Week 1

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

```
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=(10,20,30,40,50,40)
print (t1)
print("Sets")
s1={10,30,20,40,10,30,40,20,50,50}
print(s1)
```

output

```
List

[1, 2, 'ABC', 3, 'xyz', 2.3]

Dictionary

{'a': 134, 'c': 343, 'b': 266}

Tuples

(10, 20, 30, 40, 50, 40)

Sets

set([40, 50, 20, 10, 30])
```

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 Leap year")
d1 = date(3,8,2000)
d1.day()
d1.month()
d1.year()
d1.monthName()
d1.isLeapYear()
```

output

```
 ('Day = ', 3)
('Month = ', 8)
('year = ', 2000)
```

```
('Month Name:', 'August')

It is a Leap year
```

week 2

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

```
True

push operations

('size:', 3)

[11, 12, 13]

('peek', 13)

pop operations

13

12

[11]

('size:', 1)

('Runtime of the program is', 0.0009570121765136719)
```

Week 3

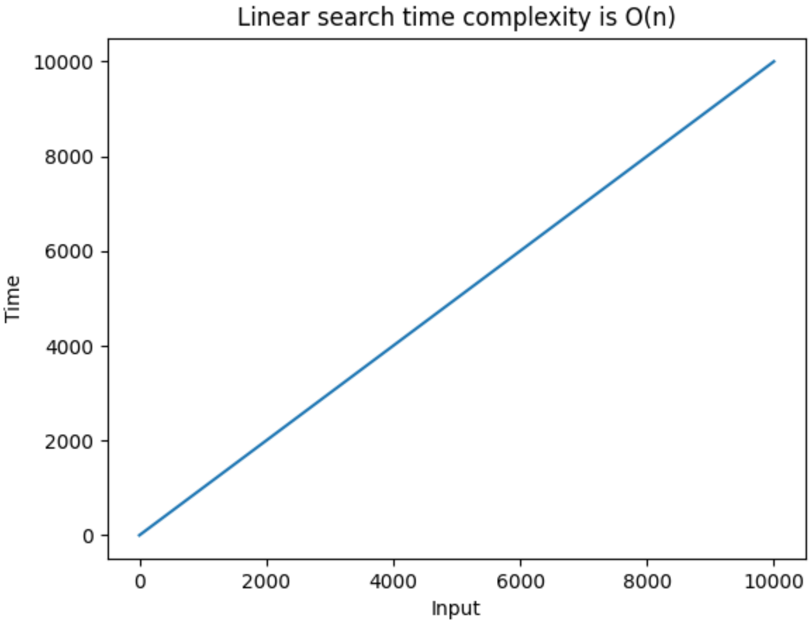
4. Implement Linear Search and compute space and time complexities, plot graph using asymptomatic notations

Linear search

```
import matplotlib
import matplotlib.pyplot as plt
matplotlib.use("TKAgg")
deflinearsearch(values,target):
    n=len(values)
    for i in range(n):
        if(values[i]==target):
            return True
    return False
values=[10,20,40,30,50,60,70,90,80]
target=8
if linearsearch(values,target):
print("Target is found in the list")
else:
print("Target is not in the list")
x=list(range(1,10000))
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()
```

<u>OutPut</u>

Target is not in the list



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

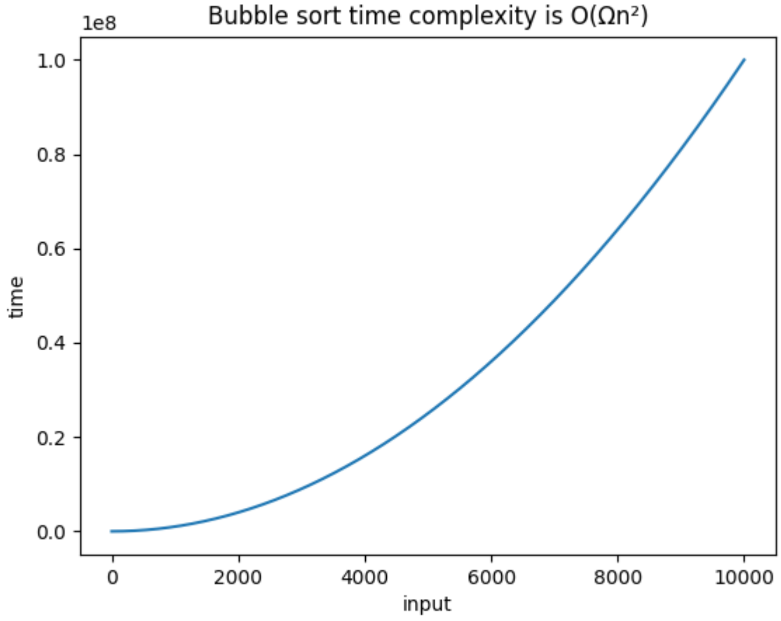
Bubble sort

