PRACTICAL WORK BOOK

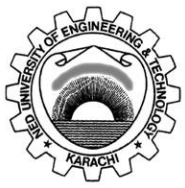
For Academic Session Fall 2018

Data Structures and Algorithms (EE-264)

For

SE Electrical

Name: Saad Sameer Khan
Roll Number: 180-41
Year: Second Section: A
Batch: 2018-19



Department of Electrical Engineering

N.E.D. University of Engineering & Technology, Karachi

CONTENTS

Laboratory Session No. 01

Objective:

To get introduced with fundamentals of programming with Python

Outcomes:

By the end of this lab, student should be able to

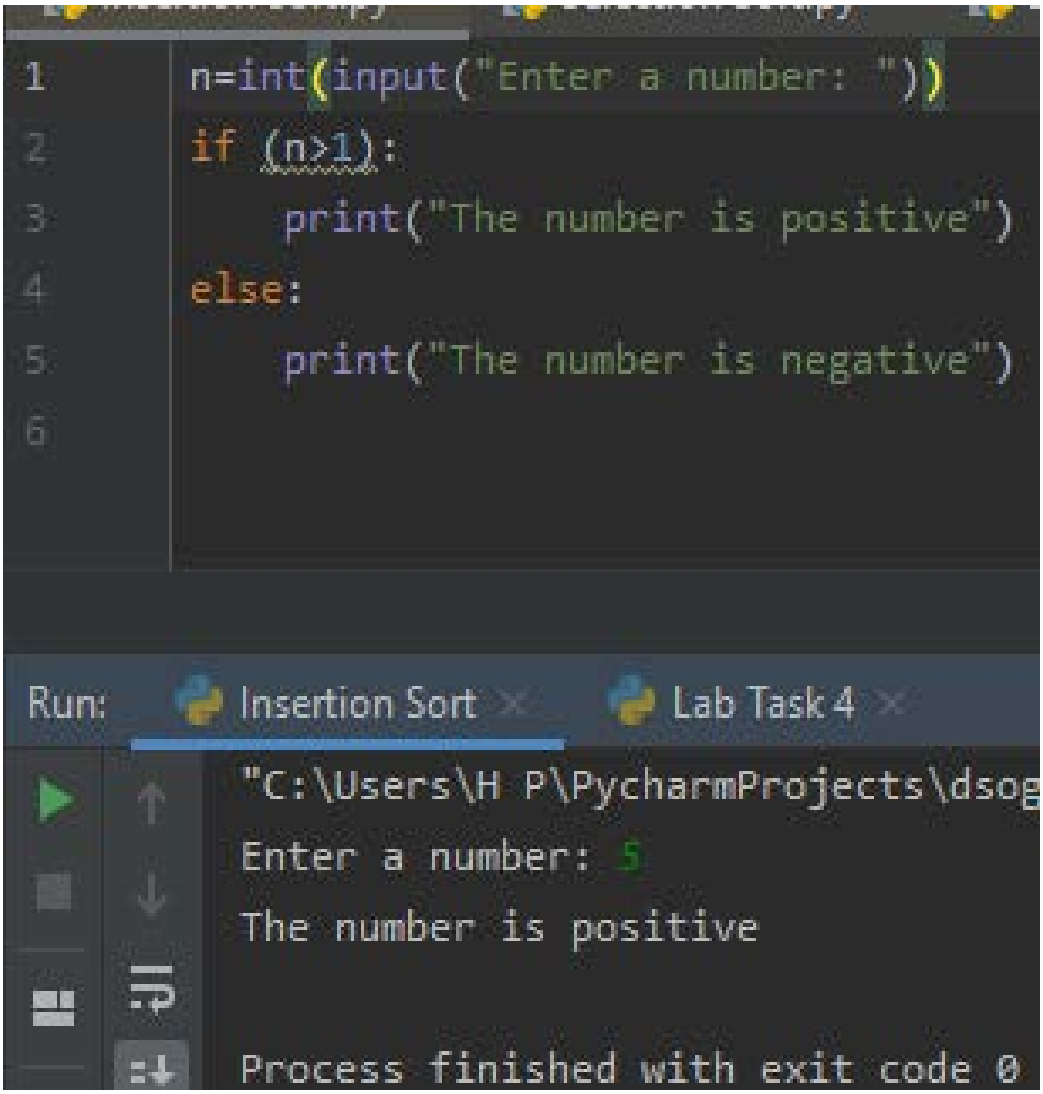
a) **Correctly code algorithms in python which may include**
   1) **Loops**
   2) **Conditions**
   3) **Lists**
   4) **User defined functions**
   5) **Importing libraries to program**

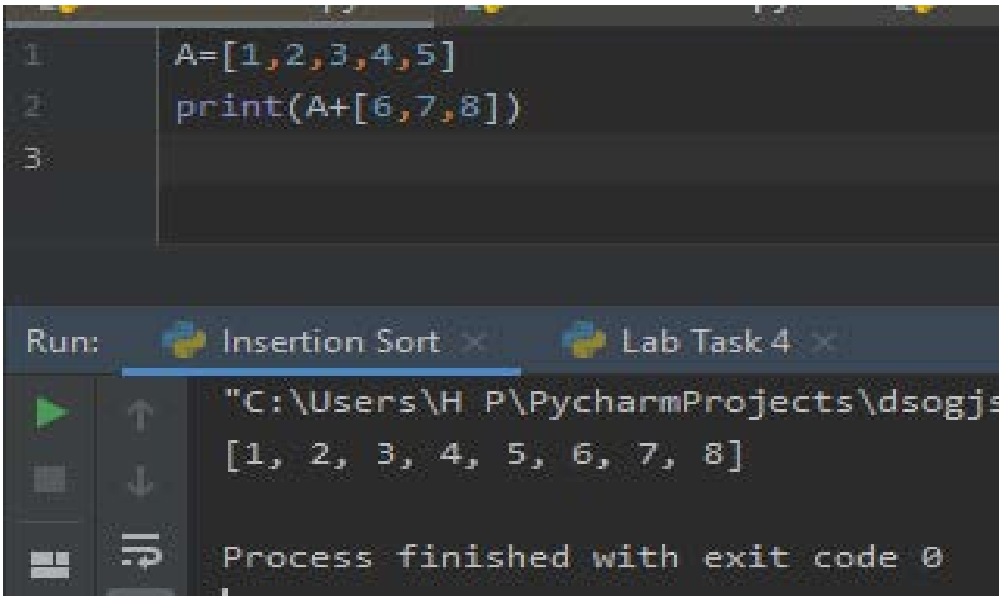
```
1   for i in range(10):
2       print("Hello World")
3
```

```
Run:  Insertion Sort  Lab Ta
"C:\Users\H P\PycharmPr
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

```
def name(n):
    print("Nice to meet you,", n)
n=str(input("What's your name? "))
name(n)
```

```
Insertion Sort  Lab Task 4
"C:\Users\H P\PycharmProjects\dsog
What's your name? Saad
Nice to meet you, Saad

Process finished with exit code 0
```


n=int(input("Enter a number: "))
if (n>1):
print("The number is positive")
else:
print("The number is negative")
Run:
Insertion Sort
Lab Task 4
"C:\Users\H P\PycharmProjects\dsog
Enter a number: 5
The number is positive
Process finished with exit code 0


1
2
3
A=[1,2,3,4,5]
print(A+[6,7,8])
Run:
Insertion Sort
Lab Task 4
"C:\Users\H P\PycharmProjects\dsogj
[1, 2, 3, 4, 5, 6, 7, 8]
Process finished with exit code 0

Laboratory Session No. 02

Objective:

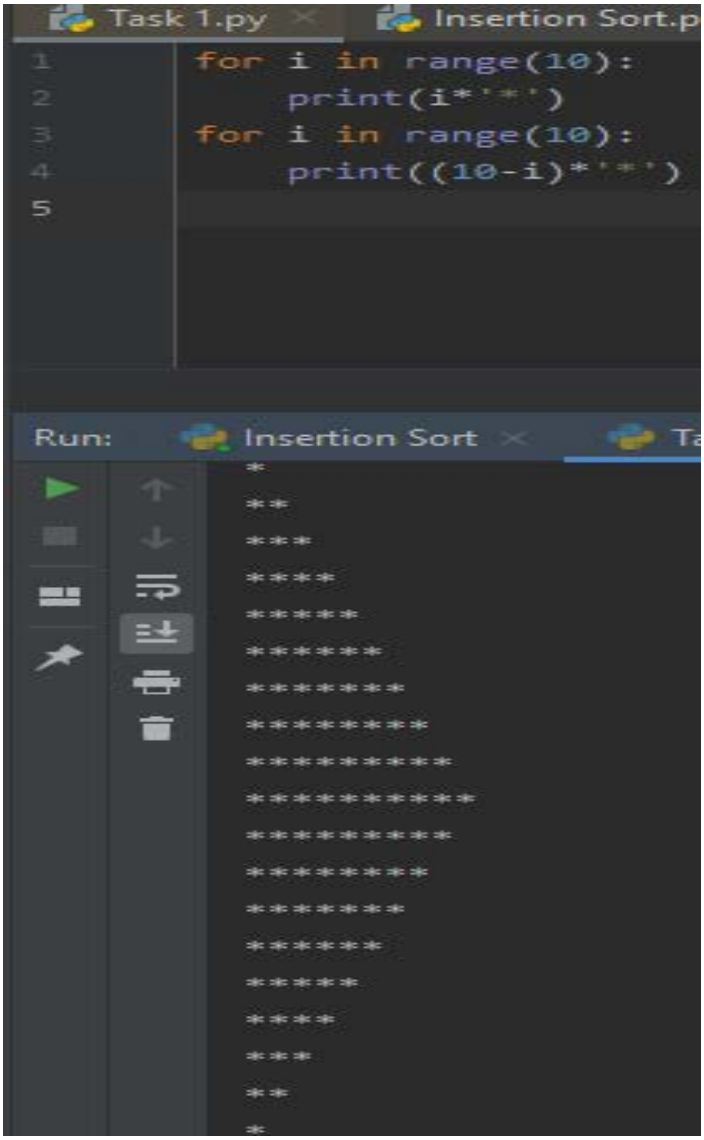
Developing and executing algorithms using Python

Outcomes:

By the end of this lab, student should be able to program different tasks given in the provided pdf.

Task 1:

