Exam Seat No. _____

THAKUR COLLEGE OF SCIENCE & COMMERCE

NAAC Accredited with Grade "A" (3rd Cycle)

THAKUR TRUSTS

ISO 9001 : 2015 Certified

Degree College

Computer Journal

CERTIFICATE

SEMESTER 2 UID No. _____

Class FYBSc(IT) Roll No. 1742 Year 2019 - 2020

This is to certify that the work entered in this journal is the work of Mst. / Ms. Raj Waghela

who has worked for the year 2019-2020 in the Computer Laboratory.

Teacher In-Charge

Head of Department

Date _____

Examiner

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
1	To search a number from the list using linear unsorted.	37	21/12/19	
2.	To search a number from the list using linear sorted.	37	21/12/19	
3.	To search a number from the given sorted list using binary search.	39.	21/12/19	
4.	~~To sort given random data by using bubble sort~~	41	~~18/1/20~~	
5.	To demonstrate the use of stack	43	18/1/20	
6.	To demonstrate Queue add and delete.	45	18/1/20	
7.	To demonstrate the use of Circular Queue in data structure.	47	18/1/20	
7	Demonstrate the use of Linked list	49	25/1/20	YL

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
8.	Evaluate Postfix using Stack	51	1/2/20	
9.	Bubble Sort	41	1/2/20	
10.	Quick sort	52	18/2/20	
11.	Selection Sort.	54.	18/2/20	
	Impetus (9.)			
12.	Merge sort	56	8/2/20	
13	Binary tree	58	8/2/20	
14	Impetus		15/2/20	

linear search unsorted.py - C:/Users/Raj/Desktop/my college/linear search unsorted.py (3.7.4)

File Edit Format Run Options Window Help

```
#LINEAR SEARCH UNSORTED
print("RAJ WAGHELA \n 1742")
A=[10,5,16,20,30,40]
print(A)
found=False
search=int(input(" enter a number to be search "))
for i in range(len(A)):
    if(search==A[i]):
        print("number found at: ",i)
        found=True
        break
if(found==False):
    print("not found")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Raj/Desktop/my college/linear search unsorted.py =====
RAJ WAGHELA
 1742
[10, 5, 16, 20, 30, 40]
 enter a number to be search 16
number found at:  2
>>>
===== RESTART: C:/Users/Raj/Desktop/my college/linear search unsorted.py =====
RAJ WAGHELA
 1742
[10, 5, 16, 20, 30, 40]
 enter a number to be search 40
number found at:  5
>>>
===== RESTART: C:/Users/Raj/Desktop/my college/linear search unsorted.py =====
RAJ WAGHELA
 1742
[10, 5, 16, 20, 30, 40]
 enter a number to be search 10
number found at:  0
>>>
```

Practical - 7

Aim: To search a number from the list using linear ~~Unsorted~~. unsorted.

Theory: The process of identifying or finding a particular record is called searching.

There are two types of search:-

1) Linear search
2) Binary search

The linear search is further classified as:

- Sorted
- Unsorted

Here we will look on the unsorted linear search.

linear search, also known as sequential search, is a process that checks every element in the list sequentially until the desired element is found. When the element to be searched are not specifically arranged in ascending or descending order. They are arranged in the random manner. That is what it calls unsorted linear search.

Unsorted linear search

→ The data is entered in the random manner.
→ User needs to specify the elements to be searched in the entered list.
→ Check the condition that whether the entered number matches, if it matches then display the location where data stored.
→ If all the elements are checked one by one and element not found then prompt message Number not found.

Practical - 2

Aim: To search a number from the list using linear sorted method.

Theory: Searching and sorting are different modes or types of data-structure.

Sorting: To basically sort the inputed data in ascending or descending manner.

Searching: To search elements and to display the same.
In searching that too in.
Linear sorted search the data is arranged in ascending to descending or descending to ascending. For That it is all ishat it meant by searching through 'sorted' that is well arranged data.

Sorted linear Search

→ The user is supposed to enter data in the sorted manner.
→ User has to given an element for searching through sorted list.
→ If element is found display with an updation as value is stored from location 'o'.
→ If data or element not found print the same.
→ In sorted order list of elements we can check one condition that whether the entered number lies from starting point till the last element if not then without any processing we can say number not the list.

linear sorted.py - C:/Users/Raj/Desktop/my college/linear sorted.py (3.7.4)

File Edit Format Run Options Window Help

```
#LINEAR SEARCH SORTED
print("RAJ WAGHELA \n 1742")
A=[1,6,17,20,38,46,55]
print(A)
found=False
search=int(input(" enter a number to be search "))
if(search<A[0]) and (search>A[len[A]]):
    print("no dosen't exist!")
else:
    for i in range(len(A)):
        if(search==A[i]):
            print("number found at: ",i)
            found=True
            break
    if(found==False):
        print("not found")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Raj/Desktop/my college/linear sorted.py ==========
RAJ WAGHELA
 1742
[1, 6, 17, 20, 38, 46, 55]
 enter a number to be search 38
number found at:  4
>>>
========== RESTART: C:/Users/Raj/Desktop/my college/linear sorted.py ==========
RAJ WAGHELA
 1742
[1, 6, 17, 20, 38, 46, 55]
 enter a number to be search 0
no dosen't exist!
>>>
```

binary search.py - C:/Users/Raj/Desktop/homepython/binary search.py (3.7.4)

File Edit Format Run Options Window Help

```
A=[7,8,10,27,28,37,45]
print('Raj Waghela\n1742')
print(A)
search=int(input("enter a number to be searched "))
l=0
r=len(A)
m=int((l+r)/2)
while(True):
    if(l==r):
        print("number not found")
        break;
    if(search==A[m]):
        print('number found at ',m)
        found=True
        break;
    else:
        if(search<A[m]):
            l=0
            r=m
            m=int((l+r)/2)
        else:
            l=m
            r=len(A)
            m=int((l+r)/2)
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Raj/Desktop/homepython/binary search.py ==========
Raj Waghela
1742
[7, 8, 10, 27, 28, 37, 45]
enter a number to be searched 0
number not found
>>>
========== RESTART: C:/Users/Raj/Desktop/homepython/binary search.py ==========
Raj Waghela
1742
[7, 8, 10, 27, 28, 37, 45]
enter a number to be searched 8
number found at  1
>>>
========== RESTART: C:/Users/Raj/Desktop/homepython/binary search.py ==========
Raj Waghela
1742
[7, 8, 10, 27, 28, 37, 45]
enter a number to be searched 28
number found at  4
>>>
```

Practical - 3

Aim: To search a number from the given sorted list using ~~bi~~ binary search

Theory: A binary search also known as a half-interval search, is an algorithm used in computer science to locate a specific value within an array. For binary search the list needed to be sorted..