```
import matplotlib

import matplotlib.pyplot as plt

matplotlib.use("TKAgg")

defbubblesort(seq):

    n=len(seq)

    for i in range(n-1):

        for j in range(n-1):

            if seq[j]>seq[j+1]:

                temp=seq[j]
```

```
seq[j]=seq[j+1]

seq[j+1]=temp

seq=[10,70,15,8,90,20]

print("before sorting: ",seq)

bubblesort(seq)

print("After sorting: ",seq)

x=list(range(1,10000))

plt.plot(x,[y*y for y in x])

plt.title("Bubble sort time complexity is O(Ωn\u00b2)")

plt.xlabel("input")

plt.ylabel("time")

plt.show()
```

<u>OutPut</u>

```
before sorting:  [10, 70, 15, 8, 90, 20]

After sorting:  [8, 10, 15, 20, 70, 90]
```



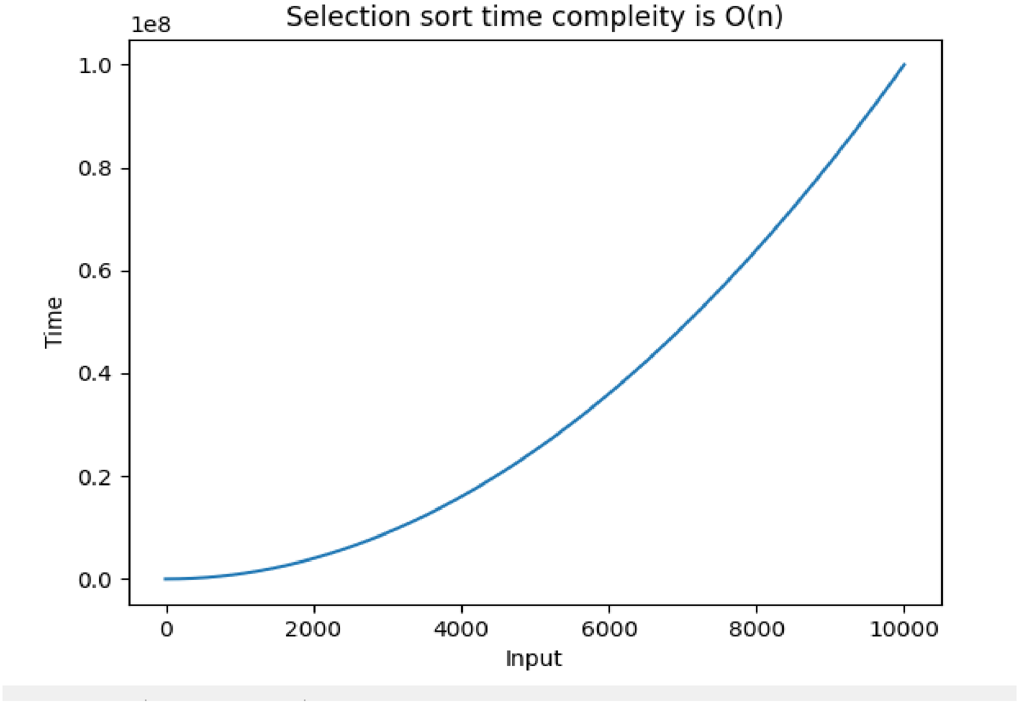
6. Implement Selection Sort and compute space and time complexities, plot graph using asymptomatic notations

