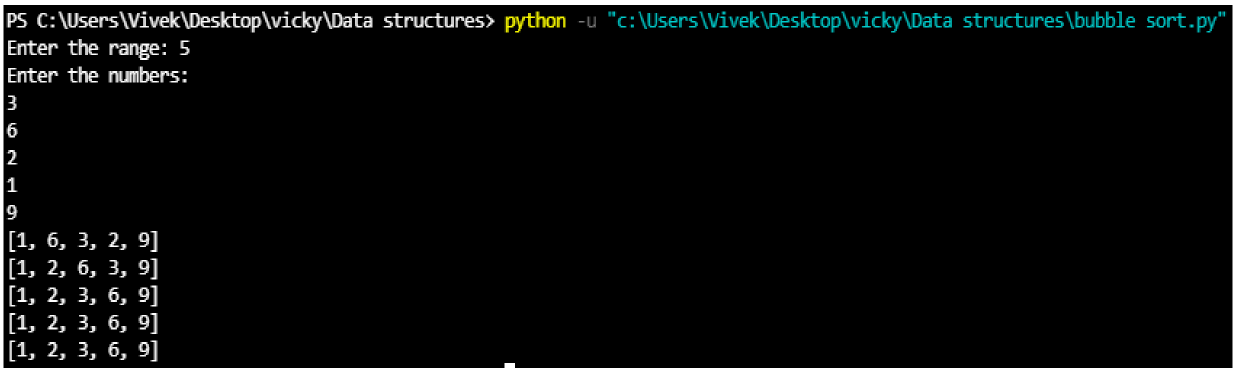
1) **Aim**: WAP to implement Bubble Sort.

Program:

```
l = []
n = int (input("Enter the range: "))
print("Enter the numbers: ")
for i in range(n):
    x = int(input())
    l.append(x)
def bubble_sort(l):
    for i in range(n):
        for j in range(i+1,n):
            if l[i]>l[j]:
                c = l[i]
                l[i]=l[j]
                l[j]=c
        print(l)
bubble_sort(l)
```

Output:



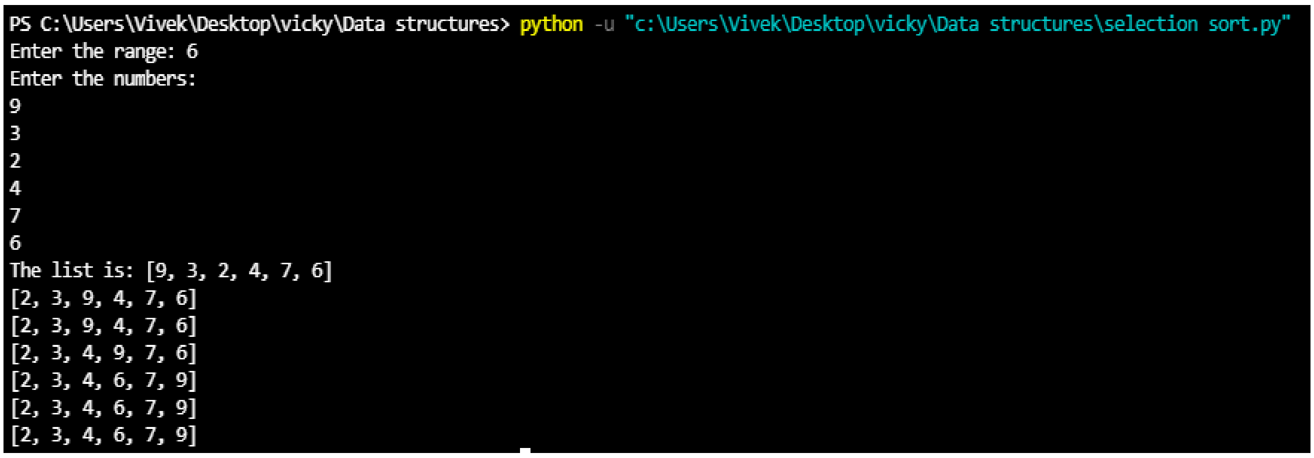
```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\bubble sort.py"
Enter the range: 5
Enter the numbers:
3
6
2
1
9
[1, 6, 3, 2, 9]
[1, 2, 6, 3, 9]
[1, 2, 3, 6, 9]
[1, 2, 3, 6, 9]
[1, 2, 3, 6, 9]
```

2) **Aim**: WAP to implement Selection Sort

Program:

```
l = []
n = int (input("Enter the range: "))
print("Enter the numbers: ")
for i in range(n):
   x = int(input())
   l.append(x)
print("The list is:",l)
def selection_sort(l):
   for i in range(n):
      min = i
      for j in range(i+1,n):
         if(l[j]<l[min]):
            min = j
            l[i],l[min]=l[min],l[i]
      print(l)
selection_sort(l)
```

Output:



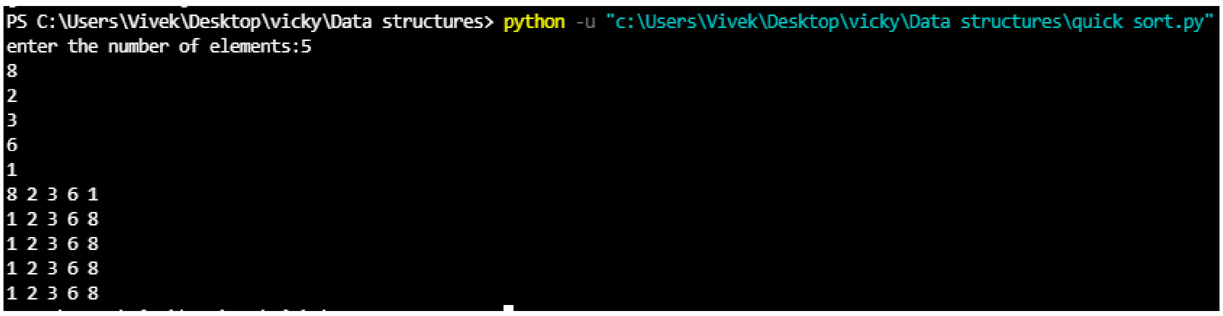
```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\selection sort.py"
Enter the range: 6
Enter the numbers:
9
3
2
4
7
6
The list is: [9, 3, 2, 4, 7, 6]
[2, 3, 9, 4, 7, 6]
[2, 3, 9, 4, 7, 6]
[2, 3, 4, 9, 7, 6]
[2, 3, 4, 6, 7, 9]
[2, 3, 4, 6, 7, 9]
[2, 3, 4, 6, 7, 9]
```

3) **Aim**: Write an application to implement Quick sort

Program:

```
def quick_sort(list,first,last):
   if(first<last):
        pivot=first
        i=first
        j=last
        while(i<j):
            while(list[i]<=list[pivot] and i<j):
                i=i+1
            while(list[j]>list[pivot]):
                j=j-1
            if(i<j):
                list[i],list[j]=list[j],list[i]
        list[pivot],list[j]=list[j],list[pivot]
        print(*list)
        quick_sort(list,first,j-1)
        quick_sort(list,j+1,last)
list=[]
x=int(input("enter the number of elements:"))
for i in range(x):
    y=int(input())
    list.append(y)
n=len(list)
print(*list)
quick_sort(list,0,n-1)
```

Output:



```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\quick sort.py"
enter the number of elements:5
8
2
3
6
1
8 2 3 6 1
1 2 3 6 8
1 2 3 6 8
1 2 3 6 8
1 2 3 6 8
```

1) Aim: Write a program to implement binary search method using recursive and non-recursive method.

   Program:

   a) With recursion

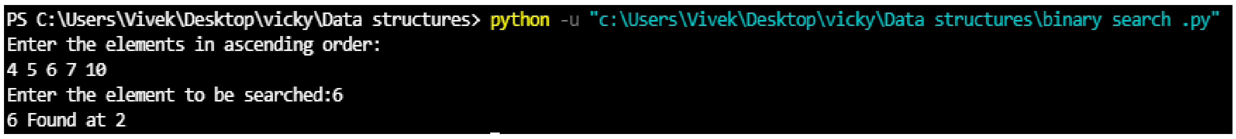
```
def binary_search(arr,l,h,key):
    if(h>=1):
        m = (l+h)//2
        if(arr[m]==key):
            return m
        elif(arr[m]<key):
            return binary_search(arr,m+1,h,key)
        else:
            return binary_search(arr,l,m-1,key)
print("Enter the elements in ascending order:")
l=[int(x) for x in input().split()]
key = int(input("Enter the element to be searched:"))
a=binary_search(l,0,len(l)-1,key)
print(str(key)+" Found at "+str(a))
```

   b) Without recursion

```
def binary_search(arr,l,h,key):
    while(l<=h):
        m = (l+h)//2
        if(arr[m]==key):
            return m
        elif(arr[m]<key):
            l = m+1
        else:
            h = m-1
print("Enter the elements in ascending order:")
l=[int(x) for x in input().split(' ')]
key = int(input("Enter the element to be searched:"))
a=binary_search(l,0,len(l)-1,key)
print(str(key)+" Found at "+str(a))
```

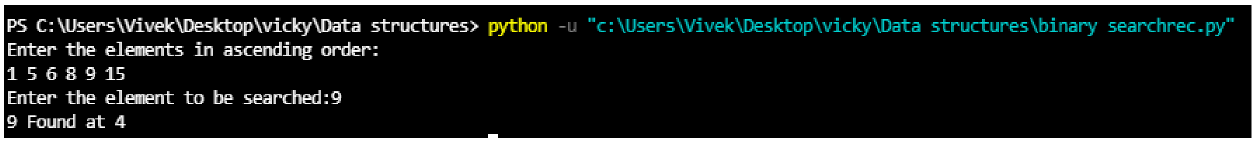
Output:

a) Without recursion:



```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\binary search .py"
Enter the elements in ascending order:
4 5 6 7 10
Enter the element to be searched:6
6 Found at 2
```

b) With recursion:



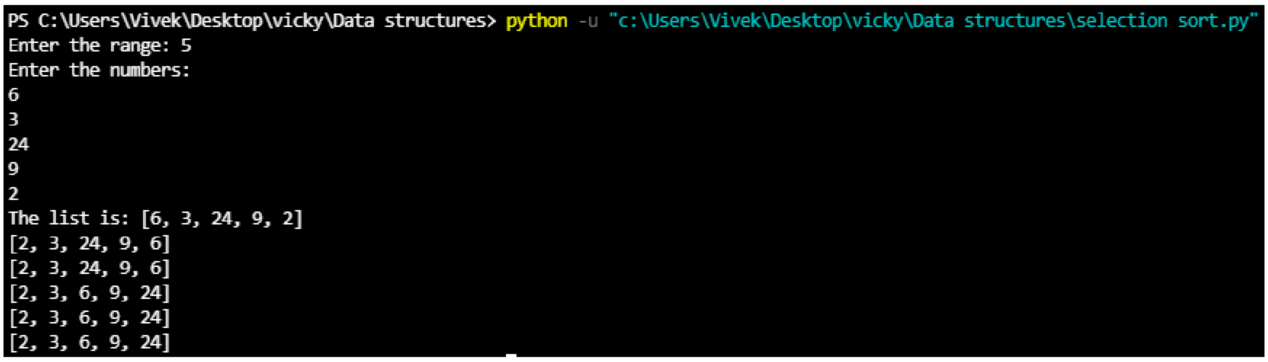
```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\binary searchrec.py"
Enter the elements in ascending order:
1 5 6 8 9 15
Enter the element to be searched:9
9 Found at 4
```

2) **Aim**: Write a program to implement selection sort using recursive method.