It help to improve the search for sorted sequence. With help of divide and conquer strategy, which entails dividing a larger problem into smaller parts and conquering the smaller part.

Binary search

→ Take a sorted list in the program.

→ By suffering the value in the list, the user is given an input of their chosen value. For searching the value position in given list.

→ Initialize a variable to zero and take the length of the list ~~as~~ and assing the length of another variable.

→ Using while statement take the boolean value True. follow the if statement if element in list and still the element to be searched is not found.

→ If middle element is element to be searched then reset flag and shows the location of element.

→ ~~else~~ else if the element is searched is less than the middle element the l = 0 or else l = m

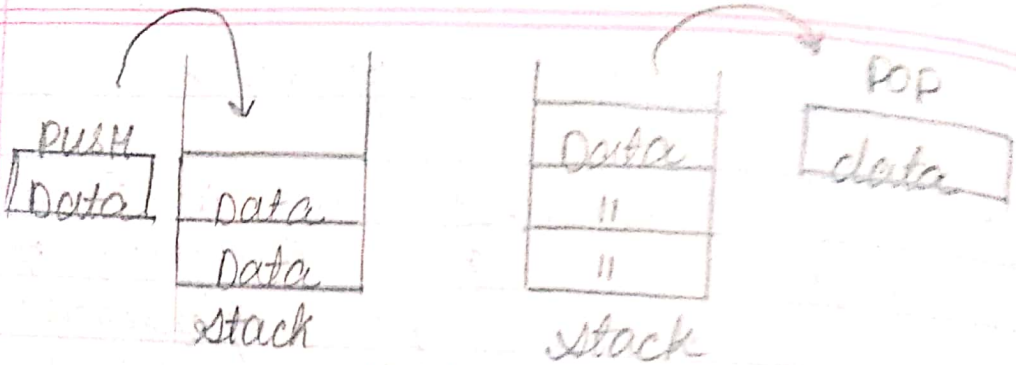
```
##stack##
print("raj waghela \n 1742")
class stack:
  global tos
  def __init__(self):
    self.l=[0,0,0,0,0,0,0]
    self.tos=-1
  def push(self,data):
    n=len(self.l)
    if self.tos==n-1:
      print("STACK IS FULL")
    else:
      self.tos=self.tos+1
      self.l[self.tos]=data
  def pop(self):
    if self.tos<0:
      print("STACK EMPTY")
    else:
      k=self.l[self.tos]
      print("data=",k)
      self.tos=self.tos-1
s=stack()
s.push(10)
s.push(20)
s.push(30)
s.push(40)
s.push(50)
s.push(60)
s.push(70)
s.push(80)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```

Practical - 5 4

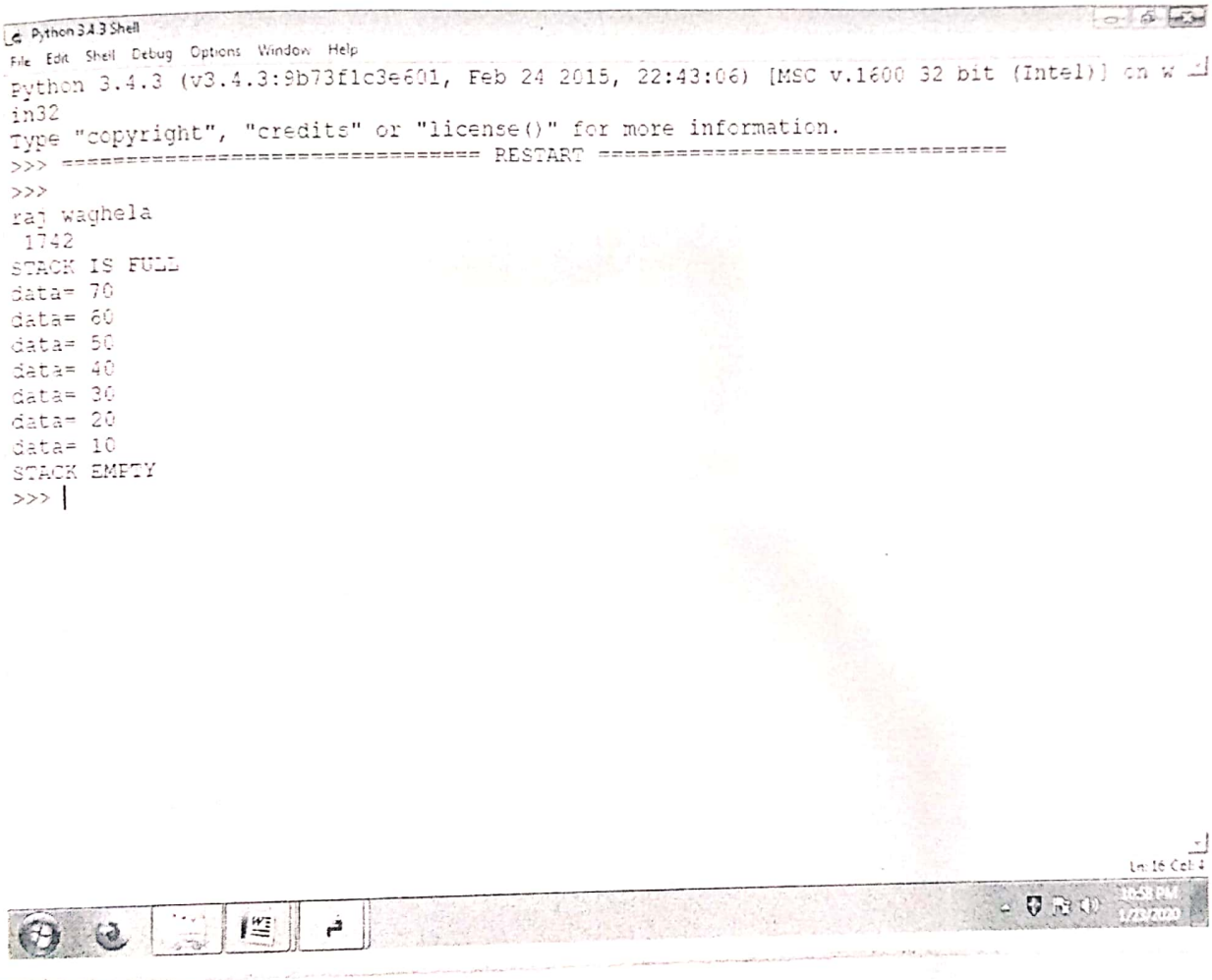
Aim: To demonstrate the use of stack.

Theory: In computer since, a stack is an abstract data type that serves as a collection of that elements with two principal operations push, which adds an elements to the collection and pop, which removes the most recently added element that was not yet removed. The order may be LIFO (last in first out) or FILO (First in last out).
Three Basic operations are performed in the stack.

1) PUSH: Adds an item in the if the stack is full then it is said to be over flow condition.
2) POP: Removes an item from the stack the items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an underflow condition.
3) Peek or Top: Returns top element of stack.
4) Is Empty: Returns true if stack is empty else false.



Last_in_First_out.



