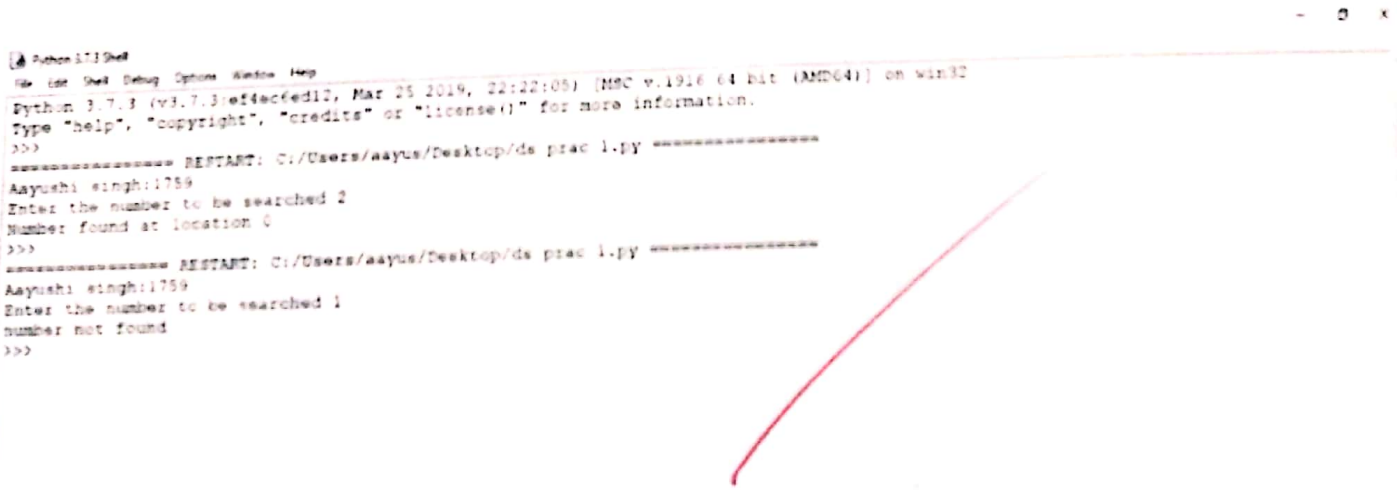
Aim:-Implement linear search to find an item in a list

Program:-

```
print("Aayushi singh:1759")
found=False
A=[2,4,7,68,78,56,80,90,34,20]
search=int(input("Enter the number to be searched"))
for i in range(len(A)):
  if (search==A[i]):
print("Number found at location",i)
    found=True
    break;
if(found==False):
print("number not found")
```



Scanned with CamScanner

PRACTICAL No - 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.

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. That is why it calls unsorted linear search.

1. The data is entered in random manner
2. User needs to specify the element to be searched
3. Check the condition whether the entered numbers matches, if the entered number

Scanned with CamScanner

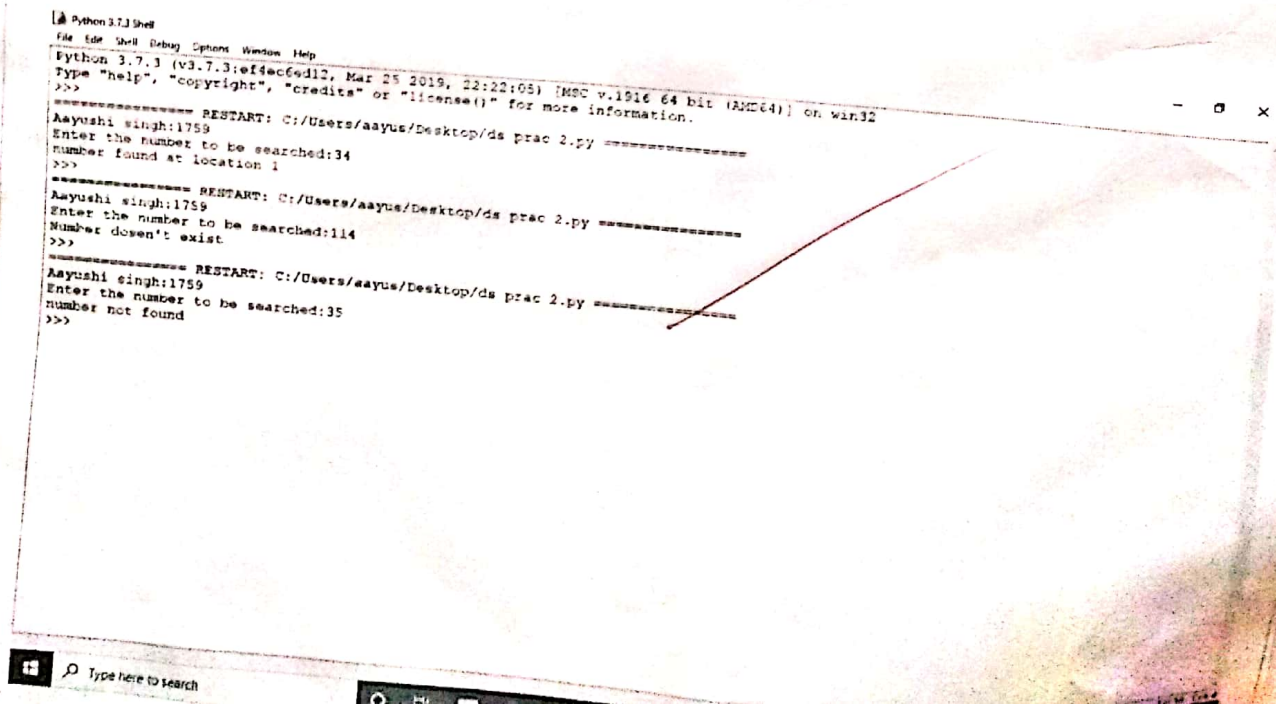
matches, if the entered number matches then display the location plus increment by 1 as data is stored from location zero.

