Exp :1 Python program to Use and demonstrate basic data structures.

A: SETS

```
 set1 =set()
set2 =set()
for i in range (1,6):
   set1.add(i)
for j in range (3,8):
   set2.add(j)
print("set1=",set1)
print("set2=",set2)
```

******OUTPUT*********

set1= {1, 2, 3, 4, 5}

set2= {3, 4, 5, 6, 7}

B: LIST

```
list=[]
print("blank list:")
print(list)
list=[10,20,14]
print("\n list of number;")
print(list)
list=["mon","tue","wed"]
print("\n list.items;")
print(list[0])
print(list[2])
list=(["jan","feb"],["mar"])
print("\n multi-demensional list:")
print(list)
```

******OUTPUT*********

```
blank list:
[]

 list of number;
[10, 20, 14]
 list.items;
mon
wed
 multi-demensional list:
(['jan', 'feb'], ['mar'])
```

C: DICTIONARAY

```
d1 = dict()
print("Empty Dict")
print(d1)
d2 = dict(a="January", b="Feb", c="March")
print(d2)
#print(Dict)
d3 = dict([(1, "january"),(2, "february")])
print("Dictionary with each item as pair")
print(d3)
```

*******OUTPUT*******

```
Empty Dict
{}
{'a': 'January', 'b': 'Feb', 'c': 'March'}
Dictionary with each item as pair
{1: 'january', 2: 'february'}
```

D:TUPLE

```
t1=(0,)
t2=(1,2,3,4)
t3=("abc",("def","ghi"))
```

```
print(t1[0])
print(t3[1:3])
print(len(t3))
tup=(1,"a","string",1+2)
print(tup)
```

*******OUTPUT*******

```
0
(('def', 'ghi'),)
2
(1, 'a', 'string', 3)
```

EXP- 2 Implement an ADT with all its operations.

```
class Date:
   def __init__(self,dd=None,mm=None,yyyy=None):
      print("constructor called")
      self.date = dd;
      self.month = mm;
      self.year = yyyy;

   def getDate(self):
      return self.date;

   def getMonth(self):
      return self.month;

   def getYear(self):
      return self.year;

   def printDate(self):
      s = str(self.date)+"-"+str(self.month)+"-"+str(self.year)
      print(s)

   def getMonthName(self):
      m=""
      if(self.month==1):
        m="January";
      elif(self.month==2):
         m="Feb";

      return m;

   def __eq__(self,otherDate):
      print("Eq called")
      if(self.date == otherDate.date and self.month==otherDate.month and
self.year==otherDate.year):
         return True
      else:
```

```
            return False

    def __lt__(self, otherDate):
        if(self.year<otherDate.year):
            return True
        elif(self.year>otherDate.year):
            return False
        elif(self.month<otherDate.month):
            return True
        elif(self.month>otherDate.month):
            return False
        elif(self.date<otherDate.date):
            return True
        elif(self.date>otherDate.date):
            return False
        else:
            return False

    def setDate(self,dd):
        self.date = dd;

    def setMonth(self, mm):
        self.month=mm;

    def setYear(self, yyyy):
        self.year=yyyy;
```

Program 2:

```
from date import Date
d1 = Date(15,2,2025)
d2 = Date(16,1,2025)
d1.printDate()
d2.printDate()
print(d1.getDate())
print(d2.getDate())

print(d1.getMonth())
```

```
print(d2.getMonth())
print(d2.getMonthName())
```

