```
n=0
a=[2,8,10,45,20,33]
print(a)
search=input("enter no to be searched:")
for i in range(len(a)):
  if (search==a[i]):
    print "no found at",i
    n=1
    break
if(n!=1):
  print("entered no not found")
print("Rohit Gupta\n1728")
```

OUTPUT:

```
>>>[2, 8, 10, 45, 20, 33]
enter no to be searched:2
no found at 0
Rohit Gupta
1728

>>>[2, 8, 10, 45, 20, 33]
enter no to be searched:20
no found at 4
Rohit Gupta
1728

>>>[2, 8, 10, 45, 20, 33]
enter no to be searched:35
entered no not found
Rohit Gupta
1728
```


DATA STRUCTURE {SEMESTER - II}

PRACTICAL NO. 1

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

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 :- 1) Sorted
2) 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.

Unsorted Linear Search :-

1.) The data is entered in random manner.
2.) User needs to specify the ~~nn~~ element to be searched in the entered list.
3.) Check the condition that whether the entered number matches if it matches then display the location and also the number.
4.) Increment the initial value i as +1 as data is stored from location zero.
5.) If all elements are checked one by one and element not found then prompt message number not found.

```
n=0
a=[1,23,45,67,89,9,87]
print(a)
search=input("enter no to be searched:")
if ((search<a[0]) or (search>a[len(a)-1])):
  print("enterd no doesn't exist")
else:
  for i in range(len(a)):
    if (search==a[i]):
      print "no found at",i
      n=1
      break
  if(n!=1):
    print("entered no not found")
print("Rohit Gupta\n1728")
```

OUTPUT:

```
>>>[1, 23, 45, 67, 89, 9, 87]
enter no to be searched:45
no found at 2
Rohit Gupta
1728

>>>[1, 23, 45, 67, 89, 9, 87]
enter no to be searched:99
entered no doesn't exist
Rohit Gupta
1728

>>>[1, 23, 45, 67, 89, 9, 87]
enter no to be searched:50
entered no not found
Rohit Gupta
1728
```


Practical No. 2.

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

Theory :- Searching and Sorting are two 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 i.e. all what is meant by searching through 'sorted' that is will arrange the data.

Sorted Linear Search :-

1) The user is supposed to enter data in stored manner.
2) User has to given an element for searching through sorted list.
3) If element is found display with an updation as value is stored from location '0'.
4) If data or element not found print the same.
5) In sorted order list of 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 is not in the list.

source code:

```
print("name:Rohit gupta\nroll no:1728")
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/Rohit gupta/Documents/ds practical 3.py ==========
name:Rohit gupta
roll no:1728
[3, 4, 21, 24, 25, 29, 64]
enter number to be searched from the list:64
('number found at location:', 7)
>>>
========== RESTART: C:/Users/Rohit gupta/Documents/ds practical 3.py ==========
name:Rohit gupta
roll no:1728
[3, 4, 21, 24, 25, 29, 64]
enter number to be searched from the list:97
number not in range
>>>
========== RESTART: C:/Users/Rohit gupta/Documents/ds practical 3.py ==========
name:Rohit gupta
roll no:1728
[3, 4, 21, 24, 25, 29, 64]
enter number to be searched from the list:7
number not in given list
>>>
```


PRACTICAL NO. 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 linear 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 or two direction.

Specifically, the key value is compared to the middle element of the array. If the key value is less than or greater than the 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 No. 4

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

Theory :- Sorting is type in which any random data is stored i.e. arranged in ascending or descending order.

Bubble sort sometimes referred to as sinking sort. It is a simple sorting algorithm that repeatedly steps through the lists, compares adjacent elements and swaps them if they are in wrong orders.

The pass through the list is repeated until the list is sorted. The algorithm which is a comparison sort is named for the way smaller or larger elements "bubble" to the top of the list.

