NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY
GREATER NOIDA-201306
(An Autonomous Institute)
School of Computer Sciences & Engineering in Emerging Technologies

Department of CSE (Data Science)

Session (2021 – 2022)

LAB FILE

ON

Data Structure using Python

(ACSE-0301)

(3rd Semester)

Submitted To:

Mrs. Sonia Arora.

Submitted By:

Name: Amritanshu Sharma
Roll No: 2001331540025

Affiliated to Dr. A.P.J Abdul Kalam Technical University, Uttar Pradesh, Lucknow.

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY GREATER NOIDA-201306

(An Autonomous Institute)

School of Computer Sciences & Engineering in Emerging Technologies

Data Structure using Python (ACSE-0301)

INDEX

19.	Program to implement the circularly Single Linked List a. Insertion b. Deletion c. Traversal d. Reversal e. Searching f. Updation			
20.	Program to implement Queue Using linked list.			
21	Program to implement Circular Queue Using linked list.			
22.	Program to implement Circular Queue Using linked list			
23.	Program to implement Stack Operation using Linked list.			
24.	Program to convert infix to postfix expression.			
25.	Program to evaluate postfix expression.			
26.	Program to compute factorial using tail recursion			
27.	Program to implement Tower of Hanoi.			
28.	Program implementing Addition of two polynomials via Linked Lists.			
29.	Program to implement binary tree using linked list a. Insertion b. Deletion c. Traversal d. Searching			
30.	Program to implement binary search tree using linked list a. Insertion b. Deletion c. Traversal d. Searching			
31.	Program to implement Heap sort in a non-recursive way			
32.	Program to implement Radix sort.			
33.	Program to implement BFS algorithm.			
34.	Program to implement DFS algorithm.			

Q1. Program to create and display Linear Array

```
n = int(input("Enter how many elements you want:"))
list = []
print("Enter numbers in array: ")
for i in range (0, n):
        z=int(input("n:"))
        list.append(z)
print("ARRAY: ", list)
```

OUTPUT:

```
Enter how many elements you want:7
Enter numbers in array: -1
n:-1
n:-2
n:-3
n:4
n:2
n:-7
n:-7
ARRAY: [-1, -2, -3, 4, 2, -6, -7]
```

Q2. Program to insert data item at any location in a linear array

```
num = int(input("Enter how many elements you want:"))
print("Enter numbers in array:")
list = []
for a in range(num):
    z = int(input("num:"))
    list.append(z)
print("ARRAY:", list)
pos=int(input("Enter position you want to enter element:"))
ele=int(input("Enter the element you want to enter:"))
list.insert(pos, ele)
print(list)
```

OUTPUT:

```
Enter how many elements you want:4
Enter numbers in array:1
num:1
num:2
num:7
num:4
ARRAY: [1, 2, 7, 4]
Enter position you want to enter element:2
Enter the element you want to enter:10
 [1, 2, 10, 7, 4]
```

Q3. Program to delete a data item from a linear array

```
no = int(input("Enter how many elements you want:"))

print("Enter numbers in array:")

list=[]

for _ in range(no):

        ele = int(input("num:"))

        list.append(ele)

print("ARRAY:", list)

pos=int(input("Enter position you want to delete element:"))

list.pop(pos)

print(list)
```

OUTPUT:

```
Enter how many elements you want:4

Enter numbers in array:1

num:1

num:8

num:3

num:4

ARRAY: [1, 8, 3, 4]

Enter position you want to delete element:2

[1, 8, 4]
```

Q4. Program to implement multiplication of two Matrixes.

```
print("Enter values for matrix - A")
row1= int(input("Number of rows, m = "))
col1= int(input("Number of columns, n = "))
a=[]
b=[]
ab=[]

for r1 in range(1, row1+1):
        l=[]
        for c1 in range(1, col1+1):
                print("Entry in row:",r1,"column:",c1)
                val=int(input())
                l.append(val)
        a.append(l)

print("Enter values for matrix - B")
row2= int(input("Number of rows, m = "))
col2=int(input("Number of columns, n = "))

for r2 in range(1, row2+1):
        l=[]
        for c2 in range(1, col2+1):
```

```
        print("Entry in row:",r2,"column:",c2)
        val2=int(input())
        l.append(val2)
    b.append(l)

if ((row1+col1)<(row2+col2)):
    for i in range(row2):
        l=[]
        for i in range(col2):
            l.append(0)
        ab.append(l)
else:
    for i in range(row1):
        l=[]
        for j in range(col1):
            l.append(0)
        ab.append(l)



print("Matrix - A =",a)
print("Matrix - B =",b)

for i in range(len(b)):
    for j in range(len(b[0])):
        for k in range(len(b)):
            ab[i][j]+=a[i][k]*b[k][j]
print("Matrix - A * Matrix- B =",ab)
```

OUTPUT:

```
Enter values for matrix - A3
Number of rows, m = 3
Number of columns, n = 3
Entry in row: 1 column: 112
Entry in row: 1 column: 27
Entry in row: 1 column: 33
Entry in row: 2 column: 14
Entry in row: 2 column: 25
Entry in row: 2 column: 36
Entry in row: 3 column: 17
Entry in row: 3 column: 28
Entry in row: 3 column: 39
Enter values for matrix - B3
Number of rows, m = 3
Number of columns, n = 4
Entry in row: 1 column: 15
Entry in row: 1 column: 28
Entry in row: 1 column: 31
Entry in row: 1 column: 42
Entry in row: 2 column: 16
Entry in row: 2 column: 27
Entry in row: 2 column: 33
Entry in row: 2 column: 40
Entry in row: 3 column: 14
Entry in row: 3 column: 25
Entry in row: 3 column: 39
Entry in row: 3 column: 41
Matrix - A = [[12, 7, 3], [4, 5, 6], [7, 8, 9]]
Matrix - B = [[5, 8, 1, 2], [6, 7, 3, 0], [4, 5, 9, 1]]
Matrix - A * Matrix- B = [[114, 160, 60, 27], [74, 97, 73, 14], [119, 157, 112, 23]]
```