Program:

```
def minindex(arr,i,j):
    if i==j:
        return i
    k = minindex(arr,i+1,j)
    return(i if arr[i]<arr[k] else k)
def selection_sort(arr,n,index=0):
    if index == n:
        return -1
    k = minindex(arr,index,n-1)
    if k!=index:
        arr[k],arr[index] = arr[index],arr[k]
    selection_sort(arr,n,index+1)
l = []
n = int (input("Enter the range: "))
print("Enter the numbers: ")
for i in range(n):
    x = int(input())
    l.append(x)
selection_sort(l,len(l))
print(l)
```

Output:



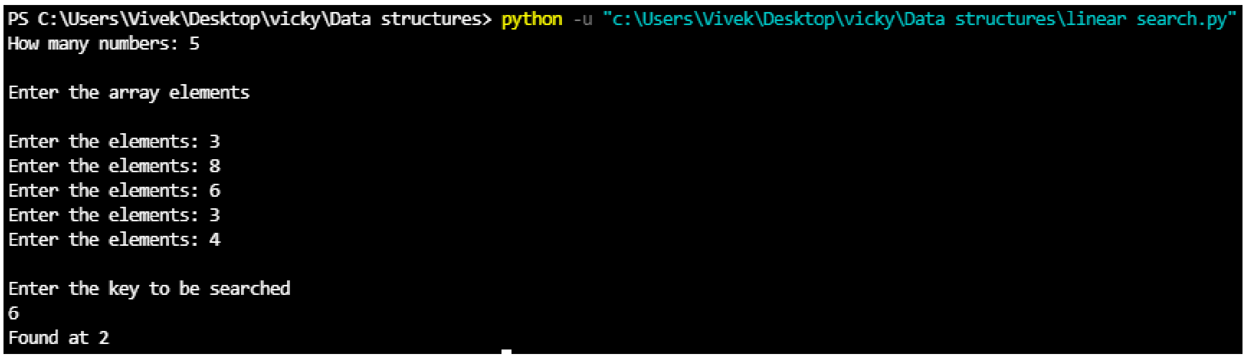
```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\selection sort.py"
Enter the range: 5
Enter the numbers:
6
3
24
9
2
The list is: [6, 3, 24, 9, 2]
[2, 3, 24, 9, 6]
[2, 3, 24, 9, 6]
[2, 3, 6, 9, 24]
[2, 3, 6, 9, 24]
[2, 3, 6, 9, 24]
```

3) **Aim**: Write a program to implement linear search method using recursive method.

Program:

```
a=list()
FOUND=0
n=int(input("How many numbers: "))
if(n>10):
   print("\nToo many Numbers\n")
print("\nEnter the array elements\n")
for i in range(n):
   y=int(input("Enter the elements: "))
   a.append(y)
key=int(input("\nEnter the key to be searched\n"))
for i in range(n):
   if(a[i]==key):
      print("Found at",i)
      FOUND=1
      break
if(FOUND==0):
   print("\nNOT FOUND...")
```

Output:



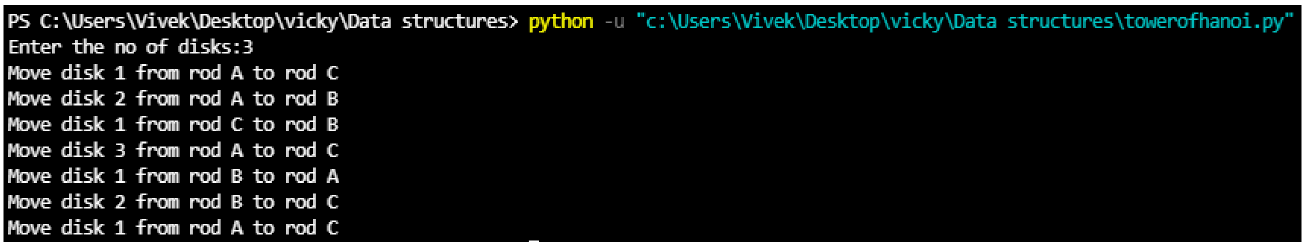
```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\linear search.py"
How many numbers: 5

Enter the array elements

Enter the elements: 3
Enter the elements: 8
Enter the elements: 6
Enter the elements: 3
Enter the elements: 4

Enter the key to be searched
6
Found at 2
```

4) **Aim**: Write a program to implement towers of Hanoi

Program:

```
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
   if n == 1:
      print("Move disk 1 from rod",from_rod,"to rod",to_rod)
      return
   TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
   print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
   TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)
n = int(input("Enter the no of disks:"))
TowerOfHanoi(n, 'A', 'C', 'B')
```

Output:



```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\towerofhanoi.py"
Enter the no of disks:3
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

1. **Write a program to implement Radix sort (prerequisite is Counting sort)**

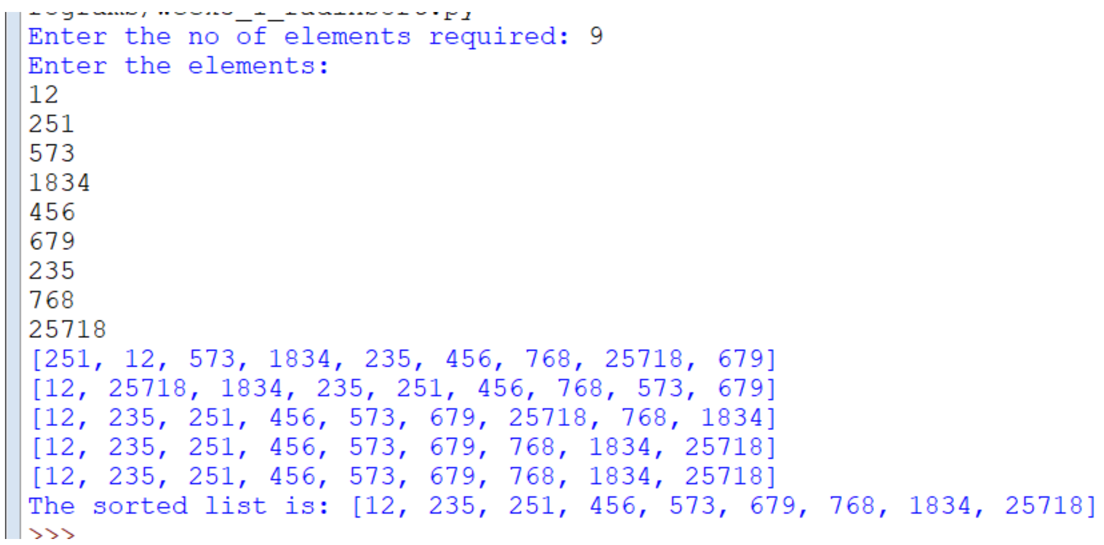
AIM: A program to implement Radix sort.

PROGRAM:

```
def count_sort(l,exp):
   n=len(l)
   output=[0]*(n)
   count=[0]*(10)
   for i in range(n):
      index=l[i]//exp
      count[index%10]+=1
   for i in range(1,10):
      count[i]+=count[i-1]
   i=n-1
   while i>=0:
      index=l[i]//exp
      output[count[index%10]-1]=l[i]
      count[index%10]-=1
      i-=1
   for i in range(n):
      l[i]=output[i]
def radix_sort(l):
   max_l=max(l)
   exp=1
   count=0
   while max_l>0:
      count+=1
      max_l=max_l//10
   for i in range(count):
      count_sort(l,exp)
      exp*=10
      print(l)
l=[]
```

```
n=int(input("Enter the no of elements required: "))

print("Enter the elements: ")

for i in range(n):

   l.append(int(input()))

radix_sort(l)

print("The sorted list is:",l)
```

OUTPUT:



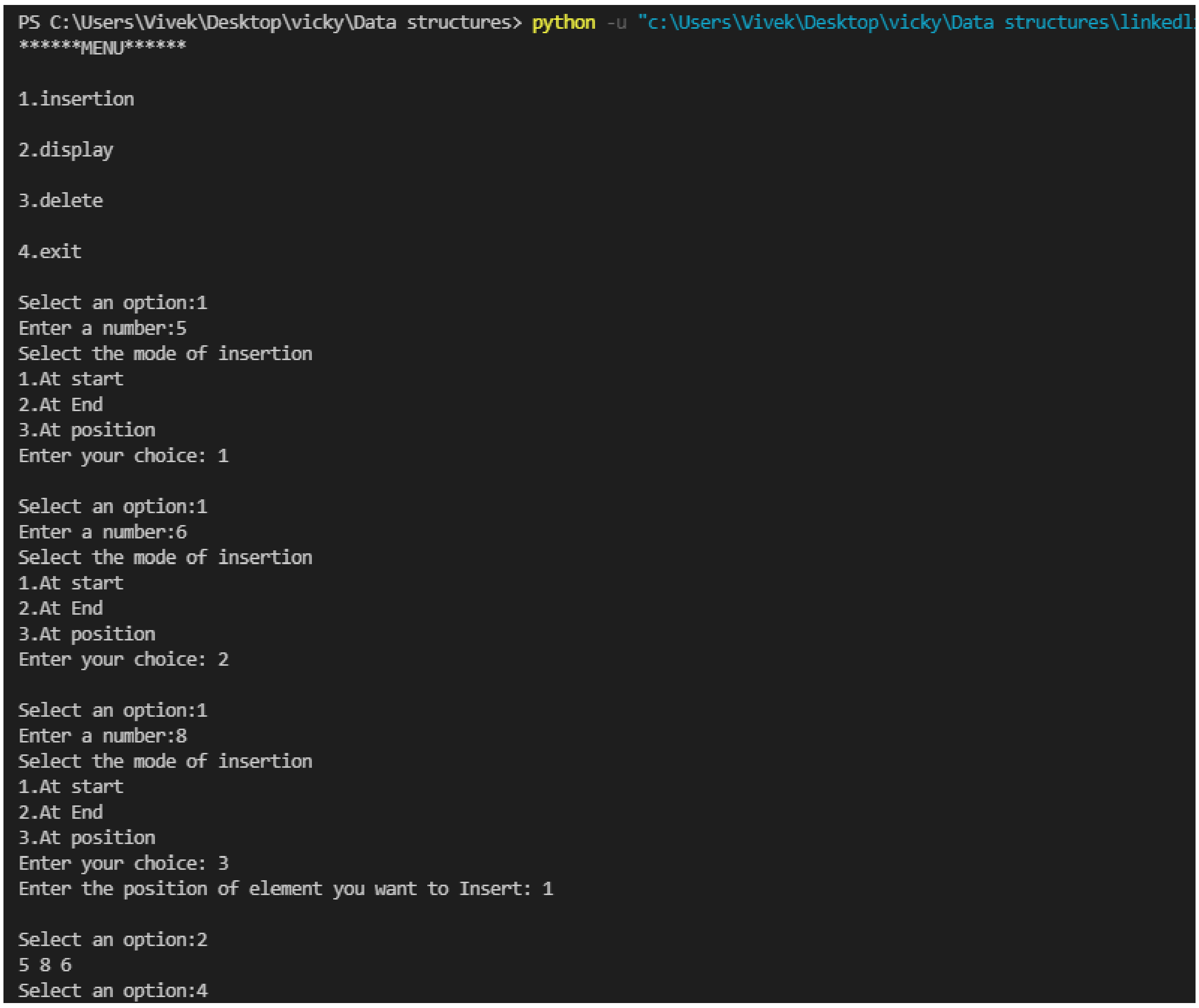
```
Enter the no of elements required: 9
Enter the elements:
12
251
573
1834
456
679
235
768
25718
[251, 12, 573, 1834, 235, 456, 768, 25718, 679]
[12, 25718, 1834, 235, 251, 456, 768, 573, 679]
[12, 235, 251, 456, 573, 679, 25718, 768, 1834]
[12, 235, 251, 456, 573, 679, 768, 1834, 25718]
[12, 235, 251, 456, 573, 679, 768, 1834, 25718]
The sorted list is: [12, 235, 251, 456, 573, 679, 768, 1834, 25718]
>>>
```

1) **Aim**: Develop an application to implement Singly linked list with following operations
   i. Insertion
   ii. Deletion
   iii. Display

Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
    def insert(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node
    def insertAfter(self, pos, new_data):
        if self.head is None:
            return
        temp = self.head
        if pos == 0:
            new_node = Node(new_data)
            new_node.next = self.head
            self.head = new_node
            return
        for i in range(pos - 1):
            temp = temp.next
            if temp is None:
                break
        new_node = Node(new_data)
        new_node.next = temp.next
        temp.next = new_node
    def insertAtEnd(self, new_data):
        new_node = Node(new_data)
        if self.head is None:
            self.head = new_node
            return
        last = self.head
        while (last.next):
            last = last.next
        last.next = new_node
    def display(self):
        temp = self.head
        while (temp):
            print (temp.data,end=" ")
            temp = temp.next
    def deleteNode(self, position):
        if self.head is None:
            return
        temp = self.head
        if position == 0:
            self.head = temp.next
            temp = None
            return
        for i in range(position - 1):
            temp = temp.next
            if temp is None:
                break
        if temp is None:
            return
        if temp.next is None:
            return
        next = temp.next.next

        temp.next = None

        temp.next = next

llist = LinkedList()

print("******MENU******")

print("\n1.insertion")

print("\n2.display")

print("\n3.delete")

print("\n4.exit")

while True:

    n=int(input("\nSelect an option:"))

    if n==1:

        a = int(input("Enter a number:"))

        c = int(input("Select the mode of insertion\n1.At start\n2.At End\n3.At position\nEnter your 
choice: "))

        if c == 1:

            llist.insert(a)

        if c == 2:

            llist.insertAtEnd(a)

        if c == 3:

            pos = int(input("Enter the position of element you want to Insert: "))

            llist.insertAfter(pos, a)

    if n==2:

        llist.display()

    if n==3:

        b = int(input("Enter the position of element you want to delete: "))

        print("\nAfter deleting an element:")

        llist.deleteNode(b)

        llist.display()

    if n==4:

        exit(0)
```