Although the algorithm is simple, it is too slow as compares one elements and checks, and if conditional fails then only swap otherwise goes on.

source code:

```
print("name:Rohit gupta \n roll no:1728")
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\Rohit gupta\Documents\ds practical 4.py ============
name:Rohit gupta
 roll no:1728
[3, 5, 6, 7, 8, 9]
>>>
```

SA

SOURCE CODE:

```
print("Rohit Gupta \n 1728")
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()
```

Output :-

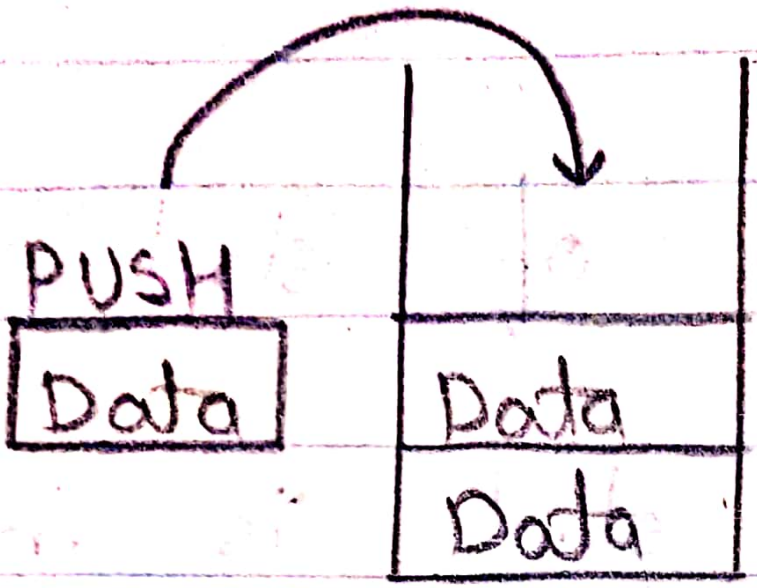
```
Rohit Gupta
1728
data= 70
data= 60
data= 50
data= 40
data= 30
data= 20
data= 10
stack empty
```


PRACTICAL NO. 5

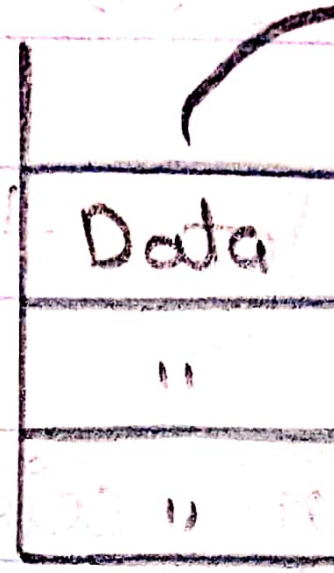
Aim :- To demonstrate the use of stack.

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 FILO (First In Last Out).
There Basic operations are performed in the stack.

- PUSH :- Adds an items in the stack. If the stack is full then it is said to be over flow condition.
- 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.
- Peek or Top :- Returns top element of stack.
- is Empty :- Returns true if stack is empty else false.



Stack



Stack

Last-in-first-

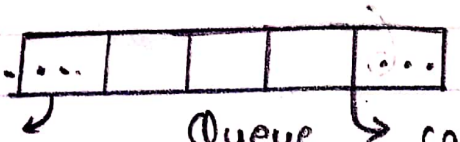
PRACTICAL NO. 6

Aim :- To demonstrate Queue add and delete.

Theory :- 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
Front points to the beginning of the queue and Rear points to the end of the queue.
Queue follows the FIFO (First in First Out) structure. According to its FIFO structure, element inserted first will also be removed first. 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.

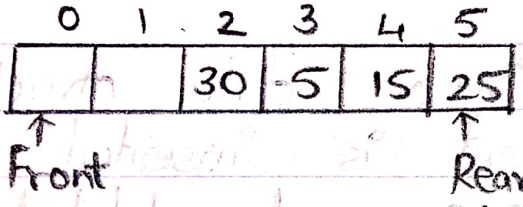
- enqueue () can be termed as add () in queue i.e. adding a element in queue.
- Dequeue () can be termed as delete or Remove. i.e. deleting or removing of element.
- front is used to get the front data item from a queue.
- Rear is used to get the last item from a queue.