Q5. Program to createa Sparse matrix.

```
def tosparse(mat):
    sparse = []
    for i in range(len(mat)):
        for j in range(len(mat[0])):
            if mat[i][j]!=0:
                temp=[]
                temp.append(i)
                temp.append(j)
                temp.append(mat[i][j])
                sparse.append(temp)
    return sparse
l=[]
print("Enter values for Matrix ")
row = int(input("Number of rows, m = "))
column = int(input("Number of columns, n = "))
for i in range(row):
    l1=[]
    for j in range(column):
        print("Entry in row:",i+1,"column:",j+1)
        a=int(input())
        l1.append(a)
    l.append(l1)
print("Matrix =",l)
print("Sparse Matrix: ")
sm=tosparse(l)
for i in range(len(sm)):
    for j in range(len(sm[0])):
        print(sm[i][j], end=' ')
    print()
```

OUTPUT:

```
Enter values for Matrix 2
Number of rows, m = 2
Number of columns, n = 3
Entry in row: 1 column: 11
Entry in row: 1 column: 22
Entry in row: 1 column: 33
Entry in row: 2 column: 14
Entry in row: 2 column: 25
Entry in row: 2 column: 36
Matrix = [[1, 2, 3], [4, 5, 6]]
Sparse Matrix:
0 0 1
0 1 2
0 2 3
1 0 4
1 1 5
1 2 6
```

Q6. Program to implement linear search in an Array.

```
s=input("Enter the list of numbers: ").split()
l=[int(i) for i in s]
n=int(input("The number to search for: "))
length=len(l)
for i in range(length):
    if n==l[i]:
        print(n," was found at index ",i,".",sep="")
        break
else:
    print(n,"was not found.")
```

OUTPUT:

```
Enter the list of numbers: 12 23 45 3 2 1
The number to search for: 3
3 was found at index 3.
```

Q7. Program to implement Binary search in an Array.

```
def binarysearch(l,low,high,num):
    if high>=low:
            mid=(high+low)//2
            if l[mid]==num:
                    return mid
            elif l[mid]>num:
                    return binarysearch(l,low,mid-1,num)
            elif l[mid]<num:
                    return binarysearch(l,mid+1, high, num)
            else:
                    return -1

arr = []
length = int(input("Enter size of list: "))
for i in range(length):
    a = int(input("Enter your number: "))
    arr.append(a)
arr.sort()
print("After sorting list is: ",arr)
num = int(input("The number to search for: "))
search = binarysearch(arr,0,len(arr),num)
```

```
if search!=1:
    print(num," was found at index ",search,".", sep=")
```

OUTPUT:

```
Enter size of list: 5
Enter your number: 12
Enter your number: 3
Enter your number: 45
Enter your number: 68
Enter your number: 95
After sorting list is: [3, 12, 45, 68, 95]3
The number to search for: 3
3 was found at index 0.
```

Q8. Program to implement Bubble Sort in a non-recursive way.

```
def bubble(l):
        n=len(l)
        for i in range(n-1):
                for j in range(n-i-1):
                        if l[j]>l[j+1]:
                                l[j],l[j+1]=l[j+1],l[j]
s=input("Enter the list of numbers: ").split()
l=[int(i) for i in s]
bubble(l)
print("Sorted list:",l)
```

OUTPUT:

```
Enter the list of numbers: 25 98 74 36 -7
Sorted list: [-7, 25, 36, 74, 98]
```

Q9. Program to implement selection sort in a non-recursive way.

```
l = [x for x in input("Enter the list of numbers: ").split(" ")]
for i in range(len(l)):
    max = int(l[i])
    temp=i
    for j in range(i+1, len(l)):
            if max>int(l[j]):
                    max=int(l[j])
                    temp=j
    r=l[i]
    l[i]=l[temp]
    l[temp]=r
print(l)
```

OUTPUT:

```
Enter the list of numbers: 44 55 2 3
['2', '3', '44', '55']
```

Q10. Program to implement Insertion sort in non -recursive way.

```
def insertionsort(a):
    length = len(a)
    for i in range(1 ,length):
        sample = a[i]
        j=i-1
        while j>=0 and sample<a[j]:
            a[j+1]=a[j]
            j-=1
        a[j+1]=sample
s = input("Enter the list of numbers: ").split()
print(s)
a = [int(i) for i in s]
insertionsort(a)
l=[str(i) for i in a]
print(l)
```

OUTPUT:

```
Enter the list of numbers: 25 98 63 78 99 54
['25', '98', '63', '78', '99', '54']
['25', '54', '63', '78', '98', '99']
```

Q11. Program to implement Merge sort in a non-recursive way.

```
def mergesort(a):
    width =1
    n=len(a)

    while (width <n):
        l=0
        while(l<n):
            r=min(l+(width *2-1),n-1)
            m=(l+r)//2
            if (width>n//2):
                m=r-(n%width)
            merge(a,l,m,r)
            l+=width*2
        width*=2
    return a
def merge(a,l,m,r):
    n1=m-l+1
    n2=r-m
    L=[0]*n1
    R=[0]*n2
    for i in range(0,n1):
        L[i]=a[l+i]
    for i in range(0,n2):
        R[i]=a[m+i+1]

    i,j,k=0,0,l
    while i<n1 and j<n2:
        if L[i]>R[j]:
            a[k]=R[j]
            j+=1
        else:
            a[k]=L[i]
            i+=1
        k+=1

    while i<n1:
        a[k]=L[i]
        i+=1
        k+=1
    while j<n2:
        a[k]=R[j]
        j+=1
        k+=1
a=[]
```

```
nu=int(input("Enter no ofelements"))

print("enter elements")

for i in range(nu):

        no=int(input())

        a.append(no)

print("Given array is ")

print(a)

mergesort(a)

print("Sorted array is ")

print(a)
```

OUTPUT:

```
Enter no ofelements3

enter elements

6

4

2

Given array is

[6, 4, 2]

Sorted array is

[2, 4, 6]
```

