Source Code :

File Edit Format Run Options Window Help

```
n=0
a=[4,29,21,64,25,24,3]
search=input("enter no to be searched")
for i in range(len(a)):
    if(search==a[i]):
        print("no found at %d",i)
        n=1
        break
if(n!=1):
    print("no not found")
print("name:gaurav singh \n roll no:1732")
```

Output :

```
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MS
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Gaurav\Documents\ds practical 1.
enter no to be searched64
('no found at %d', 3)
name:gaurav singh
 roll no:1732
>>>
========== RESTART: C:\Users\Gaurav\Documents\ds practical 1.
enter no to be searched3
('no found at %d', 6)
name:gaurav singh
 roll no:1732
>>>
========== RESTART: C:\Users\Gaurav\Documents\ds practical 1.
enter no to be searched7
no not found
name:gaurav singh
 roll no:1732
```

Practical - 1

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

Theory: The process of identifying or finding a particular record is called searching
There are two types of search
→ Linear Search
→ Binary Search
The linear search is further classified as
* SORTED * UNSORTED
Here we will look on the UNSORTED 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 elements to be searched are not specifically arranged in ascending or descending order. They are arranged in random manner. That is what it calls unsorted linear search

4/2

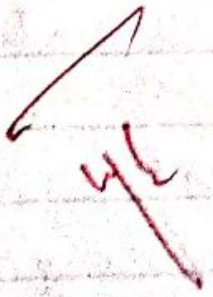
Unsorted linear search

→ The data is entered in random manner.

→ User needs to specify the element to be searched in the entered list

→ check the condition that whether the entered number matched if it matches then display the location plus increment 1 as data is stored from location zero.

→ If all elements are checked one by one and element not found then prompt message number not found.



Source code:

```
n=0
a=[3,4,21,24,25,29,64]
search=input("enter no to be searched")
if((search<a[0] or search>a[len(a)-1])):
    print("does not exist")

else:
    for i in range(len(a)):
        if(search==a[i]):
            print("no found at %d",i)
            n=1
            break
    if(n!=1):
        print("no not found")
print("name:gaurav singh \n roll no:1732"
```

Output:

```
File Edit Shell Debug Options Window Help
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Gaurav\Documents\ds practical 1.py ==
enter no to be searched64
('no found at %d', 6)
name:gaurav singh
 roll no:1732
>>>
========== RESTART: C:\Users\Gaurav\Documents\ds practical 1.py ==
enter no to be searched4
('no found at %d', 1)
name:gaurav singh
 roll no:1732
>>>
========== RESTART: C:\Users\Gaurav\Documents\ds practical 1.py ==
enter no to be searched7
no not found
name:gaurav singh
 roll no:1732
```

Practical - 2

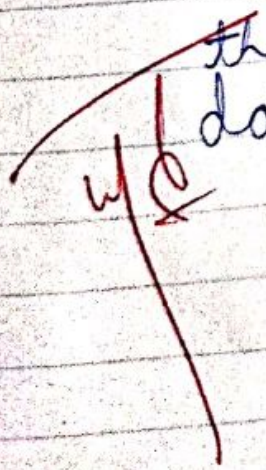
Aim: To search a number from the list using linear sorted method.

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 elments and to display the same.

Or searching that too in LINEAR SORTED search the data is arranged in ascending to descending or descending to ascending. That is all what it meant by searching through 'sorted' that is well arranged data



Sorted Linear Search

→ The user is supposed to enter data in 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 '0'.
→ If data or element not found print the same.
→ In sorted order list of elements elements we can check the 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 in the list.

source code:

```
print("name:gaurav singh\nroll no:1732")
a=[3,4,21,24,25,29,64]
print(a)
search=int(input("enter number to be searched from the list:"))
l=0
h=len(a)-1
m=int((l+h)/2)
if((search<a[l])or(search>a[h])):
    print("number not in range")
elif(search==a[h]):
    print("number found at location:",h+1)
elif(search==a[l]):
    print("number found at location:",l+1)
else:
    while(l!=h):
        if(search==a[m]):
            print("number found at location:",m+1)
            break
        else:
            if(search<a[m]):
                h=m
                m=int((l+h)/2)
            else:
                l=m
                m=int((l+h)/2)
            if(search!=a[m]):
                print("number not in given list")
                break
```