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

Aim:-Implement binary search to find an item in an ordered list

Program:-

```
print("Aayushi singh:1759")
found=False
A=[27,34,67,98,113]
search=int(input("Enter the number to be searched:"))
if(search<A[0] or search>A[len(A)-1]):
print("Number dosen't exist")
else:
   for i in range(len(A)):
      if (search==A[i]):
print("number found at location",i)
         found=True
         break;
   if(found==False):
print("number not found")
```



Scanned with CamScanner

PRACTICAL NO -2

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

Theory:- SEARCHING & SORTING are different models of data structure.

SORTING :- To basically sort the input data in ascending or it in the descending order.

SEARCHING : To search & display the desired element.

LINEAR SORTED SEARCH :-
The data is arranged in ascending to descending or descending to ascending. That is all what is meant by searching through 'sorted' that is the well arranged data.

1. The work is supposed to enter the data in sorted manner.

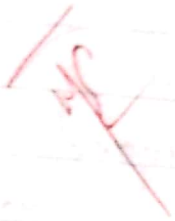
2. User has to give an element for searching through sorted list.

3. If element is found display with or updown as value is stored from location zero.

Scanned with CamScanner

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 not in the list.



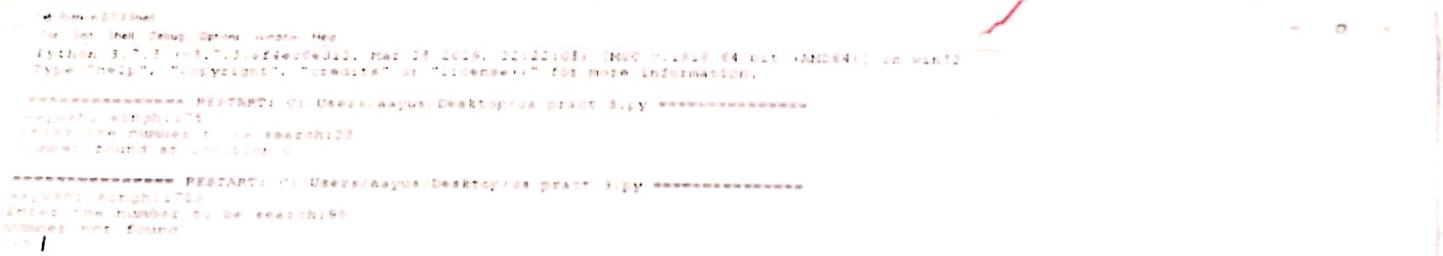
Scanned with CamScanner

Practical -3

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

Program:

```
print("Aayushi singh:1759")
A=[23,45,67,78,90,100]
search=int(input("Enter the number to be search:"))
l=0
r=len(A)-1
while True:
i=(l+r)//2
  if(l>r):
print("Number not found")
    break
  if(search==A[i]):
print("Number is found at location",i)
    break
 else:
  if(search<A[i]):
    r=i-1
 else:
 r=i+1
```



Scanned with CamScanner

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 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 from 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 and perform binary search.