*****OUTPUT*******

```
constructor called
constructor called
15-2-2025
16-1-2025
15
16
2
1
January
```

Exp 3 Implement an ADT and Compute space and time complexities.(STACK)

```
class Stack():
    def __init__(self,maxSize = 5):
        self.L = []
        self.maxSize = maxSize
        self.top = -1

    def push(self, e):
        self.top = self.top + 1
        self.L.insert(self.top,e)

    def pop_stack(self):
        e = self.L.pop(self.top)
        self.top = self.top - 1
        return e

    def peek(self):
        return self.L[self.top]

    def isFull(self):
        if (self.top==self.maxSize-1):
```

```
            return True
        else:
            return False

    def isEmpty(self):
        if(self.top==-1):
            return True
        else:
            return False

    def display(self):
        for i in range(self.top,-1,-1):
            print(self.L[i])

#main

s = Stack()
while True:
    print("1. Push")
    print("2. Pop")
    print("3. Peek")
    print("4. Display")
    print("5. Exit")

    ch = int(input("Enter choice:"))
    if(ch==1):
        if(s.isFull()):
            print("Stack Overflow")
        else:
            e = int(input("Enter element:"))
            s.push(e)

    elif(ch==2):
        if(s.isEmpty()):
            print("Stack underflow")
        else:
            e = s.pop_stack()
```

```
            print("Popped item = %d" % e)

    elif(ch==3):
        if(s.isEmpty()):
            print("Stack Empty Error")
        else:
            e = s.peek()
            print("Top of stack = %d" % e)

    elif(ch==4):
        s.display()

    elif(ch==5):
        break

    else:
        print("Invalid choice")
```

```
*******OUTPUT********
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice:1
Enter element:25
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice:4
25
1. Push
2. Pop
3. Peek
4. Display
```

```
5. Exit
Enter choice:2
Popped item = 25
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice:3
Stack Empty Error
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice:1
Enter element:24
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice:3
Top of stack = 24
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice:3
Top of stack = 24
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice:5
```

Time and space Complexity

Function Name	Time Complexity
Push	O(1)
Pop	O(1)
Peek	O(1)
isEmpty	O(1)
isFull	O(1)

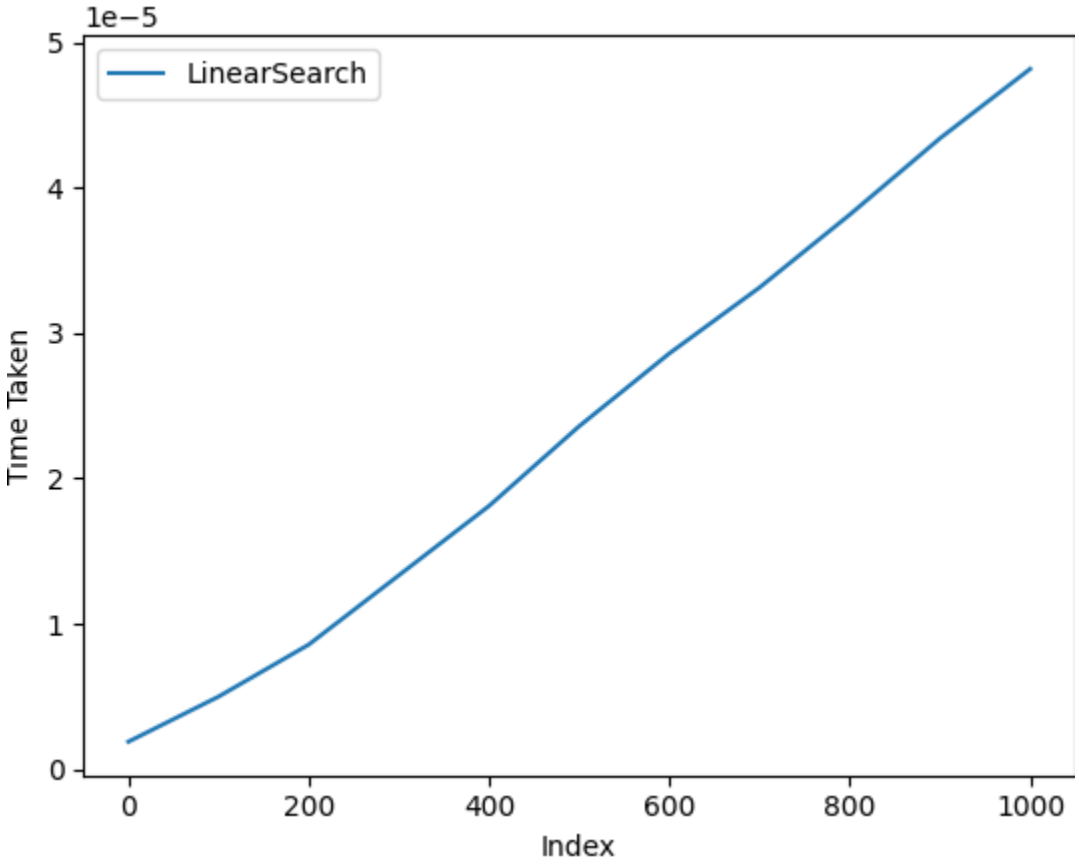
Display is not proper function of stacks, The time complexity for display is O(n)