output:

```
>>>
========== RESTART: C:/Users/Gaurav/Documents/ds practical 3.py
==========
name:gaurav singh
roll no:1732
[3, 4, 21, 24, 25, 29, 64]
enter number to be searched from the list:64
('number found at location:', 7)
>>>
========== RESTART: C:/Users/Gaurav/Documents/ds practical 3.py
==========
name:gaurav singh
roll no:1732
[3, 4, 21, 24, 25, 29, 64]
enter number to be searched from the list:97
number not in range
>>>
========== RESTART: C:/Users/Gaurav/Documents/ds practical 3.py
==========
name:gaurav singh
roll no:1732
[3, 4, 21, 24, 25, 29, 64]
enter number to be searched from the list:7
number not in given list
```

Practical - 3

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

Theory: A binary search also known as a half-interval search, is an algorithm used in computer science to locate a specified value (key) within an array. For the search to be binary the array must be sorted in either ascending or descending order.

At each step of the algorithm a comparison is made and the procedure branches into one of two directions.

specifically, the key value is compared to the middle element of the array. If the key value is less than or greater than this middle element, the algorithm knows which half of the array to continue searching in because the array is sorted.

This process is repeated on progressively smaller segments of the array until the value is located. Because each step in the algorithm divides the array size in half, a binary search will complete successfully in logarithmic time

Practical-4

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

Theory: Sorting is type is which any random data is sorted i.e arranged in ascending or descending order.
BUBBLE sort sometimes ~~refff~~ referred to as sinking sort.
Is a simple sorting algorithm that repeatedly steps through the lists, compares adjacent elements and swape them if they are in wrong order.
The pass through the list is repeated until the list is sorted.
The algorithm which is a comparision sort is named for way smaller or larger elements "bubble" to the top of the list.
Although the algorithm is simple, it is too slow as it compares one element checks if condition fails then only swaps otherwise goes on

source code:

```
print("name:gaurav singh \n roll no:1732")
a=[7,3,8,9,5,6]
for p in range(len(a)-1):
    for c in range(len(a)-1-p):
        if (a[c]>a[c+1]):
            t=a[c]
            a[c]=a[c+1]
            a[c+1]=t
print a
```

output:

```
>>>
============ RESTART: C:\Users\Gaurav\Documents\ds practical 4.py
============
name:gaurav singh
 roll no:1732
[3, 5, 6, 7, 8, 9]
>>>
```

Example:
First pass
(5 1 4 2 8) → (1 5 4 2 8) Here algorithm compare the first two elements and swaps since 5 > 1.
(1 5 4 2 8) → (1 4 5 2 8) swap since 5 > 4
(1 4 5 2 8) → (1 4 2 5 8) swap since 5 > 2
(1 4 2 5 8) → (1 4 2 5 8) Now since these elements are already in order (8 > 5) algorithm does not swap them.

Second pass:
(1 4 2 5 8) → (1 4 2 5 8)
(1 4 2 5 8) → (1 2 4 5 8) swap since 4 > 2
(1 2 4 5 8) → (1 2 4 5 8)

Third pass:
(1 2 4 5 8) It checks and gives the data in sorted order.

Practical - 5

Aim: Demonstrate the use of stack()

Theory: In computer science, a stack is an abstract data type that serves as a collection of elements with two principal operations push, which adds an element 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 FIFO (First In First out).

There are 3 basic operators performed in the stack.

i) Push :- Adds an item in the stack. If the stack is full then it is said to be over flow condition