```
## Queue add and Delete ##
class Queue:
  print("raj waghela \n 1742")
  global r
  global f
  def __init__(self):
    self.r=0
    self.f=0
    self.l=[0,0,0,0,0,0]
  def add(self,data):
    n=len(self.l)
    if self.r<n-1:
      self.l[self.r]=data
      self.r=self.r+1
    else:
      print("Queue is full")
  def remove(self):
    n=len(self.l)
    if self.f<n-1:
      print(self.l[self.f])
      self.f=self.f+1
    else:
      print("Queue is empty")
Q=Queue()
Q.add(30)
Q.add(40)
Q.add(50)
Q.add(60)
Q.add(70)
Q.add(80)
```

Practical - 05

AIM: To demonstrate Queue add and delete.

Theory: 1) Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other ~~end~~ end called as FRONT.

2) Front points to the beginning of the queue and Rear points to the end of the queue.

3) Queue follows the FIFO (First_in_ First_out) structure.

4) According to its FIFO structure element inserted first ~~will~~ will also be removed first.

5) In a Queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue) because queue is open at both of its ends.

6) enqueue() can be termed as add() i.e adding a element in queue.

7) Dequeue() can be termed as delete or ~~Pop~~ Remove. i.e deleting or removing of element.

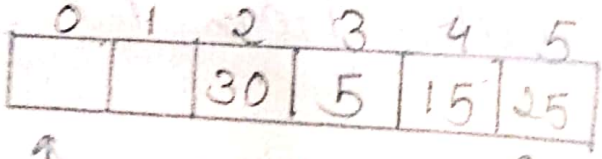
8) Front is used to get the front data item from a queue.

9) Rear is used to get the last item from a queue.

On both sides

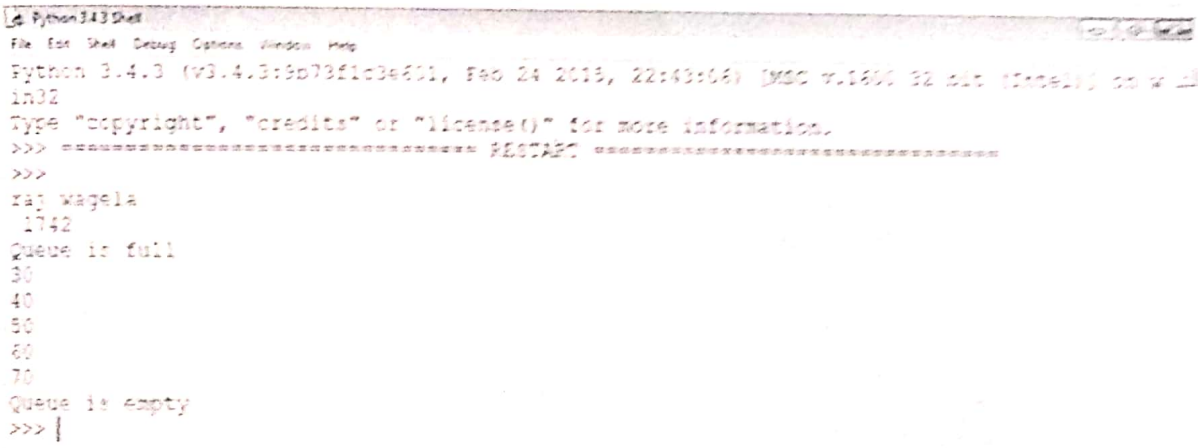
Can have ends

Queue



Front = 2
Rear = 5

```
Q.remove()

Q.remove()

Q.remove()

Q.remove()

Q.remove()