Output:



```
PS C:\Users\Vivek\Desktop\vicky\Data structures> python -u "c:\Users\Vivek\Desktop\vicky\Data structures\linkedl
******MENU******

1.insertion

2.display

3.delete

4.exit

Select an option:1
Enter a number:5
Select the mode of insertion
1.At start
2.At End
3.At position
Enter your choice: 1

Select an option:1
Enter a number:6
Select the mode of insertion
1.At start
2.At End
3.At position
Enter your choice: 2

Select an option:1
Enter a number:8
Select the mode of insertion
1.At start
2.At End
3.At position
Enter your choice: 3
Enter the position of element you want to Insert: 1

Select an option:2
5 8 6
Select an option:4
```

2) **Aim**: Develop an application to implement doubly linked list with following operations

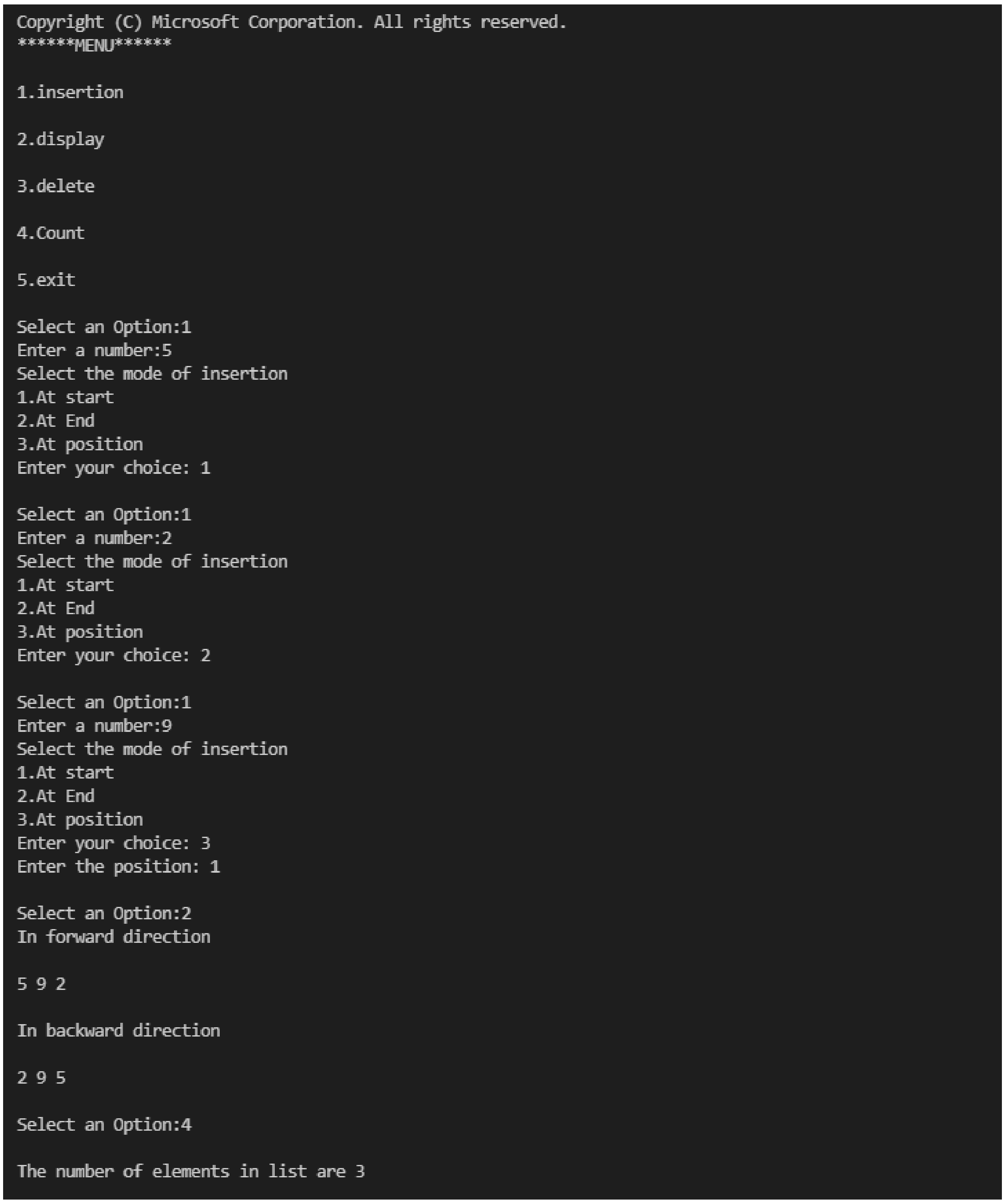
i. insertion
ii. Deletion
iii. Display(forward and backward)

Code:

```
class Node:
   def __init__(self, data):
      self.data = data
      self.next = None
      self.prev = None
class LinkedList:
   def __init__(self):
      self.head = None
   def insert(self, new_data):
      new_node = Node(new_data)
      if self.head == None :
         new_node.prev = None
         new_node.next = None
      else:
         new_node.prev = None
         new_node.next = self.head
         self.head.prev = new_node
      self.head = new_node
   def insertAfter(self, pos, new_data):
      new_node = Node(new_data)
      if self.head == None :
         new_node.prev = None
         new_node.next = None
         self.head = new_node
```

```
        if pos == 0:
            new_node.prev = None
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
            return
        temp = self.head
        for i in range(pos - 1):
            temp = temp.next
            if temp is None:
                break
        new_node.next = temp.next
        (temp.next).prev = new_node
        temp.next = new_node
        new_node.prev = temp
    def insertAtEnd(self, new_data):
        new_node = Node(new_data)
        if self.head is None:
            new_node.prev = None
            new_node.next = None
            self.head = new_node
        ptr = self.head
        while (ptr.next != None):
            ptr = ptr.next
        ptr.next = new_node
        new_node.prev = ptr
        new_node.next = None
    def display(self):
        temp = self.head
        print("In forward direction\n")
        while (temp.next!=None):
            print (temp.data,end=" ")
            temp = temp.next
        print(temp.data)
        print("\nIn backward direction\n")
        while(temp.prev!=None):
            print (temp.data,end=" ")
            temp = temp.prev
        print(temp.data)
    def delnodefirst(self):
        if self.head is None:
            print("List Empty!!\n")
        else:
            temp = self.head
            self.head = self.head.next
            if(self.head!=None):
                self.head.prev = None
    def deleteNode(self, position):
        if self.head is None:
            print("List Empty!!\n")
        temp = self.head
        if position == 0:
            temp = self.head
            self.head = self.head.next
            if(self.head!=None):
                self.head.prev = None
        for i in range(position - 1):
            temp = temp.next
            if temp is None:
                break
        temp.prev.next = temp.next
        temp.next.prev = temp.prev
    def delnodelast(self):
        if self.head is None:
            print("List Empty!!\n")
        else:
            temp = self.head
            while(temp.next != None):
                temp = temp.next
            temp.prev.next = None
    def count(self):
        temp = self.head
        num = 0
        while(temp != None):
            num +=1
            temp = temp.next
        print("\nThe number of elements in list are",num)
llist = LinkedList()
print("******MENU******")
print("\n1.insertion")
print("\n2.display")
print("\n3.delete")
print("\n4.Count")
print("\n5.exit")
while True:
    n=int(input("\nSelect an Option:"))
    if n==1:
        a = int(input("Enter a number:"))
        c = int(input("Select the mode of insertion\n1.At start\n2.At End\n3.At position\nEnter your choice: "))
        if c==1:
            llist.insert(a)
        elif c==2:
            llist.insertAtEnd(a)
        elif c==3:
            pos = int(input("Enter the position: "))
            llist.insertAfter(pos,a)
    if n==2:
        llist.display()
    if n==3:
        d = int(input("Select the mode of Deletion\n1.At start\n2.At End\n3.At position\nEnter your choice: "))
        if(d == 1):
            llist.delnodefirst()
        elif(d==2):
            llist.delnodelast()
        elif(d==3):
            b = int(input("Enter the position of element you want to delete: "))
            llist.deleteNode(b)
        print("\nAfter deleting an element:")
        llist.display()
    if n==4:
        llist.count()
    if n==5:
        exit(0)
```

Output:



```
Copyright (C) Microsoft Corporation. All rights reserved.
******MENU******

1.insertion

2.display

3.delete

4.Count

5.exit

Select an Option:1
Enter a number:5
Select the mode of insertion
1.At start
2.At End
3.At position
Enter your choice: 1

Select an Option:1
Enter a number:2
Select the mode of insertion
1.At start
2.At End
3.At position
Enter your choice: 2

Select an Option:1
Enter a number:9
Select the mode of insertion
1.At start
2.At End
3.At position
Enter your choice: 3
Enter the position: 1

Select an Option:2
In forward direction

5 9 2

In backward direction

2 9 5

Select an Option:4

The number of elements in list are 3
```

160120733167
M.Sai Teja

WEEK – 5

1. Develop an application to implement circular linked list with following operations.
a. Insertion
b. Deletion
c. Display
d. Count
e. Search

AIM: A program to implement circular linked list with given following operations.
PROGRAM:
```
class Node:
def __init__(self,data):
self.data=data
self.next=None
class CLL:
def __init__(self):
self.header=None
self.tail=None
def insertion(self):
item=int(input("Enter the value: "))
newnode=Node(item)
if self.header is None:
self.header=newnode
self.tail=newnode
else:
choice=int(input("1. Start\t2. Middle\t3. End\nEnter where you want to insert: "))
if choice==1:
newnode.next=self.header
self.header=newnode
self.tail.next=newnode
elif choice==2:
ptr=self.header
pos=int(input("Enter the position: "))
for i in range(1,pos-1):
ptr=ptr.next
newnode.next=ptr.next
ptr.next=newnode
elif choice==3:
self.tail.next=newnode
newnode.next=self.header
self.tail=newnode
def display(self):
ptr=self.header
if self.header==None:
print("list is empty")
else:
while ptr!=self.tail:
print(ptr.data,end=' ')
ptr=ptr.next
print(ptr.data)
```

