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

```
numberl  = 100
print(”The type of number1 ”,type(number 1))
 integer_1 = 100
integer_2 = 50
print (integer_1*integer_2)
print(integer_1+integer_2)
print(integer_1-integer_2)
print(integer_1/integer_2)
number2 = 100.0
print(”The type of number2 ”,type(number 2))
float_1 = 12.539
float_2 = 6.78
print (float_1*float_2)
print(float_1+float_2)
print(float_1-float_2)
print (float_1/float_2)
str = ’Jack Daniels'
print(”The type of str ",type(str)) string_1 = "Hello"
string_2 = " World"
print (string_1 + string_2)
result = 100 < 200
print(”The type of result ",type(result))
has_passed  = False
marks = 80
if (marks > 50):
  has_passed = True
print  (has_passed)
```

OR

```
print("List")
l1=[1,2,"ABC",3, "xyz",2.3]
print(l1)
print("Dictionary”)
d1={"a":134,"b":266,"c":343}
print(d1)
print("Tuples")
t1=(10,20,30,40,50)
print(t1)
print("Sets")
s1={10,30,20,40,10,30,40,20,50,50}
print(s1)
```

OUTPUT

2.Implement an ADT with all its operations.

```
class date:
   def init (self,a,b,c):
     self.d=a
     self.m=b
     self.y=c
    def day(self):
        print("Day = ", self.d)
   def month(self):
        print("Month = ", self.m)
   def year(self):
       print("year=",self.y)
   def monthName(self):
       months = ["Unknown","January","Febuary","March","April","May","June","July",
     "August","September","October","November","December"]
    print("Month Name:",months[self.m])
   def isLeapYear(self):
      if (self.y % 400 == 0) and (self.y % 100 =0):
         print("It is a Leap year")
      elif (self.y % 4 == 0) and (self.y % 100 != 0):
         print("It is a Leap year")
      else:
         print("It is not a Leapyear")
d1 = date(3,8,2000)
d1.day()
d1.month()
d1.year()
d1.monthName()
d1.isLeapYear()
```

OUTPUT

3. Implement an ADT and Compute space and time complexities.

```
import time
class stack:
    def_init_(self):
        self.items = []
    def isEmpty(self): return
        self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items) - 1]
    def size(self):
        return len(self.items)
    def display(self):
        return (self.items)
s=stack()
start = time.time()
print(s.isEmpty())
print("push operations")
s.push(11)
s.push(12)
s.push(13)
print("size:",s.size())
print(s.display())
print("peek",s.peek())
print("pop operations")
print(s.pop())
print(s.pop())
print(s.display())
print("size:",s.size())
end = time.time()
print("Runtime of the program is", end - start)
```

OUTPUT

4.Implement Linear Search and compute space and time complexities, plot graph using asymptomatic notations

```
import time
import matplotlib.pyplot as plt
start=time.time()
def linearsearch(a, key):
    n = len(a)
    for i in range(n):
        if a[i] == key:
            return i;
    return -1
a = [13,24,35,46,57,68,79]
start = time.time()
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
i = linearsearch(a,k)
if   i == -1:
     print("Search UnSuccessful")
else:
     print("Search Successful key found at location:",i+1)
end = time.time()
print("Runtime of the program is", end-start)
x=list(range(1,1000))
plt.plot(x ,[y for y in x] )
plt.title("Linear Search- Time Complexity is O(n)")
plt.xlabel("Input")
plt.ylabel("Time")
plt.show()
```

OUTPUT

5.Implement Bubble Sort and compute space and time complexities, plot graph using asymptomatic notations

```
import time
import matplotlib.pyplot as plt
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]:
                temp = a[j]
                a[j] = a[j+1]
                a[j+1] = temp
x = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",x)
bubblesort(x)
print("After sorting:",x)
x=list(range(1,10000))
plt.plot(x,[y*y for y in x] )
plt.title(”Bubble Sort- time complexity is 0(n2)")
p1t.x1abel("Input")
p1t.ylabel("Time")
p1t.show()
```