```
import matplotlib

import matplotlib.pyplot as plt

matplotlib.use("TKAgg")

defselectionsort(seq):

    n=len(seq)

    for i in range(n):

        min=i

        for j in range(i+1,n):

            if seq[j]<seq[min]:

                temp=seq[min]

seq[min]=seq[j]

seq[j]=temp

seq=[56,53,32,66,21,78,965,64,2,54,2]

print("Before sorting:",seq)

selectionsort(seq)

print("After sorting:",seq)

x=list(range(1,10000))

plt.plot(x,[y*y for y in x])

plt.title("Selection sort time compleity is O(n)")

plt.xlabel("Input")

plt.ylabel("Time")

plt.show()
```

output

before sorting: [56,53,32,66,21,78,965,64,2,54,2]

After sorting: [2,2,21,32,53,54,56,64,66,78,965]



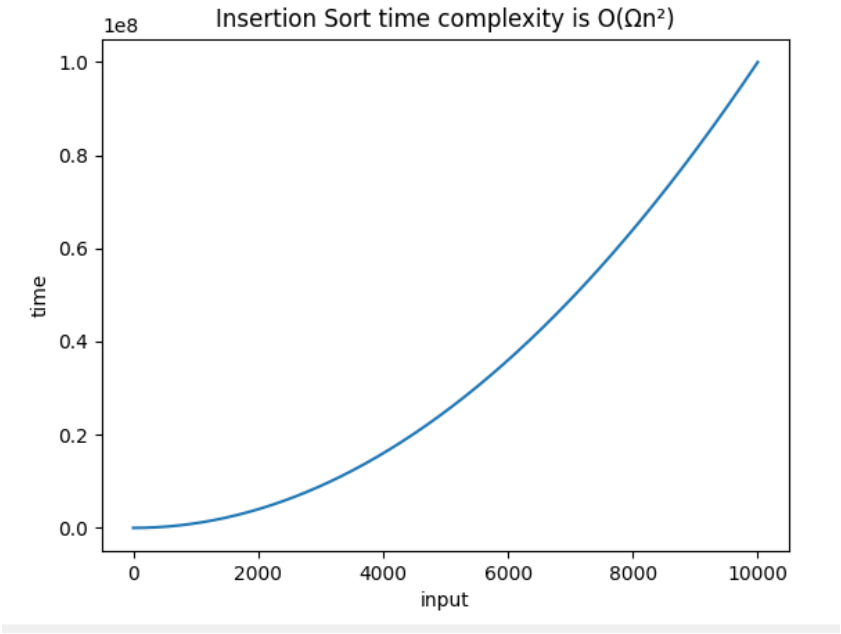
7. Implement Insertion Sort and compute space and time complexities, plot graph using asymptomatic notations

```
import matplotlib

import matplotlib.pyplot as plt

matplotlib.use("TKAgg")

definsertionsort(a):

  n=len(a)

  for i  in range(1,n):

     k=a[i]

    j=i-1
```

```
        while(j>=0 and a[j]>k):
            a[j+1]=a[j]
            j=j-1
            a[j+1]=k
x=[19,78,34,24,48,92,2]
print("Unsorted array is:",x)
insertionsort(x)
print("Sorted array is: ",x)
x=list(range(1,10000))
plt.plot(x,[y*y for y in x])
plt.title("Insertion Sort time complexity is O(Ωn\u00b2)")
plt.xlabel("input")
plt.ylabel("time")
plt.show()
```

output