Q12.Program to implement Merge sort in a recursive way

```
def merge(left,right):
    if not len(left) or not(right):
        return left or right

    result=[]
    i,j=0,0
    while (len(result)<len(left)+ len(right)):
        if left[i]<right[j]:
            result.append(left[i])
            i+=1
        else:
            result.append(right[j])
            j+=1
        if i==len(left) or j ==len(right) :
            result.extend(left[i:] or right[j:])
            break
    return result

def mergesort(list):
    if len(list)<2:
        return list
    middle=len(list)//2
    left=mergesort(list[:middle])
    right=mergesort(list[middle:])
    return merge(left,right)
l=[]
```

```
number=int(input("Enter no ofelements"))
print("enter elements")
for i in range(number):
        tx=int(input())
        l.append(tx)
b=mergesort(l)
print(b)
```

OUTPUT:

```
Enter no ofelements5
enter elements
9
6
7
8
1
[1, 6, 7, 8, 9]
```

Q13-Program to implement Quick sort in a recursive way.

```
def partition(arr,low,high):
    i=(low-1)
    pivot=arr[high]
    for j in range(low,high):
        if arr[j]<=pivot:
            i+=1
            arr[i],arr[j]=arr[j],arr[i]
    arr[i+1],arr[high]=arr[high],arr[i+1]
    return (i+1)
def quicksort(arr,low,high):
    if len(arr)==1:
        return arr
    if low<high:
        pi=partition(arr,low,high)
        quicksort(arr,low,pi-1)
        quicksort(arr,pi+1,high)


a=[]
no=int(input("Enter no ofelements"))
print("enter elements")
for i in range(no):
    iz=int(input())
    a.append(iz)
print("Unsorted Array")
print(a)
n=len(a)
```

```
quicksort(a,0,n-1)
print("Sorted Array in Ascending Order:")
print(a)
```

OUTPUT:

```
Enter no ofelements4
enter elements
2
3
1

6
Unsorted Array
[2, 3, 1, 6]
Sorted Array in Ascending Order:
[1, 2, 3, 6]
```

Q14. Program to implement Queue Using array

```
q=[]
def Enqueue():
    if len(q)==size:
        print("Queue is Full!!!!")
    else:
        element=input("Enter the element:")
        q.append(element)
def dequeue():
    if not q:
        print("Queue is Empty!!!")
    else:
        e=q.pop(0)
            # print("element removed!!",e)
def display():
    for i in q:
        print(i+' ')
size=int(input("Enter the size of Queue:"))

while True:
    print("Select the Operation:")
    print("1.Enqueue 2.Dequeue 3. Display 4. Quit")
    choice=int(input())
    if choice ==1:
        Enqueue()
    elif choice ==2:
        dequeue()
    elif choice ==3:
        display()
    elif choice ==4:
        break
    else:
        print("Invalid Option!!!")
```

OUTPUT:

```
Enter the size of Queue:3
Select the Operation:1
1.Enqueue 2.Dequeue 3. Display 4. Quit1

Enter the element:5
Select the Operation:1
1.Enqueue 2.Dequeue 3. Display 4. Quit1

Enter the element:6
Select the Operation:2
1.Enqueue 2.Dequeue 3. Display 4. Quit2

Select the Operation:3
1.Enqueue 2. Dequeue 3. Display 4. Quit3

6 4
Select the Operation:4
1.Enqueue 2.Dequeue 3. Display 4. Quit4
```

Q15. Program to implement circular queue using array

```
class CircularQueue():

  # constructor
  def __init__(self, size): # initializing the class
    self.size = size

    # initializing queue with none
    self.queue = [None for i in range(size)]
    self.front = self.rear = -1

  def enqueue(self, data):

    # condition if queue is full
    if ((self.rear + 1) % self.size == self.front):
      print(" Queue is Full\n")

    # condition for empty queue
    elif (self.front == -1):
      self.front = 0
      self.rear = 0
      self.queue[self.rear] = data
    else:

      # next position of rear
      self.rear = (self.rear + 1) % self.size
      self.queue[self.rear] = data

  def dequeue(self):
    if (self.front == -1):
      print ("Queue is Empty\n")

    # condition for only one element
    elif (self.front == self.rear):
      temp=self.queue[self.front]
      self.front = -1
      self.rear = -1
      return temp
    else:
      temp = self.queue[self.front]
      self.front = (self.front + 1) % self.size
      return temp
  def display(self):

    # condition for empty queue
    if(self.front == -1):
      print ("Queue is Empty")

    elif (self.rear >= self.front):
      print("Elements in the circular queue are:", end = " ")
      for i in range(self.front, self.rear + 1):
        print(self.queue[i], end = " ")
      print ()

    else:
      print ("Elements in Circular Queue are:",end = " ")
      for i in range(self.front, self.size):
        print(self.queue[i], end = " ")
      for i in range(0, self.rear + 1):
        print(self.queue[i], end = " ")
      print ()

    if ((self.rear + 1) % self.size == self.front):
      print("Queue is Full")
```

```
# __main__ Driver-Code

ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
```

OUTPUT:

```
Elements in Circular Queue are: 14 22 13 -6
Deleted value = 14
Deleted value = 22
Elements in Circular Queue are: 13 -6
Elements in Circular Queue are: 13 -6 9 20 5
Queue is Full
```

Q16-Program to implement Stack operations using array

```
class StackUsingArray:
  def __init__(self):
    self.stack = []
  def push(self, element):
    self.stack.append(element)
  def pop(self):
    if(not self.isEmpty()):
      lastElement = self.stack[-1]
      del(self.stack[-1])
      return lastElement
    else:
      return("Stack Already Empty")
  def isEmpty(self):
    return self.stack == []

  def printStack(self):
    print(self.stack)



if __name__ == "__main__":
  s = StackUsingArray()
  while(True):
    el = int(input("1 for Push\n2 for Pop\n3 to check if it is Empty\n4 to print Stack\n5 to
exit\n"))
    if(el == 1):
      item = input("Enter Element to push in stack\n")
      s.push(item)
    if(el == 2):
      print(s.pop())
    if(el == 3):
      print(s.isEmpty())
    if(el == 4):
      s.printStack()
    if(el == 5):
      break
```