#



Queue can have ends on both side

#



Front = 2
Rear = 5

SOURCE CODE:

```
Print("Rohit Gupta 1728")
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
      if self.f<self.r:
        print(self.l[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)
```

OUTPUT :

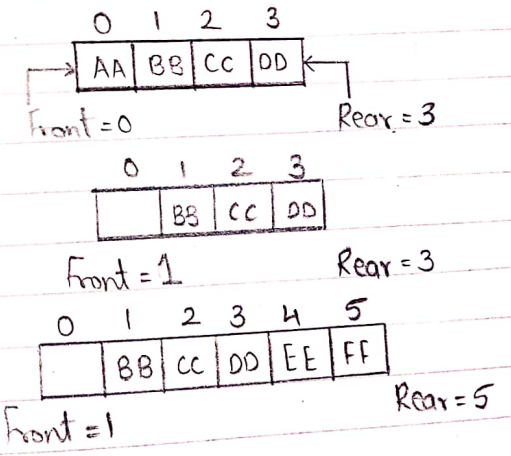
```
Rohit Gupta 1728

data added: 44
data added: 55
data added: 66
data added: 77
data added: 88
data added: 99
data removed: 44
```


PRACTICAL NO. 7

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

Theory :- 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. To overcome this limitation we can implement queue as circular queue. In circular queue we go on adding the element to the queue and reach the end of the array. The next element is stored in the first slot of array.

Example :-



Front = 2 Rear = 0

```
#linked list
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("Rohit Gupta\n1728")
OUTPUT:
>>>20
30
40
50
60
70
80
Rohit Gupta
1728
```

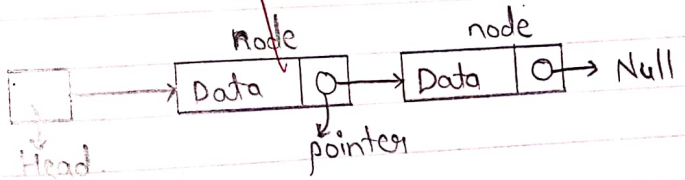

PRACTICAL NO. 8

Aim :- To demonstrate the use of Linked list in data structure.

Theory :- A linked list is a sequence of data structures. 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 :

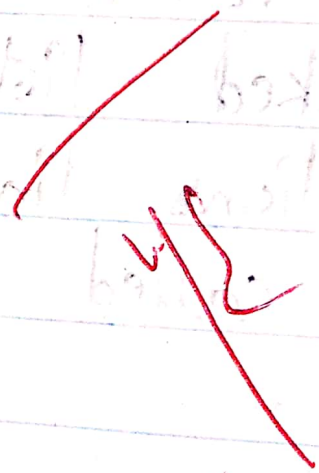


Types of linked list :-

→ Simple.
→ Doubly.
→ Circular.

Basic Operation :

→ Insertion
→ Deletion
→ Display
→ Search
→ Delete



```
#Postfix Evaluation

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="13 5 3 7 - + *"
r=evaluate(s)
print("The evaluated value is:",r)

print("Rohit Gupta \n 1728")
```

OUTPUT:

```
The evaluated value is: 13
Rohit Gupta
 1728
```


PRACTICAL NO. 9

Aim :- To evaluate postfix expression using stack.

Theory :- Stack is an (ADT) and works on LIFO (Last_in_first_out) ie. PUSH & POP operations. A postfix expression is a collection of operators and operands in which the operators is placed after the operands.

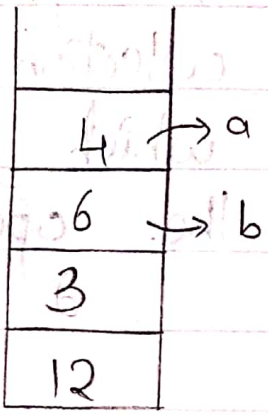
Steps to be followed :-

1.) Read all the symbols one by one from left to right in the given postfix expression.
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 oprands in two different variables (operand 1 & operands 2). Then perform reading symbol operation using operand 1 & operand 2 and PUSH result back on to the stack.
4.) Finally! Perform a pop operation and display the popped value as final result.