```
Unsorted array is: [19, 78, 34, 24, 48, 92, 2]
Sorted array is:  [2, 19, 24, 34, 48, 78, 92]
```


Insertion Sort time complexity is $O(\Omega n^2)$
1e8
1.0
0.8
0.6
0.4
0.2
0.0
time
0
2000
4000
6000
8000
10000
input

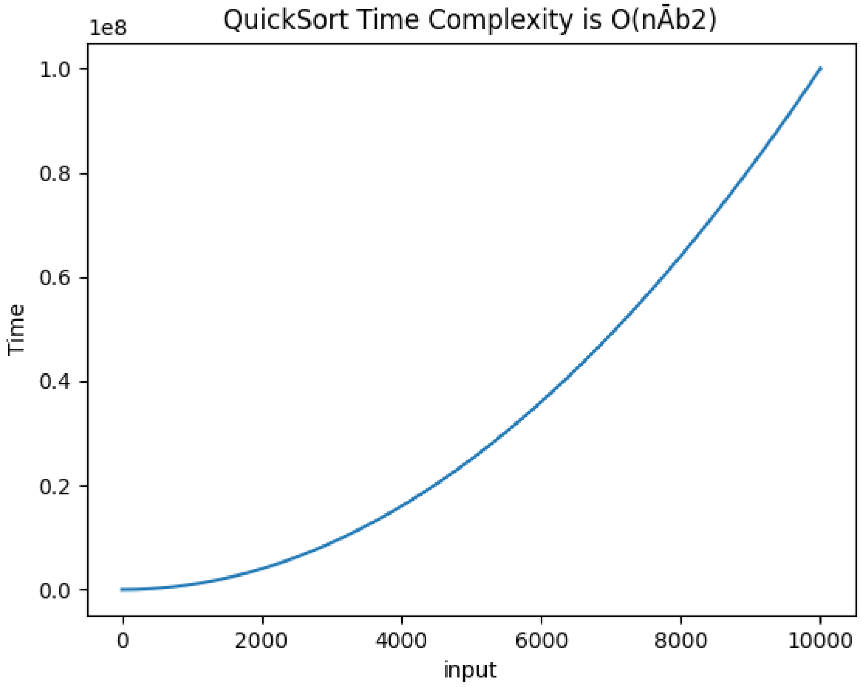
Week 4

8. Implement quick sort and compute space and time complexities, plot graph using asymptomatic notations.

```
import matplotlib

import matplotlib.pyplot as plt

matplotlib.use('TKAgg')

import math

def partition(array,low,high):

    pivot=array[low]

    left=low+1

    right=high

    while(True):

        while left<right and pivot>array[left]:

            left+=1

        while right>=left and array[right]>pivot:

         right-=1

        if left<right:

                array[left],array[right]=array[right],array[left]

        else:

            break

    array[low],array[high]=array[right],array[low]

    return right

defquick_sort(array,low,high):

    if low>high:

        return array
    else:
        pi=partition(array,low,high)
quick_sort(array,low,pi-1)
quick_sort(array,pi+1,high)
array=[10,70,15,8,60,90,20]
print("before sorting: ",array)
quick_sort(array,0,len(array)-1)
print("after sorting: ",array)
x=list(range(1,10000))
plt.plot(x,[y*y for y in x])
plt.title("QuickSort Time Complexity is O(n\400b2)")
plt.xlabel("input")
plt.ylabel("Time")
plt.show()
```

output

```
before sorting:  [10, 70, 15, 8, 60, 90, 20]
after sorting:  [8, 8, 10, 10, 15, 15, 60]
```



9. Implement Binary Search using Recursion and compute space and time

```
import matplotlib

import matplotlib.pyplot as plt

matplotlib.use('TKAgg')

defbinarysearch(a, low, high, key):

 if low <= high:

    mid = (high + low) // 2

    if a[mid] == key:

print("Search Successful key found at location:",mid+1)

        return

elif key < a[mid]:

binarysearch(a, low, mid-1, k)