Q.remove()
```



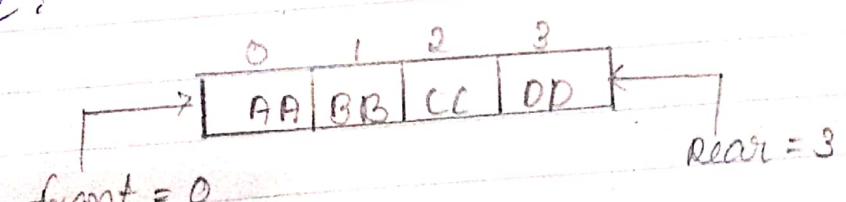
```
#(circular queue)
print("raj waghela \n 1755")
class Queue:
 global r
 global f
 def __init__(self):
  self.r=0
  self.f=0
  self.l=[0,0,0,0,0,0]
 def add(self,data):
  n=len(self.l)
  if self.r<=n-1:
   self.l[self.r]=data
   print("data added:",data)
   self.r=self.r+1
  else:
   s=self.r
   self.r=0
   if self.r<self.f:
    self.l[self.r]=data
    self.r=self.r+1
   else:
    self.r=s
    print("Queue is full")
def remove(self):
 n=len(self.l)
 if self.f<=n-1:
  print("data removed:",self.l[self.f])
  self.f=self.f+1
else:
 s=self.f
 self.f=0
```

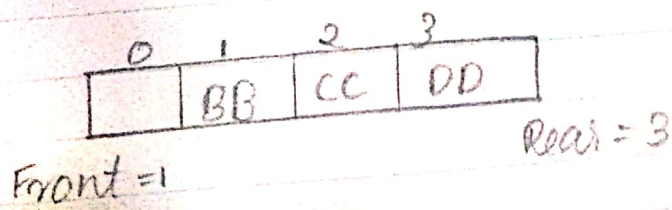
Practical - 16

AIM: To demonstrate the use of circular queue in data-structure.

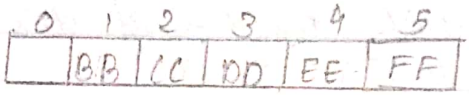
Theory: 1) The queue that we implement using an array suffer from one limitation. In that implementation there is a possibility that the queue is reported as full, even though in actuality there might be empty slots at the beginning of the queue.
2) To overcome this limitation we can implemente queue as circular queue.
3) In circular queue we go on adding the element to the queue and reach the end of the array.
4) The next element is stored in the first slot of the array.

Example:

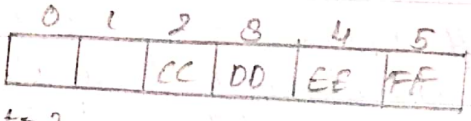




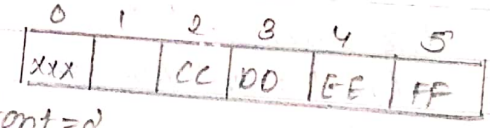
TA:



front=1 Rear=5

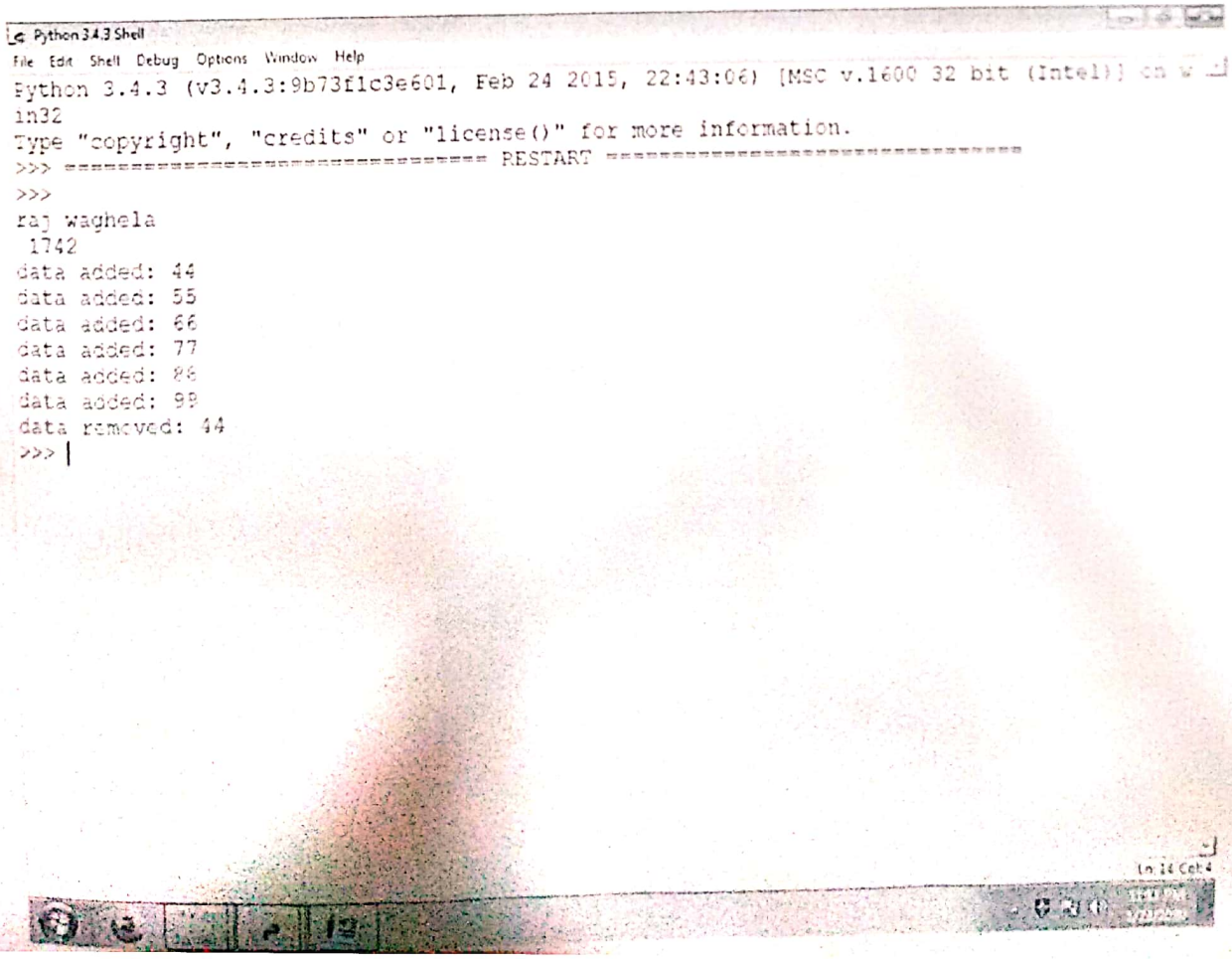


front=2 Rear=5



front=2 Rear=5