OUTPUT:

```
1 for Push
2 for Pop
3 to check if it is Empty
4 to print Stack
5 to exit
1
Enter Element to push in stack
4
1 for Push
2 for Pop
3 to check if it is Empty
4 to print Stack
5 to exit
1
Enter Element to push in stack
5
1 for Push
2 for Pop
3 to check if it is Empty
4 to print Stack
5 to exit
1
Enter Element to push in stack
7
1 for Push
2 for Pop
3 to check if it is Empty
4 to print Stack
5 to exit
3
False
1 for Push
2 for Pop
3 to check if it is Empty
4 to print Stack
5 to exit
2
7
```

```
1 for Push
2 for Pop
3 to check if it is Empty
4 to print Stack
5 to exit
4
['4', '5']
1 for Push
2 for Pop
3 to check if it is Empty
4 to print Stack
5 to exit
5
```

Q17. Program to implement single linked list

a. Insertion **b. Deletion** **c. Traversal** **d. Reversal**
e. Searching **f. Updation** **g. Sorting** **h. Merging**

```
# 17(a). Insertion
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def inser(self,value):
        newN=Node(value)
        if self.head==None:
            self.head=newN
        else:
            newN.next=self.head
            self.head=newN
    def disp(self):
        n=self.head
        print("The Inserted elements at the front end are :")
        while n!=None:
            print(n.data)
            n=n.next
y=1
l=linkedlist()
while(y!=3):
    y=int(input("Select a Operation: 1.Insertion 2.Display 3.Quit "))
    if y==1:
        s=int(input("Enter element "))
        l.inser(s)
    elif y==2:
        l.disp()
    elif y==3:
        break
    else:
        print("Invalid Option!!!")
```

OUTPUT:

```
Select a Operation: 1.Insertion 2 Display 3 Quit 1
Enter element 23
Select a Operation: 1 Insertion 2 Display 3 Quit 1
Enter element 121
Select a Operation: 1 Insertion 2 Display 3 Quit 1
Enter element 34
Select a Operation: 1 Insertion 2 Display 3 Quit 2
The Inserted elements at the front end are :
34
121
23
```

#17(b). Deletion

```
class node:
    def __init__(self,data):
        self data=data
        self next=None
class ll:
    def __init__(self):
        self head=None
        self v=-1
    def inser(self,value):
        newN=node(value)
        if self head==None:
            self head=newN
            self v=self v+1
        else:
            newN next=self head
            self head=newN
            self v=self v+1
    def disp(self):
        n=self head
        while n!=None:
            print(n data)
            n=n next
    def delet(self,x):
        temp=self head
        if x==0:
            temp next=temp next next
        elif x>self v:
            print("Position is more than number of nodes")
        else:
            for i in range(1,x+1):
                if i==x:
                    temp next=temp next next
                    break
Y=1
l=ll()
while(Y!=4):
    Y=int(input("Select an Operation:\n1 Insert\n2 Deletion\n3 Display\n4 Quit\t"))
    if Y==1:
        s=int(input("Enter Element "))
```

```
            l inser(s)
        elif Y==2:
            z=int(input("Enter a position "))
            l delet(z)
        elif Y==3:
            l disp()
        elif Y==4:
            break
        else:
            print("Invalid Option!!!")
```

OUTPUT:

```
Select an Operation:
1 Insert
2 Deletion
3 Display
4 Quit→1
Enter Element 9
Select an Operation:
1 Insert
2 Deletion
3 Display
4 Quit→1
Enter Element 10
Select an Operation:
1 Insert
2 Deletion
3 Display
4 Quit→2
Enter a position 2
Position is more than number of nodes
Select an Operation:
1 Insert
2 Deletion
3 Display
4 Quit→3
10
9
Select an Operation:
1 Insert
```

```
2 Deletion
3 Display
4 Quit→4
```

```
#17(c). Traversal
class Node:
    def __init__(self,data):
        self data=data
        self next=None
class ll:
    def __init__(self):
        self head=None
    def inser(self,value):
        newN=Node(value)
        if self head==None:
            self head=newN
        else:
            newN next=self head
            self head=newN
    def display(self):
        temp=self head
        while temp!= None:
            print(temp data)
            temp=temp next
l=ll()
n=int(input("Enter how many elements would you like to add: "))
for i in range(n):
    s=int(input("Enter data elements: "))
    l inser(s)
print("The linked list is: ")
l display()
```

OUTPUT

```
Enter how many elements would you like to add: 5
Enter data elements: 1
Enter data elements: 2
Enter data elements: 3
Enter data elements: 4
Enter data elements: 5
The linked list is:
5
4
3
2
1
```

#17(d). Reversal

```
class node:
    def __init__(self,data):
        self data=data
        self next=None
class ll:
    def __init__(self):
        self head=None
    def inser(self,value):
        newN=node(value)
        if self head==None:
            self head=newN
        else:
            newN next=self head
            self head=newN
    def display(self):
        temp=self head
        while temp!=None:
            print(temp data)
            temp=temp next
    def reverse(self):
        current=self head
        nnext=None
        prev=None
        while(current!=None):
            nnext=current next
            current next=prev
            prev=current
            current=nnext
        self head=prev
```

```
l=ll()
y=1
while(y!=4):
    y=int(input("Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit "))
    if y==1:
        s=int(input("Enter number "))
        l inser(s)
    elif y==2:
        l reverse()
    elif y==3:
        l display()
    elif y==4:
        break
```

OUTPUT:

```
Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit 1
Enter number 5
Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit 1
Enter number 8
Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit 1
Enter number 15
Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit 3
15
8
5
Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit 2
Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit 3
5
8
15
Select a option: 1 Insertion 2 Reversal 3 Display 4 Quit 4
```

#17(e). Searching

```
class node:
    def __init__(self,data):
        self data=data
        self next=None
class ll:
    def __init__(self):
        self head=None
    def inser(self,value):
        newN=node(value)
        if self head==None:
            self head=newN
        else:
            newN next=self head
            self head=newN
    def search(self,v):
        temp=self head
        c=0
        while temp!=None:
            if temp data==v:
                c=c+1
                break
            temp=temp next
        if c==1:
            print("Item found")
        else:
            print("item not found")

    def display(self):
        temp=self head
        while temp!=None:
            print(temp data)
            temp=temp next
l=ll()
y=1
while y!=4:
    y=int(input("Select Operation:\n1 Insertion\n2 Searching\n3 Display\n4 Quit\t"))
    if y==1:
        s=int(input("Enter elements "))
        l inser(s)
    elif y==2:
        j=int(input("Enter a key to search "))
        l search(j)
    elif y==3:
        l display()
    elif y==4:
        break
```