```
for i in range(10):
    print(i*'*')
for i in range(10):
    print((10-i)*'*')
```

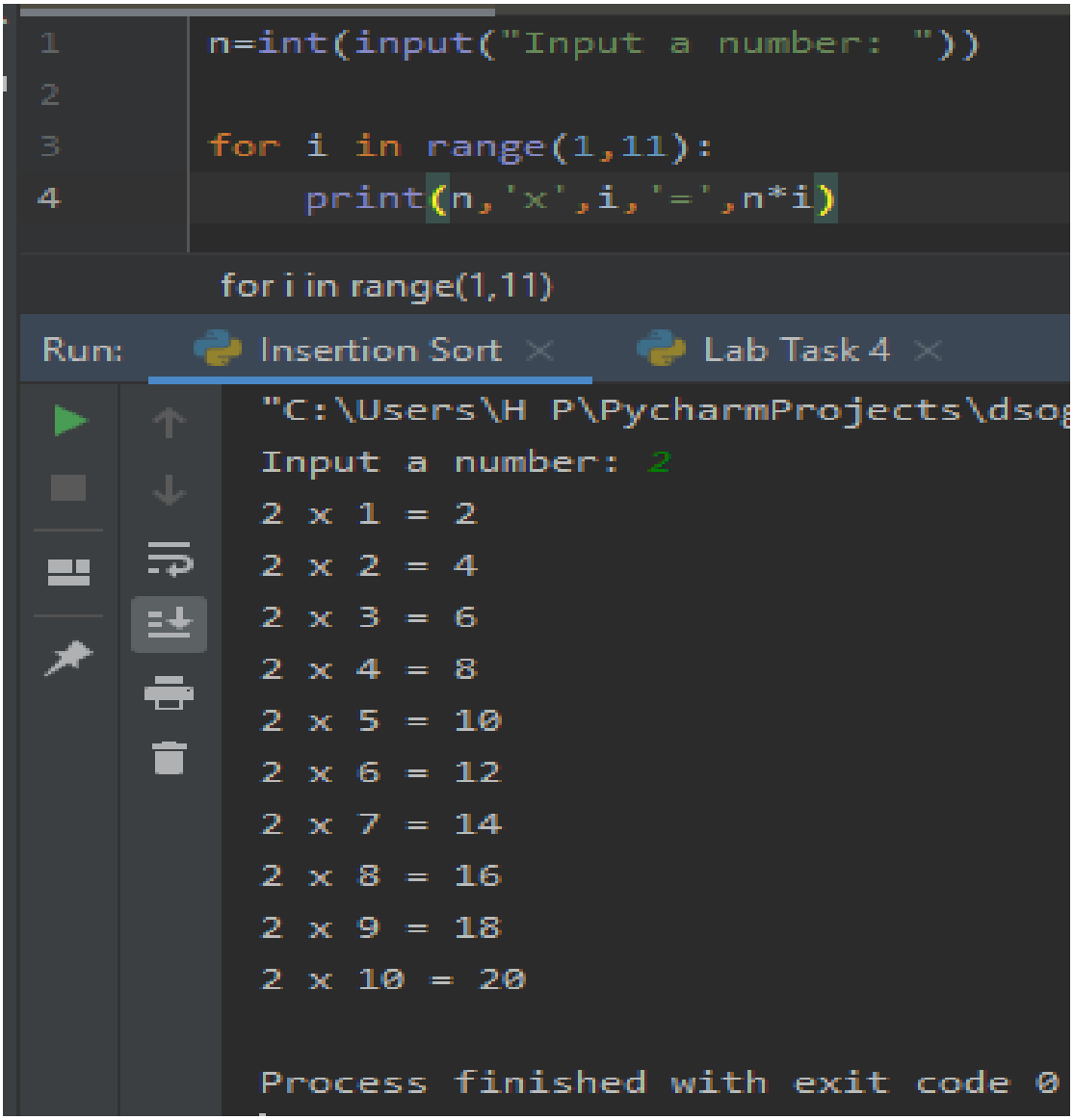


Task 2:

n=int(input("Input a number: "))

for i in range(1,11):

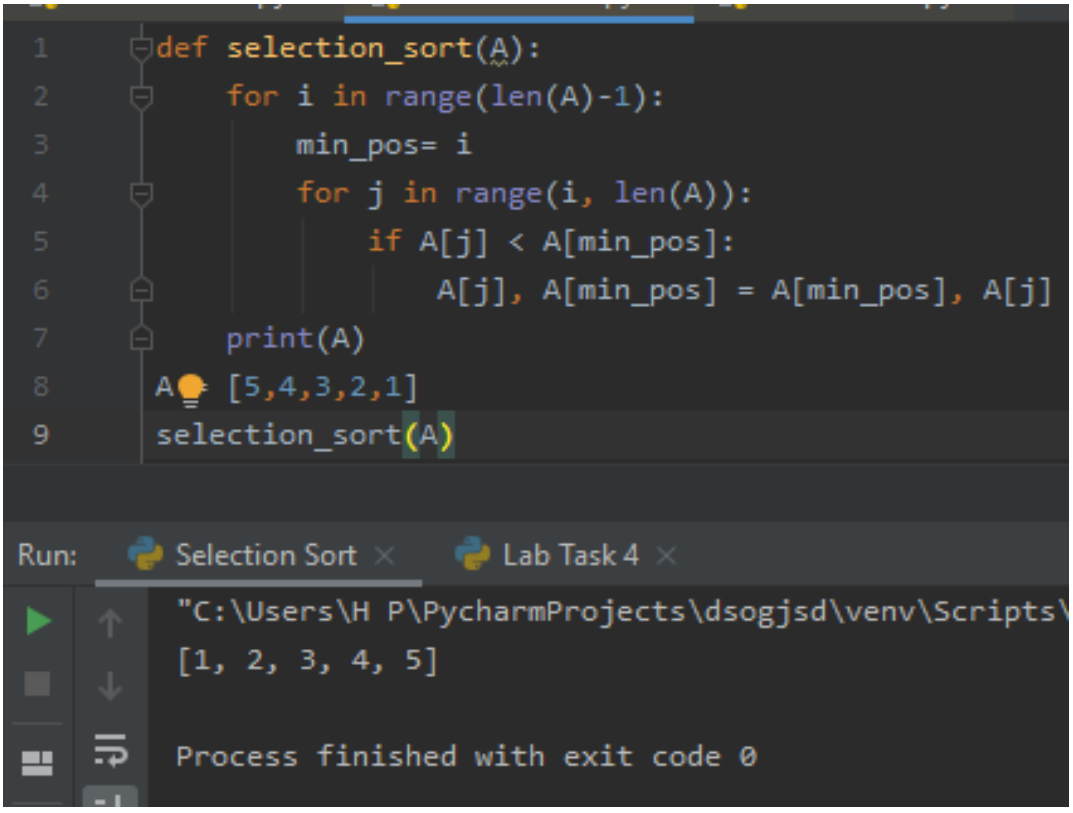
print(n,'x',i,'=',n*i)



Task 3:

A=[12,32,64,28,99]

```
for i in range(len(A)):
    min_idx=i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx=j
    A[i],A[min_idx]=A[min_idx],A[i]
print(A)
```



Task 4:

```
def searchkey(x):
    list=[ ‘a’ , ‘a’ , ‘b’ , ‘c’ , ‘c’ , ‘f’ ]
    count=0
    if x in A:
        print(x, " is in the list.")
        for p in A:
            if (p == x):
                count+=1
        print(x, “occurs”, count, “times.”)
    else:
        print(x, “is not in the list. ”)
n=str(input(“Enter an item: “)
searchkey(n)
```