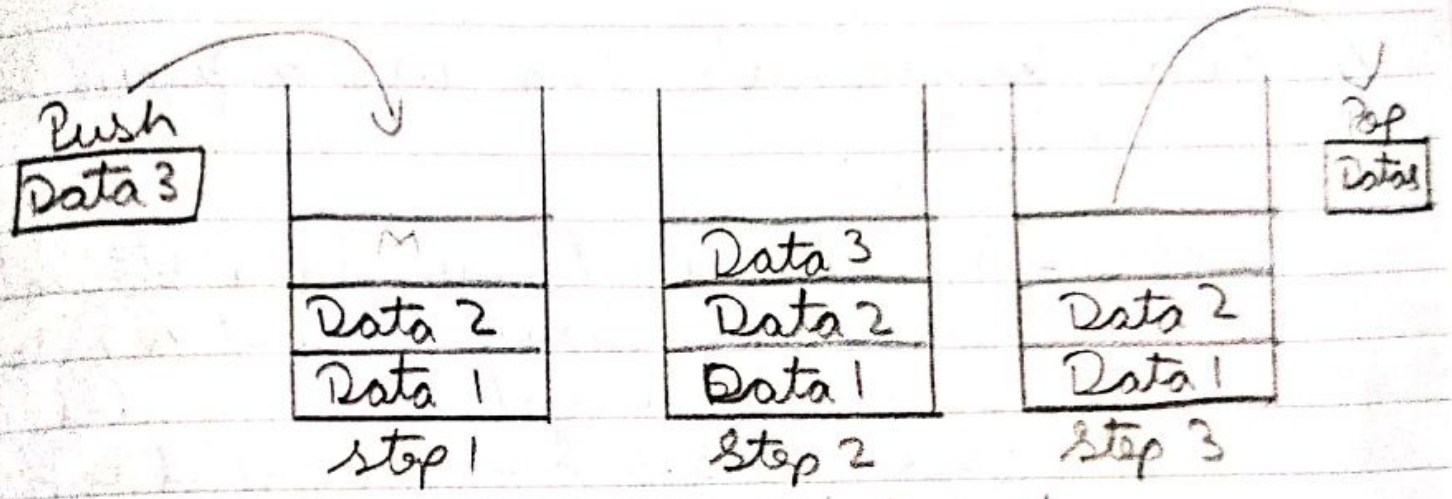
ii) Pop :- Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an underflow condition.

iii) Peek or Top :- Returns Top element of stack.

iv) is Empty :- Return True if stack is empty else else false.

```
#Stack
print("Name:Gaurav Singh \nRoll No:1732")
class Stack():
        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 += 1
        self.l[self.tos] = data
        def pop(self):
        if self.tos < 0:
        print("Stack is Empty")
        else:
        k = self.l[self.tos]
        print("data = ",k)
        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()

#Output:
Name: Gaurav Singh
RollNo:1732
Stack is Full
data = 70
data = 60
data = 50
data = 40
data = 30
data = 20
data = 10
Stack is Empty
```



Last in First out

Practical-6

Aim: Demostrate the use of Queue and delete

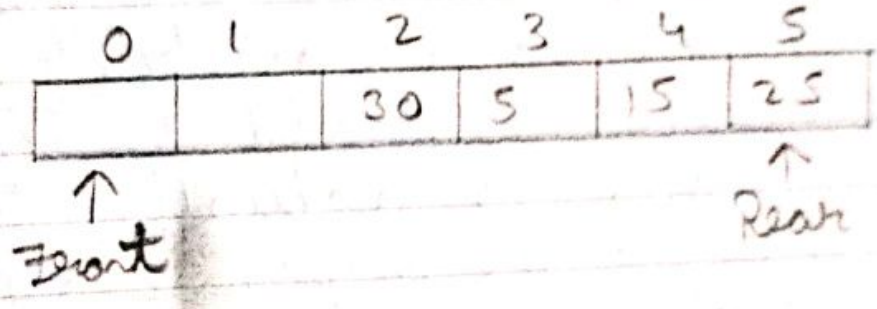
Theory:- 1) Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end called as FRONT.
2) Front points to beginning of the Queue and Rear points to the end of the Queue.
3) Queue follows the FIFO (First in First out) structure.
4) According to its FIFO structure, element inserted first will also be removed first.
5) In a Queue, one end is always used to delete data (dequeue) and the other is used to insert data (enqueue) because queue is open at both of its ends.
6) Enqueue() can be termed as add() in queue i.e adding an element in queue and dequeue() can be termed as delete or remove i.e deleting or removing the element.
7) Front is used to get the front data iten from a queue and Rear is used to get the last iten from a queue.

```
#Queue and Delete
print("Name: Gaurav Singh \nRollNo.:1732")
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
        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 += 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)
q.remove()
q.remove()
q.remove()
q.remove()
q.remove()
q.remove()

#Output:
Name: Gaurav Singh
Roll No.:1732
Queue is Full
30
40
50
60
70
Queue is Empty
```