160120733167
M.Sai Teja

```
def deletion(self):
ptr=self.header
if self.header is None:
print("List is empty")
else:
n=int(input("\n1.Start\t2.Middle\t3.End\nEnter the position of the element to delete: "))
if n==1:
self.header=self.header.next
self.tail.next=self.header
if n==2:
pos=int(input("Enter the position: "))
for i in range(pos-1):
ptr=ptr.next
ptr.next=ptr.next.next
if n==3:
while ptr.next!=self.tail:
ptr=ptr.next
self.tail=ptr
self.tail.next=self.header
DATE: 4-11-2021
def search(self):
ptr=self.header
key=int(input("Enter the element to be searched: "))
while ptr.next!=self.tail and ptr.data!=key:
ptr=ptr.next
if ptr.data!=key and ptr.next==self.header:
print("Not found")
elif ptr.data==key:
print("Found")
def count(self):
ptr=self.header
if self.header is None:
count=0
else:
count=1
while ptr!=self.tail:
count+=1
ptr=ptr.next
print("count of elements is: ",count)
l=CLL()
while True:
n=int(input("\n1.Insert\t2.Display\t3.Deletion\t4.Search\t5.Count\t6.Exit\nEnter your choice: "))
if n==1:
l.insertion()
elif n==2:
l.display()
elif n==3:
l.deletion()
elif n==4:
```

160120733167
M.Sai Teja

```
l.search()
elif n==5:
l.count()
elif n==6:
exit()
else:
print("Wrong choice, try again")
```

OUTPUT:

```
1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 2
1 2 4

1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 3

1.Start 2.Middle        3.End
Enter the position of the element to delete: 1

1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 2
2 4

1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 3

1.Start 2.Middle        3.End
Enter the position of the element to delete: 3

1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 2
2

1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 3

1.Start 2.Middle        3.End
Enter the position of the element to delete: 2
Enter the position: 2

1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 2
2

1.Insert        2.Display       3.Deletion      4.Search        5.Count 6.Exit
Enter your choice: 5
count of elements is:  1
```

2. Develop an application to implement doubly circular linked list with following operations.
a. Insertion
b. Deletion
c. Display (forward and backward)
AIM: A program to implement doubly circular linked list with given following operations.

160120733167
M.Sai Teja

PROGRAM:

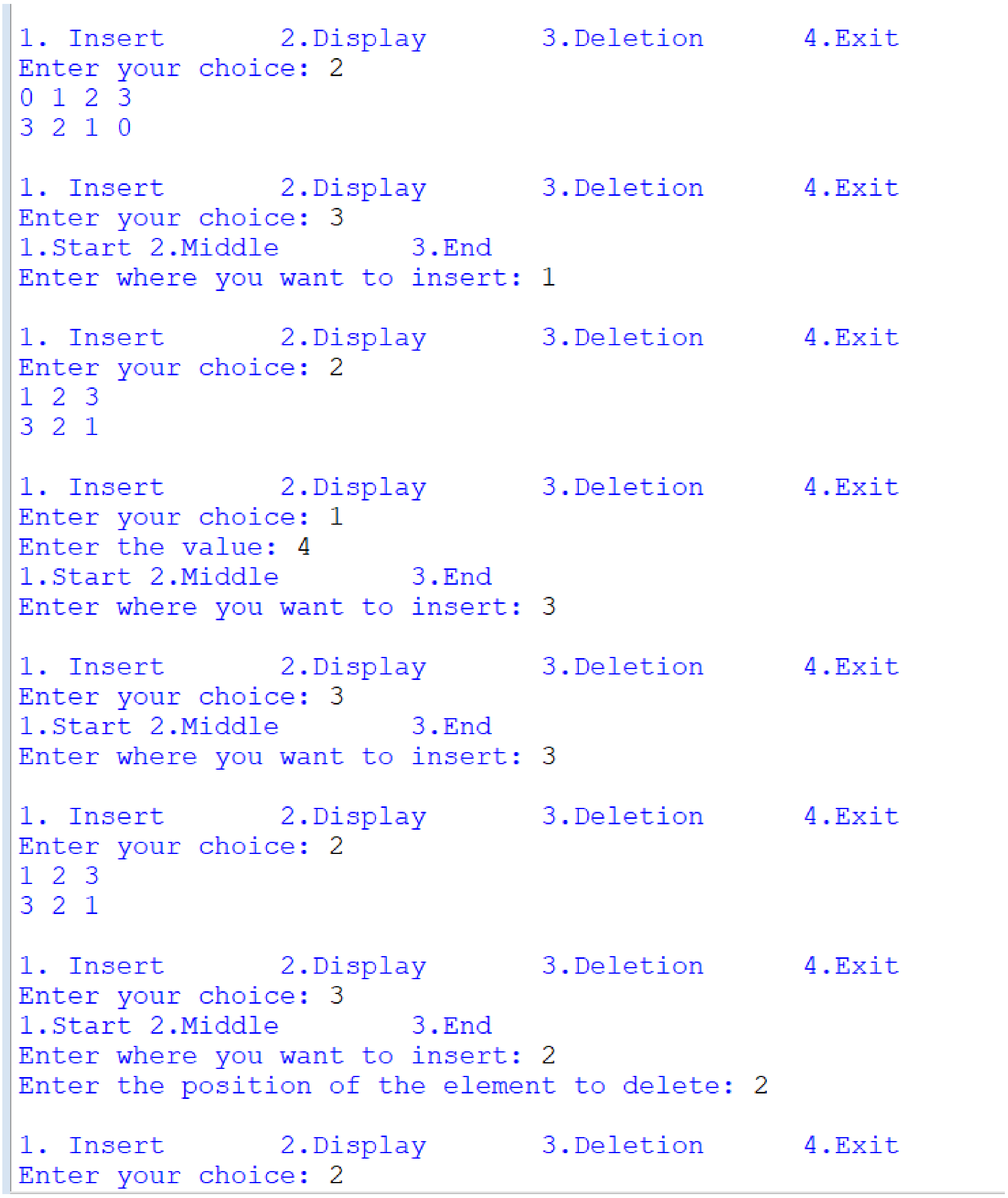
```
class Node:
def __init__(self,data):
self.data=data
self.next=None
self.prev=None
class DCLL:
def __init__(self):
self.head=None
self.tail=None
def insertion(self):
item=int(input("Enter the value: "))
newnode=Node(item)
if self.head is None:
self.head=newnode
self.prev=newnode
self.tail=newnode
else:
choice=int(input("1.Start\t2.Middle\t3.End\nEnter where you want to insert: "))
if choice==1:
newnode.next=self.head
self.head.prev=newnode
self.head=newnode
self.tail.next=newnode
if choice==2:
ptr=self.head
pos=int(input("Enter position: "))
for i in range(1,pos-1):
ptr=ptr.next
newnode.next=ptr.next
ptr.next.prev=newnode
ptr.next=newnode
newnode.prev=ptr
if choice==3:
self.tail.next=newnode
newnode.prev=self.tail
self.tail=newnode
self.tail.next=self.head
def display(self):
ptr=self.head
if self.head is None:
print("list is empty")
else:
while ptr!=self.tail:
print(ptr.data,end=' ')
ptr=ptr.next
print(ptr.data)
while ptr!=self.head:
print(ptr.data,end=' ')
```

160120733167
M.Sai Teja

```
ptr=ptr.prev
print(ptr.data)
def deletion(self):
ptr=self.head
if self.head is None:
print("list is empty")
else:
choice=int(input("1.Start\t2.Middle\t3.End\nEnter where you want to insert: "))
if choice==1:
self.head=self.head.next
self.head.prev=self.tail
if choice==2:
pos=int(input("Enter the position of the element to delete: "))
for i in range(0,pos-1):
ptr=ptr.next
ptr1=ptr.prev
ptr2=ptr.next
ptr1.prev=ptr2
ptr2.next=ptr1
if choice==3:
while ptr.next!=self.tail:
ptr=ptr.next
self.tail=ptr
ptr.next=self.head
self.head.prev=self.tail
l=DCLL()
while True:
n=int(input("\n1. Insert\t2.Display\t3.Deletion\t4.Exit\nEnter your choice: "))
if n==1:
l.insertion()
elif n==2:
l.display()
elif n==3:
l.deletion()
elif n==4:
exit()
else:
print("Wrong choice, try again")
```

OUTPUT:

160120733167
M.Sai Teja



```
1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 2
0 1 2 3
3 2 1 0

1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 3
1.Start 2.Middle        3.End
Enter where you want to insert: 1

1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 2
1 2 3
3 2 1

1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 1
Enter the value: 4
1.Start 2.Middle        3.End
Enter where you want to insert: 3

1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 3
1.Start 2.Middle        3.End
Enter where you want to insert: 3

1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 2
1 2 3
3 2 1

1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 3
1.Start 2.Middle        3.End
Enter where you want to insert: 2
Enter the position of the element to delete: 2

1. Insert        2.Display       3.Deletion      4.Exit
Enter your choice: 2
```

Week -6

AIM: A program to implement skip list and its operations (insertion, Display)

DESCRIPTION:

A skip list is a probabilistic data structure. The skip list is used to store a sorted list of elements or data with a linked list. It allows the process of the elements or data to view efficiently. In one single step, it skips several elements of the entire list, which is why it is known as a skip list. It is an extended version of the linkedlist. It allows the user to search, remove and insert the element very quickly. It consists of different levels where the elements are arranged randomly which makes the search operations easier.

Insertion operation:

Insert(list, searchKey)

```
1.  local update[0...MaxLevel+1]
2.  x := list -> header
3.  for i := list -> level downto 0 do
4.  while x -> forward[i] -> key  forward[i]
5.  update[i] := x
6.  x := x -> forward[0]
7.  lvl := randomLevel()
8.  if lvl > list -> level then
9.  for i := list -> level + 1 to lvl do
10. update[i] := list -> header
11. list -> level := lvl
12. x := makeNode(lvl, searchKey, value)
13. for i := 0 to level do
14. x -> forward[i] := update[i] -> forward[i]
15. update[i] -> forward[i] := x
```

Display operation:

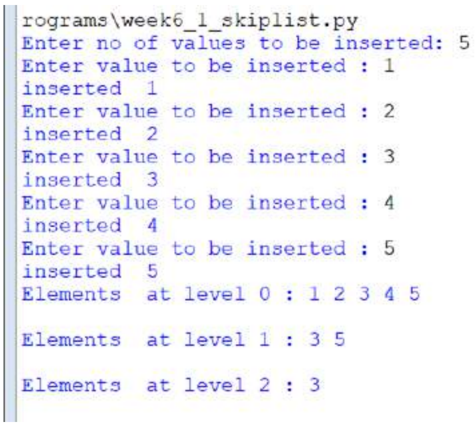
```
1.  def display(self):
```

```
2.  ptr=self.header
3.  for l in range(self.level+1):
4.  print("Elements  at level",l,":",end=" ")
5.  node=ptr.next[l]
6.  while(node!=None):
a.  print(node.data,end=" ")
b.  node=node.next[l]
7.  print("\n")
```