```
def searchkey(x):
    A=['a','a','b','c','c','f']
    count=0
    if x in A:
        print(x," is in the list.")
        for p in A:
            if (p == x):
                count += 1
        print(x, "occurs ",count," times.")
    else:
        print(x, "is not in the list.")
n=str(input("Enter an item: "))
searchkey(n)
```

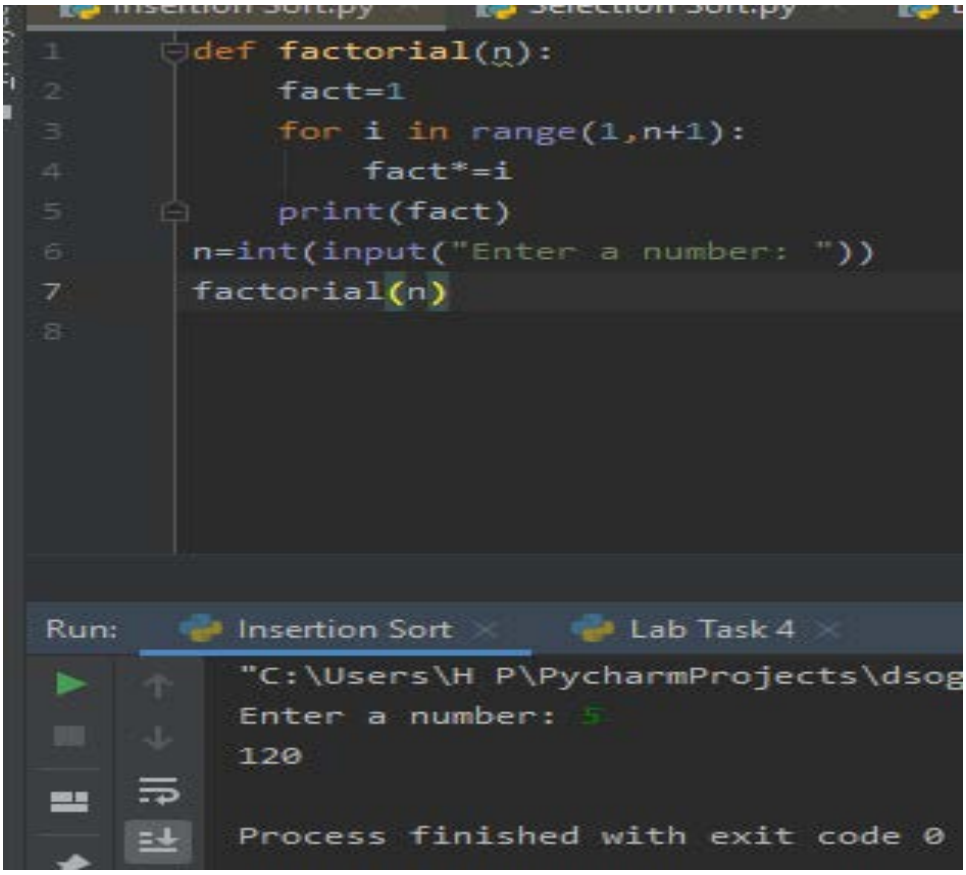
searchkey()

Run: Insertion Sort × Lab Task 4 ×

```
"C:\Users\H P\PycharmProjects\dsogjsd\venv\Scripts
Enter an item: a
a  is in the list.
a occurs  2  times.
```

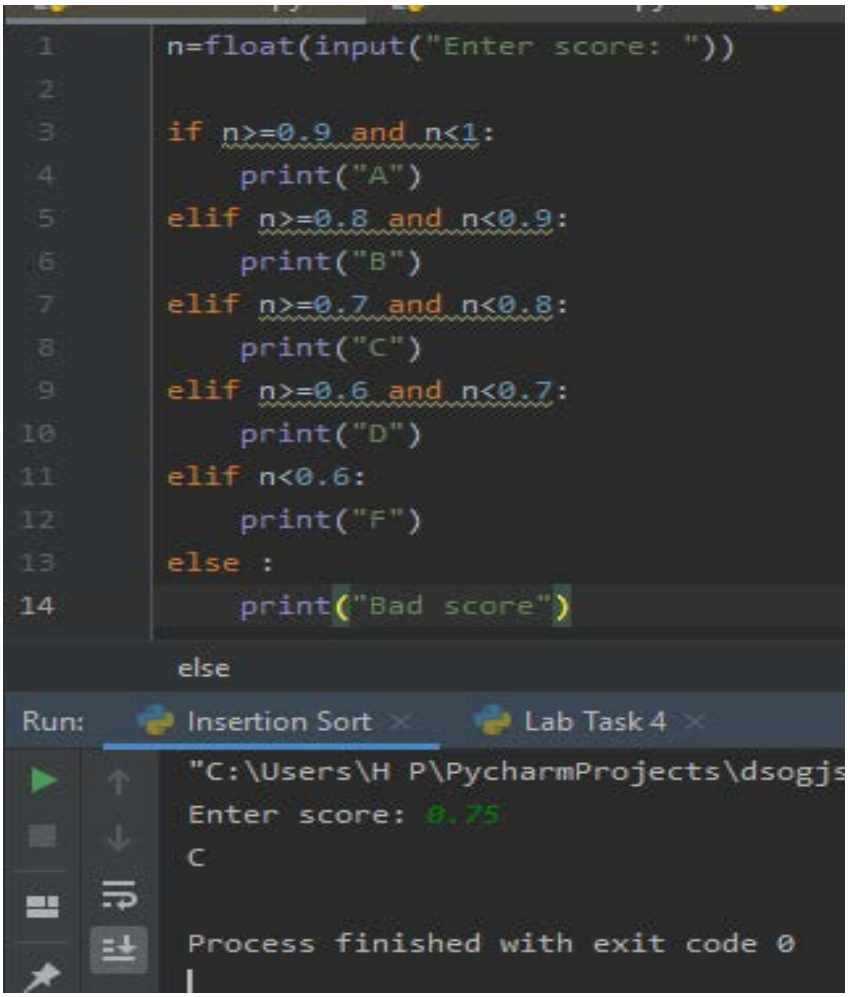
Task 5:

```
def factorial(n):
    fact=1
    for i in range(1,n+1):
        fact*=i
    print(fact)
n=int(input("Enter a number: "))
factorial(n)
```



Task 6:

```
n=float(input("Enter score: "))


if n>=0.9 and n<1:
   print("A")
elif n>=0.8 and n<0.9:
   print("B")
elif n>=0.7 and n<0.8:
   print("C")
elif n>=0.6 and n<0.7:
   print("D")
elif n<0.6:
   print("F")
else :
   print("Bad score")
```


n=float(input("Enter score: "))
if n>=0.9 and n<1:
print("A")
elif n>=0.8 and n<0.9:
print("B")
elif n>=0.7 and n<0.8:
print("C")
elif n>=0.6 and n<0.7:
print("D")
elif n<0.6:
print("F")
else :
print("Bad score")
else
Run:
Insertion Sort
Lab Task 4
"C:\Users\H P\PycharmProjects\dsogjs
Enter score: 0.75
C
Process finished with exit code 0

Task 7:

```
def Score(n):
    if n>=0.9 and n<1:
        print("A")
    elif n>=0.8 and n<0.9:
        print("B")
    elif n>=0.7 and n<0.8:
        print("C")
    elif n>=0.6 and n<0.7:
        print("D")
    elif n<0.6:
        print("F")
    else :
        print("Bad score")
n=int(input(“Enter a score: “)
Score(n)
```

Laboratory Session No. 03

Objective:

To get introduced with the time complexity of different algorithms.

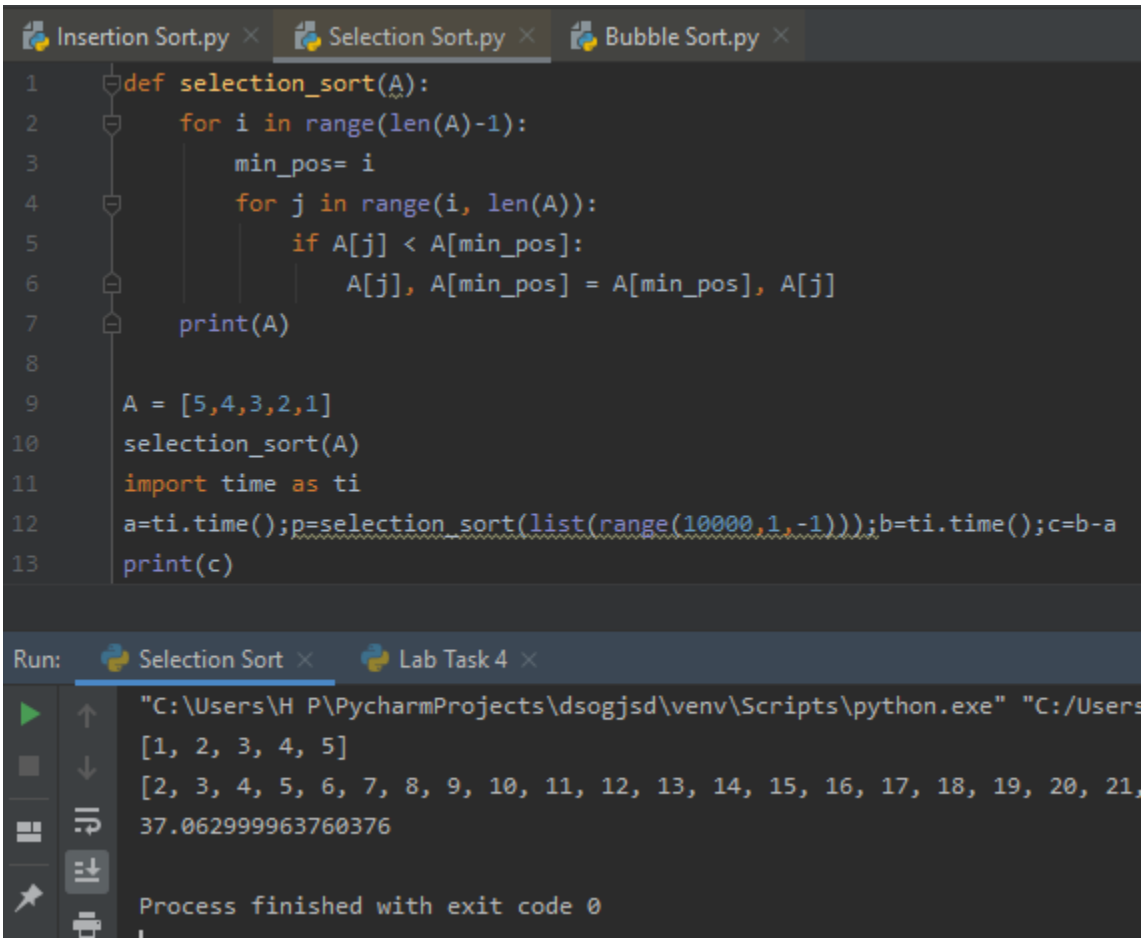
Outcomes:

By the end of this lab, student should be able to

- **Find out the time taken by different algorithms for their completion.**

Task 1:

```
def selection_sort(A):
    for i in range(len(A)-1):
        min_pos= i
        for j in range(i, len(A)):
            if A[j] < A[min_pos]:
                A[j], A[min_pos] = A[min_pos], A[j]
    print(A)
A = [5,4,3,2,1]
selection_sort(A)
import time as ti
a=ti.time();p=selection_sort(list(range(10000,1,-1)));b=ti.time();c=b-a
print(c)
```



Task 2:

```
def bubble_sort(A):
    for i in range(n):
        for j in range(n-i-1):
            if A[j]>A[j+1]:
                A[j],A[j+1]=A[j+1],A[j]
    print(A)
```

A=[6,5,4,3,2,1]

bubble_sort(A)

import time as ti

a=ti.time();p=bubble_sort(list(range(10000,1,-1)));b=ti.time();c=b-a

print(c)

```
def bubble_sort(A):
    for i in range(len(A)):
        for j in range(len(A)-i-1):
            if A[j]>A[j+1]:
                A[j],A[j+1]=A[j+1],A[j]
    print(A)

A=[6,5,4,3,2,1]
bubble_sort(A)
import time as ti
a=ti.time();p=bubble_sort(list(range(10000,1,-1)));b=ti.time();c=b-a
print(c)
```

```
Run:  Bubble Sort  Lab Task 4
"C:\Users\H P\PycharmProjects\dsogjsd\venv\Scripts\python.exe" "C:/
[1, 2, 3, 4, 5, 6]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
58.22159171104431

Process finished with exit code 0
```

Task 3:

```
def insertion_sort(A):
    for i in range(1,len(A)):
        j=i-1
        while j>=0 and A[j]>A[j+1]:
            A[j],A[j+1]=A[j+1],A[j]
            j=j-1
    print(A)
A=[8,6,5,4,7,3,2,9,54]
insertion_sort(A)
import time as ti
a=ti.time();p=insertion_sort(list(range(10000,1,-1)));b=ti.time();c=b-a
print(c)
```

```
1   def insertion_sort(A):
2       for i in range(1,len(A)):
3           j=i-1
4           while j>=0 and A[j]>A[j+1]:
5               A[j],A[j+1]=A[j+1],A[j]
6               j=j-1
7       print(A)
8   A=[8,6,5,4,7,3,2,9,54]
9   insertion_sort(A)
10  import time as ti
11  a=ti.time();p=insertion_sort(list(range(10000,1,-1)));b=ti.time();c=b-a
12  print(c)
```

```
Run:  Insertion Sort  Lab Task 4
"C:\Users\H P\PycharmProjects\dsogjsd\venv\Scripts\python.exe" "C:/Users
[2, 3, 4, 5, 6, 7, 8, 9, 54]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
78.00110054016113

Process finished with exit code 0
```

Laboratory Session No. 04

Objective:

To develop and apply the recursive divide and conquer approach in sorting

Note:

1) **You are supposed to first elaborate the debugging options given in python using *spyder IDE*. Show the execution of merge algorithm in debugging mode**
2) **Given the pseudocode of merge-sort, translate it in *python***
3) **Show the recursive calls for a given size of input array in merge-sort using debugging tools**
4) **Compare the running time with the sorting algorithms in Lab#03**

Task 1:

Debugging with Spyder IDE

There are many features in Spyder. Some of them are:

Adjusting breakpoints through keyboard shortcuts or other methods.

The current frame may be highlited.

Accessing variables using the Variable Explorer and run many commands in the iPython Console.

Control over the execution of the debugging process via shortcuts which may be configured. And many more features.

Execution Of Merge Algorithm in Spyder:

```
1 def merge(A,B):
2     n1, n2 =len(A),len(B)
3     A=A+[float('inf')];B=B+[float('inf')]
4     i,j=0,0; l = list()
5     for k in list(range(n1+n2)):
6         if A[i]<=B[j]:
7             l=l+[A[i]]
8             i=i+1
9         else:
10            l=l+[B[j]]
11            j=j+1
12    return l
13
14 print(merge([1,3,5],[0,2,4]))
```

```
ipdb> > c:\users\mhuza\desktop\ned-ee se\3rdsemester\ee-264  data structures
and algorithms dsa\spyder\untitled0.py(5)merge()
      3     A=A+[float('inf')];B=B+[float('inf')]
      4     i,j=0,0; l = list()
----> 5     for k in list(range(n1+n2)):
      6         if A[i]<=B[j]:
      7             l=l+[A[i]]
```

```
ipdb> > c:\users\mhuza\desktop\ned-ee se\3rdsemester\ee-264  data structures
and algorithms dsa\spyder\untitled0.py(6)merge()
      4     i,j=0,0; l = list()
      5     for k in list(range(n1+n2)):
----> 6         if A[i]<=B[j]:
      7             l=l+[A[i]]
      8             i=i+1
```

```
ipdb> > c:\users\mhuza\desktop\ned-ee se\3rdsemester\ee-264  data structures
and algorithms dsa\spyder\untitled0.py(10)merge()
      8             i=i+1
      9         else:
---> 10             l=l+[B[j]]
     11             j=j+1
```

Task 2:

```
def Merge(a, p, q, r):
    (l, ri)=([], [])
    for i in range(p, q):
        l.append(a[i])
    for j in range(q, r):
        ri.append(a[j])

    l.append(float('inf'))
    ri.append(float('inf'))
    i=0
    j=0
    for k in range(p, r):
        if l[i] <= ri[j]:
            a[k] = l[i]
            i += 1
        else:
            a[k] = ri[j]
            j += 1

def MergeSort(a, p, r):
    if p + 1 < r:
        q = divide(r + p)
        MergeSort(a, p, q)
        MergeSort(a, q, r)
        Merge(a, p, q, r)

def divide(number):
    Q, R = divmod(number, 2)
    return Q + R

a = [5,4,3,2,1]
MergeSort(a, 0, len(a))
print (a)

import time as ti
a = ti.time()
p = MergeSort(list(range(10000,1,-1)),0,len(list(range(10000,1,-1))))
b = ti.time()
c = b-a
print('\n\n Time =',c)
```

```
In [24]: runfile('C:/Users/mhuza
merge sort the long way.py', wdi
Algorithms DSA/Spyder')
[1, 2, 3, 4, 5]