EXP 4: Implement Linear Search compute space and time complexities,

```
import matplotlib.pyplot as plt
import time

def linearSearch(a, key):
  for i in range(len(a)):
    if(key==a[i]):
      return i
  return -1

#main

a = []
for i in range(1000):
  a.append(i)


x = [i * 100 for i in range(11)]

y = []
```

```
for i in range(11):
  key = x[i]
  begin = time.time()
  s = linearSearch(a,key)
  end = time.time()
  total = end - begin
  y.append(total)

plt.plot(x,y, label = "LinearSearch")
plt.legend()
plt.xlabel("Index")
plt.ylabel("Time Taken")
plt.show()    #NOTE EXCUTED IN PYTHON 3 ONLY
```

*****OUTPUT******



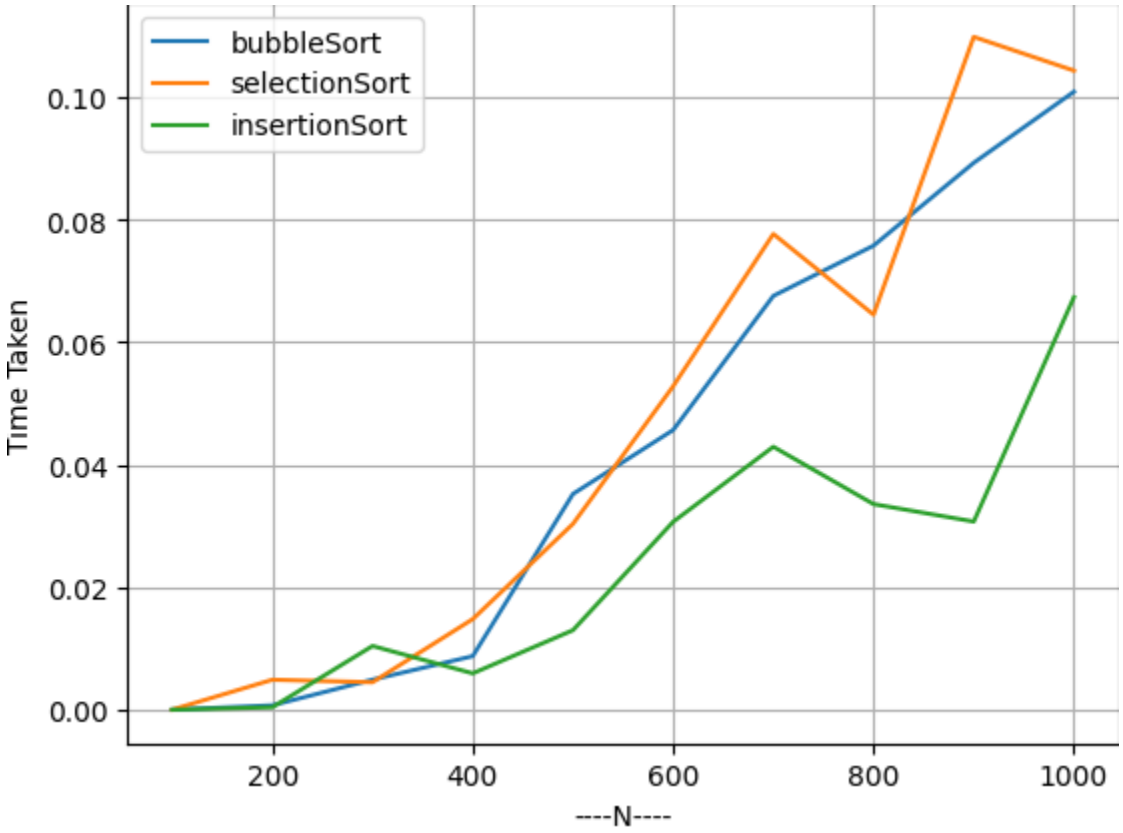
EXP 5 Implement Bubble, Selection, insertion sorting algorithms compute space and time complexities, plot graph using asymptomatic notations.

```
import random
import matplotlib.pyplot as plt
import time
import copy

def bubbleSort(a):
    N = len(a)
    for i in range(N-1):
        for j in range(N-1-i):
            if(a[j]>a[j+1]):
                a[j],a[j+1] = a[j+1],a[j]

def selectionSort(a):
    N = len(a)
    for i in range(N-1):
        minIndex = i
        for j in range(i+1, N):
            if(a[j]<a[minIndex]):
                minIndex = j

        a[i],a[minIndex]= a[minIndex],a[i]


def insertionSort(a):
    N = len(a)
    for i in range(1, N):
        key = a[i]
        j = i - 1
        while j>=0 and key<a[j]:
            a[j+1] = a[j]
            j = j - 1

        a[j+1] = key
```

```
#main
x = [i*100 for i in range(1,11)]
y = [[],[],[]]
t = (bubbleSort, selectionSort, insertionSort)
##############################

a = []
b = []
c = []

for i in range(10):
    N = i * 100
    L = []
    for j in range(N):
        e = random.randint(1,10000)
        L.append(e)
    a.append(L)

b = copy.deepcopy(a)
c = copy.deepcopy(a)

ele = [a,b,c]

for i in range(10):
    k = 0
    for func in t:
        begin = time.time()
        func(ele[k][i])
        end = time.time()
        total = end - begin
        y[k].append(total)
        k = k + 1


plt.plot(x,y[0], label ="bubbleSort")
plt.plot(x,y[1], label ="selectionSort")
plt.plot(x,y[2], label ="insertionSort")
```

```
plt.legend()
plt.grid()

plt.xlabel("----N----")
plt.ylabel("Time Taken")

plt.show()
```