    else:
```

```
binarysearch(a, mid + 1, high, k)
else:
print("Search UnSuccessful")
a = [13,24,35,46,57,68,79]
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
binarysearch(a, 0, len(a)-1, k)
x=list(range(1,10000))
plt.plot(x,[y*y for y in x])
plt.title("Binary Search -Time Complexity is O(log n)")
plt.xlabel("input")
plt.ylabel("Time")
plt.show()
#output
```

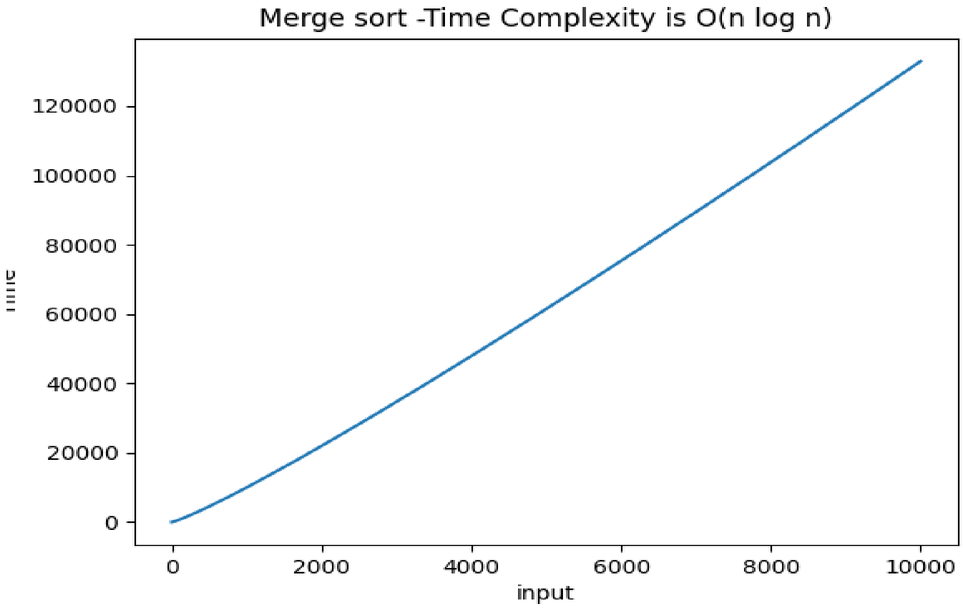
```
the array elements are: [13, 24, 35, 46, 57, 68, 79]
enter the key element to search:35
Search Successful key found at location: 3
```

10.Implement merge sort and compute space and time complexities, plot graph using asymptomatic notations.

```
import matplotlib
import matplotlib.pyplot as plt
matplotlib.use('TKAgg')
import math
defmergesort(arr):
    if len(arr)==1:
        return arr
    mid=int(len(arr)/2)
first_half=mergesort(arr[:mid])
second_half=mergesort(arr[mid:])
    return simplemerge(first_half,second_half)
defsimplemerge(L,r):
i=j=0
    temp=[]
    while i<len(L)and j<len(r):
        if L[i]<r[j]:
temp.append(L[i])
i=i+1
        else:
temp.append(r[j])
            j=j+1
    while i<len(L):
temp.append(L[i])
i=i+1
    while j<len(r):
temp.append(r[j])
        j=j+1
    return temp
arr=[40,80,10,50,30,20,70,60]
print("before sorting: ",arr)
result=mergesort(arr)
```

```
print("After sorting:",result)

x=list(range(1,10000))

plt.plot(x,[y*math.log(y,2) for y in x])

plt.title("Merge sort -Time Complexity is O(n log n)")

plt.xlabel("input")

plt.ylabel("Time")

plt.show()

#output
```

```
before sorting:  [40, 80, 10, 50, 30, 20, 70, 60]

After sorting: [10, 20, 30, 40, 50, 60, 70, 80]
```



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

```
Enter the term:10
('The Fibonacci sequence is:', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
('The Fibonacci value is:', 55)
```

Week 5

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)
    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
   cur = self.first
   while(cur):
    print(cur.data)
    cur = cur.next
  def search(self,item):
   if(self.first== None):
     print("list is empty")
     return
   cur = self.first
   while cur != None:
     if cur.data == item:
        print("Item is Present in the Linked list")
        return
     else:
        cur = cur.next
   print("Item is not present in the Linked list")
#Singly Linked List
sll = SinglyLinkedList()
while(True):
 ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
 if(ch == 1):
   item = input("Enter the element to insert:")
   sll.insertFirst(item)
   sll.display()
 elif(ch == 2):
   sll.removeFirst()
   sll.display()
 elif(ch == 3):
   item = input("Enter the element to search:")
   sll.search(item)
 elif(ch == 4):
   sll.display()
 else:
   break
```