OUTPUT:

```
Select Operation:
1 Insertion
2 Searching
3 Display
4 Quit→1
Enter elements 7
Select Operation:
1 Insertion
2 Searching
3 Display
4 Quit→1
Enter elements 9
Select Operation:
1 Insertion
2 Searching
3 Display
4 Quit→2
Enter a key to search 7
Item found
Select Operation:
1 Insertion
2 Searching
3 Display
4 Quit→3
9
7
Select Operation:
1 Insertion
2 Searching
3 Display
4 Quit→4
```

#17(f). Updation

```
class node:
    def __init__(self,data):
        self data=data
        self next=None
class ll:
    def __init__(self):
        self head=None
    def inser(self,value):
        newN=node(value)
        if self head==None:
            self head=newN
        else:
            newN next=self head
            self head=newN
    def display(self):
        temp=self head
        while temp!=None:
            print(temp data)
            temp=temp next
    def update(self,pos,val):
        temp=self head
        if pos==0:
            temp data=val
        else:
            temp=temp next
            for i in range(1,pos+1):
                if i==pos:
                    temp data=val
                    break
                temp=temp next
l=ll()
y=1
while y!=4:
    y=int(input("Select Operation\n1 Insertion\n2 Updation\n3 Display\n4 Quit\t"))
    if y==1:
        s=int(input("Enter element "))
        l inser(s)
    elif y==2:
        j=int(input("Enter the index to update "))
        k=int(input("Enter a value to update "))
        l update(j,k)
    elif y==3:
        l display()
    elif y==4:
        break
```

OUTPUT:

```
Select Operation
1 Insertion
2 Updation
3 Display
4 Quit→1
Enter element 2
Select Operation
1 Insertion
2 Updation
3 Display
4 Quit→1
Enter element 7
Select Operation
1 Insertion
2 Updation
3 Display
```

```
4 Quit→3
7
2
Select Operation
1 Insertion
2 Updation
3 Display
4 Quit→2
Enter the index to update 1
Enter a value to update 9
Select Operation
1 Insertion
2 Updation
3 Display
4 Quit→3
7
9
```

Q18. Program to implement doubly linklist

#18(a). Insertion

```
class Node:
    def __init__(self,data):
        self data=data
        self next=None
        self prev=None
class dll:
    def __init__(self):
        self head=None
    def inser(self,value):
        new=Node(value)
        if self head==None:
            self head=new
        else:
            temp=self head
            while temp next!= None:
                temp=temp next
            temp next=new
            new prev=temp
    def display(self):
        temp=self head
        while temp!= None:
            print(temp data)
            temp=temp next
l=dll()
y=1
while y!=3:
    y=int(input("Select Opertion\n1 Insertion\n2 Display\n3 Quit\t"))
    if y==1:
        s=int(input("enter element "))
        l inser(s)
    elif y==2:
        print("Adding a node to the end of the list: ")
        l display()
    elif y==3:
        break
```

OUTPUT:

```
Select Opertion
1 Insertion
2 Display
3 Quit→1
enter element 3
Select Opertion
1 Insertion
2 Display
3 Quit→1
enter element 5
Select Opertion
1 Insertion
2 Display
3 Quit→2
Adding a node to the end of the list:
3
5
Select Opertion
1 Insertion
2 Display
3 Quit→3
```

```
# Traversal
class Node:
    def __init__(self,data):              self data=data
        self prev=None
        self next=None
class dll:
    def __init__(self):
        self head=None
    def inser(self,value):
        newN=Node(value)
        if self head==None:
            self head=newN
        else:
            newN next=self head
            self head prev=newN
            self head=newN
    def forward(self):
        temp=self head
        print("Traversal in forward direction")
        while temp!=None:
            print(temp data)
            temp=temp next
    def backward(self):
        temp=self head
        print("Traversal in reverse direction")
        while temp next!=None:
            temp=temp next
        while temp!=None:
            print(temp data)
            temp=temp prev
l=dll()

y=int(input("Enter Number of Elements to Insert in DoublyLinkedList "))

for i in range(y):

    x=int(input("Enter Element "))

    l inser(x)

l forward()
```

```
l backward()
```

OUTPUT:

```
Enter Number of Elements to Insert in DoublyLinkedList 3
Enter Element 4
Enter Element 1
Enter Element 6
Traversal in forward direction
6
1
4
Traversal in reverse direction
4
1
6
```

# (e) Searching

```
class Node:
    def __init__(self,data):
        self data=data
        self prev=None
        self next=None
class dll:
    def __init__(self):
        self head=None
    def inser(self,value):
        newN=Node(value)
        if self head==None:
            self head=newN
        else:
            temp=self head
            while temp next!=None:
                temp=temp next
            temp next=newN
            newN prev=temp
    def search(self,v):
        temp=self head
        c=0
        s=0
        while temp!=None:
            s=s+1
            if temp data==v:
                c=c+1
                break
            temp=temp next
        if c==1:
            print("Node is present in the list at the position :",s)
        else:
            print("Node is not present in the list")
    def display(self):
        temp=self head
        while temp!=None:
            print(temp data)
            temp=temp next
```

```
l=dll()
y=1
while y!=4:
        y=int(input("Select a operation:\n1 Insertion\n2 Searching\n3 Display\n4 Quit\t"))
        if y==1:
                x=int(input("Enter Element "))
                l inser(x)
        elif y==2:
                z=int(input("Enter Element to Search "))
                l search(z)
        elif y==3:
                l display()
        elif y==4:
                break
```