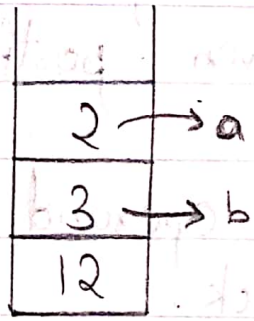
Value of postfix expression :-

S = 12 3 6 4 − + *

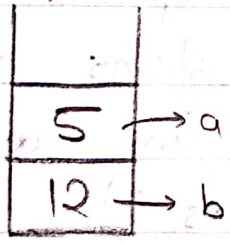
Stack :



b - a = 6 - 4 = 2 // Store again in stack



b + a = 3 + 2 = 5 // Store again in stack.



b * a = 12 * 5 = 60

```
print("Name:Rohit Gupta \nRoll No.:1728")

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:Rohit Gupta
Roll No.:1728
[42, 45, 54, 55, 66, 89, 67, 80, 100]
```


PRACTICAL NO. 10

Aim :- Sorting the numbers using Quick Sort..

Theory :- QuickSort is a fast sorting algorithm, which is used not only for educational purposes, but widely applied in practice. On the average it has O(n log n) complexity, making quicksort suitable for sorting big data volumes. The idea of the algorithm is quite simple and once you realize it, you can write quicksort as fast as bubble sort

Algorithm :-

The divide and conquer strategy is used in quicksort. Below the recursion step is described.

1) **Choose a pivot value :**

We take the value of the middle element as pivot value but it can be any value, which is in range of sorted values, even if it doesn't present in the array.

2) **Partition :-** Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.

3.) Sort both parts:- Apply quicksort algorithm recursively to the left and right parts

```
print("Name: Rohit Gupta \nRoll No.: 1728")
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:
            nn=Node(val)
            h=self.root
            while True:
                if nn.data<h.data:
                    if h.l!=None:
                        h=h.l
                    else:
                        h.l=nn
                        print(nn.data,"addedon left of",h.data)
                        break
                else:
                    if h.r!=None:
                        h=h.r
                    else:
                        h.r=nn
                        print(nn.data,"data 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)
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)
```


PRACTICAL NO. 11

Aim :- To demonstrate the use of binary tree and traversal.

Theory :- Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree :-

1.) In-order Traversal
2.) Pre-order Traversal
3.) Post-order Traversal.

1.) In-order Traversal :-

In this traversal method, the left subtree is visited first, then the root and later the right subtree. We should always remember that every node may represent a subtree itself.

Algorithm :-

Until all nodes are traversed :-
Step 1 :- Recursively traverse left subtree.
Step 2 :- Visit root node.
Step 3 :- Recursively traverse right subtree.

2) Pre-order Traversal :-
In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Algorithm :-
Until all nodes are traversed :-
Step 1 :- Visit root node.
Step 2 :- Recursively traverse left subtree.
Step 3 :- Recursively traverse right subtree.

3) Post-order Traversal :-
In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

Algorithm :-
Until all nodes are traversed :-
Step 1 :- Recursively traverse left subtree.
Step 2 :- Recursively traverse right subtree.
Step 3 :- Visit root node.

```
print("inorder")
t.inorder(t.root)
print("postorder")
t.postorder(t.root)
```

```
OUTPUT
Name: Rohit Gupta
Roll No.: 1728
80 addedon left of 100
70 addedon left of 80
85 data on right of 80
10 addedon left of 70
78 data on right of 70
60 data on right of 10
88 data on right of 85
15 addedon left of 60
12 addedon 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
12
15
60
10
78
70
88
85
80
100
```

```
print("Name:Rohit Gupta \nRoll No.:1728")
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+=1
                    j+=1
                    k+=1
            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:Rohit Gupta
Roll No.:1728
[12, 23, 34, 56, 42, 45, 78, 86, 98]
```


Practical No. 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 algorithm.

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-array into one.