#output

```
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :1
Enter the element to insert:5
5
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :1
Enter the element to insert:4
4
5
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :4
4
5
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :2
('the deleted item is', 4)
5
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :3
Enter the element to search:5
```

Item is Present in the Linked list

Week 6

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):
        cur = self.first
        while(cur.next):
          cur = cur.next
        cur.next = temp
     else:
        self.first = temp
   def __iter__(self):
    cur = self.first
    while cur:
      yield cur.data
      cur = cur.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

```
9

98

welcome

govt polytechnic koppal

456.35

545

5
```

Week 7

14. Implementation of Doubly linked list (DLL)(Traversing the Nodes, searching for a Node, Appending Nodes, Deleting Nodes):

```
class Node:
  def __init__(self, data = None):
    self.data = data
    self.next = None
    self.prev = None
class DoublyLinkedList:
  def __init__(self):
    self.first = None
  def insertAtEnd(self, data):
    temp = Node(data)
    if(self.first == None):
      self.first=temp
    else:
      cur = self.first
      while(cur.next != None):
        cur = cur.next
      cur.next = temp
      temp.prev = cur
  def deleteFirst(self):
   if(self.first== None):
     print("list is empty")
   elif(self.first.next == None):
    print("the deleted item is",self.first.data)
    self.first = None
 else:
    cur=self.first
    self.first=self.first.next
    self.first.prev = None
    print("the deleted item is",cur.data)
def display(self):
 if(self.first== None):
   print("list is empty")
   return
cur = self.first
while(cur):
 print(cur.data, end = " ")
 cur = cur.next
def search(self,item):
 if(self.first== None):
   print("list is empty")
   return
cur = self.first
while cur != None:
 if cur.data == item:
  print("Item is present in the Linked list")
  return
 else:
      cur = cur.next
    print("Item is not present in the Linked list")
#Doubly Linked List
dll = DoublyLinkedList()
while(True):
  ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
  if(ch == 1):
     item = input("Enter the element to insert:")
     dll.insertAtEnd(item)
     dll.display()
  elif(ch == 2):
     dll. deleteFirst()
     dll.display()
  elif(ch == 3):
     item = input("Enter the element to search:")
     dll.search(item)
  elif(ch == 4):
     dll.display()
  else:
     break
```

15. Implementation of Circular linked list (CLL)(Traversing the Nodes, searching for a Node, Appending Nodes, and Deleting Nodes):

```
class Node:
   def __init__(self, data = None):
     self.data = data
     self.next = None
class CircularLinkedList:
   def __init__(self):
     self.first = None
   def insertAtEnd(self, data):
     temp = Node(data)
     if(self.first == None):
       self.first = temp
       self.first.next = temp
     else:
       cur = self.first
       while(cur.next != self.first):
         cur = cur.next
       cur.next = temp
       temp.next = self.first
   def deleteAtEnd(self):
     if(self.first== None):
       print("list is empty")
     elif(self.first.next == self.first):
       print("the deleted item is",self.first.data)
       self.first = None
     else:
       cur=self.first
       while(cur.next != self.first):
         pr = cur
```

```
         cur = cur.next
       pr.next = self.first
       print("the deleted item is",cur.data)
  def display(self):
    if(self.first== None):
       print("list is empty")
       return
    cur = self.first
    while(True):
       print(cur.data)
       cur = cur.next
       if(cur == self.first):
          break
  def search(self,item):
    if(self.first== None):
       print("list is empty")
       return
    cur = self.first
    while cur.next != self.first:
      if cur.data == item:
       print("Item is present in the linked list")
       return
    else:
      cur = cur.next
      print("Item is not present in the linked list")
```

```
#Circular Linked List

cll = CircularLinkedList()

while(True):

  ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))

  if(ch == 1):

    item = input("Enter the element to insert:")

    cll.insertAtEnd(item)

    cll.display()

  elif(ch == 2):

    cll.deleteAtEnd()

    cll.display()

  elif(ch == 3):

    item = input("Enter the element to search:")

    cll.search(item)

  elif(ch == 4):

    cll.display()

  else:

    break
```