Scanned with CamScanner

PRACTICAL No-5

Aim:- To demonstrate the use of stack

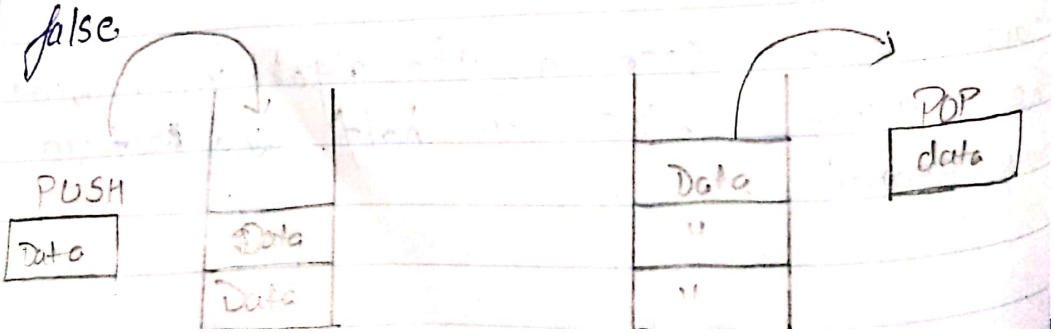
Theory:- In computer science, a stack is an abs
data type that serves as a collection of
elements with two principal operations push,
which adds an element to the collection
and pop, which removes the most recent
added element that was not yet removed.
The order may be FIFO (First In First Out)
FILO (First In Last Out) Three Basic opera
are performed in the stack.

•Push: Adds an item in the stack. If the sta
is full then it is said to be over flow cond

•POP: Removes an item from the stack. The items
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



Scanned with CamScanner

Practical -4

```
## Stack ##
print("Aayushi singh:1759")
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 is empty")
        else:
            k=self.l[self.tos]
            print("data=",k)
            self.tos=self.tos-1
s=stack()
s.push(10)
s.push(20)
```

Scanned with CamScanner

Practical - 5

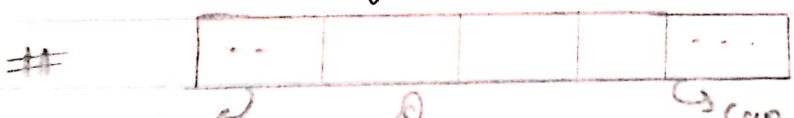
```
## Queue add and Delete ##
print("Aayushi Singh:1759")
class Queue:
  global r
  global f
  def __init__(self):
    self.r=0
    self.f=0
    self.l=[0,0,0,0,0,0]
  def add(self,data):
    n=len(self.l)
    if self.r<n-1:
      self.l[self.r]=data
      self.r=self.r+1
    else:
      print("Queue 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)
```

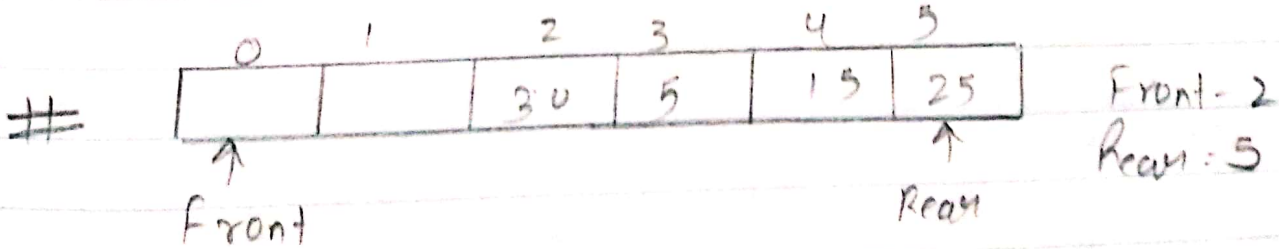
Scanned with CamScanner

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 others 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.



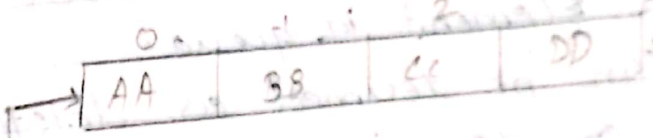


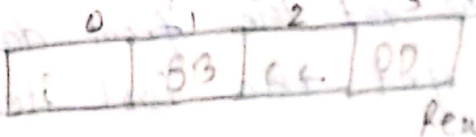
58 PRACTICAL No -7

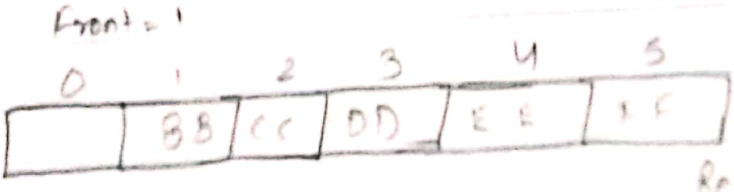
Aim :- To demonstrate the use of circular queue in data-structure.