OUTPUT:

```
Select a operation:
1 Insertion
2 Searching
3 Display
4 Quit→1
Enter Element 2
Select a operation:
1 Insertion
2 Searching
3 Display
4 Quit→1
Enter Element 4
Select a operation:
1 Insertion
2 Searching
3 Display
4 Quit→3
2
4
Select a operation:
1 Insertion
2 Searching
3 Display
4 Quit→2
```

```
Enter Element to Search 2
Node is present in the list at the position : 1
Select a operation:
1 Insertion
2 Searching
3 Display
4 Quit→4
```

```
# (d)Reversal
class Node:
 def __init__(self, data):
  self.data = data
  self.next = None
  self.prev = None
class LinkedList:
 def __init__(self):
  self.head = None
 def push_back(self, newElement):
  newNode = Node(newElement)
  if(self.head == None):
   self.head = newNode
   return
  else:
   temp = self.head
   while(temp.next != None):
    temp = temp.next
   temp.next = newNode
   newNode prev = temp
 def reverseList(self):
  if(self.head != None):
   prevNode = self.head
   tempNode = self.head
   curNode = self.head next
   prevNode.next = None
   prevNode.prev = None
   while(curNode != None):
    tempNode = curNode.next
    curNode next = prevNode
    prevNode.prev = curNode
    prevNode = curNode
    curNode = tempNode
   self.head = prevNode
 def PrintList(self):
  temp = self.head
  if(temp != None):
   print("The list contains:", end=" ")
   while (temp != None):
    print(temp.data, end=" ")
    temp = temp.next
   print()
  else:
   print("The list is empty ")
MyList = LinkedList()
MyList.push_back(10)
MyList.push_back(20)
MyList.push_back(30)
MyList.push_back(40)
MyList.push_back(50)



MyList.PrintList()
MyList.reverseList()
MyList.PrintList()
```

OUTPUT:

```
The list contains: 10 20 30 40 50
The list contains: 50 40 30 20 10
```

Q20. Program to implement Queue Using linked list

```
class Node:

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

if __name__== '__main__':
  q = Queue()
  q.EnQueue(10)
  q.EnQueue(20)
  q.DeQueue()
  q.DeQueue()
  q.EnQueue(30)
  q.EnQueue(40)
  q.EnQueue(50)
  q.DeQueue()
  print("Queue Front " + str(q.front data))
  print("Queue Rear " + str(q.rear data))
```

OUTPUT:

```
Queue Front 40
Queue Rear 50
```

Q21. Program to implement Circular Queue Using linked list

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

class Queue:
  def __init__(self):
    front = None
    rear = None

def enQueue(q, value):
  temp = Node()
  temp.data = value
  if (q.front == None):
    q.front = temp
  else:
    q.rear.link = temp

  q.rear = temp
  q.rear.link = q.front

def deQueue(q):
  if (q.front == None):
    print("Queue is empty")
    return (-999999999999)

  value = None
  if (q.front == q.rear):
    value = q.front.data
    q.front = None
    q.rear = None
  else:
    temp = q.front
    value = temp.data
    q.front = q.front.link
    q.rear.link = q.front

  return value

def displayQueue(q):
  temp = q.front
  print("Elements in Circular Queue are: ",end = " ")
  while (temp.link != q.front):
    print(temp.data, end = " ")
    temp = temp.link
  print(temp.data)

if __name__ == '__main__':

  q = Queue()
  q.front = q.rear = None

  enQueue(q, 14)
  enQueue(q, 22)
  enQueue(q, 6)
  displayQueue(q)

  print("Deleted value = ", deQueue(q))
  print("Deleted value = ", deQueue(q))

  displayQueue(q)

  enQueue(q, 9)
  enQueue(q, 20)
  displayQueue(q)
```

OUTPUT:

```
Elements in Circular Queue are: 14 22 6
Deleted value = 14
Deleted value = 22
Elements in Circular Queue are: 6
Elements in Circular Queue are: 6 9 20
```

Q22. Program to implement Priority Queue Using linked list

```
class PriorityQueueNode:

 def __init__(self, value, pr):

  self.data = value
  self.priority = pr
  self.next = None

class PriorityQueue:A

  def __init__(self):

    self.front = None


  def isEmpty(self):

    return True if self.front == None else False


  def push(self, value, priority):


    if self.isEmpty() == True:

      self.front = PriorityQueueNode(value,
                priority)

      return 1

    else:

      if self.front priority > priority:


        newNode = PriorityQueueNode(value,
                priority)

        newNode.next = self.front
        self.front = newNode

        return 1

      else:

        temp = self.front

        while temp.next:


          if priority <= temp.next.priority:
            break

          temp = temp.next

        newNode = PriorityQueueNode(value,
                priority)
        newNode.next = temp.next
        temp.next = newNode

        return 1


  def pop(self):


    if self.isEmpty() == True:
      return

    else:
      self.front = self.front.next
      return 1

  def peek(self):

    if self.isEmpty() == True:
      return
    else:
      return self.front data

  def traverse(self):
    if self.isEmpty() == True:
      return "Queue is Empty!"
    else:
      temp = self.front
      while temp:
        print(temp.data, end = " ")
        temp = temp.next

if __name__ == "__main__":
  pq = PriorityQueue()
  pq.push(4, 1)
  pq.push(5, 2)
  pq.push(6, 3)
  pq.push(7, 0)
  pq.traverse()
  pq.pop()
```

OUTPUT:

```
7 4 5 6
```

Q23. Write a program to implement stack using the list

```
stacks = []
a = input("Enter element, 'XXX' to end: ")
while (a!="XXX"):
    stacks.append(a)
    a = input("Enter element, 'XXX' to end: ")
print("Initial stack","\n",stacks,sep=")
print("Elements poped from stack:")
for i in range(3):
    print(stacks.pop())
print("Stack after elements are poped:","\n",stacks,sep=")
```

OUTPUT:

```
Enter element, 'XXX' to end: a
Enter element, 'XXX' to end: b
Enter element, 'XXX' to end: c
Enter element, 'XXX' to end: XXX
Initial stack
['a', 'b', 'c']
Elements poped from stack:
c
b
a
Stack after elements are poped:
[]
```

Q24. Write a Python Program to convert infix expression to postfix expression