~~#~~ or Both sides

2 Queue → can have ends



Front = 2

~~Front =~~

Rear = 5

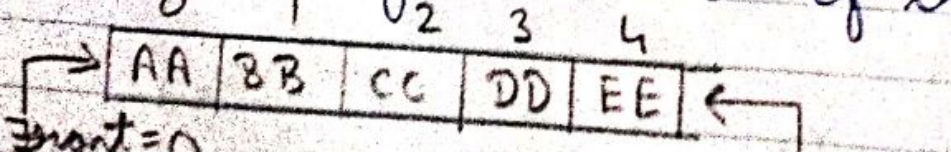
Practical - 7

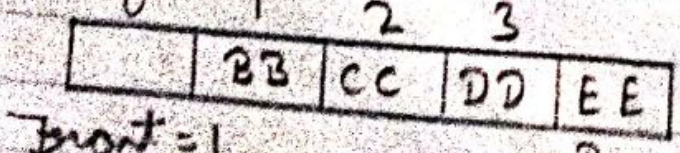
Aim :- Demostrate the use of Circular Queue in Data structure

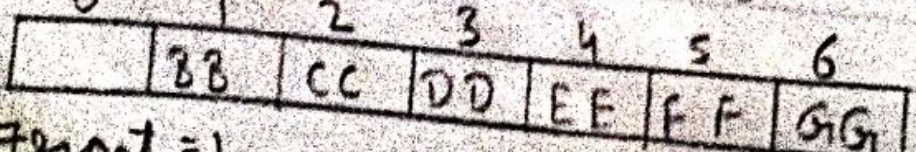
Theory: 1) The queue that are implemented using an array suffer from one limitation. In that implementation there is a possibility that the queue is reported as full, even though is actually there might be empty slots at the beginning of the queue.

2) To overcome the limitation, we can implement queue as Circular queue.

3) In circular queue, we go on adding the element to the queue and reach the end of the array and the next element is sorted is the first slot of the array







```
#Circular Queue
print("Name: Gaurav Singh \nRoll No 1732")
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 += 1
        else:
        s = self.r
        self.r = 0
        if self.r < self.f:
        self.l[self.r] = data
        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 += 1
        else:
        s = self.f
        self.f = 0
        if self.f < self.r:
        print(self.l[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)
```

Output :

```
Name: Gaurav Singh
Roll No 1732
data added :  44
data added :  55
data added :  66
data added :  77
data added :  88
data added :  99
data removed :  44
```

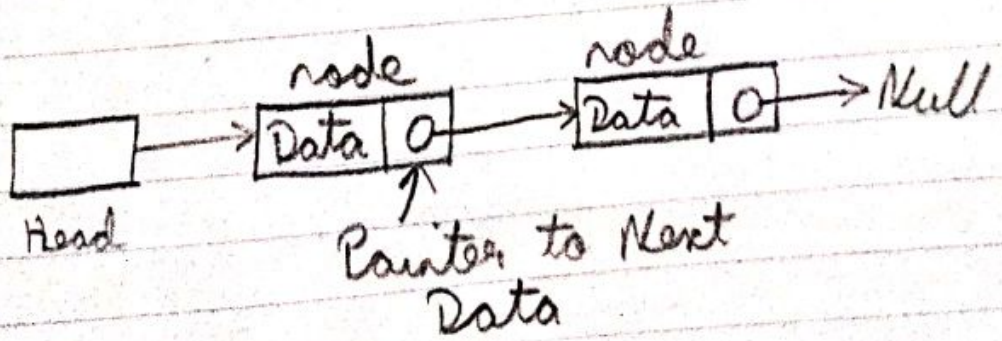
Practical - 8

Aim :- Demonstrate the use of linked lists in Data structure.

Theory :- 1) A linked list is a sequence of data structures linked list is a sequence of ~~links~~ links which contains items and 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 :-



```
#LinkedList
print("Name: Gaurav Singh \nRoll No.:1732")
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()

#Output:
Name: Gaurav Singh
Roll No.:1732
20
30
40
50
60
70
80
```

Types of linked list :- 1) Simple 2) Doubly 3) Circular

Basic operation :-

1) Insertion
2) Deletion
3) Display
4) Search
5) Delete

Practical - 9

Aim :- Evaluation of Postfix Expression using stack.