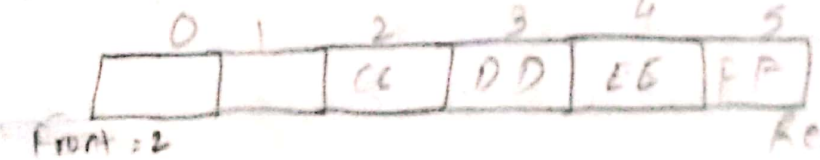
Theory :- The queue that we implement using an array suffer from one limitation. In that implementation there is a possibility that the queue is reported as full, even though in actuality there might be empty slots at the beginning of the queue. To overcome this limitation we can implement queue as circular 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 the array.

Example :









Scanned with CamScanner

Practical-6

```
#(circular queue)
print("Aayushi singh:1759")
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])
```

Scanned with CamScanner

Practical 7

```
print("Aayushi Singh:1759")
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:
```

Scanned with CamScanner

PRACTICAL No-8

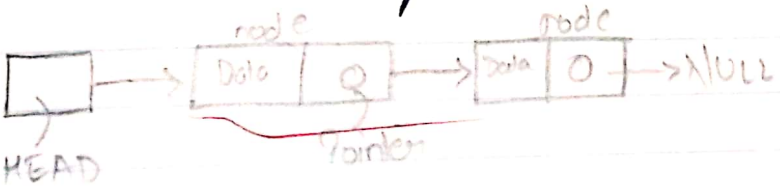
AIM: To demonstrate the use of Linked list in data structure

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 Operations

- Insertion
- Deletion

Scanned with CamScanner

- Display
- Search
- Delete.

Scanned with CamScanner

```
    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(30)
start.addB(20)
start.addB(10)
start.display()
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Aayushi Singh:1758
10
20
30
50
>>>
```

Scanned with CamScanner

```
print("Aayushi singh:1759")
def evalute(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))
```

Scanned with CamScanner

PRACTICAL - 9

AIM :- To evaluate Postfix expression 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 operator is placed after the operands.

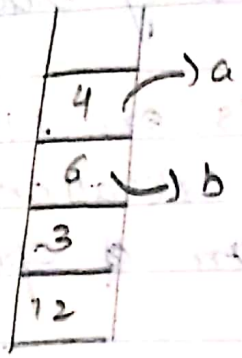
Steps to be followed :-
1. Read all the symbols one by one from left to right in the given postfix expression.
2. If the reading symbol is operand the push it on to the stack.
3. If the reading symbol is operator (+, -, *, /, etc) then perform Two pop operation and store the two popped operands in two variables (operand 1 & operand 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 :

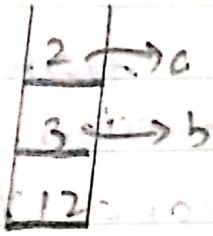
9 = 12 3 6 4 - + *

Scanned with CamScanner

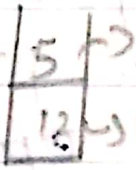
Stack



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



b + a = 3 + 2 = 5 // store result in



b * a = 12 * 5 = 60

```
    return stack.pop()

s="8 7 6 + *"

r=evalute(s)

print("The Evaluted Value is:",r)
```

```
Python 3.8.3 (v3.8.3:6f8c8320e9, Feb 19 2020, 23:45:06) [MSC v.1916 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART: ==========
>>>
Recent stack:756
The Evaluted Value is: 104
>>>
```

Scanned with CamScanner

SA

Practical 8

```
print("Aayushi Singh:1759")
a=[10,8,9,5,3,1,2]
print(a)
for i in range(len(a)-1):
  for j in range(len(a)-1-i):
    if(a[j]>a[j+1]):
      t=a[j]
      a[j]=a[j+1]
      a[j+1]=t
print(a)
```

```
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/Jai/Desktop/New folder/pi.py ==========
Aayushi Singh:1759
[10, 8, 9, 5, 3, 1, 2]
[1, 2, 3, 5, 8, 9, 10]
>>>
```

Scanned with CamScanner

PRACTICAL No:-10

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

Theory:- SORTING is type in which any random data is sorted i.e arranged in ascending or descending order BUBBLE sort sometimes referred to as sinking sort. Is a simple sorting algorithm that repeatedly steps through the lists, compare adjacents elements and swap them if they are wrong order. The pass through the list is repeated until the list is on sorted, The algorithm which is a comparision sort is named for the way smaller or larger elements "bubble" to the top of the list.