 Time = 0.06999349594116211
```

Task 3:

First call:

Name	Type	Size	
A	list	6	[5, 4, 3, 2, 1, 0]
a	int	1	3
n	int	1	6
s	list	0	[]

Second call:

Name	Type	Size	
A	list	3	[5, 4, 3]
a	int	1	1
n	int	1	3
s	list	0	[]

Third call:

Name	Type	Size	
A	list	1	[5]
n	int	1	1
s	list	1	[5]

Fourth call:

Name	Type	Size	
A	list	2	[4, 3]
a	int	1	1
n	int	1	2
s	list	0	[]

And so on….

Task 4:

S.no	***Sort***	***Time taken to sort 10,000 numbers***
1.	Bubble Sort	58.221592
2.	Insertion Sort	78.001100
3.	Selection Sort	37.06999
4.	Merge Sort	0.0890052

As we can clearly see above, Merge Sort is the fastest.

Laboratory Session No. 05

Objective:

Extending the divide-and-conquer approach on sorting and searching problems

Note:

1) **Show that divide-and-conquer based approach can be applied on bubble-sort, selection sort and insertion sort.**
2) **Compare linear search with binary search. Analyze the binary search algorithm.**
3) **Apply binary search technique and modify insertion sort algorithm. Analyze the modified algorithm and compare it with the insertion sort algorithm.**

Task 1:

Divide And Conquer (Insertion Sort):

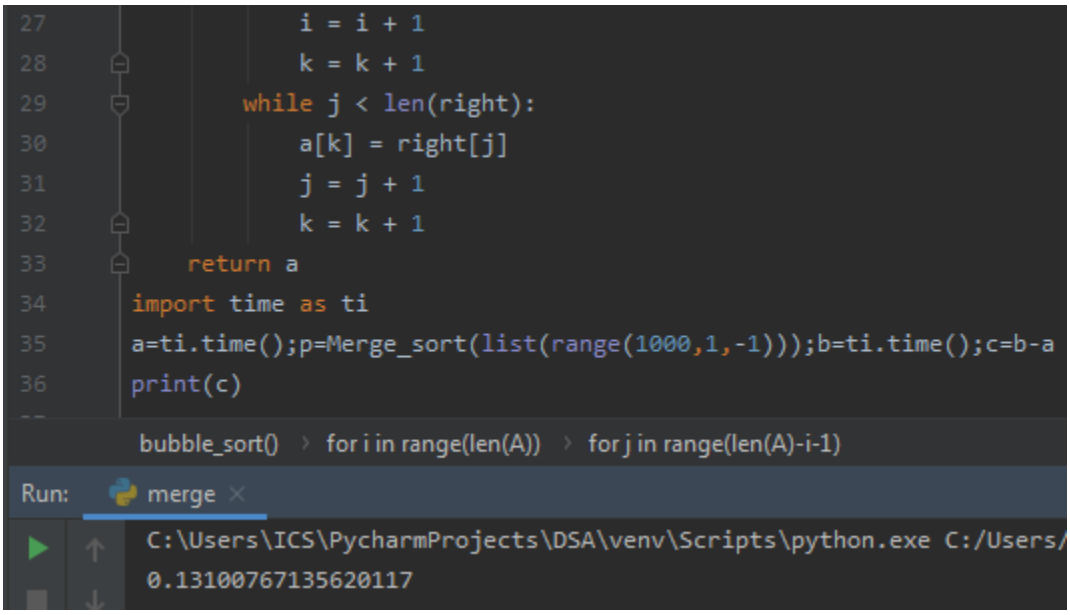
```python
def insertion_sort(A):
    for i in range(1,len(A)):
        j=i-1
        while j>=0 and A[j]>A[j+1]:
            A[j],A[j+1]=A[j+1],A[j]
            j=j-1
    return A

def Merge_sort(a):
    if len(a) > 1:
        q=len(a)//2
        left=a[:q]
        right=a[q:]
        insertion_sort(left)
        insertion_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                a[k] = left[i]
                i = i + 1
            else:
                a[k] = right[j]
                j = j + 1
```

```
                k = k + 1
            while i < len(left):
                a[k] = left[i]
                i = i + 1
                k = k + 1
            while j < len(right):
                a[k] = right[j]
                j = j + 1
                k = k + 1
        return a
import time as ti
a=ti.time();p=Merge_sort(list(range(1000,1,-1)));b=ti.time();c=b-a
print(c)
```

```
Run:  merge
C:\Users\ICS\PycharmProjects\DSA\venv\Scripts\python.exe C:/Users/
0.18001031875610352
```

Divide And Conquer (Bubble Sort):

```
def bubble_sort(A):
    for i in range(len(A)):
        for j in range(len(A)-i-1):
            if A[j]>A[j+1]:
                A[j],A[j+1]=A[j+1],A[j]
    return A


def Merge_sort(a):
    if len(a) > 1:
        q=len(a)//2
        left=a[:q]
        right=a[q:]
        bubble_sort(left)
        bubble_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                a[k] = left[i]
                i = i + 1
            else:
                a[k] = right[j]
                j = j + 1
                k = k + 1
        while i < len(left):
            a[k] = left[i]
```


i = i + 1
k = k + 1
while j < len(right):
a[k] = right[j]
j = j + 1
k = k + 1
return a
import time as ti
a=ti.time();p=Merge_sort(list(range(1000,1,-1)));b=ti.time();c=b-a
print(c)
bubble_sort() › for i in range(len(A)) › for j in range(len(A)-i-1)
Run: merge
C:\Users\ICS\PycharmProjects\DSA\venv\Scripts\python.exe C:/Users/
0.13100767135620117

Divide And Conquer (Selection Sort):

```
def selection_sort(A):
    for i in range(len(A)-1):
        min_pos= i
        for j in range(i, len(A)):
            if A[j] < A[min_pos]:
                A[j], A[min_pos] = A[min_pos], A[j]
    return(A)
def Merge_sort(a):
    if len(a) > 1:
        q=len(a)//2
        left=a[:q]
        right=a[q:]
        selection_sort(left)
        selection_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                a[k] = left[i]
                i = i + 1
            else:
                a[k] = right[j]
                j = j + 1
            k = k + 1

        while i < len(left):
            a[k] = left[i]
            i = i + 1
            k = k + 1

        while j < len(right):
            a[k] = right[j]
            j = j + 1
            k = k + 1
    print(a)
```

```
import time as ti
a=ti.time();p=Merge_sort(list(range(1000,1,-1)));b=ti.time();c=b-a
print(c)
```

```
merge ×
C:\Users\ICS\PycharmProjects\DSA\venv\Scripts\python.exe C:/Users/
0.09500527381896973
```

Task 2:

Comparing Linear Search with Binary Search:

In **binary search** it searches a sorted array by repeatedly dividing the search interval in half. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Code for Binary Search:

```
In [13]: import time
         def binarySearch(alist, item):
             first = 0
             last = len(alist)-1
             found = False
             while first<=last and not found:
                 midpoint = (first + last)//2
                 if alist[midpoint] == item:
                     found = True
                 else:
                     if item < alist[midpoint]:
                         last = midpoint - 1
                     else:
                         first = midpoint + 1
             return found
```

```
In [19]: a=time.time(); y=binarySearch(list(range(1000, 0, -2)), 788); b=time.time(); c=b-a
         print(y)
         print(c)

         False
         0.0010042190551757812
```

In computer science, **linear search** or **sequential search** is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

Code for Linear/Sequential Search:

```
In [8]: def linear_search(arr, x):

            for i in range(len(arr)):

                if arr[i] == x:
                    return i
                return ("Invalid Search")
```

```
In [11]: import time as ti
```

```
In [12]: a=time.time(); y=linear_search(list(range(1000, 0, -2)), 788); b=time.time(); c=b-a
         print(y)
         print(c)
```

```
Invalid Search
0.0010044574737548828
```

As we can see, binary search takes less time to search an element as compared to Linear Search. Hence, Binary Search is more efficient.

Task 3:

In [18]:
```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        temp = arr[i]
        pos = binary_search(arr, temp, 0, i) + 1
        for k in range(i, pos, -1):
            arr[k] = arr[k - 1]
        arr[pos] = temp
def binary_search(arr, key, start, end):
   #key
   if end - start <= 1:
        if key < arr[start]:
            return start - 1
        else:
            return start
   mid = (start + end)//2
   if arr[mid] < key:
      return binary_search(arr, key, mid, end)
   elif arr[mid] > key:
      return binary_search(arr, key, start, mid)
   else:
      return mid
# main
arr = [8,5,7,9,3,2]
insertion_sort(arr)
print("Sorted array is:", arr)
```