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

Steps to be followed :-
1) Read all the symbols one by one from left to right in the given postfix evaluation.
2) If the reading symbol is operand then push it on to the stack.
3) If the reading symbol is operator (+, -, *, /, etc) then perform Two pop operations and store the two popped operand on two different variable (operand 1 and operand 2). Then perform reading symbol operator using operand 1 and operand and push result back on to the stack.
4) Finally. Perform a pop operation and display the popped value as final result.

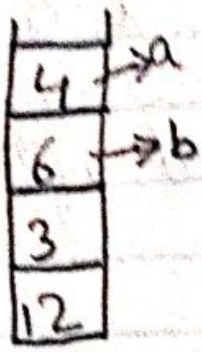
```
#PostFix Evaluation
print("Name: Gaurav Singh \nRoll No.:1732")
def evaluate(s):
        k = s.split()
        n = len(k)
        stack = []
        for i in range(n):
        if k[i].isdigit():
        stack.append(int(k[i]))
        elif k[i] == '+':
        a = stack.pop()
        b = stack.pop()
        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(b)*int(a))
        else:
        a = stack.pop()
        b = stack.pop()
        stack.append(int(b)/int(a))
        return stack.pop()
s = "5 6 4 2 * - +"
r = evaluate(s)
print("The Evaluated Value is ",r)

#Output:
Name: Gaurav Singh
Roll No.:1732
The Evaluated Value is  3
```

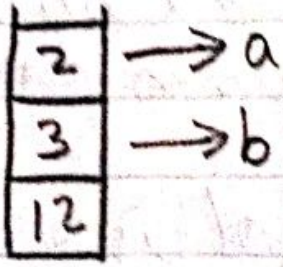
* Value of Postfix Evaluation :-

S = 1 2 3 6 4 ⊖ + *

Stack :-



b - a = 6 - 4 = 2



b + a = 3 + 2 = 5

Practical-10

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

Theory: Quicksort is an efficient ~~sor~~ sorting algorithm. Type of a Divide & Conquer algorithm. It pick an element as pivot and partitions the given array around ~~on~~ the picked 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 ~~than~~ (greater than x) after x. All this should be done in linear time.

```
print("Name: Gaurav singh \nRoll No.:1732")

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=[42,54,45,67,89,66,55,80,100]
quickSort(alist)
print(alist)
```

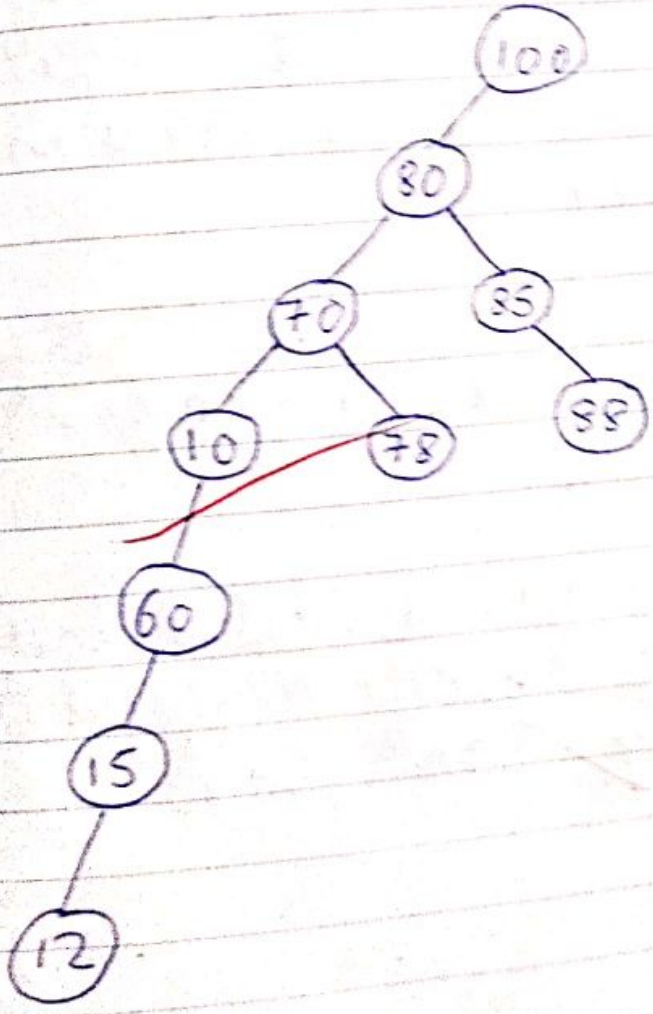
Output:

```
Name: Gaurav  singh
Roll No.:1732
[42, 45, 54, 55, 66, 89, 67, 80, 100]
```