Algorithm is simple, it is too slow as it compares one element checks if condition fails the only swaps otherwise goes on.

Example:-

1st pass

(5, 14, 28) (15 4 28) Here algorithm compares the first two elements and swaps

(5, 14, 8) ⊙

Second. —> (4528). Here Object takes time.

Sin : 5>0

(5, 14, 28) => (14528) swap sinu - 5>4

(14 528) => (14258) swap sinu 5>2

Scanned with CamScanner

8.D

(14528) → (14285) swap ~~seking~~ Sin 4>2
(12458) → (1245-8)

third part
[2, 4, 5, 8] It checks 8. and gives the
(1, 2458 → (1258)

Practical 11

```
print("Aayushi Singh 1759")
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]
```

Scanned with CamScanner

PRACTICAL - 11

Aim:- To evaluate 'ie' to sort the given data in Quick sort.

Theory :- Quicksort is an efficients posting algorithm. It type of Divide and conquers algorithm. It picks an element as pivot and partition the given array around the picked pivot. There are many difficult ways.

1) Always pick first element as pivot.

2) Always pick last elements as pivot.

3) Pick a random element as pivot

4) Pick median as pivot. The key process in quicksor is para. Target of partition is give an array and an element 'x' of array as pivot, put x at its correct position in sorted sorted array and put all smaller elements than x) before x, and put all greata elements greater than x after x all. This should be done in linear time.

Scanned with CamScanner

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

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Agyekum Singh...708
[42, 45, 54, 55, 66, 67, 80, 89, 100]
>>>
```

Scanned with CamScanner

Practical:-12

```
## Binary Tree and Traversal ##
print("Aayushi singh:1759")
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.ll=None:
            h=h.l
          else:
            h.l=newnode
            print(newnode.data,"added on left of",h.data)
            break
    else:
```

Scanned with CamScanner

PRACTICAL NO:-12

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.

#1

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

INORDER: The left-subtree is visited 1st then root and later the right sub tree. We should always remember that every node may represent a subtree itself. output produced is sorted key values in ASCENDING ORDER

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

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

Scanned with CamScanner

```
            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.inorder(start.l)
            self.inorder(start.r)
            print(start.data)
T=Tree()
T.add(300)
T.add(80)
T.add(70)
T.add(85)
T.add(10)
T.add(79)
T.add(64)
T.add(88)
T.add(15
```

Scanned with CamScanner

```
T.add(13)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
```

```
========== RESTART: C:/Users/aayus/Desktop/jaipractical1.py ==========
Aayushi singh:1759
80 added on left of 300
70 added on left of 80
85 added on right of 80
10 added on left of 70
79 added on right of 70
64 added on right of 10
88 added on right of 85
15 added on left of 64
13 added on left of 15
preorder
300
80
70
10
64
15
13
79
85
88
inorder
10
13
15
64
70
70
79
80
85
88
300
postorder
10
13
15
64
70
79
80
85
88
300
>>>
```

Scanned with CamScanner

PRACTICAL No-15

Merge Sort :-

theory :- Merge Sort is a sorting technique based on divide and conquer techniques 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 values 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.

Scanned with CamScanner

```
print("Aayushi Singh:1759")
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
  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:
```

Scanned with CamScanner

```
        m=int((l+(r-1))/2)
        mergesort(arr,l,m)
        mergesort(arr,m+1,r)
        sort(arr,l,m,r)
arr=[17,26,35,56,78,45,87,99,44]
print(arr)
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

```
============== RESTART: C:/Users/aayus/Desktop/jaipractical2.py ==============
Aayushi Singh:1759
[17, 26, 35, 56, 78, 45, 87, 99, 44]
[17, 26, 56, 56, 44, 45, 78, 87, 99]
>>>
```

Scanned with CamScanner

PRACTICAL No - 14

Aim:- To sort Given random data by using selection sort.

Theory:-

Selection sort is a simple sorting algorithm. This sorting algorithm is an inplace 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.

Scanned with CamScanner

```
#Selection sort#
print("Aayushi Singh")
a=[3,4,5,6,7,8,9,10]
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(a)
```

```
============== RESTART: C:/Users/aayus/Desktop/jaipractical3.py ==============
Aayushi Singh
[3, 4, 5, 6, 7, 8, 9, 10]
[3, 4, 5, 6, 7, 8, 9, 10]
>>>
```

Scanned with CamScanner