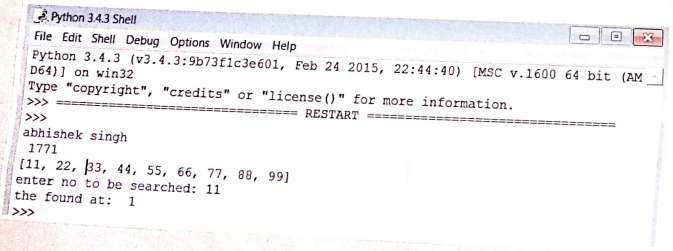
Program 1.

```
print("abhisheksingh \n 1771")
a=[11,22,33,44,55,66,77,88,99]
j=0
print(a)
search=int(input("enter no to be searched: "))
for i in range(len(a)):
if(search==a[i]):
print("the found at: ",i+1)
        j=1
break
if(j==0):
print ("number not found!")
```

output.



PRACTICAL No: 1

AIM :- To Search a Number from the list using linear search.

THEORY :- The process of retrieving or finding a set of Search particular records is called Searching. There are two types of Search
- Linear Search
- Binary Search

The linear Search is further classified as
* Sorted * Unsorted.

Here we look on UNSORTED linear Search.
Linear Search is also know as Sequential Search it is the process that each Emeny Sequentially until the Desire element should found. if Elements to be searched are not arrange in ascending order. They are arranged in Random manner That is what Called unsorted Linear Search.

es

Unsorted Linear Search,

↳ The data Entered in Random manner.
↳ user needs to specify the Elements to be search in Entered list
↳ Check The Condition that the Entered Number matched. Then. Display the location & increment 1 as Data is stored from location Zero.
↳ if all Elements are checked one by one & Element - not found then prompt message number not found.

Program2.

```
print("abhisheksingh \n1771")
a=[3,4,21,24,25,29,64]
j=0
print(a)
search=int(input("enter the number to be searched: "))
if((search<a[0]) or (search>a[6])):
print("number doesnt exist")
else:
for i in range (len(a)):
if(search==a[i]):
print("number found at: ",i+1)
        j=1
break
if(j==0):
print("number not found!")
```

output

SORTED LINEAR SEARCH.

- The Loaner is Supposed to Enter data is Sorted manner
- User has to given an Element for Searching through Sorted list
- if Element is found display with an updation as value is Sorted from location '0'
- if data or Element is not found print the Same
- In Sorted order list of Element which Can we Can check the Condition that whether the Entered number lies from the Starting point till the last Element if not then without any processing will Can Say number not in the list

Source code:

```
print("Abhishek")
a=[3,12,23,36,49,66,78]
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:

Case1:
Abhishek
1706
[3, 12, 23, 36, 49, 66, 78]
Enter number to be searched from the list:36
Number found at location: 4
Case2:
Abhishek
1706
[3, 12, 23, 36, 49, 66, 78]
Enter number to be searched from the list:99
Number not in RANGE!
Case3:
Abhishek
1706
[3, 12, 23, 36, 49, 66, 78]
Enter number to be searched from the list:11
Number not in given list!

PRACTICAL – 3

AIM: To search a number from the given Sorted list using Binary search.

THORY :- 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 either in ascending order or descending order.

At each step of the algorithm, a comparison is made & the procedure branches into two directories

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 value element, the algorithm of the array to continue searching in because the array is sorted. This process is repeated on progressively smaller segment of the array until the value is located.

Source code:

```
print("Abhishek jha")
a=[11,37,10,61,12,8]
print("Before BUBBLE SORT elemets list: \n ",a)
for passes in range (len(a)-1):
  for compare in range (len(a)-1-passes):      if(a[compare]>a[compare+1]):
        temp=a[compare]
        a[compare]=a[compare+1]
        a[compare+1]=temp
print("After BUBBLE SORT elemnts list: \n",a)
```

Output:

```
Abhishek jha
1740
Before BUBBLE SORT elemets list:
 [11, 37, 10, 61, 12, 8]
After BUBBLE SORT elemnts list:
[8, 10, 11, 12, 37, 61]
```

Example:

First pass

(5 1 4 2 8) → (1 5 4 2 8) Here algorithm Compa
the first two Element & swaps since 5>
(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 Elem
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 sin 4>2
(1 2 4 5 8) → (1 2 4 5 8)

Third pass

(1 2 4 5 8) It Checks & gives the sorted
data, in sorted order.

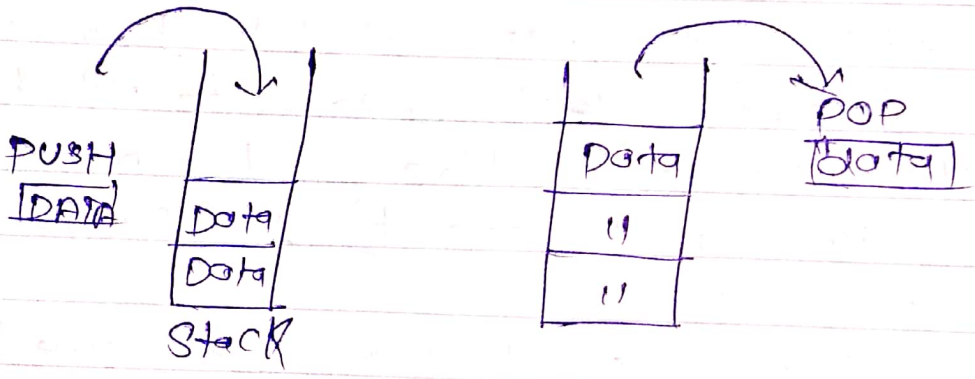
```
## Stack ##
print("abhishek")
class stack:
    global tos
    def __init__(self):
        self.l=[0,0,0,0,0,0,0]
        self.tos=-1
    def push(self,data):

        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
Abhishek
Stack is full
Data= 70
Data= 60
Data= 50
Data= 40
Data= 30
Data= 20
Data= 10
Stack empty
```

- Peek or Top : Returns top Element o Stack.
- Is Empty : Returns true if Stack is Empty else false.



Last in first out