OUTPUT

6. Implement Selection Sort and compute space and time complexities, plot graph using asymptomatic notations

```
import time
import matplotlib.pyplot as plt
start=time.time()
def selectionsort(a):
    n = len(a)
    for i in range(n-2):
        min = i
        for j in range(i+1,n):
            if a[j]<a[min]:
                min=j
        temp = a[i]
        a[i] = a[min]
        a[min] = temp
x = [34, 46,43,27,57,41,45,21,70]
print("Before sorting:",x)
selectionsort(x)
print("After sorting:",x)
x=list(range(1,10000))
plt.plot(x,[y*y for y in x] )
p1t.title("Selection Sort- time complexity is O(n^2)")
p1t.x1abel("Input")
p1t.ylabel("Time")
plt.show()
```

OUTPUT

7. Implement Insertion Sort and compute space and time complexities, plot graph using asymptomatic notations

```
import time
import matplotlib.pyplot as plt
def insertionsort(a):
    n = len(a)
    for i in range(1,n-1):
        v=a[i]
        j = i-1
        while j>=0 and a[j]>v:
            a[j+1] = a[j]
             j=j-1
            a[j+1] = v
x = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",x)
insertionsort(x)
print("After sorting:",x)
x=list(range(1,10000))
plt.plot(x,[y  for y in x] )
p1t.title("Insertion Sort- time complexity is O(n)")
p1t.x1abel("Input")
p1t.ylabel("Time")
plt.show()
```

OUTPUT

8.Implement Merge Sort and compute space and time complexities, plot graph using asymptomatic notations

```
 import time
 import matplotlib.pyplot as plt
  import math
start=time.time()
def  mergsort(mylist):
if(len>i):
       mid=len(mylist)//2
       left=mylist[mid:]
       right=mylist[:mid]
mergsort(left)
mergsort(right)
i=0
k=0
j=0
while i>len(left)and j <len(right):
      if left[i]<=right[j]:
           mylist[k]=left[i]
           i += 1
      else:
           mylist[k]=right[j]
            j += 1
            k += 1
while i<len(left):
         mylist[k]=left[i]
         i += 1
         k += 1
while j<len(right):
        mylist[k]=right[j]
        j += 1
        k+= 1
a=[20,8,9,66,55]
k=input("enter the element:")
print(a)
end=time.time()
x=list(range(1,1000))
print("runtime of the program is{end - start}")
plt.plot("mergsort=o(n)")
plt.plot(x,[y*math.log(y,2)for y in x])
plt.xlabel("input")
plt.ylabel("time")
plt.show()
```

OUTPUT

9.Implement Quick Sort and compute space and time complexities, plot graph using asymptomatic notations

```
import time
import matplotlib.pyplot as plt
import math
start=time.time()
import time
import matplotlib.pyplot as plt
import math
def quicksort(alist,start,end):
   if end–start > 1:
      p=partition(alist,start,end)
      quicksort(alist,start,p)
      quicksort(alist,p+1,end)
def partition(alist,start,end):
    pivot = alist[start]
     i = start +1
     j = end – 1
while True:
    while (i <= j and alist[i] <= pivot):
            i = i + 1
    while (i <= j and alist[i] >= pivot):
            j = j – 1
    if i <= j:
       alist[i],alist[j] = alist[j],alist[i]
    else:
       alist[start],alist[j] = alist[j],alist[start]
       return j
alist = input("enter the list of  number: ").split()
alist= [int(x) for x in alist]
quicksort(alist,0,len(alist))
print("sorted list:  ", end=' ')
print(alist)
```

```
end=time.time()
x=list(range(1,1000))
print("runtime of the program is{end - start}")
plt.plot("mergsort=o(n)")
plt.plot(x,[y*math.log(y,2)for y in x])
plt.xlabel("input")
plt.ylabel("time")
plt.show()
```