```
Sorted array is: [2, 3, 5, 7, 8, 9]
```

In [19]:
```
import time as ti
```

In [20]:
```
a = ti.time();p = insertion_sort(list(range(10000,1,-1)));b = ti.time();c = b-a
```

In [21]:
```
c
```

Out[21]:
```
6.491865158081055
```

By looking at the time efficiency, we can clearly observe that it increased as compared to the efficiency of the insertion sort we did in lab 3.

Laboratory Session No. 06

Objective:

Apply Asymptotic Notations to the Sorting Algorithms.

Note:

Find the suitable values of c_1 and c_2 to show that following algorithms can be tightly bound to their respective asymptotic notations using graph in excel.

1) **Insertion Sort**
2) **Merge Sort**

Insertion Sort:

n	T(n)	T(n)/n^2
1000	0.312000573	0.000000312000573
10000	33.404896259	0.00000033404896
15000	79.022519826	0.0000003512119
20000	143.370200157	0.0000003584255004
25000	212.512155055	0.000000340019448

From above values, we find out the average constant: **c=0.0000003391412763**

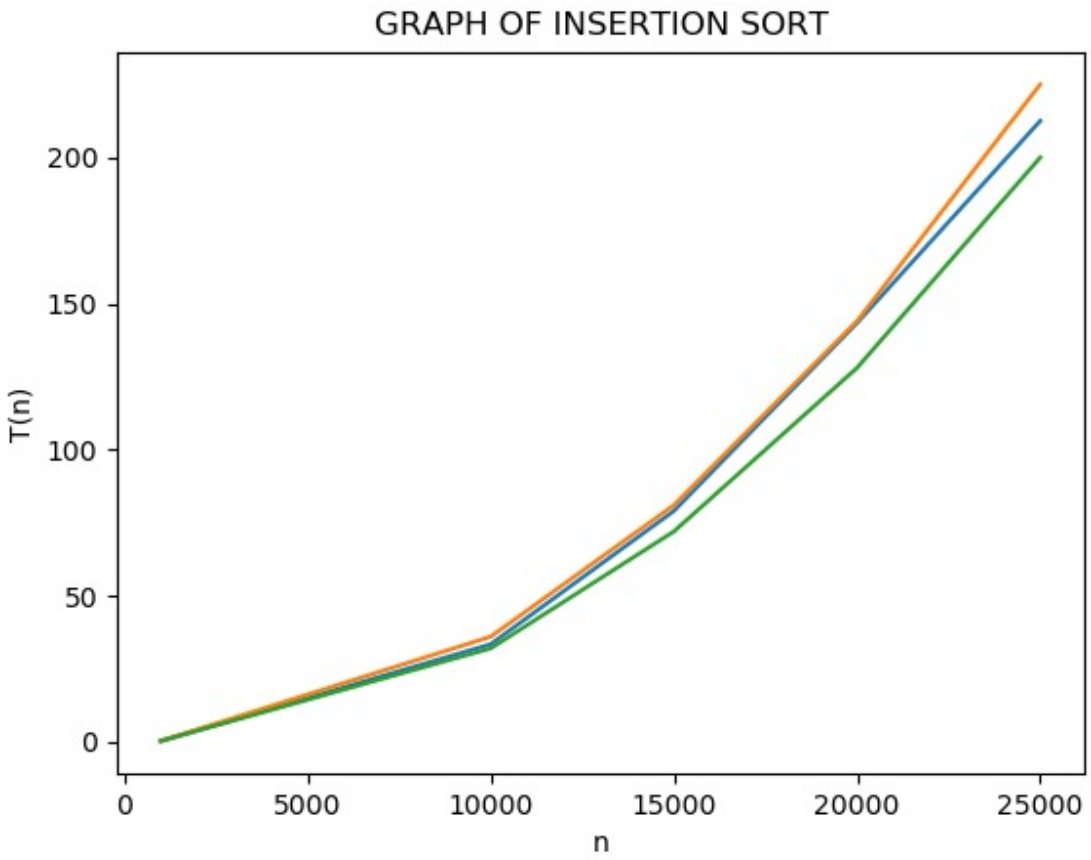
We now choose the upper bound limit as: **c1=0.00000036**

And the lower bound limit as: **c2=0.00000032**

Code for Time Plot:

```
import matplotlib.pyplot as plt
x = [1000,10000,15000,20000,25000]
y1=[0.312,33.404,79.0225,143.370,212.512]
y2=[0.36,36,81,144,225]
y3=[0.32,32,72,128,200]
plt.plot(x, y1)
plt.plot(x, y2)
plt.plot(x,y3)
plt.xlabel('n')
plt.ylabel('T(n)')
plt.title('GRAPH OF INSERTION SORT')
plt.show()
```

Now the graph is shown as:



Merge Sort:

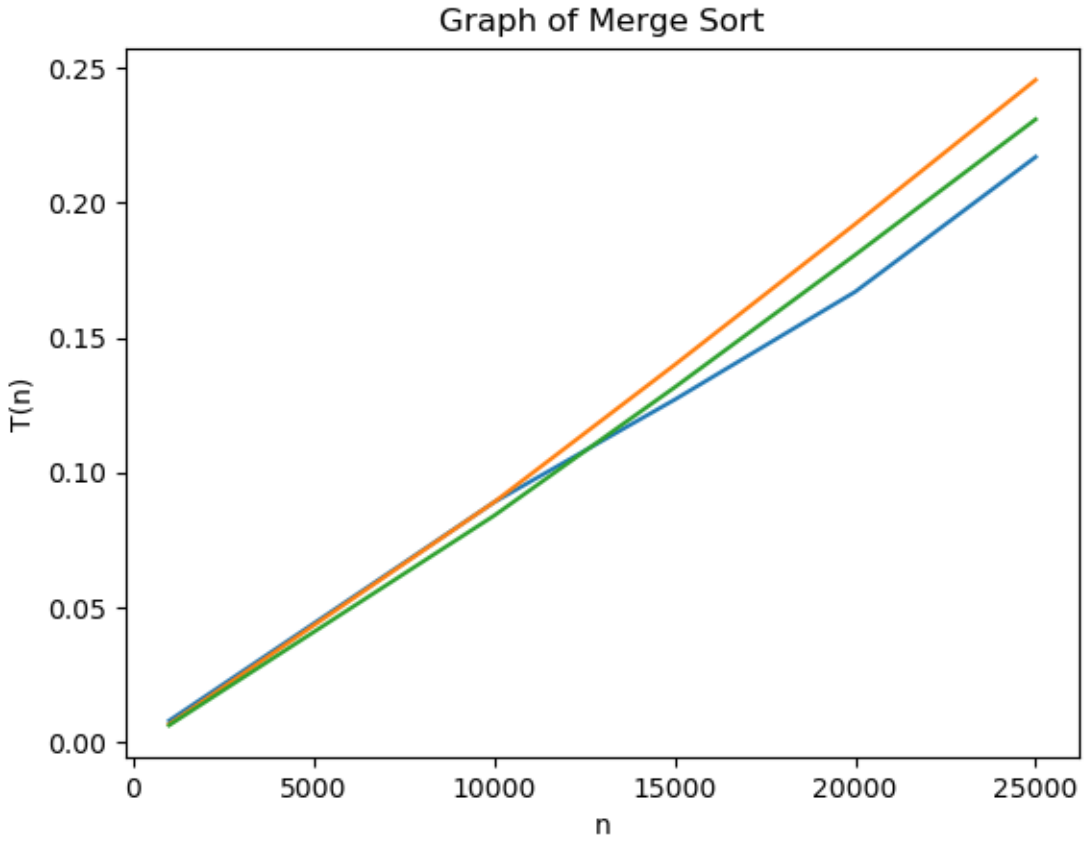
n	T(n)	T(n)/nlog2n
1000	0.008000612	0.000000802808
10000	0.0890052318572998	0.000000669831
15000	0.12700748443603516	0.0000006103485
20000	0.1670093536376953	0.0000005844509
25000	0.21701264381408691	0.000000594162859

From above values, we find out the average constant: c=0.000000652320618

Upper Bound Limit: **c1=0.00000067232**; Lower Bound Limit: **c2=0.000000632321**

Now the graph is shown as:

```
1  import matplotlib.pyplot as plt
2  x=[1000,10000,15000,20000,25000]
3  y1=[0.008,0.089,0.127,0.167,0.217]
4  y2=[0.0067,0.089,0.1399,0.1921,0.24555]
5  y3=[0.0063,0.084,0.13157,0.18068,0.23094]
6  plt.plot(x,y1)
7  plt.plot(x,y2)
8  plt.plot(x,y3)
9  plt.xlabel('n')
10 plt.ylabel('T(n)')
11 plt.title('Graph of Merge Sort')
12 plt.show()
```



Laboratory Session No. 07

Objective:

Introduction to object oriented programming.

Note:

1) **Compare OOP with POP. Discuss significance of OOP. The concept of classes and objects.**
2) **Using Jupyter notebook, create classes in Python**
3) **Elaborate the usage of *__init__* method and *self* construct in python(using any suitable example)**

Task 1:

Object-Oriented Programming Languages (OOP):

Object-oriented programming (**OOP**) is a programming language model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

POP stands for **Procedural Oriented Programming**. This paradigm focuses on procedures or functions that are required to perform the computation. It focuses on the process, rather than on data.
In POP, the program is divided into multiple functions. Each function has a clearly defined purpose. A function is a set of instructions to perform a certain task. These functions share global variables. Data is exchanged among functions.

The **main difference** between OOP and POP is that the **OOP divides the program into multiple objects to solve the problem while the POP divides the program into multiple procedures or functions to solve the problem.**

Main Emphasis

While OOP emphasis on objects, POP emphasizes on functions. This is one main difference between OOP and POP.

Program Decomposition

OOP divides the program into multiple objects. POP divides the program into multiple functions.

Modification

Modification is easier in OOP as the objects are independent. Modifications in POP can affect the entire program. Therefore, modifications are difficult in POP.

Communication

In OOP, the objects communicate with each other by passing messages. In POP, the functions communicate with each other by passing parameters.

Data Control

In OOP, each object controls its own data. In POP, the functions share global variables.

Data Hiding

It is possible to hide data in OOP. It avoids illegal access to the data or the attributes. Therefore, OOP makes data more secure. On the other hand, there is no data hiding mechanism in POP. This is another important difference between OOP and POP.

Assess Specifiers

OOP has access specifiers such as private, protected, and public to change the visibility of attributes and methods. There are no access specifiers in POP. This is also an important difference between OOP and POP.

Code Reusability

Although OOP has inheritance to reuse the already existing code, there is no inheritance in POP.

Programming Languages

C++, Java, and Python are a few languages that support OOP. C, Pascal, FORTRAN, and COBAL are few languages that support POP.

Conclusion

OOP and POP are two programming paradigms. The main difference between OOP and POP is that OOP divides the program into multiple objects to solve the problem while POP divides the program into multiple procedures or functions to solve the problem.

Task 2:

```
In [1]: class Student():
            pass

In [2]: a=Student() #assining the class Student into a data 'a'

In [3]: type(a) #we can find the type of data by using type function
Out[3]: __main__.Student

In [4]: x=3

In [5]: type(x)
Out[5]: int

In [6]: a.name='peter' #assigning name

In [7]: a.name
Out[7]: 'peter'

In [8]: a.roll=75

In [9]: a.roll
```

```
In [9]: a.roll
Out[9]: 75

In [13]: a.cpga=3.4

In [14]: a.cgpa
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-14-3150cdd9379d> in <module>()
----> 1 a.cgpa

AttributeError: 'Student' object has no attribute 'cgpa'

In [15]: a.year='SE'

In [16]: a.year
Out[16]: 'SE'

In [17]: dir(a)
```

Task 3:

```
In [50]: class Student:
             def __init__(self,nam,cgp,rol):#constructor
                 self.name = nam
                 self.cgpa = cgp
                 self.roll = rol
             def name_print(self):#Self refers to the object being currently called
                 print("The name assigned is",self.name)
             def cgpa_print(self):
                 print("The cgpa of",self.name,'is',self.cgpa)
             def cgpa_update(self,val):
                 self.cgpa=val
             def all_print(self):
                 print(self.name,'has roll#',self.roll,'and cgpa equal to',self.cgpa)

In [51]: a= Student('bebe',3.99,41)

In [52]: a.name_print()
         The name assigned is bebe

In [53]: a.all_print()
         bebe has roll# 41 and cgpa equal to 3.99

In [54]: a.cgpa_update(3.59)
```

```
In [54]: a.cgpa_update(3.59)

In [55]: a.all_print()
         bebe has roll# 41 and cgpa equal to 3.59

In [56]: p=[a]

In [57]: p[0].name
Out[57]: 'bebe'

In [59]: p[0].roll
Out[59]: 41

In [61]: b= Student('paul',3.1,53)

In [62]: p.append(b)

In [63]: p
Out[63]: [<__main__.Student at 0x1e2f49d4780>, <__main__.Student at 0x1e2f4a2ff28>]
```

Laboratory Session No. 08

Objective:

To implement the following open-ended problem in python

Develop a system which can perform following basic banking related tasks

a) ***Customer account could be created with name, NIC, account number and initial balance. All such attributes should be placed in a class***
b) ***Balance of any costumer could be updated***
c) ***Customer data could be sorted name wise and balance wise(any previously used sorting procedure may be applied)***

Creating a class:

```
def insertion_sort(A):
    for i in range(1,len(A)):
        j=i-1
        while j>=0 and A[j]>A[j+1]:
            A[j],A[j+1]=A[j+1],A[j]
            j=j-1
    return A

class bankaccount():
    total = 0
    data =[]
    def __init__ (self, nam, nic, accnum, currbal):
        self.name = nam
        self.cnic = nic
        self.accountnumber = accnum
        self.currentbalance = currbal
        bankaccount.data.append([self.name , self.currentbalance])


    def name_print(self):
        print("The name assigned is", self.name)

    def NIC_print(self):
        print("The CGPA of", self.name, " is ", self.cnic)
```

```
26      def accountnumber_print(self):
27          print("the account number of", self.name, "is", self.accountnumber)
28
29      def update_balance(self,val):
30          self.currentbalance = val
31
32      def currentbalance_print(self):
33          print("The current balance is: ", self.currentbalance)
```

```
def datasort():
    insertion_sort(bankaccount.data)
    print("The data of our customer is recorded as:\n")
    for i in range(len(bankaccount.data)):
        print(str(i+1) + ") " + bankaccount.data[i][0] + " has current balance of Rs." + str(bankaccount.data[i][1]))


def all_print(self):
    print(self.name, "with CNIC", self.cnic, "and account #", self.accountnumber, "currently has a balance of: ",
          self.currentbalance)
```

Sorting balance wise:

```
l = bankaccount( "Saad" , "004-43434" , 100 , 945)
m = bankaccount( "Khan" , "0454-66434" , 99 , 56000)
n = bankaccount( "Maaz" , "004-456447" , 88 , 1500)
o = bankaccount( "Hamza" , "0066-9999" , 77 ,6366)
p = bankaccount( "Rafay" , "0049-9999" , 5 ,8569)
bankaccount.datasort()
```

```
C:\Users\ICS\PycharmProjects\DSA\venv\Scr
1) Khan has current balance of Rs.56000
2) Rafay has current balance of Rs.8569
3) Hamza has current balance of Rs.6366
4) Maaz has current balance of Rs.1500
5) Saad has current balance of Rs.945
```

Laboratory Session No. 09

Objective:

To implement fundamental data structures in Python (using list)

Note:

Using *list* in python, implement the following

a) **Stacks(push and pop operations)**
b) **Queues(enqueue and dequeuer operations)**
c) **A dynamic set 'S' having following functionalities**
    a. **Search (S, key)**
    b. **Insert (an object)**
    c. **Delete (an object)**
    d. **Minimum(S)**
    e. **Maximum(S)**

Stacks (using List):

```
1  stack = ["Khan", "Akbar", "Jamal"]
2  stack.append("Ali")
3  stack.append("Iqbal")
4  print(stack)
5  print(stack.pop())
6  print(stack)
7  print(stack.pop())
8  print(stack)
```

```
Run:  plot
C:\Users\ICS\PycharmProjects\DSA\venv\Scrip
['Khan', 'Akbar', 'Jamal', 'Ali', 'Iqbal']
Iqbal
['Khan', 'Akbar', 'Jamal', 'Ali']
Ali
['Khan', 'Akbar', 'Jamal']
```

Queues (using List):