PROGRAM:

```
from random import *
class Node:
    def __init__(self,data,level):
        self.data=data
        self.next=[None]*(level+1)
class Skiplist:
    def __init__(self,maxlevel,P):
        self.maxlevel=maxlevel
        self.P=P
        self.header=Node(-1,maxlevel)
        self.level=0
    def randomlevel(self):
        lvl=0
        while random()<self.P and lvl<self.maxlevel:
            lvl+=1
        return lvl
    '''def randomlevel(self):
        lvl=randint(0,self.maxlevel)
        return lvl'''
    def insertion(self):
        key=int(input("Enter value to be inserted : "))
        update=[None]*(self.maxlevel+1)
        curr=self.header
        for x in range(self.level,-1,-1):
            while curr.next[x] and curr.next[x].data<key:
                curr=curr.next[x]
            update[x]=curr
        curr=curr.next[0]
        if curr==None or curr.data!=key:
            rlevel=self.randomlevel()
            if rlevel>self.level:
                for i in range(self.level+1,rlevel+1):
                    update[i]=self.header
                self.level=rlevel
        n=Node(key,rlevel)
        for x in range(rlevel+1):
            n.next[x]=update[x].next[x]
            update[x].next[x]=n
        print("inserted ",key)
    def display(self):
        ptr=self.header
        for l in range(self.level+1):
            print("Elements  at level",l,":",end=" ")
            node=ptr.next[l]
            while(node!=None):
                print(node.data,end=" ")
                node=node.next[l]
            print("\n")
n=int(input("Enter no of values to be inserted: "))
maxlevel=2**(n-1)
list=Skiplist(maxlevel,0.4)
for x in range(n):
    list.insertion()
list.display()
```

OUTPUT:



```
rograms\week6_1_skiplist.py
Enter no of values to be inserted: 5
Enter value to be inserted : 1
inserted  1
Enter value to be inserted : 2
inserted  2
Enter value to be inserted : 3
inserted  3
Enter value to be inserted : 4
inserted  4
Enter value to be inserted : 5
inserted  5
Elements  at level 0 : 1 2 3 4 5

Elements  at level 1 : 3 5

Elements  at level 2 : 3
```

TIME COMPLEXITY:

Insertion operation:

Average case: O(log n)

Worst case: O(n)

Display operation:

Average and worst case: O(n)

CONCLUSION: The code is error free and runs as expected.

Week -7

Q1

AIM: A program to implement stack using array and its operations

DESCRIPTION:

1. Define a class Stack with initial attributes of array, size and top and assign top value to be -1
2. Now define a push method in Stack class. Check whether top value is equal to (size-1) value, if it is true, then print “Overflow”, else input a value, store it in the array and increment the top value
3. Similarly define a pop method to remove the top value and decrement the top value by 1. If the top value is -1, then print “Underflow”
4. Also define a display method to print the values of the array stored in the stack method.
5. Now outside the class, input the size of the array and define a object of Stack class with the size of array given. Perform the push, pop and display operations with the keys given to them.

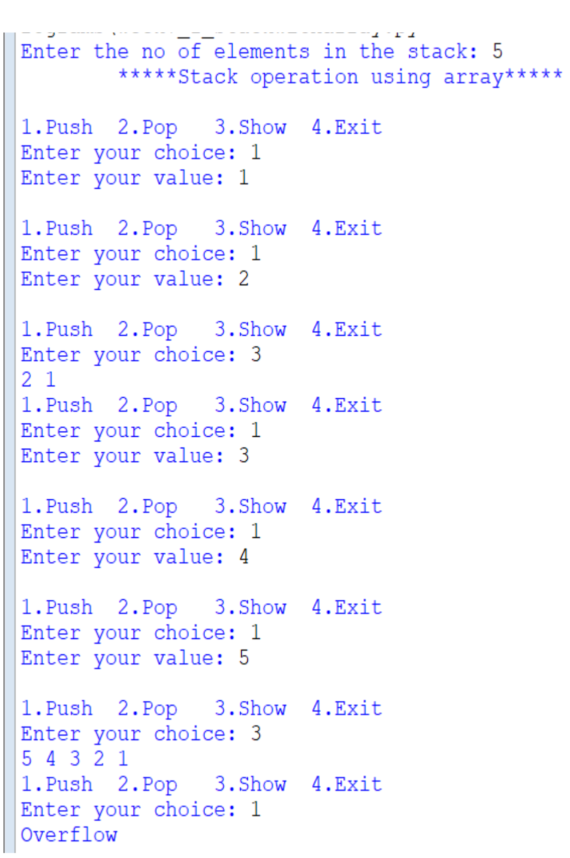
PROGRAM:

```
class Stack:
    def __init__(self,size):
        self.s=[0]*(size)
        self.size=size
        self.top=-1
    def push(self):
        if self.top == self.size-1:
            print("Overflow")
        else:
            self.top+=1
            key=int(input("Enter your value: "))
            self.s[self.top]=key
    def pop(self):
        if self.top == -1:
            print("stack is empty")
        else:
            self.top-=1
    def show(self):
        if self.top == -1:
            print("Stack is empty")
        else:
            temp=self.top
            while temp!=-1:
                print(self.s[temp],end=' ')
                temp-=1

n=int(input("Enter the no of elements in the stack: "))
print("\t*****Stack operation using array*****\t")
s=Stack(n)
while True:
    choice=int(input("\n1.Push\t2.Pop\t3.Show\t4.Exit\nEnter your choice: "))
    if choice == 1:
        s.push()
    elif choice == 2:
        s.pop()
    elif choice == 3:
        s.show()
    elif choice == 4:
        break
    else:
        print("Wrong choice, try again")
```

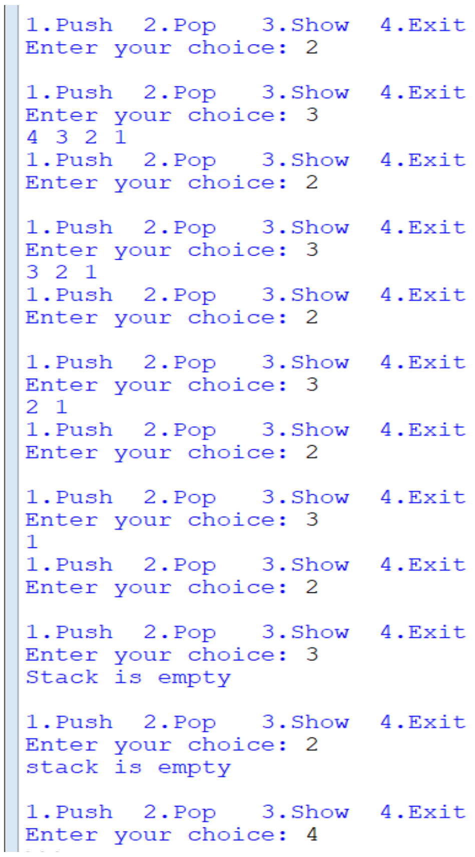
TIME COMPLEXITY:

1. For insertion: O(1)
2. For deletion: O(1)
3. For display: O(n)

OUTPUT:



```
Enter the no of elements in the stack: 5
        *****Stack operation using array*****

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 1

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 3

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 4

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 5

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
5 4 3 2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Overflow
```



```
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
4 3 2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
3 2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
Stack is empty

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2
stack is empty

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 4
```

CONCLUSION: The code is error free and it runs as expected.

Q2

AIM: A program to implement stack using linked list and its operations

DESCRIPTION:

1. Define a Node class with initial attributes data and next. Assign data value to given value and next to Null value
2. Now define a Stack class with initial head attribute and assign it to Null value
3. Define a push method in stack class and input the value to be stored. Create a Node object and store the value in it. Check if the head is null, if it is, then assign head to new node object created. Else assign next attribute of new node to head variable, and make the new node as head.
4. Similarly define a pop method, where it will remove a node from the stack. First check if stack is empty, then print "Stack is empty" else assign head variable to next node of the linked list.
5. Define a show method to print the stack.
6. Now outside the class, create a stack object and perform its operations for push, pop and showing the stack.

PROGRAM:

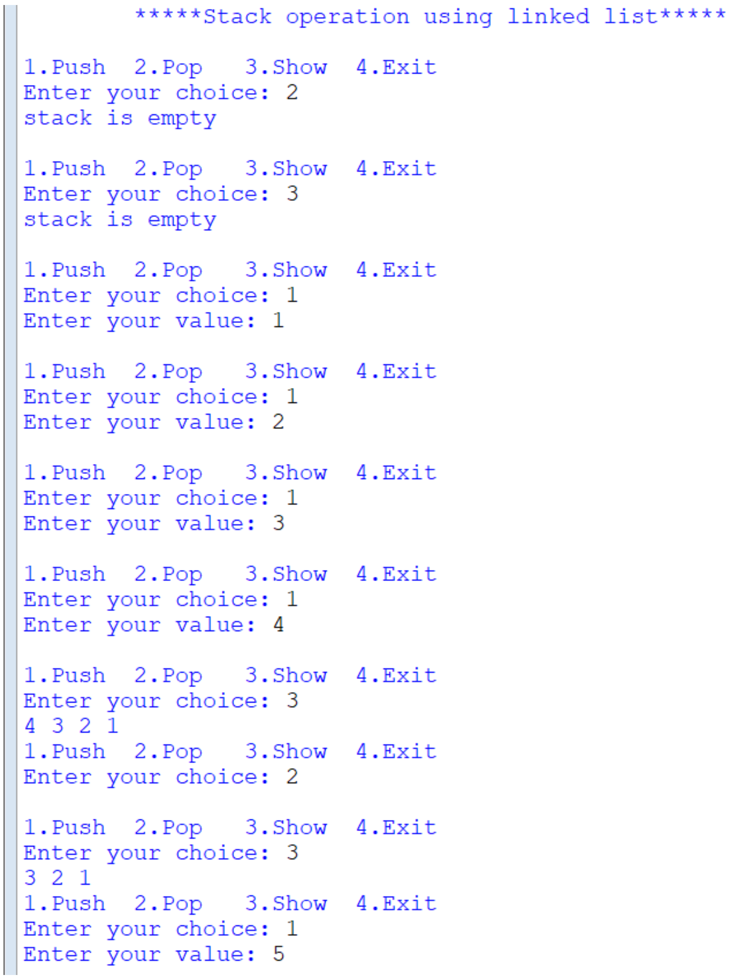
```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class Stack:
    def __init__(self):
        self.head=None
    def push(self):
        value=int(input("Enter your value: "))
        newnode=Node(value)
        if self.head is None:
            self.head=newnode
        else:
            newnode.next=self.head
            self.head=newnode
    def pop(self):
        if self.head is None:
            print("stack is empty")
        else:
            self.head=self.head.next
    def show(self):
        ptr=self.head
        if self.head is None:
            print("stack is empty")
        else:
            while ptr!=None:
                print(ptr.data,end=' ')
                ptr=ptr.next
print("\t*****Stack operation using linked list*****\t")
s=Stack()
while True:
    choice=int(input("\n1.Push\t2.Pop\t3.Show\t4.Exit\nEnter your choice: "))
    if choice == 1:
        s.push()
    elif choice == 2:
        s.pop()
    elif choice == 3:
        s.show()
    elif choice == 4:
        break
```

```
else:
    print("Wrong choice, try again")
```

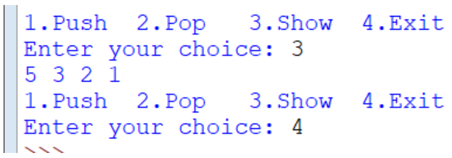
TIME COMPLEXITY:

1. For insertion: O(1)
2. For deletion: O(1)
3. For display: O(n)

OUTPUT:



```
        *****Stack operation using linked list*****

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2
stack is empty

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
stack is empty

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 1

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 3

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 4

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
4 3 2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 2

1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
3 2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 1
Enter your value: 5
```



```
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 3
5 3 2 1
1.Push  2.Pop   3.Show  4.Exit
Enter your choice: 4
```

CONCLUSION: The code is error free and it runs as expected.

Week -8

Q1

AIM: write a program to implement infix to postfix expression.

