swaps** |
| 4 |  |  |  |
| 8 |  |  |  |
| 16 |  |  |  |
| 32 |  |  |  |
| 64 |  |  |  |
| 128 |  |  |  |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*