```
1  from collections import deque
2  queue = deque(["Saad", "Maaz", "Daniyal"])
3  print(queue)
4  queue.append("Hamza")
5  print(queue)
6  queue.append("Hashir")
7  print(queue)
8  print(queue.popleft())
9  print(queue.popleft())
10 print(queue)
```

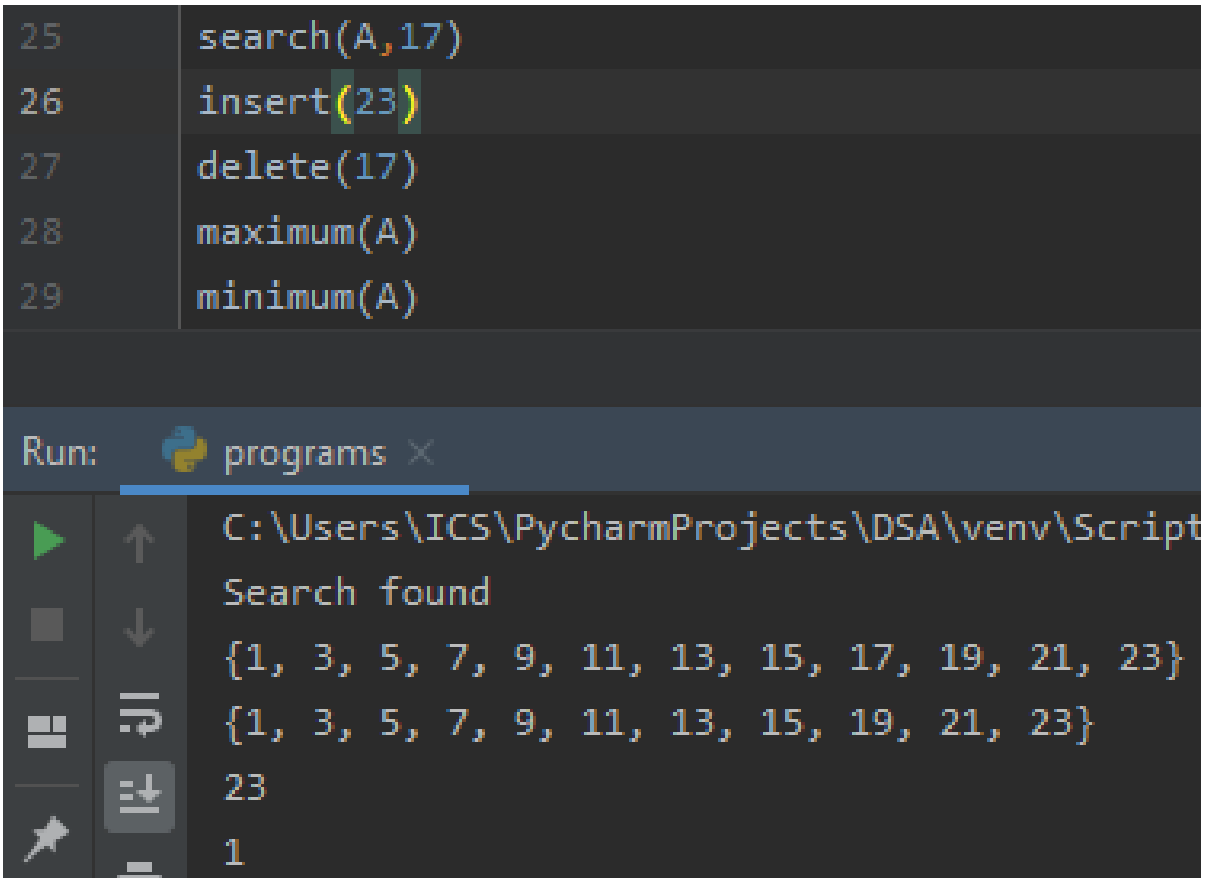
```
Run:  plot
C:\Users\ICS\PycharmProjects\DSA\venv\Scripts\python.e
deque(['Saad', 'Maaz', 'Daniyal'])
deque(['Saad', 'Maaz', 'Daniyal', 'Hamza'])
deque(['Saad', 'Maaz', 'Daniyal', 'Hamza', 'Hashir'])
Saad
Maaz
deque(['Daniyal', 'Hamza', 'Hashir'])
```

Dynamic Sets:

CODE:

```
A={1,3,5,7,9,11,13,15,17,19,21}

def search(A,key):
    temp={key}
    temp=A.intersection(temp)
    if len(temp)>=1:
        print('Search found')
    else:
        print('Not found')

def insert(Object):
    A.add(Object)
    print(A)

def delete(Object):
    A.discard(Object)
    print(A)

def maximum(A):
    print(max(A))

def minimum(A):
    print(min(A))
```

RESULT:


25 search(A,17)
26 insert(23)
27 delete(17)
28 maximum(A)
29 minimum(A)
Run: programs
C:\Users\ICS\PycharmProjects\DSA\venv\Script
Search found
{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23}
{1, 3, 5, 7, 9, 11, 13, 15, 19, 21, 23}
23
1

Laboratory Session No. 10

Objective:

Accomplish the following open-ended tasks:

Using Node class, develop

1. **Stacks**
2. **Queues**
3. **Singly connected linked-list with following features:**
    a. **Add nodes**
    b. **Traverse all nodes starting from top node**
    c. **Search any key value in all nodes**
    d. **Insert node between any two nodes**

Stacks:

Push Operation:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
        self.head = None

    def isempty(self):
        if self.head == None:
            return True
        else:
            return False

    def push(self, data):
        if self.head == None:
            self.head = Node(data)
        else:
            newnode = Node(data)
            newnode.next = self.head
            self.head = newnode
```

Pop Operation:

```
24      def pop(self):
25          if self.isempty():
26              return None
27          else:
28              poppednode = self.head
29              self.head = self.head.next
30              poppednode.next = None
31              return poppednode.data
32
33      def peek(self):
34          if self.isempty():
35              return None
36          else:
37              return self.head.data
38
39      def display(self):
40          iternode = self.head
41          if self.isempty():
42              print("Stack Underflow")
43          else:
44              while (iternode != None):
45                  print(iternode.data, "->", end=" ")
46                  iternode = iternode.next
47              return
```

Result:

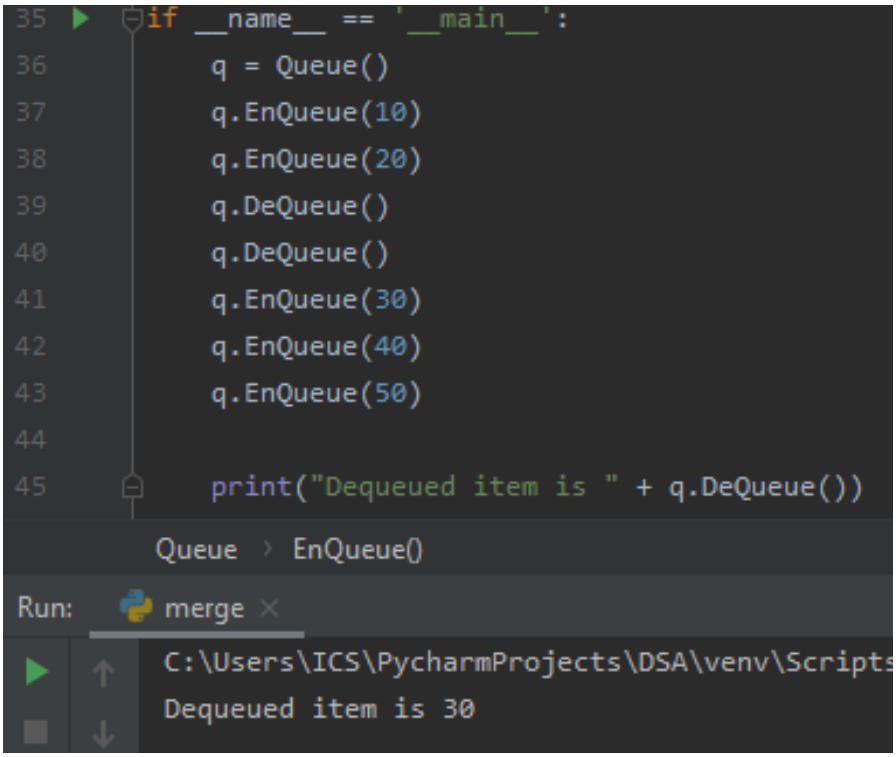
```
MyStack = Stack()

MyStack.push(1)
MyStack.push(9)
MyStack.push(3)
MyStack.push(48)

MyStack.display()

print("\nTop element is ", MyStack.peek())

MyStack.pop()
MyStack.pop()

MyStack.display()

print("\nTop element is ", MyStack.peek())
```

```
programs
C:\Users\ICS\PycharmProjects\DSA\venv\Scripts
48 -> 3 -> 9 -> 1 ->
Top element is  48
9 -> 1 ->
Top element is  9
```

Queues:

```
class Queue:

    def __init__(self):
        self.front = self.rear = None

    def isEmpty(self):
        return self.front == None

    def EnQueue(self, item):
        temp = Node(item)

        if self.rear == None:
            self.front = self.rear = temp
            return
        self.rear.next = temp
        self.rear = temp

    def DeQueue(self):

        if self.isEmpty():
            return
        temp = self.front
        self.front = temp.next

        if (self.front == None):
            self.rear = None
        return str(temp.data)
```

Result:


if __name__ == '__main__':
q = Queue()
q.EnQueue(10)
q.EnQueue(20)
q.DeQueue()
q.DeQueue()
q.EnQueue(30)
q.EnQueue(40)
q.EnQueue(50)
print("Dequeued item is " + q.DeQueue())
Queue › EnQueue()
Run:
merge
C:\Users\ICS\PycharmProjects\DSA\venv\Scripts
Dequeued item is 30

Linked Lists Operations:

a) ADD NODES:

```
1   class Node:
2
3       def __init__(self, data=None):
4           self.data = data
5           self.next = None
6
7   class Linked_list:
8
9       def __init__(self):
10          self.head=node()
11
12      def append(self,data):
13          new_node=node(data)
14          temp=self.head
15          while temp.next!=None:
16              temp=temp.next
17          temp.next=new_node
```

b) Traversing NODES:

```
19      def display(self):
20          elems=[]
21          temp_node=self.head
22          while temp_node.next!=None:
23              temp_node=temp_node.next
24              elems.append(temp_node.data)
25          print(elems)
```

c) Searching a KEY in NODES:

```
26      def insert(self,index,data):
27          temp_node=self.head
28          prior_node=self.head
29          temp_idx=0
30          while True:
31              temp_node=temp_node.next
32              if temp_idx==index:
33                  new_node=node(data)
34                  prior_node.next=new_node
35                  new_node.next=temp_node
36                  return
37              prior_node=temp_node
38              temp_idx+=1
```

d) Inserting a NODE between TWO NODES:

```
39      def search(selfself,val):
40          current=self.head
41          while current != None:
42              if current.data==val:
43                  return True
44              current=current.next
45          return False
```