Description:

Infix expression: The expression of the form a op b. When an operator is in-between every pair of operands. Postfix expression: The expression of the form a b op. When an operator is followed for every pair of operands. To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

Program:

```
class Conversion:
    def __init__(self,capacity):
        self.capacity=capacity
        self.top=-1
        self.precedence={'+':1,'-':1,'*':2,'/':2,'^':3}
        self.array=[]
        self.output=[]
    def peek(self):
        if self.top!=-1:
            return self.array[-1]
    def pop(self):
        if self.top!=-1:
            self.top-=1
            return self.array.pop()
    def push(self,data):
        self.top+=1
        self.array.append(data)
```

```
def notgreater(self, i):
    try:
        a = self.precedence[i]
        b = self.precedence[self.peek()]
        return True if a  <= b else False
    except KeyError:
        return False

def infixtopostfix(self,exp):
    for i in exp:
        if i.isalpha():
            self.output.append(i)
        elif i=='(':
            self.push(i)
        elif i==')':
            #print(*self.array)
            while(self.top!=-1 and self.peek()!='('):

                #print(">",a)
                self.output.append(self.pop())
            if (self.top!=-1 and self.peek()=='('):
                self.pop()



        else:
            while(self.top!=-1 and self.notgreater(i)):


                self.output.append(self.pop())
                self.push(i)
        while(self.top!=-1):
            self.output.append(self.pop())
        print("Postfix expression is :")
        print("".join(self.output))
exp = input("Enter infix expression to be converted: ")
I = Conversion(len(exp))


I.infixtopostfix(exp)
```

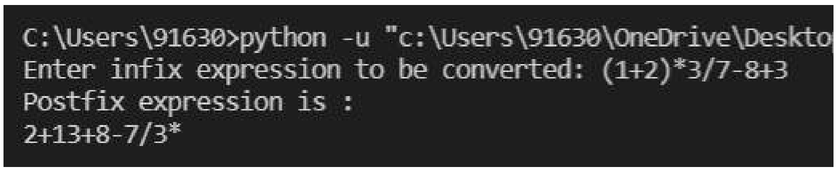
Time complexity:

Worst case time complexity: O(n^2)

Average case time complexity: O(n^2)

Best case time complexity: O(n^2)

Output:



```
C:\Users\91630>python -u "c:\Users\91630\OneDrive\Deskto
Enter infix expression to be converted: (1+2)*3/7-8+3
Postfix expression is :
2+13+8-7/3*
```

Conclusion: The code is error free and runs as expected

Q2

AIM: write a program to evaluate postfix expression.

Description: The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. We have discussed infix to postfix conversion.

Program:

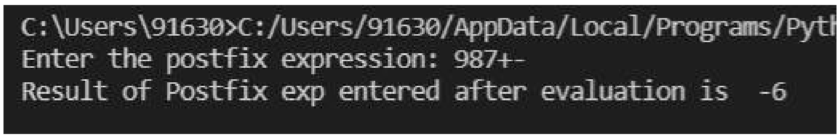
```
class Postfix:
    def __init__(self):
        self.top=-1
        self.array=[]
    def push(self,d):
        self.top+=1
        self.array.append(d)
    def pop(self):
        if self.top!=-1:
            self.top-=1
            return self.array.pop()
    def evaluate(self,exp):
        for x in exp:
            if x.isdigit():
                self.push(x)
            else:
                A=self.pop()
                B=self.pop()
                self.push(str(eval(B+x+A)))
            #print(*self.array)
        return int(self.pop())
expression=input("Enter the postfix expression: ")
```

P=Postfix()

print("Result of Postfix exp entered after evaluation is ",P.evaluate(expression))

Time complexity: Time Complexity: O(n) where n is the size of the given string/expression

Output:



```
C:\Users\91630>C:/Users/91630/AppData/Local/Programs/Pyth
Enter the postfix expression: 987+-
Result of Postfix exp entered after evaluation is  -6
```

Conclusion: The code is error free and runs as expected.

Week -9

Q1

AIM: Write a menu driven program to implement Binary tree with the following operations.

i. Insertion
ii. Preorder
iii. Inorder
iv. Postorder

Description:

1. START
2. Create a node class for having left and right attributes for each object of binary tree class
3. Now create a binary tree class and define the required methods as mentioned in the given problem.

Insertion:

```
i.     START
ii.    Enter the value to be inserted in data variable.
iii.       if self.root is None:
iv.            self.root=Node(data)
v.         else:
vi.            ptr=self.root
vii.           n=int(input("\n1. Left \t 2. Right\nEnter which side u want
       to insert: "))
viii.          while (ptr.left is not None) or (ptr.right is not None):
ix.
x.                 if n==1 and ptr.left is not None:
xi.                    ptr=ptr.left
xii.                   n=int(input("\n1. Left \t 2. Right\nEnter which side u
       want to insert: "))
xiii.              elif n==2 and ptr.right is not None:
xiv.                   ptr=ptr.right
xv.                    n=int(input("\n1. Left \t 2. Right\nEnter which side u
       want to insert: "))
xvi.               elif (n==1 and ptr.left is None) or (n==2 and ptr.right is
       None):
xvii.                  break
```

```
xviii.          else:
xix.                print("Wrong choice, try again")
xx.         newnode=Node(data)
xxi.        if n==1:
xxii.           ptr.left=newnode
xxiii.      elif n==2:
xxiv.           ptr.right=newnode
xxv.    STOP
```

Preorder Traversal

```
i.      def preorder(self,root):
ii.         if (root):
iii.            print(root.data,end=' ')
iv.             self.preorder(root.left)
v.              self.preorder(root.right)
```

Inorder Traversal

```
i.      def inorder(self,root):
ii.         if (root):
iii.            self.inorder(root.left)
iv.             print(root.data,end=' ')
v.              self.inorder(root.right)
```

Postorder Traversal

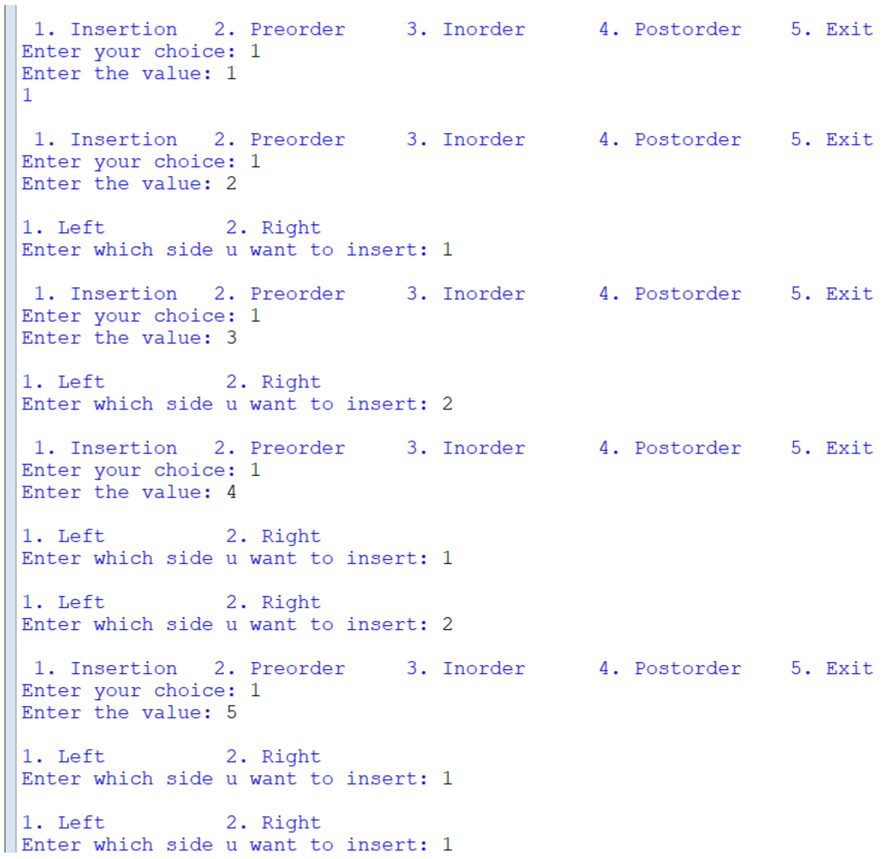
```
i.      def postorder(self,root):
ii.         if (root):
iii.            self.postorder(root.left)
iv.             self.postorder(root.right)
v.              print(root.data,end=' ')
```

4. Now outside the class, create a binary tree object and do the given operations as required using a while loop
5. STOP

Program:

```
class Node:
  def __init__(self,data):
    self.data=data
    self.left=None
    self.right=None
class Binary_tree:
  def __init__(self):
        self.root=None
    def insertion(self):
        data=int(input("Enter the value: "))
        if self.root is None:
            self.root=Node(data)
            print(self.root.data)
        else:
            ptr=self.root
            n=int(input("\n1. Left \t 2. Right\nEnter which side u want to insert: "))
            while (ptr.left is not None) or (ptr.right is not None):


                if n==1 and ptr.left is not None:
                    ptr=ptr.left
                    n=int(input("\n1. Left \t 2. Right\nEnter which side u want to
insert: "))
                elif n==2 and ptr.right is not None:
                    ptr=ptr.right
                    n=int(input("\n1. Left \t 2. Right\nEnter which side u want to
insert: "))
                elif (n==1 and ptr.left is None) or (n==2 and ptr.right is None):
                    break
                else:
                    print("Wrong choice, try again")
            newnode=Node(data)
            if n==1:
                ptr.left=newnode
            elif n==2:
                ptr.right=newnode
    def preorder(self,root):
        if (root):
            print(root.data,end=' ')
            self.preorder(root.left)
            self.preorder(root.right)
    def inorder(self,root):
        if (root):
            self.inorder(root.left)
            print(root.data,end=' ')
            self.inorder(root.right)
    def postorder(self,root):
        if (root):
            self.postorder(root.left)
            self.postorder(root.right)
            print(root.data,end=' ')


bt=Binary_tree()
while True:
    n=int(input("\n 1. Insertion\t2. Preorder\t3. Inorder\t4. Postorder\t5. 
Exit\nEnter your choice: "))
    if n==1:
        bt.insertion()
    elif n==2:
        print("The preorder traversal is: ")
        bt.preorder(bt.root)
    elif n==3:
        print("The Inorder traversal is: ")
        bt.inorder(bt.root)
    elif n==4:
        print("The postorder traversal is: ")
        bt.postorder(bt.root)
    elif n==5:
        exit()
    else:
        print("Wrong choice, try again")
```

Output:



```
 1. Insertion   2. Preorder     3. Inorder      4. Postorder    5. Exit
Enter your choice: 1
Enter the value: 1
1

 1. Insertion   2. Preorder     3. Inorder      4. Postorder    5. Exit
Enter your choice: 1
Enter the value: 2

1. Left         2. Right
Enter which side u want to insert: 1

 1. Insertion   2. Preorder     3. Inorder      4. Postorder    5. Exit
Enter your choice: 1
Enter the value: 3

1. Left         2. Right
Enter which side u want to insert: 2

 1. Insertion   2. Preorder     3. Inorder      4. Postorder    5. Exit
Enter your choice: 1
Enter the value: 4

1. Left         2. Right
Enter which side u want to insert: 1

1. Left         2. Right
Enter which side u want to insert: 2

 1. Insertion   2. Preorder     3. Inorder      4. Postorder    5. Exit
Enter your choice: 1
Enter the value: 5

1. Left         2. Right
Enter which side u want to insert: 1

1. Left         2. Right
Enter which side u want to insert: 1
```

Time complexity:

Insertion :O(log(n)) to base 2

Conclusion: The code is error free and it runs as expected

Week -10

Q1

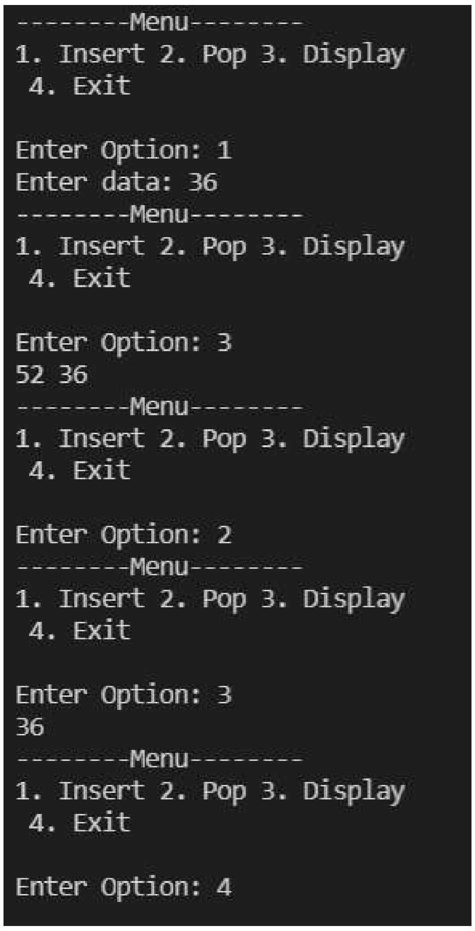
AIM: To WAP to implement Queues using arrays.