Practical-11

Aim: Binary Tree and Traversal

Theory: A Binary tree is a special type of tree in which every node or vertex has either no child or one child node or two child nodes.
A binary tree is an important class of a tree data structure in which a node can have at most two children.



Diagramatic Representation of Binary Search Tree

```
#Binary Tree and Traversal
print("Name:gaurav Singh \nRoll No.:1732")
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)
                                                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)
```

Traversal is a process to visit all the nodes of a tree and may print their values too. There are 3 ways which we use to traverse a tree

- ↳ INORDER
- ↳ PREORDER
- ↳ POSTORDER

IN-ORDER: The left-subtree is visited 1st then the root & later the right subtree we should always remember that every node may represent a subtree itself. output produced is sorted key values in ASCENDING order.

PRE-ORDER: The root node is visited 1st then the left subtree and finally the right subtree.

POST-ORDER: The root node is visited last left subtree, then the right subtree and finally root node.

```
        def postorder(self,start):
                if start!=None:
                        self.inorder(start.l)
                        self.inorder(start.r)
                        print(start.data)
T=Tree()
T.add(100)
T.add(80)
T.add(70)
T.add(85)
T.add(10)
T.add(78)
T.add(60)
T.add(88)
T.add(15)
T.add(12)
print("Preorder")
T.preorder(T.root)
print("Inorder")
T.inorder(T.root)
print("Postorder")
T.postorder(T.root)
```

```
Output:
Name:gaurav Singh
Roll No.:1732
80 added on left of 100
70 added on left of 80
85 added on right of 80
10 added on left of 70
78 added on right of 70
60 added on right of 10
88 added on right of 85
15 added on left of 60
12 added on left of 15
Preorder
100
80
70
10
60
15
12
78
85
88
Inorder
10
12
15
60
70
78
80
85
88
100
Postorder
10
12
15
60
70
78
80
85
88
100
```

Practical - 12

Aim: Merge Sort.

Theory: Merge sort is a sorting technique based on divide and conquer technique with worst-case time complexity being $O(n \log n)$ it one of the most respected algorithms

- Merge sort first divides the array into equal halves, and then combines them in a sorted manner.

It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging 2 halves. The merge(arr, l, m, r) is key process that assumes that arr[l....m] and arr[m+1....r] are sorted and merges the two sorted sub-arrays into one.

```
#Merge Sort
print("Name:gaurav Singh \nRoll No.:1732")
def sort(arr,l,m,r):
        n1=m-l+1
        n2=r-m
        L=[0]*(n1)
        R=[0]*(n2)
        for i in range(0,n1):
                L[i]=arr[l+i]
        for j in range(0,n2):
                R[j]=arr[m+1+j]
        i=0
        j=0
        k=l
        while i<n1 and j<n2:
                if L[i]<=R[j]:
                        arr[k]=L[i]
                        i+=l
                        j+=l
                        k+=l
                else:
                        arr[k]=R[j]
                        j+=1
                        k+=1
                while i<n1:
                        arr[k]=L[i]
                        i+=1
                        k+=1
                while j<n2:
                        arr[k]=R[j]
                        j+=1
                        k+=1
def mergesort(arr,l,r):
        if l<r:
                m=int((l+(r-1))/2)
                mergesort(arr,l,m)
                mergesort(arr,m+1,r)
                sort(arr,l,m,r)
arr=[12,23,34,56,78,45,86,98,42]
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

Output:

```
Name:gaurav Singh
Roll No.:1732
[12, 23, 34, 56, 42, 45, 78, 86, 98]
```