```
        if self.f<=self.r:
            print(self.i[self.f])
            self.f=self.f+1
        else:
            print("Queue is empty")
            self.f=s
Q=Queue()
Q.add(44)
Q.add(55)
Q.add(66)
Q.add(77)
Q.add(88)
Q.add(99)
Q.remove()
Q.add(66)
```



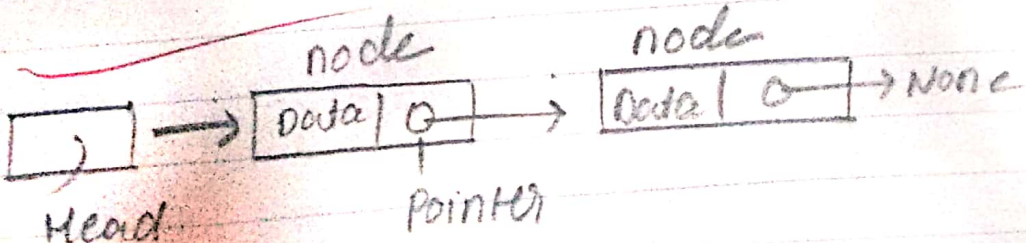
```
print("Linkedlist Program:")
class node:
    global data
    global next
    def __init__(self,item):
        self.data=item
        self.next=None
class linkedlist:
    global s
    def __init__(self):
        self.s=None
    def addL(self,item):
        newnode=node(item)
        if self.s==None:
            self.s=newnode
        else:
            head=self.s
            while head.next!=None:
                head=head.next
            head.next=newnode
    def addB(self,item):
        newnode=node(item)
        if self.s==None:
            self.s=newnode
        else:
            newnode.next=self.s
            self.s=newnode
    def display(self):
        head=self.s
        while head.next!=None:
            print(head.data)
            head=head.next
        print(head.data)
start=linkedlist()
start.addL(50)
start.addL(60)
start.addL(70)
start.addL(80)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
print("Raj waghela\n1742")
```

Aim : To demonstrate the use of linked list in datastructure.

Theory : A linked list is a sequence of data structure linked list is a sequence of links which contains items each link contains a connection to another link.

- LINK - Each link of a linked list can store a data called an element
- NEXT - Each link of a linked list contains a link to the next link called NEXT.
- Linked list - A linked list contains the connection link to the first link called First.

LINKED - LIST Representation:

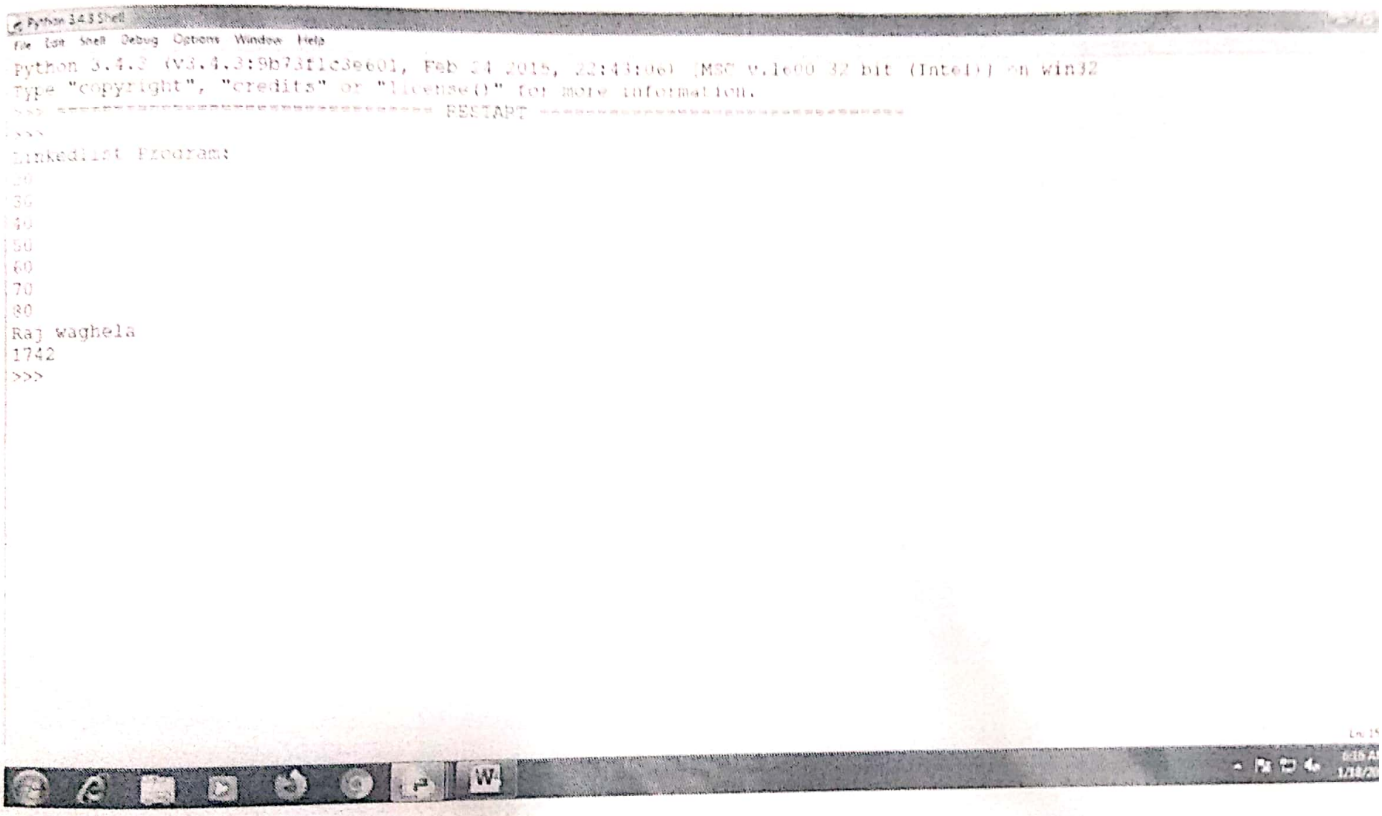


Type of linked list :-

→ Simple
→ Doubly
→ Circular

Basic Operations :-

→ Insertion
→ Deletion
→ Display
→ Search
→ Delete