Description: A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). Queues using arrays can be implemented using lists in python. We can use the append() and pop() methods to add or remove the elements respectively. We can define an attribute size to impose Overflow and Underflow conditions.

Code:

```
class Queue:
    def __init__(self):
        self.Q = []
    def Insert(self, data):
        self.Q.append(data)
    def Pop(self):
        if not len(self.Q):
            print("List is empty")
        else:
            self.Q.pop(0)
    def Display(self):
        print(*self.Q)
Q = Queue()
while True:
    print("--------Menu--------\n1. Insert 2. Pop 3. Display\n 4. Exit\n")
    choice = int(input("Enter Option: "))
    if choice == 1:
        Q.Insert(int(input("Enter data: ")))
    elif choice == 2:
        Q.Pop()
    elif choice == 3:
        Q.Display()
    else:
        exit()
```

Output:



```
--------Menu--------
1. Insert 2. Pop 3. Display
 4. Exit

Enter Option: 1
Enter data: 36
--------Menu--------
1. Insert 2. Pop 3. Display
 4. Exit

Enter Option: 3
52 36
--------Menu--------
1. Insert 2. Pop 3. Display
 4. Exit

Enter Option: 2
--------Menu--------
1. Insert 2. Pop 3. Display
 4. Exit

Enter Option: 3
36
--------Menu--------
1. Insert 2. Pop 3. Display
 4. Exit

Enter Option: 4
```

Result: The code is error free and runs as expected.

Q2

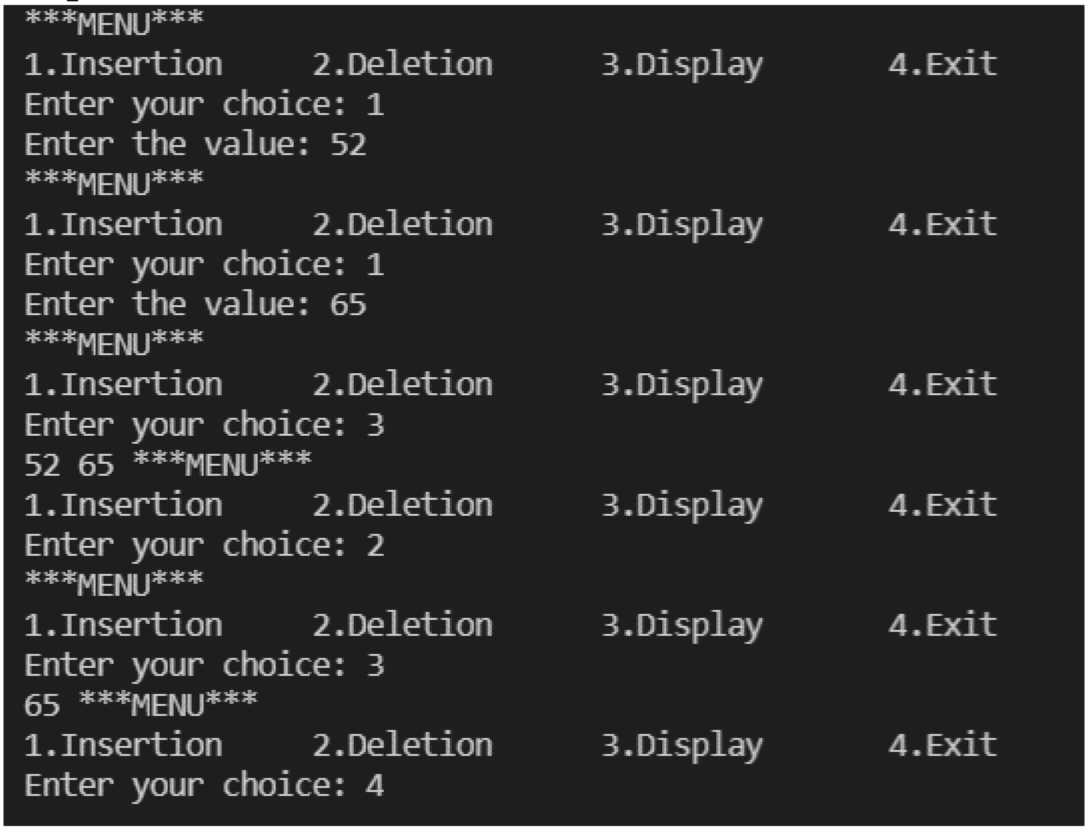
AIM: To WAP to implement Queue using linked list.

Description: A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). Queues can be implemented using Linked lists. Class Node is declared with necessary attributes – ‘next’, ‘data’ which helps to define the structure of every node created. Linked list contains the methods that we want to perform on the Queue.

Code:

```
class Node:
    def __init__(self,data):
        self.data=data;self.next=None
class Queue:
    def __init__(self):
        self.head=None
        self.tail=None
    def Enqueue(self):
        key=int(input("Enter the value: "))
        newnode=Node(key)
        if self.head is None:
            self.head=newnode
            self.tail=self.head
        else:
            self.tail.next=newnode
            self.tail=newnode
    def Dequeue(self):
        if self.head is None:
            print("The queue is empty")
        else:
            self.head=self.head.next
    def Display(self):
        if self.head is None:
            print("The queue is empty")
        else:
            ptr=self.head
            while ptr!=None:
                print(ptr.data,end=" ")
                ptr=ptr.next
q=Queue()
while True:

n=int(input("***MENU***\n1.Insertion\t2.Deletion\t3.Display\t4.Exit\nEnter
your choice: "))
    if n==1:
        q.Enqueue()
    elif n==2:
        q.Dequeue()
    elif n==3:
        q.Display()
    elif n==4:
        exit()
    else:
        print("Wrong choice, try again")
```

Output:



```
***MENU***
1.Insertion     2.Deletion      3.Display       4.Exit
Enter your choice: 1
Enter the value: 52
***MENU***
1.Insertion     2.Deletion      3.Display       4.Exit
Enter your choice: 1
Enter the value: 65
***MENU***
1.Insertion     2.Deletion      3.Display       4.Exit
Enter your choice: 3
52 65 ***MENU***
1.Insertion     2.Deletion      3.Display       4.Exit
Enter your choice: 2
***MENU***
1.Insertion     2.Deletion      3.Display       4.Exit
Enter your choice: 3
65 ***MENU***
1.Insertion     2.Deletion      3.Display       4.Exit
Enter your choice: 4
```

Result: The code is error free and runs as expected.

Q3

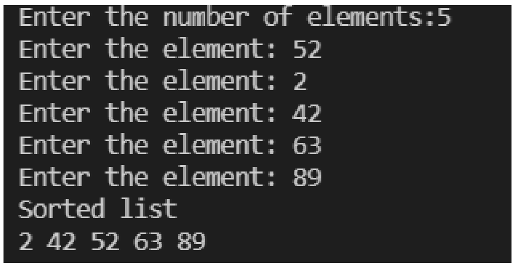
AIM: To WAP to implement Merge Sort.

Description: Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

Code:

```
def Merge(a,low,mid,high):
    c=list()
    i=low
    j=mid+1
    while (i<=mid and j<=high):
        if a[i]<a[j]:
            c.append(a[i])
            i+=1
        else:
            c.append(a[j])
            j+=1
    while i<=mid:
        c.append(a[i])
        i+=1
    while j<=high:
        c.append(a[j])
        j+=1
    for i in range(len(c)):
        a[low+i]=c[i]
def MergeSort(list,low,high):
    mid=(low+high)//2
    if (low<high):
        MergeSort(list,low,mid)
        MergeSort(list,mid+1,high)
        Merge(list,low,mid,high)
    return list
l=[]
n=int(input("Enter the number of elements:"))
for i in range(n):
    y=int(input("Enter the element: "))
    l.append(y)
x=len(l)
MergeSort(l,0,x-1)
print("Sorted list")
print(*l)
```

Output:



```
Enter the number of elements:5
Enter the element: 52
Enter the element: 2
Enter the element: 42
Enter the element: 63
Enter the element: 89
Sorted list
2 42 52 63 89
```

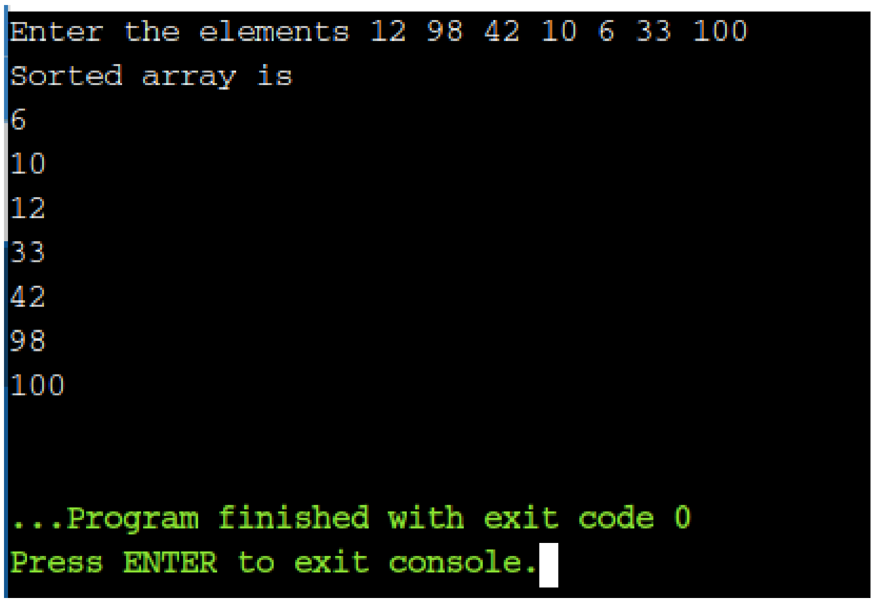
Result: The code is error free and runs as expected.