OUTPUT

10.Implement Binary Search and compute space and time complexities, plot graph using asymptomatic notations

```
import time
import matplotlib.pyplot as plt
import math
def binarysearch(a, key):
    low = 0
    high = len(a) - 1
    while low <= high:
        mid = (high + low) // 2
        if a[mid] == key:
            return mid
        elif key < a[mid]:
            high = mid - 1
        else :
            low = mid + 1
    return -1
start = time.time()
a = [13,24,35,46,57,68,79]
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
r = binarysearch(a,k)
if r == -1:
    print("Search UnSuccessful")
else:
    print("Search Successful key found at location:",r+1)
end = time.time()
print("Runtime of the program is:", end-start)
x=list(range(1,500))
plt.plot(x ,[y*math.log(y,2) for y in x] )
plt.title("Binary Search- Time Complexity is O(log n)")
plt.show()
```

OUTPUT

11.Implement Fibonacci sequence with dynamic programming.

```
def fib(n):
    if n<=1:
        return n
    f = [0, 1]
    for i in range(2, n+1):
        f.append(f[i-1] + f[i-2])
    print("The Fibonacci sequence is:",f)
    return f[n]
n=int(input("Enter the term:"))
print("The Fibonacci value is:",fib(n))
```

OUTPUT

12.Implement singly linked list (Traversing the Nodes, searching for a Node, Prepending Nodes, and Removing Nodes)

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

class SinglyLinkedList:
   def__init_(self):
      self.first = None

   def insertFirst(self, data):
      temp = Node(data)
      if(self.first == None):
          self.first=temp
      else:
         temp.next=self.first
         self.first=temp

   def removeFirst(self):
      if(self.first== None):
         print("list is empty")
      else:
         cur=self.first
         self.first=self.first.next
         print("the deleted item is",cur.data)

   def display(self):
      if(self.first== None):
         print("list is empty")
         return
      current = self.first
      while(current):
      print(current.data, end = " ")
      current = current.next

   def search(self,item):
      if(self.first== None):
         print("list is empty")
         return
      current = self.first
      found = False
      while current != None and not found:
         if current.data == item:
            found = True
         else:
            current = current.next
```

```
        if(found):
            print("Item is present in the linkedlist")
        else:
        print("Item is not present in the linked list")

#Singly Linked List
ll = SinglyLinkedList()
while(True):
    c1 = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
    if(c1 == 1):
        item = input("Enter the element to insert:")
        ll.insertFirst(item)
     elif(c1 == 2):
        ll.removeFirst()
    elif(c1 == 3):
        item = input("Enter the element to search:")
        ll.search(item)
    elif(c1 == 4):
        ll.display()
    else:
        break
```

OUTPUT

13.Implement singly linked list using Iterators.

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

class LinkedList:
    def __init__(self):
        self.first = None

    def   insert(self,data):
        temp = Node(data)
        if(self.first):
            current   =   self.first
            while(current.next):
                current = current.next
            current.next = temp
        else:
            self.first = temp
    def__iter__(self):
        current = self.first
        while current:
            yield current.data
            current = current.next

# Linked List Iterators
ll = LinkedList()
ll.insert(9)
ll.insert(98)
ll.insert("welcome")
ll.insert("govt polytechnic koppal")
ll.insert(456.35)
ll.insert(545)
ll.insert(5)
for x in ll:
    print(x)
```

OUTPUT

14.Implement Stack Data Structure.

```
s = []
def push():
    if len(s) == size:
        print("Stack is Full")
    else:
        item = input("Enter the element:")
        s.append(item)
def pop():
    if(len(s) == 0):
        print("Stack is Empty")
    else:
        item = s[-1]
        del(s[-1])
        print("The deleted element is:",item)
def display():
    if(len(s)== 0):
        print("Stack is Empty")
    else:
        for i in reversed(s):
            print(i)
size=int(input("Enter the size of Stack:"))
while(True):
    choice = int(input("1-Push 2-POP 3-DISPLAY 4-EXIT Enter your choice:"))
    if(choice == 1):
        push()
    elif(choice == 2):
        pop()
    elif(choice == 3):
        display()
    else:
        break
```