```
print("Raj waghela\n1742\nPostFix Program:")
def evaluate(s):
    k=s.split()
    n=len(k)
    stack=[]
    for i in range(n):
        if k[i].isdigit():
            stack.append(int(k[i]))
        elif k[i]=='+':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)+int(a))
        elif k[i]=='-':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)-int(a))
        elif k[i]=='*':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(a)*int(b))
        else:
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)/int(a))
    return stack.pop()
s="8 6 9 * +"
r=evaluate(s)
print("the evaluated values is: ",r)
```

Aim: To evaluate postfix evaluation using stack.

Theory: Stack is an ADT and works on LIFO (Last in first out) i.e Push & Pop operations.
A postfix expression is a collection of operators and operands in which the operators is placed after the operands.

Steps to be followed :-

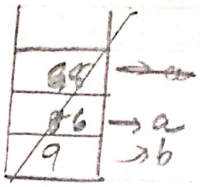
1. Read all the symbols one by one from left to right in the given expression.
2. If the reading symbol is operand the push it on the stack.
3. If the reading symbol is operator (+, -, *, /, etc) then perform TWO popped operands in two different variables (operand1 & operand2) Then perform reading symbol operator using operand1 & operand2 and push Result on the stack.
4. Finally! Perform a pop operations and display the popped value as final result.

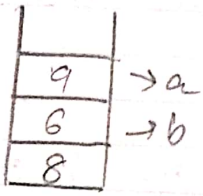
Value of postfix expression :
3 2 8 4 - + *

12

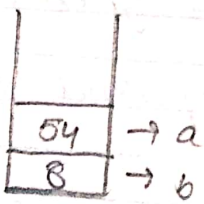
~~stack:~~ s = "8 6 9 * +"

stack

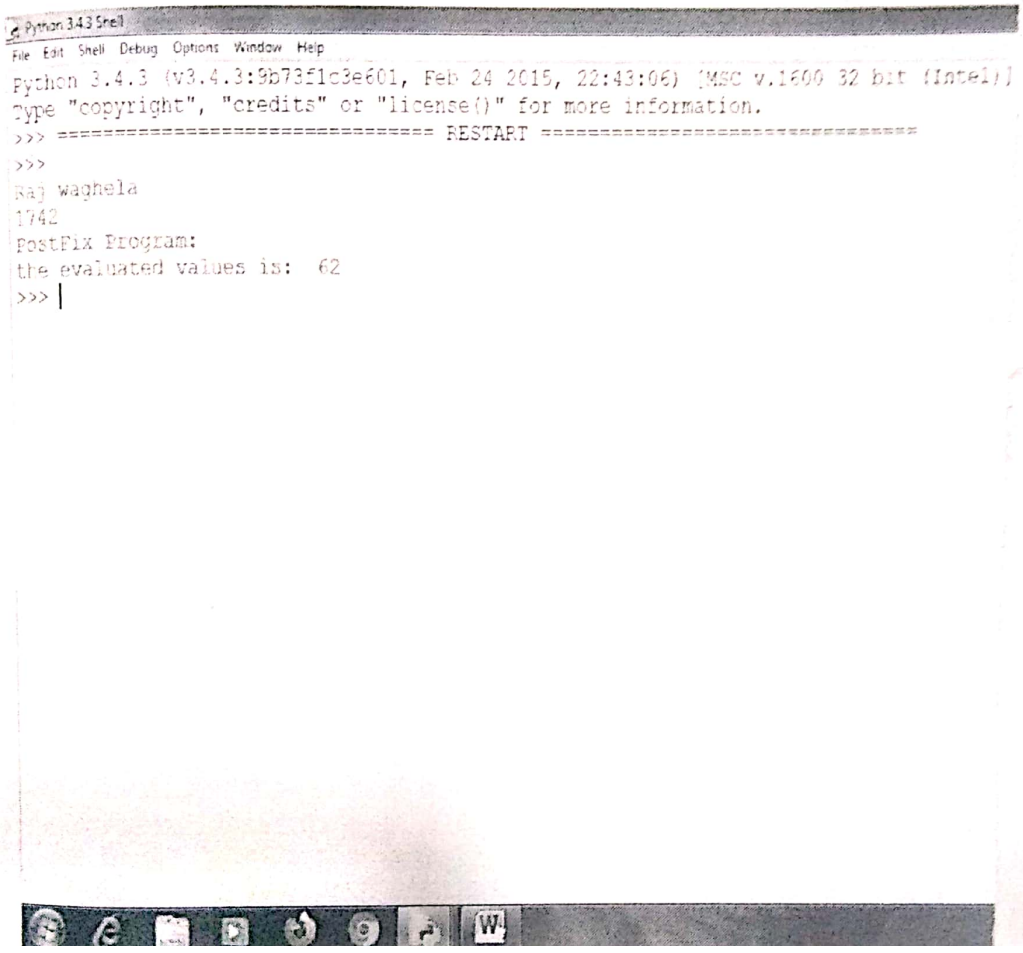




a * b = 9 * 6 = 54



a + b = 54 + 8 = 62