*****OUTPUT******



EXP 6 Implement Binary Search using recursion Compute space and time complexities,

a. Using Iteration

```
def binSearch(a, key):
    low = 0
    high = len(a) - 1

    while(low<=high):
        mid = (low+high)//2
        if(key == a[mid]):
            return "Key found"

        elif(key > a[mid]):
            low = mid + 1

        elif(key < a[mid]):
            high = mid - 1


    return "Key not found"

#main
L = []
n = int(input("Enter n:"))

print("Enter %d elements in sorted order" %n)

for i in range(n):
    e = int(input("Enter element:"))
    L.append(e)

key = int(input("Enter key to search:"))
s = binSearch(L,key)
print(s)
```

Output

Enter n:5
Enter 5 elements in sorted order
Enter element:1
Enter element:2
Enter element:3
Enter element:4
Enter element:5
Enter key to search:30
Key not found

Enter n:5
Enter 5 elements in sorted order
Enter element:10
Enter element:20
Enter element:30
Enter element:40
Enter element:50
Enter key to search:40
Key found

---

b. Using Recursion

```
def binSearch(a, key, low, high):
    if(low>high):
        return "Key not found"

    mid = (low+high)//2
    if(key == a[mid]):
        return "Key Found"

    elif(key>a[mid]):
        s = binSearch(a,key, mid+1, high)
        return s

    elif(key<a[mid]):
        s = binSearch(a,key, low, mid-1)
        return s


#main
L = []
n = int(input("Enter n:"))

print("Enter %d elements in sorted order" %n)

for i in range(n):
    e = int(input("Enter element:"))
    L.append(e)

key = int(input("Enter key to search:"))
s = binSearch(L,key,0,n-1)
print(s)
```

Output

Enter n:5
Enter 5 elements in sorted order
Enter element:10
Enter element:20
Enter element:30
Enter element:40
Enter element:50
Enter key to search:25
Key not found

Enter n:5
Enter 5 elements in sorted order
Enter element:10
Enter element:20
Enter element:30
Enter element:40
Enter element:50
Enter key to search:20
Key Found

EXP 7: Implement Merge and quick sorting algorithms compute space and time complexities, plot graph using asymptomatic notations and compare all solutions.