OUTPUT

15.Implement bracket matching using stack.

```
def bracketmatching(expr):
    stack = []
    for char in expr:
        if char in ["(", "{", "["]:
            stack.append(char)
        else:
            if not stack:
                return False
            current_char = stack.pop()
            if current_char == '(':
                if char != ")":
                    return False
            if current_char == '{':
                if char != "}":
                    return False
            if current_char == '[':
                if char != "]":
                    return False
    if stack:
        return False
    return True
expr = "{()}[]"
if bracketmatching(expr):
    print("Matching")
else:
    print("Not Matching")
```

OUTPUT

16.Program to demonstrate recursive operations (factorial/ Fibonacci)

a) Factorial

```
def fact(n):
    if n == 1:
        return 1
    else:
        return (n * fact(n-1))
n=int(input("Enter the number:"))
print("The factorial of a number is:",fact(n))
```

OUTPUT

b) Fibonacci

```
def fib(n):
    if n<=1:
        return n
    return fib(n-1) + fib(n-2)
n=int(input("Enter the range:"))
print("The fibonacci value is:",fib(n))
```

OUTPUT

17.Implement solution for Towers of Hanoi.

```
def towerofhanoi(n, source, destination, auxiliary):
    if n==1:
        print ("Move disk 1 from source",source,"to destination",destination)
        return
    towerofhanoi(n-1, source, auxiliary, destination)
    print ("Move disk",n,"from source",source,"to destination",destination)
    towerofhanoi(n-1, auxiliary, destination, source)
n = 4
towerofhanoi(n,'A','B','C')
```

OUTPUT

18.Implement Queue Data Structure.

```
q=[]
def enqueue():
    if len(q)==size:
        print("Queue is Full")
    else:
        item=input("Enter the element:")
        q.append(item)
def dequeue():
    if not q:
        print("Queue is Empty")
    else:
        item=q.pop(0)
        print("Element removed is:",item)
def display():
    if not q:
         print("Queue is Empty")
    else:
        print(q)
size=int(input("Enter the size of Queue:"))
while True:
    choice=int(input("1.Insert 2.Delete 3. Display 4. Quit Enter your choice:"))
    if choice==1:
        enqueue()
    elif choice==2:
        dequeue ()
    elif choice==3:
        display()
    else:
        break
```

OUTPUT

19.Implement Binary Search Tree and its Operations

```
class BSTNode:
        def __init__(self,data):
                self.data = data
                self.leftChild = None
                self.rightChild = None
def insertNode(root,value):
        if root.data == None:
                root.data = value
        elif value < root.data:
             if root.leftChild is None:
                    root.leftChild = BSTNode(value)
                else:
                    insertNode(root.leftChild,value)
        elif value >= root.data:
            if root.rightChild is None:
                  root.rightChild = BSTNode(value)
            else:
                insertNode(root.rightChild,value)
        return str(value)," The node has been successfully inserted "
def searchNode(root,value):
    if value < root.data:
       if root.leftChild is None:
          return str(value)," NOT found in the tree"
       return searchNode(root.leftChild,value)
    elif value > root.data:
        if root.rightChild == None:
            return str(value)," NOT found in the tree"
        return searchNode(root.rightChild,value)
     else:
         return str(value)," found in the tree"
def   inOrder(root):
      if not root:
         return
      inOrder(root.leftChild)
      print(root.data,end="->")
      inOrder(root.rightChild)
def   preOrder(root):
      if not root:
         return
      print(root.data,end="->")
      preOrder(root.leftChild)
      preOrder(root.rightChild)
```

```
def postOrder(root):
    if not root:
        return
    postOrder(root.leftChild)
    postOrder(root.rightChild)
    print(root.data,end="->")
newTree = BSTNode(None)

print( insertNode(newTree,70) )
print( insertNode(newTree,50) )
print( insertNode(newTree,90) )
print( insertNode(newTree,30) )
print( insertNode(newTree,80) )
print( insertNode(newTree,100) )
print( insertNode(newTree,20) )
print( insertNode(newTree,40) )
print( insertNode(newTree,60) )

print("IN-ORDER TRAVERSAL OF TREE",end=".......\n")
inOrder(newTree)
print(end="\n")

print("PRE-ORDER TRAVERSAL OF TREE",end=".......\n")
preOrder(newTree)
print(end="\n")

print("POST-ORDER TRAVERSAL OF TREE",end=".......\n")
postOrder(newTree)
print(end="\n")

print(searchNode(newTree,20) )
print(searchNode(newTree,100))
print(searchNode(newTree,200))
```