```
a=[13,12,15,18,19,11]
print(a)
for i in range(len(a)-1):
    for j in range(len(a)-1):
        if(a[j]>a[j+1]):
            t=a[j]
            a[j]=a[j+1]
            a[j+1]=t
print("Numbers after Bubble sort: \n",a)
print("Raj Waghela 1742")
```

~~Practical 4~~ Practical - 9

Aim: To sort given random data by using bubble sort.

Theory: 1) Sorting is type in which any random data is sorted i.e arranged in ascending or descending order.

2) Bubble sort sometimes refered is as sinking sort.

3) It is a simple sorting algorithm that repeatedly steps through the lists, compares adjacent elements and swaps them if they are in ~~order~~ wrong order.

4) The pass through the lists is repeated until the list is sorted.

5) The algorithm which is an comparision sort is named for the way smaller or larger elements "bubble" to the top of the list.

6) Although the algorithm is simple, it is too slow as ~~if~~ it compares one elements checks if ~~condition~~ fails then only swaps otherwise goes on.

7) Example:

First pass

(51428) ⟶ (15428) Here algorithm compares the first two elements and swaps since 5>1.

(15428) ⟶ (14528) swap since 5>4

(14528) ⟶ (14258) swap since 5>2

(14258) ⟶ (14258) Now since these elements are already in order (8>5) algorithm does not swap them.

Second pass:

(14258) ⟶ (14258)

(14258) ⟶ (12458) Swap since 4>2

(12458) ⟶ (12458)

Third pass:

(12458) It checks and gives the data is sorted order.

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/Admin/Documents/temp7.py ================
[13, 12, 15, 18, 19, 11]
Numbers after Bubble sort:
 [11, 12, 13, 15, 18, 19]
Raj Waghela 1742
>>>
```

```
def quickSort(alist):
    quickSortHelper(alist,0,len(alist)-1)
def quickSortHelper(alist,first,last):
    if first<last:
        splitpoint=partition(alist,first,last)
        quickSortHelper(alist,first,splitpoint-1)
        quickSortHelper(alist,splitpoint+1,last)
def partition(alist,first,last):
    pivotvalue=alist[first]
    leftmark=first+1
    rightmark=last
    done=False
    while not done:
        while leftmark<=rightmark and alist[leftmark]<=pivotvalue:
            leftmark=leftmark+1
        while alist[rightmark]>=pivotvalue and rightmark>=leftmark:
            rightmark=rightmark-1
        if rightmark<leftmark:
            done=True
        else:
            temp=alist[leftmark]
            alist[leftmark]=alist[rightmark]
            alist[rightmark]=temp
    temp=alist[first]
    alist[first]=alist[rightmark]
    alist[rightmark]=temp
    return rightmark
alist=[13,12,15,18,19,11]
print(alist)
```

Practical - 10

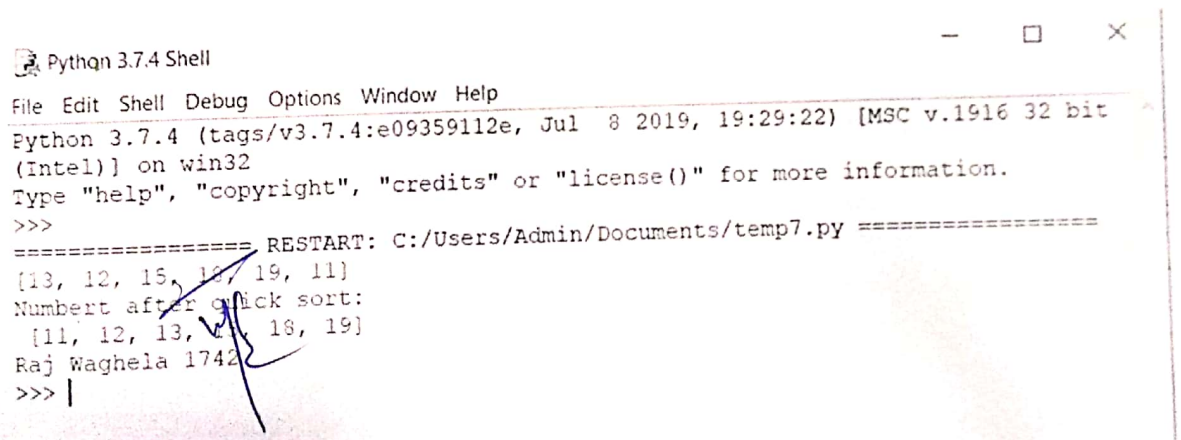
Aim: To 'evaluate' i.e to sort the given data in Quick sort.

Theory: Quicksort is an efficient sorting algorithm. Type of a Divide & Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quick sort that pick pivot in different ways.

1) Always pick first element as pivot.
2) Always pick last element as pivot.
3) Pick a random element as pivot.
4) Pick median as pivot.

The key process in quicksort is partition(). Target of partitions is given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, & put all greater ~~elements~~ (greater than x) after x. All this should be done in linear time.

```
quickSort(alist)
print("Numbert after quick sort: \n",alist)
print("Raj Waghela 1742")
```

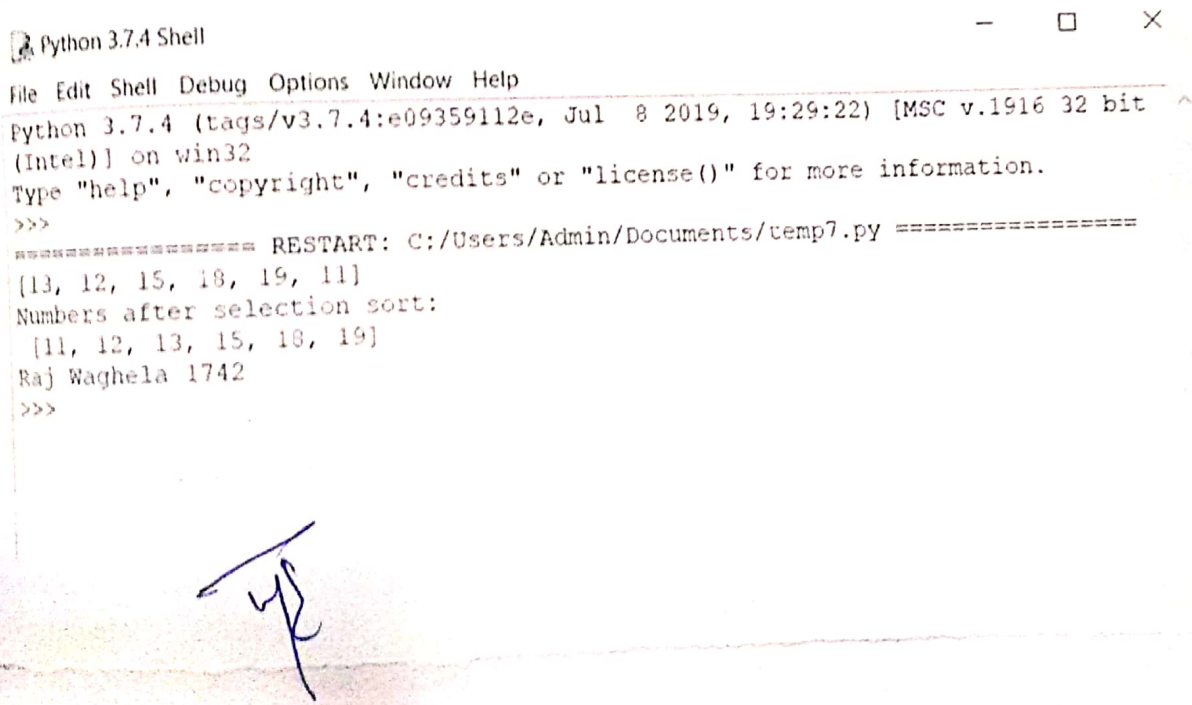


```
a=[13,12,15,18,19,11]
print(a)
for i in range(len(a)-1):
    for j in range(len(a)-1):
        if(a[j]>a[i+1]):
            t=a[j]
            a[j]=a[i+1]
            a[i+1]=t
print("Numbers after selection sort: \n",a)
print("Raj Waghela 1742")
```