#output

```
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :1

Enter the element to insert:10

10

Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :1

Enter the element to insert:20

10
```

```
20

Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :1

Enter the element to insert:30

10

20

30

Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :1

Enter the element to insert:40

10

20

30

40

Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :2

('the deleted item is', 40)

10

20

30

Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :2

('the deleted item is', 30)

10

20

Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :3

Enter the element to search:10

Item is present in the linked list

Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :1
Enter the element to insert:50
10
20
50
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :4
10
20
50
Enter your choice 1-insert 2-delete 3-search 4-display 5-exit :5
```

week 8

16. 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():
    size = len(s)
    if(size== 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

```
Enter the size of Stack:10
1-Push 2-POP 3-DISPLAY 4-EXIT Enter your choice:1
Enter the element:2
1-Push 2-POP 3-DISPLAY 4-EXIT Enter your choice:3
2
1-Push 2-POP 3-DISPLAY 4-EXIT Enter your choice:2
('The deleted element is:', 2)
1-Push 2-POP 3-DISPLAY 4-EXIT Enter your choice:
```

17. 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

```
Matching
```

Week 9

18. 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

```
Enter the number:4
('The factorial of a number is:', 24)
```

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

```
Enter the range:3
('The fibonacci value is:', 2)
```

19. 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

```
 ('Move disk 1 from source', 'A', 'to destination', 'C')
('Move disk', 2, 'from source', 'A', 'to destination', 'B')
('Move disk 1 from source', 'C', 'to destination', 'B')
('Move disk', 3, 'from source', 'A', 'to destination', 'C')
('Move disk 1 from source', 'B', 'to destination', 'A')
('Move disk', 2, 'from source', 'B', 'to destination', 'C')
('Move disk 1 from source', 'A', 'to destination', 'C')
('Move disk', 4, 'from source', 'A', 'to destination', 'B')
('Move disk 1 from source', 'C', 'to destination', 'B')
('Move disk', 2, 'from source', 'C', 'to destination', 'A')
('Move disk 1 from source', 'B', 'to destination', 'A')
('Move disk', 3, 'from source', 'C', 'to destination', 'B')
```

```
('Move disk 1 from source', 'A', 'to destination', 'C')

('Move disk', 2, 'from source', 'A', 'to destination', 'B')

('Move disk 1 from source', 'C', 'to destination', 'B')
```

Week 10

20. 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:# or if len(q) == 0
     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

```
Enter the size of Queue:5
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:1
Enter the element:5
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:1
Enter the element:4
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:1
Enter the element:3
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:1
Enter the element:2
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:1
Enter the element:1
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:1
Queue is Full
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:2
('Element removed is:', 5)
1.Insert 2.Delete 3. Display 4. Quit Enter your choice:3
[4, 3, 2, 1]
```

21. Implement Priority Queue Data Structure.

```
class PriorityQEntry(object):
  def __init__(self, item, priority):
    self.item = item
    self.priority = priority
class PriorityQueue:
  def __init__(self):
    self.qList = list()
  def isEmpty(self):
    return len(self) == 0
  def __len__(self):
    return len(self.qList)
  def enqueue(self, item, priority):
    entry = PriorityQEntry(item, priority)
    if self.__len__() == 0:
     self.qList.append(entry)
    else:
       for x in range(0, len(self)):
         if entry.priority >= self.qList[x].priority:
           if x == (len(self)- 1):
             self.qList.insert(x + 1, entry)
           else:
             continue
         else:
           self.qList.insert(x, entry)
           return True
  def dequeue(self):
    assert not self.isEmpty(), "Cannot dequeue from an empty queue."
    return self.qList.pop(0)
  def display(self):
    for x in self.qList:
      print (str(x.item)+"-"+str(x.priority))
q=PriorityQueue()
print("Enque")
q.enqueue(25,3)
q.enqueue(50,2)
q.enqueue(75,1)
q.enqueue(100,6)
q.display()
print("Deque")
q.dequeue()
q.dequeue()
q.display
```