DS WEEK 11 LAB RECORD

1)
AIM: To develop an application to implement heap sort to sort data in ascending order

CODE:
```
def heapify(arr, n, i):
        largest = i
        l = 2 * i + 1
        r = 2 * i + 2
        if l < n and arr[i] < arr[l]:
                largest = l
        if r < n and arr[largest] < arr[r]:
                largest = r
        if largest != i:
                arr[i],arr[largest] = arr[largest],arr[i]
                heapify(arr, n, largest)
def heapSort(arr):
        n = len(arr)
        for i in range(n // 2 - 1, -1, -1):
                heapify(arr, n, i)
        for i in range(n-1, 0, -1):
                arr[i], arr[0] = arr[0], arr[i]
                heapify(arr, i, 0)
arr = [int(i) for i in  input("Enter the element").split()]
heapSort(arr)
n = len(arr)
print ("Sorted array is")
for i in range(n):
        print ("%d" %arr[i])
```

OUTPUT:



```
Enter the elements 12 98 42 10 6 33 100
Sorted array is
6
10
12
33
42
98
100


...Program finished with exit code 0
Press ENTER to exit console.
```

WEEK 12

1. **Write a program to implement circular queue with following operations**
    a. **Insertion**
    b. **Deletion**
    c. **Display**

AIM: A program to implement circular queue with the given operations

DESCRIPTION:

Circular queue is a linear data structure in which the operations are performed based on FIFO (First in First out) principle and the last position is connected back to the first position to make a circle.

Operations on Circular Queue:

- enQueue(value): This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position. Check whether queue is full - check (rear==size-1 && front == 0) || (rear == front-1)). If it is full then display queue is full. If queue is not full then, check if (rear==size-1 && front!=0) if it true then set rear =0 and insert element
- deQueue(): This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position. Check whether queue is empty or not. Then if it is not empty, check if front is equal to size-1 and then if it true, set front = 0 and return the element.

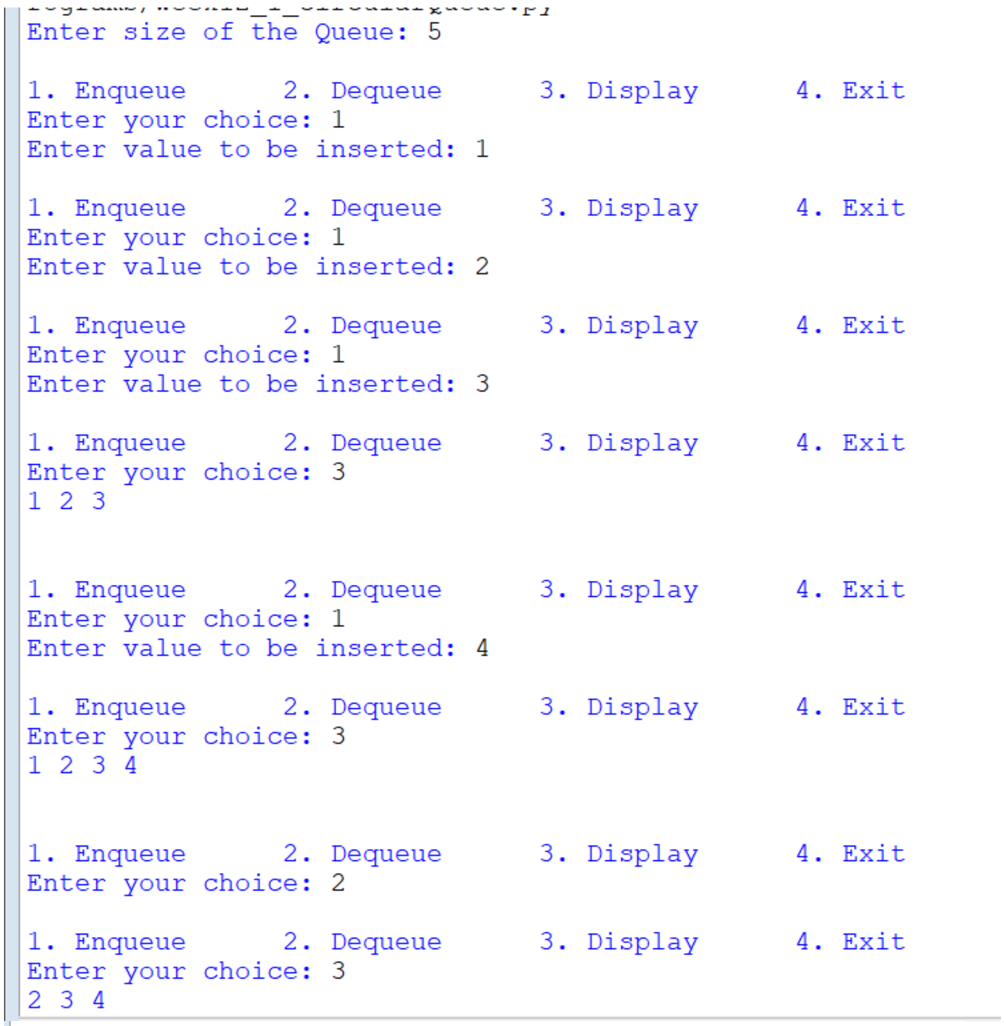
PROGRAM:

```
class circular:
  def __init__(self,size):
    self.size=size
    self.cq=[None]*size
    self.front=-1
    self.rear=-1
  def insert(self):
    data=int(input("Enter value to be inserted: "))
    if (self.front==(self.rear+1)%self.size):
      print("Overflow")
    elif (self.front==-1 and self.rear==-1):
      self.front=0
      self.rear=0
```

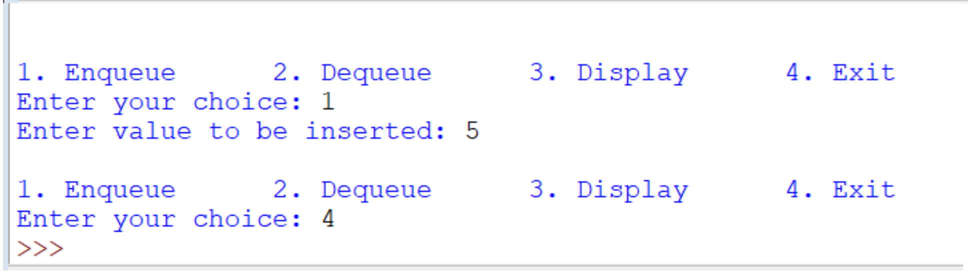
```
else:
         self.rear=(self.rear+1)%self.size
      self.cq[self.rear]=data
   def delete(self):
      if self.rear==-1:
         print("Underflow")
      else:
         if (self.front==self.rear):
            self.front=-1
            self.rear=-1
         elif (self.front==self.size-1):
            self.front=0
         else:
            self.front=(self.front+1)%self.size
   def display(self):
      if self.rear==-1:
         print("Queue is empty")
      if self.front>=self.rear:
         for x in range(self.front,self.size):
            print(self.cq[x],end=' ')
         for x in range(0,self.rear+1):
            print(self.cq[x],end=' ')
      else:
         for x in range(self.front,self.rear+1):
            print(self.cq[x],end=" ")
         print("\n")
n=int(input("Enter size of the Queue: "))
c=circular(n)
```

```
while True:
   co=int(input("\n1. Enqueue\t2. Dequeue\t3. Display\t4. Exit\nEnter your choice: "))
   if co==1:
      c.insert()
   elif co==2:
      c.delete()
   elif co==3:
      c.display()
   elif co==4:
      break
   else:
      print("Wrong choice, try again")
```

OUTPUT:



```
Enter size of the Queue: 5

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 1
Enter value to be inserted: 1

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 1
Enter value to be inserted: 2

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 1
Enter value to be inserted: 3

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 3
1 2 3


1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 1
Enter value to be inserted: 4

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 3
1 2 3 4


1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 2

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 3
2 3 4
```



```

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 1
Enter value to be inserted: 5

1. Enqueue       2. Dequeue      3. Display      4. Exit
Enter your choice: 4
>>>
```

CONCLUSION: The code is error free and runs as expected.

TIME COMPLEXITY: Time complexity of enqueue(), dequeue() operation is O(1).

2. **Write an application to implement hashing, when collision occurred use linear probing method to resolve collision and after storing elements find location of specified element in hashing.**

AIM: An application to implement hashing using linear probing method to resolve collision and after storing elements, find location of specified element in hashing.

DESCRIPTION:

Hashing is implemented in two steps:

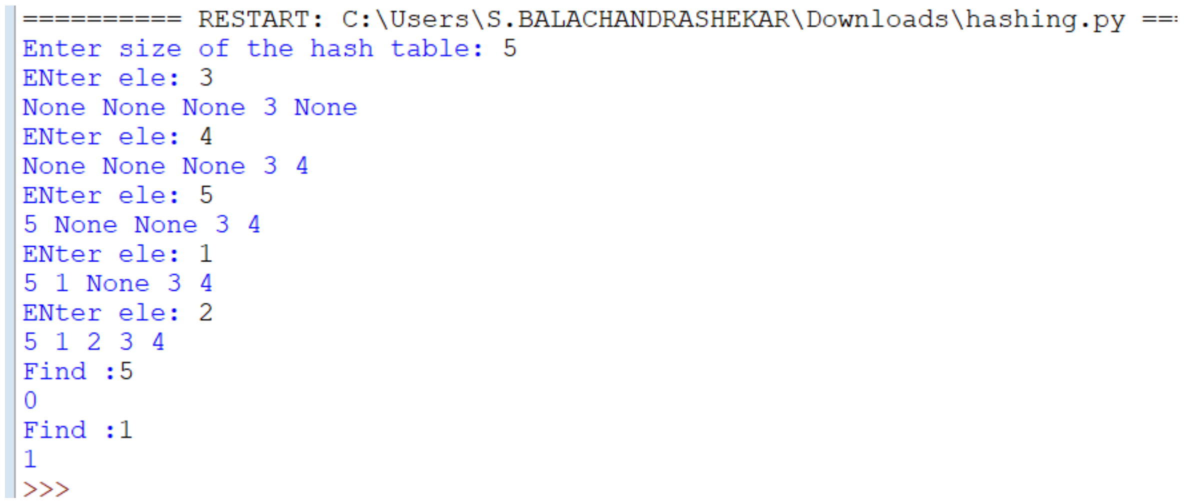
1. An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.
2. The element is stored in the hash table where it can be quickly retrieved using hashed key.
   hash = hashfunc(key)
   index = hash % array_size
   In this method, the hash is independent of the array size and it is then reduced to an index (a number between 0 and array_size − 1) by using the modulo operator (%).
   Linear probing is when the interval between successive probes is fixed (usually to 1). Let's assume that the hashed index for a particular entry is index. The probing sequence for linear probing will be:
   index = index % hashTableSize
   index = (index + 1) % hashTableSize
   index = (index + 2) % hashTableSize
   index = (index + 3) % hashTableSize

PROGRAM:

```
class Hashtable:
  def __init__(self,size) -> None:
    self.a=[None]*size
    self.size=size
  def add_element(self,ele):
    if self.a[ele%self.size]==None:
      self.a[ele%self.size]=ele
    elif self.a[ele%self.size]!=None:
      i=1
      while(True):
        if self.a[(ele+i)%self.size]==None:
          self.a[(ele+i)%self.size]=ele
          break
```

```
            i+=1

    def search_index(self,ele):

        if self.a[ele%self.size]!=ele:

            i=0

            while(True):


                if self.a[(ele+i)%self.size]==None:

                    self.a[(ele+i)%self.size]=ele

                    return ((ele+i)%self.size)

                i+=1

        else:

            return ele%self.size

size=int(input("Enter size of the hash table: "))

H=Hashtable(size)

for x in range(size):

    a=int(input("ENter ele: "))

    H.add_element(a)

    print(*H.a)

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

print(H.search_index(ele))

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

print(H.search_index(ele))
```

OUTPUT:



```
========== RESTART: C:\Users\S.BALACHANDRASHEKAR\Downloads\hashing.py ==
Enter size of the hash table: 5
ENter ele: 3
None None None 3 None
ENter ele: 4
None None None 3 4
ENter ele: 5
5 None None 3 4
ENter ele: 1
5 1 None 3 4
ENter ele: 2
5 1 2 3 4
Find :5
0
Find :1
1
>>>
```

CONCLUSION: The code is error free and runs as expected.

TIME COMPLEXITY:

Worst-case: O(n)

Average and best cases: O(1).