A: Merge

```
def partition(a, low, high):
    pivot = a[high]
    j = low
    i = j - 1

    for j in range(low, high):
        if(a[j]<pivot):
```

```
            i = i + 1
            a[i],a[j] = a[j],a[i]

    a[i+1], a[high] = a[high], a[i+1]

    return i+1
##########################################
def quickSort(a, low, high):
    if(low>=high):
        return
    PI = partition(a, low, high)
    quickSort(a, low, PI - 1)
    quickSort(a, PI+1, high)

##########################################
#main
a = []
n = int(input("Enter n:"))
print("Enter %d elements" %n)

for i in range(n):
    e = int(input("Enter element:"))
    a.append(e)
print("Before Sorting")
print(a)
quickSort(a, 0, n-1)
print("Sorted Array is")
print(a)
**********OUTPUT*******
Enter n:3
Enter 3 elements
Enter element:10
Enter element:8
Enter element:30
Before Sorting
[10, 8, 30]
Sorted Array is
[8, 10, 30]
```

B: **quick sorting**

```
def partition(a, low, high):
    pivot = a[high]
    j = low
    i = j - 1
    for j in range(low, high):
        if(a[j]<pivot):
            i = i + 1
            a[i],a[j] = a[j],a[i]

    a[i+1], a[high] = a[high], a[i+1]
    return i+1
##########################################
def quickSort(a, low, high):
    if(low>=high):
        return
    PI = partition(a, low, high)
    quickSort(a, low, PI - 1)
    quickSort(a, PI+1, high)

a = []
n = int(input("Enter n:"))
print("Enter %d elements" %n)

for i in range(n):
    e = int(input("Enter element:"))
    a.append(e)
```

```
print("Before Sorting")
print(a)
quickSort(a, 0, n-1)
print("Sorted Array is")
print(a)
```

*********OUTPUT*********

Enter n:3
Enter 3 elements
Enter element:10
Enter element:15
Enter element:8
Before Sorting
[10, 15, 8]
Sorted Array is
[8, 10, 15]

EXP 8: Implement Fibonacci sequence with dynamic programming.

```
def fib(n):
    a = []
    a.append(0)
    a.append(1)

    for i in range(2, n):
        a.append( a[i-1] + a[i-2] )

    return a

#main

n = int(input("Enter n:"))
a = fib(n)
```

```
print("%d terms of fib series are" %n)
print(a)
```

Output

Enter n:10

10 terms of fib series are

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

EXP 9: Program to demonstrate recursive operations Factorial Using Recursion

```
def fact(n):
    if(n==0):
        return 1
    f = fact(n-1)
    return n * f

#main

n = int(input("Enter n:"))
f = fact(n)
print("Factorial of %d is %d" %(n,f))
```

Output

Enter n:5

Factorial of 5 is 120

EXP 10: Implement solution for Towers of Hanoi.

```
def TOH(n, source="source", dest="destination", aux="auxillary"):
    if(n==0):
        return
    TOH(n-1, source, aux, dest)
    print("Move %d disk from %s tower to %s tower" %(n,source,dest))
    TOH(n-1, aux, dest, source)


#main
n = int(input("Enter number of towers:"))
TOH(n)
```

*****OUTPUT******

```
Enter number of towers:3
Move 1 disk from source tower to destination tower
Move 2 disk from source tower to auxillary tower
Move 1 disk from destination tower to auxillary tower
Move 3 disk from source tower to destination tower
Move 1 disk from auxillary tower to source tower
Move 2 disk from auxillary tower to destination tower
Move 1 disk from source tower to destination tower
```