```
OPERATORS = set(['+', '-', '*', '/', '(', ')', '^'])
PRIORITY = {'+':1, '-':1, '*':2, '/':2, '^':3}
def ip(ex):
    stack =
    OUTPUT = ''
    for ch in ex: []
        if ch not in OPERATORS:
            OUTPUT+=ch
        elif ch=='(':
            stack append('(')
        elif ch==')':
            while stack and stack[-1]!='(':
                OUTPUT+=stack pop()
            stack pop()
        else:
            while stack and stack[-1]!='(' PRIORITY[ch]<=PRIORITY[stack[-]]:
                OUTPUT+=stack pop()
            stack append(ch)
    while stack:
        OUTPUT+=stack pop()
    return OUTPUT
ex = input("Enter infix expression")
print(f"infix expression: {ex}")
obj = ip(ex)
print(f"postfix expression: {obj}")
```

OUTPUT:

```
Enter infix expression a*b+(c/d)
infix expression: a*b+(c/d)
postfix expression: ab*cd/+
```

Q25. Write a Python Program to evaluate postfix expression

```
operators = ['+', '-', '%', '*', '/', '**']
def pi(ex):
    stack = []
    for i in ex:
        if i not in operators:
            stack append(i)
        else:
            a=stack pop()
            b=stack pop()
            if (i == '+'):
                res = int(b)+ int(a)
            elif (i=='-'):
                res = int(b)-int(a)
            elif (i=='*'):
                res = int(b)*int(a)
            elif (i=='&'):
                res = int(b)% int(a)
            elif (i == '/'):
                res = int(b)/int(a)
            elif (i =='**'):
                res = int(b)**int(a)
            stack.append(res)
    return('' join(map(str, stack)))


ex = input("Enter Postfix expression")

print("Result of Postfix expression", ex, "is",pi(ex))
```

OUTPUT:

```
Enter Postfix expression231*+9-

Result of Postfix expression 231*+9- is -4
```

Q26. Program to compute factorial using tail recursion

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

OUTPUT:

```
Enter a number: 5
The factorial of 5 is 120
```

Q27. Program to implement Tower of Hanoi

```
def towerofhanoi(n,src,des,aux):
        if n==1:
                print("Move disk 1 from source",src,"to destination",des)
                return
        towerofhanoi(n-1,src,aux,des)
        print("Move disk",n,"from source",src,"to destination",des)
        towerofhanoi(n-1,aux,des,src)
n=int(input(" enter no of disk"))
towerofhanoi(n,"A","B","C")
```

OUTPUT:

```
enter no of disk3
Move disk 1 from source A to destination B
Move disk 2 from source A to destination C
Move disk 1 from source B to destination C
Move disk 3 from source A to destination B
Move disk 1 from source C to destination A
Move disk 2 from source C to destination B
Move disk 1 from source A to destination B
```

Q28. Program to implement addition of two polynomials using linklist

```
def add(A, B, m, n):
    size = max(m, n);
    sum = [0 for i in range(size)]
    for i in range(0, m, 1):
        sum[i] = A[i]
    for i in range(n):
        sum[i] += B[i]
    return sum
def printPoly(poly, n):
    for i in range(n):
        print(poly[i], end = "")
        if (i != 0):
            print("x^", i, end = "")
        if (i != n - 1):
            print(" + ", end = "")
if __name__ == '__main__':
    A = [5, 0, 10, 6]
    B = [1, 2, 4]
    m = len(A)
    n = len(B)
    print("First polynomial is")
    printPoly(A, m)
    print("\n", end = "")
    print("Second polynomial is")
    printPoly(B, n)
    print("\n", end = "")
    sum = add(A, B, m, n)
    size = max(m, n)

    print("sum polynomial is")
    printPoly(sum, size)
```

OUTPUT:

```
First polynomial is
5 + 0x^ 1 + 10x^ 2 + 6x^ 3
Second polynomial is
1 + 2x^ 1 + 4x^ 2
sum polynomial is
6 + 2x^ 1 + 14x^ 2 + 6x^ 3
```

Q29. Program to implement binary tree using linklist :

# (a). Insertion

```
class Node:
 def __init__(self, data):
  self left = None
  self right = None
  self data = data

 def insert(self, data):

  if self data:
   if data < self data:
    if self left is None:
     self left = Node(data)
    else:
     self left insert(data)
   elif data > self data:
    if self right is None:
     self right = Node(data)
    else:
     self right insert(data)
  else:
   self data = data



 def PrintTree(self):
  if self left:
   self left PrintTree()
```

```
    print( self data),
    if self right:
      self right PrintTree()



root = Node(12)
root insert(6)
root insert(14)
root insert(3)
root PrintTree()
```

OUTPUT:

```
3
6
12
14
```

# (d). Searching

```
class Node:

  def __init__(self, data):

    self left = None
    self right = None
    self data = data
  def insert(self, data):

    if self data:
      if data < self data:
        if self left is None:
          self left = Node(data)
        else:
          self left insert(data)
      elif data > self data:
        if self right is None:
          self right = Node(data)
        else:
          self right insert(data)
    else:
      self data = data

  def findval(self, lkpval):
    if lkpval < self data:
      if self left is None:
        return str(lkpval)+" is not Found"
```

```
   return self left findval(lkpval)
  elif lkpval > self data:
   if self right is None:
    return str(lkpval)+" is not Found"
   return self right findval(lkpval)
  else:
   return str(self data) + " is found"
 def PrintTree(self):
  if self left:
   self left PrintTree()
  print(self data),
  if self right:
   self right PrintTree()

root = Node(27)
root insert(14)
root insert(35)
root insert(31)
root insert(10)
root insert(19)
print(root findval(7))
print(root findval(14))
```

OUTPUT:

```
7 is not found
17 is found
```

#(b). Deletion

```
class Node:
    def __init__(self,data):
        self data = data
        self left = None
        self right = None
def inorder(temp):
    if(not temp):
        return
    inorder(temp left)
    print(temp data, end = " ")
    inorder(temp right)
def deleteDeepest(root,d_node):
    q = []
    q append(root)
    while(len(q)):
        temp = q pop(0)
        if temp is d_node:
            temp = None
            return
        if temp right:
            if temp right is d_node:
                temp right = None
                return
            else:
                q append(temp right)
        if temp left:
            if temp left is d_node:
                temp left = None
                return
            else:
                q append(temp left)

def deletion(root, key):
    if root == None :
        return None
    if root left == None and root right == None:
        if root key == key :
            return None
        else :
            return root
    key_node = None
    q = []
    q append(root)
    temp = None
    while(len(q)):
        temp = q pop(0)
        if temp data == key:
            key_node = temp
        if temp left:
            q append(temp left)
        if temp right:
            q append(temp right)
    if key_node :
        x = temp data
        deleteDeepest(root,temp)
        key_node data = x
    return root

if __name__=='__main__':
    root = Node(10)
    root left = Node(11)
    root left left = Node(7)
    root left right = Node(12)
    root right = Node(9)
    root right left = Node(15)
    root right right = Node(8)
    print("The tree before the deletion:")
    inorder(root)
    key = 11
    root = deletion(root, key)
    print()
    print("The tree after the deletion;")
    inorder(root)
```

OUTPUT:

```
The tree before deletion:
7 11 12 10 15 9 8
The tree after the deletion:
7 8 12 10 15 9
```

Q30. Program to implement binary search tree using Linked Lists:

a Insertion b Deletion c Traversal d Searching

```
class Node:
    def __init__(self, key):
        self key = key
        self left = None
        self right = None
def inorder(root):
    if root is not None:
        inorder(root left)
        print(root key, end=" ")
        inorder(root right)
def insert(node, key):
    if node is None:
        return Node(key)
    if key < node key:
        node left = insert(node left, key)
    else:
        node right = insert(node right, key)
    return node
def deleteNode(root, key):
    if root is None:
        return root
    if key < root key:
        root left = deleteNode(root left, key)
        return root
    elif(key > root key):
        root right = deleteNode(root right, key)
        return root

    if root left is None and root right is None:
        return None
    if root left is None:
        temp = root right
        root = None
        return temp
    elif root right is None:
        temp = root left
        root = None
        return temp
    succ = root right
    while succ left != None:
        succParent = succ
        succ = succ left
    if succParent != root:
        succParent left = succ right
    else:
        succParent right = succ right
    root key = succ key
    return root
def search(root,key):
    if root is None or root val == key:
        return root
    if root val < key:
        return search(root right,key)
        return search(root left,key)

# driver code
root = None
root = insert(root, 50)
root = insert(root, 30)
root = insert(root, 20)
root = insert(root, 40)
root = insert(root, 70)
root = insert(root, 60)
root = insert(root, 80)

print(f"Traversal of the tree \n {inorder(root)}\n")
print("Inorder traversal of the given tree")
inorder(root)

print("\nDelete 20")
root = deleteNode(root, 20)
print("Inorder traversal of the modified tree")
inorder(root)
print("\nDelete 30")
root = deleteNode(root, 30)
print("Inorder traversal of the modified tree")
inorder(root)
print("\nDelete 50")
root = deleteNode(root, 50)
print("Inorder traversal of the modified tree")
inorder(root)
```

OUTPUT:

```
20 30 40 50 60 70 80 Traversal of the tree
 None
Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
Delete 50
Inorder traversal of the modified tree
40 60 70 80
```

Q31. Program to implement Heap sort in non-recursive way

```
def buildMaxHeap(arr, n):
    for i in range(n):
        if arr[i] > arr[int((i - 1) / 2)]:
            j = i
            while arr[j] > arr[int((j - 1) / 2)]:
                (arr[j],
                arr[int((j - 1) / 2)]) = (arr[int((j - 1) / 2)],arr[j])
                j = int((j - 1) / 2)


def heapSort(arr, n):
    buildMaxHeap(arr, n)
    for i in range(n - 1, 0, -1):
        arr[0], arr[i] = arr[i], arr[0]
        j, index = 0, 0
        while True:
            index = 2 * j + 1
            if (index < (i - 1) and
                arr[index] < arr[index + 1]):
                index += 1
            if index < i and arr[j] < arr[index]:
                arr[j], arr[index] = arr[index], arr[j]
            j = index
            if index >= i:
                break
if __name__ == '__main__':
    arr = [10, 20, 15, 17, 9, 21]
```

```
n = len(arr)
print("Given array: ")
for i in range(n):
    print(arr[i], end = " ")
print()
heapSort(arr, n)
print("Sorted array: ")
for i in range(n):
    print(arr[i], end = " ")
```

OUTPUT:

```
Given array:
10 20 15 17 9 21
Sorted array:
9 10 15 17 20 21
```

Q32- Program to implement Radix sort

```
def countingSort(arr, exp1):
  n = len(arr)
  OUTPUT = [0] * (n)
  count = [0] * (10)
  for i in range(0, n):
    index = arr[i] // exp1
    count[index % 10] += 1
  for i in range(1, 10):
    count[i] += count[i - 1]
  i = n - 1
  while i >= 0:
    index = arr[i] // exp1
    OUTPUT[count[index % 10] - 1] = arr[i]
    count[index % 10] -= 1
    i -= 1
  i = 0
  for i in range(0, len(arr)):
    arr[i] = OUTPUT[i]
def radixSort(arr):
  max1 = max(arr)
  exp = 1
  while max1 / exp > 0:
    countingSort(arr, exp)
    exp *= 10
arr = [170, 45, 75, 90, 802, 24, 2, 66]
radixSort(arr)
```

```
for i in range(len(arr)):
  print(arr[i])
```

OUTPUT:

```
2
24
45
66
75
90
170
802
```

Q33. Program to implement BFS algorithm

```
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, end = " ")
  for neighbour in graph[m]:
   if neighbour not in visited:
    visited.append(neighbour)
    queue.append(neighbour)
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

OUTPUT:

```
Following is the Breadth-first Search
5 37 2 4 8
```

Q34. Program to implement DFS algorithm

```
#Depth First Search
graph = {
  'A' : ['B','C'],
  'B' : ['D','E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['G'],
  'F' : [],
  'G' : [],}
visited =[]
queue = []
def dfs(visited,graph,node):
 visited.append(node)
 queue.insert(0,node)
 while(queue):
  s = queue.pop(0)
  print(s,end=" ")
  for next in graph[s]:
   if next not in visited:
    queue.insert(0,next)
    visited.append(next)
dfs(visited,graph,'A')
```

OUTPUT:

```
A C F B E G D
```