Practical - 11

Topic : Selection Sort.

Theory : Selection sort is a simple sorting algorithm. This sorting algorithm is an in place comparison based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.
The smallest element is selected from the unsorted array and swapped with the leftmost element, and that elements becomes part of the sorted array. This process continues moving unsorted array boundry by one element to the right.
The algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$, where n is the number of items.



```
def mergeSort(arr):

  if len(arr)>1:
    mid = len(arr)//2
    lefthalf = arr[:mid]
    righthalf = arr[mid:]

    mergeSort(lefthalf)
    mergeSort(righthalf)

    i=j=k=0

    while i < len(lefthalf) and j < len(righthalf):
      if lefthalf[i] < righthalf[j]:
        arr[k]=lefthalf[i]
        i=i+1

      else:
        arr[k]=righthalf[j]
        j=j+1
      k=k+1

    while i < len(lefthalf):
      arr[k]=lefthalf[i]
      i=i+1
      k=k+1

    while j < len(righthalf):
      arr[k]=righthalf[j]
      j=j+1
      k=k+1
```

Topic : Merge Sort

Theory :
- Merge Sort uses the divide and conquer technique. In merge sort we divide the list into nearly equal sublists, each of which are then again divided into two sublists.
- We continue this procedure until there is only one element in the individual sublist
- Once we have individual elements in the sublists, we consider these sublists as subproblems which can be solved. Since there is only one element in the sublist, the sublist is already sorted. So now we merge the subproblems.
- Now while merging the subproblems, we compare the 2 sublists and create the combined list by comparing the element of the sublists. After comparison we place the elements in their correct position in the original sublists.

```
arr = [11,10,15,28,18,69,14,42,24]

print("THE UNSORTED LIST : ", arr)

mergeSort(arr)

print("\nTHE MERGE SORTED LIST : ",arr)

print("Raj Waghela 1742")
```

```
Python 3.7.4 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Admin\Documents\temp1.py ==========
THE UNSORTED LIST :  [11, 10, 15, 28, 18, 69, 14, 42, 24]

THE MERGE SORTED LIST :  [10, 11, 14, 15, 18, 24, 28, 42, 69]
Raj Waghela 1742
>>>
```

4/8

```
class Node:
    global r
    global l
    global data
    def __init__(self,l):
        self.l=None
        self.data=l
        self.r=None
class Tree:
    global root
    def __init__(self):
        self.root=None
    def add(self,val):
        if self.root==None:
            self.root=Node(val)
        else:
            newnode=Node(val)
            h=self.root
            while True:
                if newnode.data<h.data:
                    if h.l!=None:
                        h=h.l
                    else:
                        h.l=newnode
                        print(newnode.data,"added on
left of",h.data)
```

Practical - 13

Topic - Binary Tree

Theory: Binary tree is a tree which supports maximum of 2 children for any node within the tree. Thus any particular node can have either 0 or 1 or 2 children.

Leaf Node: Nodes which do not have any children

Internal Node: Nodes which are non-leaf nodes

Traversing can be defined as a process of visiting every node of the tree exactly once

Preorder: (i) Visit the root node.
(ii) Traverse the left subtree. The left subtree in turn might have left and right subtrees.
(iii) Traverse the right subtree. The right subtree in turn might have left and right subtrees.

Inorder: i) Traverse the left subtree. The left subtree in turn might have left and right subtrees.
ii) Visit the root node.
iii) Traverse the right subtree. The right subtree in turn might have left and right subtrees

Postorder : i) Traverse the left subtree. The left subtree inturn might have left and right subtrees.

ii) Traverse the right subtree. The right subtree. The right subtree inturn might have left and right subtrees

iii) Visit the root node.

right

p

```
                    break
                else:
                    if h.r!=None:
                        h=h.r
                    else:
                        h.r=newnode
                        print(newnode.data,"added on
right of",h.data)
                        break
    def preorder(self,start):
        if start!=None:
            print(start.data)
            self.preorder(start.l)
            self.preorder(start.r)
    def inorder(self,start):
        if start!=None:
            self.inorder(start.l)
            print(start.data)
            self.inorder(start.r)
    def postorder(self,start):
        if start!=None:
            self.postorder(start.l)
            self.postorder(start.r)
            print(start.data)
print("Raj waghela\n1742\nBinary Tree and Traversal")
t=Tree()
t.add(100)
```

```
t.add(90)
t.add(75)
t.add(76)
t.add(18)
t.add(83)
t.add(57)
t.add(89)
t.add(14)
t.add(5)
print("preorder")
t.preorder(t.root)
print("inorder")
t.inorder(t.root)
print("postorder")
t.postorder(t.root)
```