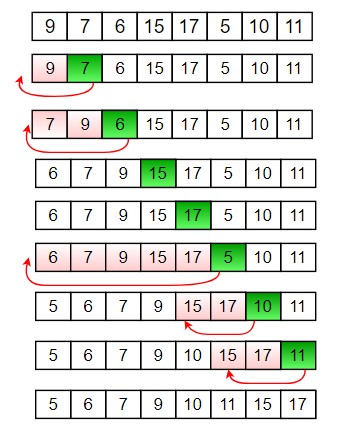
## **Practical 1**

**Aim:** Write a program to implement insertion sort and find the running time of the algorithm.

**Theory:**

1. Insertion sort is a sorting algorithm in which the elements are transferred one at a time to the right position.
2. In other words, an insertion sort helps in building the final sorted list, one item at a time, with the movement of higher-ranked elements.
3. An insertion sort has the benefits of simplicity and low overhead.

**Example:**



**Algorithm:**

INSERTION-SORT (A)

1. for j <- 2 to length[A]
2. do key <- A[j]
3. Insert A[j] into the sorted sequence A[1 . . j - 1].
4. i <- j - 1
5. while i > 0 and A[i] > key
6. do A[i + 1] <- A[i]
7. i <- i - 1
8. A[i + 1] <- key

**Code:**

def insertionSort(arr):

# Traverse through 1 to len(arr)

for i in range(1, len(arr)):

key = arr[i]

# Move elements of arr[0..i-1], that are

# greater than key, to one position ahead

# of their current position

j = i-1

while j >=0 and key < arr[j] :

arr[j+1] = arr[j]

j -= 1

arr[j+1] = key

#sorting the array [12, 11, 13, 5, 6] using insertionSort

arr = [12, 11, 13, 5, 6]

insertionSort(arr)

lst = [] #empty list to store sorted elements

print("Sorted array is : ")

for i in range(len(arr)):

lst.append(arr[i]) #appending the elements in sorted order

print(lst)

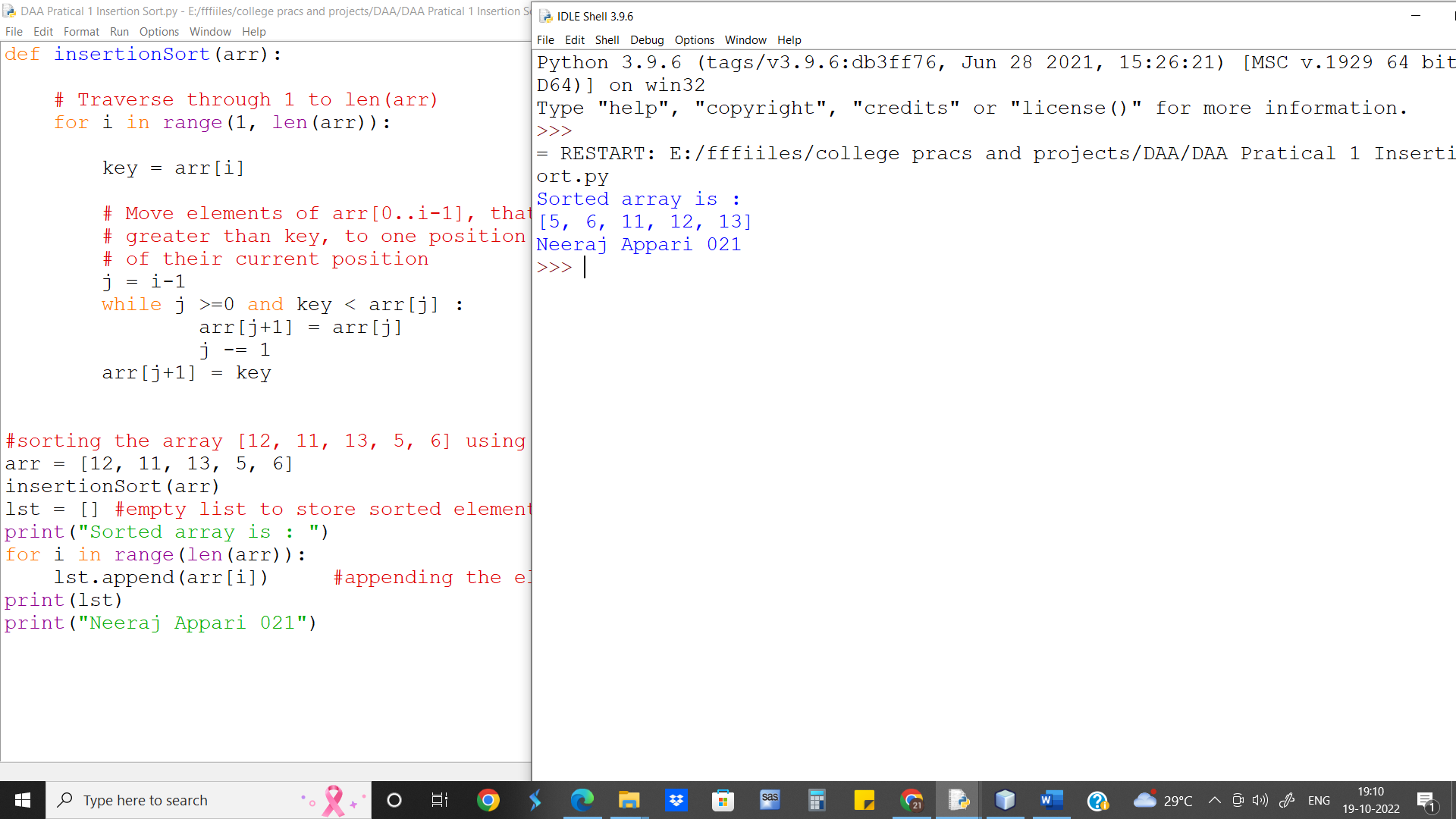
print("Neeraj Appari 021")

**Output**:

Sorted array is :

[5, 6, 11, 12, 13]

Neeraj Appari 021



**Run Time analysis of above Algorithm:**

Run Time in Seconds: 0.001309500

Runtime for Insertion sort is O(n2)

**Loop Invariant:**

In [insertion sort](https://www.geeksforgeeks.org/insertion-sort/), loop invariant condition is that the subarray A[0 to i-1] is always sorted.

**Conclusion:** Insertion Sort works best with small number of elements. The worst-case runtime complexity of Insertion Sort is o(n2) similar to that of Bubble Sort. However, Insertion Sort is considered better than Bubble sort.