OUTPUT

20.Implement Breadth first search Algorithm

```
from queue import Queue
graph = { 'A' : [ 'B' , 'D' , 'E' , 'F' ],
          'D' : [ 'A' ],
          'B' : ['A','F','C'],
          'F' : ['B' , 'A' ],
          'C' : ['B'],
          'E' : ['A']
          }
print("GIVEN GRAPH IS: ")
print(graph)

def BFS_TRAVERSAL(input_graph , source):
    Q = Queue()
    visited_vertices = list()

    Q.put(source)
    visited_vertices.append(source)

    while not Q.empty():
        vertex=Q.get()
        print(vertex,end="->")
        for u in input_graph[vertex]:
            if u not in visited_vertices:
                Q.put(u)
                visited_vertices.append(u)

print("BFS TRAVERSAL OF GRAPH WITH SOURCE A IS :")
BFS_TRAVERSAL(graph , 'A')
```

OUTPUT

21 Implement Depth first search Algorithm

```
graph = { 'A' : [ 'B' , 'D' , 'E' , 'F' ],
          'D' : [ 'A' ],
          'B' : ['A','F','C'],
          'F' : ['B' , 'A' ],
          'C' : ['B'],
          'E' : ['A']
          }
print("GIVEN GRAPH IS: ")
print(graph)

def  DFS_TRAVERSAL(input_graph , source):
     stack = list()
     visited_list = list()

     stack.append(source)
     visited_list.append(source)

     while stack:
          vertex=stack.pop()
          print(vertex,end="->")
          for u in input_graph[vertex]:
               if u not in visited_list:
                    stack.append(u)
                    visited_list.append(u)

print("DFS TRAVERSAL OF GRAPH WITH SOURCE A IS :")
DFS_TRAVERSAL(graph , 'A')
```

OUTPUT

22.Implement Hash Functions

```
class Hash:
 def  __init__(self):
     self.buckets=[[],[],[],[],[]]
 def insert(self,key):
       bindex = key % 5
       self.buckets[bindex].append(key)
       print(key,"inserted in Bucket No.",bindex+1)
   def search(se1f,key):
       bindex = key % 5
       if key in self.buckets[bindex]:
          print(key,"present in bucket No.",bindex+1)
       else:
          print(key,"is not present in any of the buckets")
   def display(self):
       for i in range(0,5):
          print("\nBucket No.",i+1,end=":")
          for j in self.buckets[i]:
             print(j,end="->")
hsh = Hash()
while True:
   print("\nHash operations 1.Insert 2.Search 3.Display 4.Quit")
   ch=int(input("Enter your choice:"))
   if ch == 1:
      key=int(input("Enter key to be inserted:"))
      hsh.insert(key)
   elif ch == 2:
      key=int(input("\nEnter key to be searched:"))
      hsh.search(key)
   elif ch == 3:
      hsh.display()
   else:
      break
```

OUTPUT