#output

```
75-1
50-2
25-3
100-6
Deque
```

Week 11

22. Program to implement binary search tree and its operation

```
class BST:
def __init__(self, value):
self.left = None
self.right = None
self.value = value

definsert(self, data):
    if data <self.value:
      if self.left is None:
self.left = BST(data)
      else:
self.left.insert(data)
elif data >self.value:
      if self.right is None:
self.right = BST(data)
      else:
self.right.insert(data)

defsearch(self, data):
    if self.value == data:
print("Node is Found")
      return
elif data <self.value:
```

```
            if self.left:
self.left.search(data)
            else:
print("Node is not present in the tree")
        else:
            if self.right:
self.right.search(data)
            else:
print("Node is not present in the tree")

definorder(self):
        if self.left:
self.left.inorder()
print(self.value, end=" ")
        if self.right:
self.right.inorder()

defdelete(self, data):
        if self.value is None:
print("Tree is Empty")
            return self
        if data <self.value:
            if self.left:
self.left = self.left.delete(data)
            else:
```

```
print("The given node is not present in the tree")
elif data >self.value:
      if self.right:
self.right = self.right.delete(data)
      else:
print("The given node is not present in the tree")
    else:
      if self.left is None:
        temp = self.right
self.right = None
        return temp
elifself.right is None:
        temp = self.left
self.left = None
        return temp
      else:
min_node = self.right.find_min()
self.value = min_node.value
self.right = self.right.delete(min_node.value)
    return self
deffind_min(self):
    current = self
    while current.left:
      current = current.left
    return current
```

```
root = BST(10)

data_list = [6, 12, 1, 16, 98, 3, 7]

for i in data_list:

root.insert(i)

print("Inorder traversal before deleting the node:")

root.inorder()

print()

root.delete(6)

print("After deleting the node:")

root.inorder()
```

output

```
Tree elements are   1   3   6   7    10   12   16  98

After deleting the node   1  3   7   10   12   16   98
```

Week 12

23.Program for implementations of BFS

```
graph={
'5':['3','7'],
'3':['2','4'],
'7':['8'],
'2':[],
'4':['8'],
'8':[]
}
visited=[]
queue=[]
def bfs(visited,graph,node):
  visited.append(node)
  queue.append(node)
  while queue:
   m=queue.pop(0)
   print(m)
   for neighbour in graph[m]:
    if neighbour not in visited:
      visited.append(neighbour)
      queue.append(neighbour)
print("bfs")
bfs(visited,graph,'5')
```

#output

```
following is the breath first search

5      3      7     2      4      8
```

24.Program for implementations of DFS

```
def dfs(graph,start,visited=None):
  if visited is None:
    visited=set()
  visited.add(start)
  print(start)
  for next in graph[start] - visited:
    dfs(graph, next, visited)
  return visited
graph={'0':set(['1',2]),
       '1':set(['0','3','4']),
       '2':set(['0']),
       '3':set(['1']),
       '4':set(['2','3'])}
dfs(graph,'0')
,graph,'5')
```

output

```
following is the depth first search

0      1    2    4    3
```

Week 13

25.Program to implement hash functions

```
def display_hash(hashtable):
  for i in range(len(hashtable)):
    print(i)
    for j in hashtable[i]:
        print("-->")
        print(j)
hashtable=[[]for _ in range(10)]
def hashing(keyvalue):
  return keyvalue%len(hashtable)
def insert(hashtable,keyvalue,value):
  hash_key=hashing(keyvalue)
  hashtable[hash_key].append(value)
insert(hashtable,10,"allahabad")
insert(hashtable,25,"mumbai")
insert(hashtable,20,"mathura")
insert(hashtable,9,"delhi")
insert(hashtable,21,"punjub")
insert(hashtable,21,"noida")
display_hash(hashtable)
```

output

```
0→allahabad-→mathura
1→punjab-→noida
2
```

```
3
4
5→mumbai
6
7
8
9→delhi
```