```
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)
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()
```

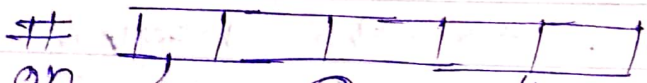

PRACTICAL NO:- 6

AIM :- To demonstrate Queue add & delete

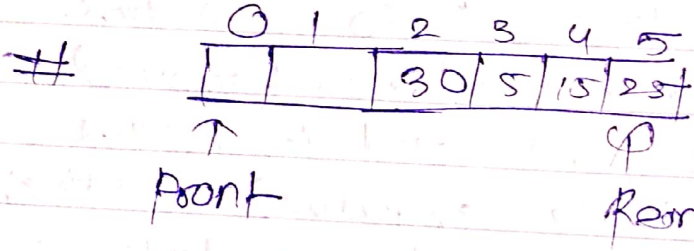
THEORY :- Queue is a linear data structure where the first element is inserted from one end called REAR & deleted from other end called as FRONT.
Front point to the begining of the Queue Follows as FIFA (First in. First out) structure
According to its FIFO structure Element Inserted first will also removed first
In a queue, one end is always used to insert data (enqueue) & the other is used to delete data (dequeue)
Because queue is opened at both of its Ends
Enqueue() can be termed as adds i.e 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

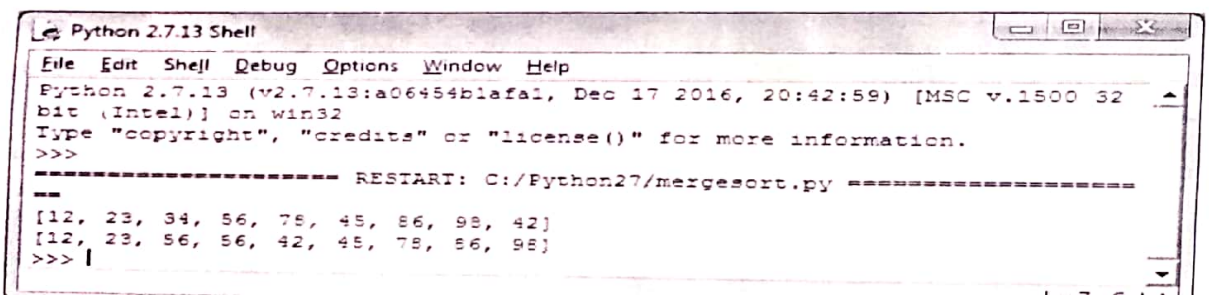


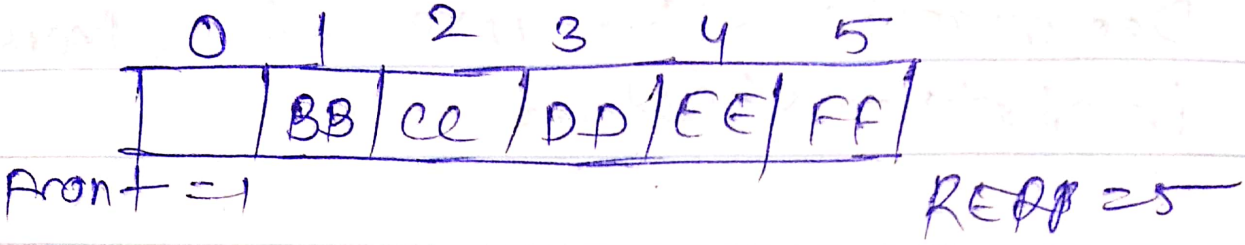
on Both side Queue can have End.

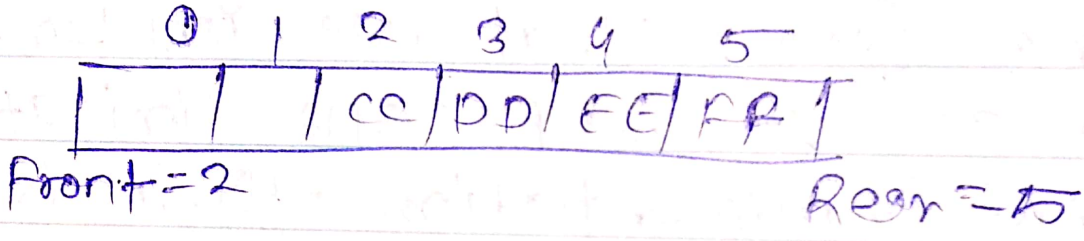


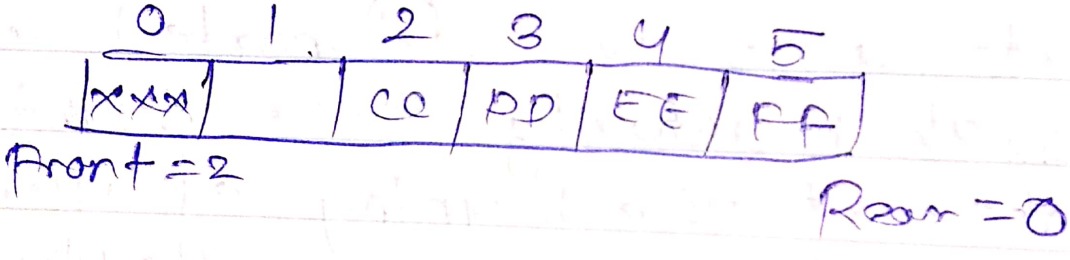
Front = 2
Rear = 5.

```
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
defmergesort(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]
print(arr)
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```









```
print("Abhishek")
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="12 3 6 4 - + *"
r=evaluate(s)
print("The evaluated value is:",r)
```

OUTPUT:

Abhishek

The evaluated value is: 60

PRACTICAL NO:- 8 **43**

Aim:- To Evaluate postfix Expression using stack

THEORY:- Stack is an [ADT] work LIFO ie push & pop operation is placed after the ope

Step to the be followed.

1) Read all the symbols one by one from left to Right to the given postfix Expression.
2) if the Reading Symbol is open then push it on to the stack.
3) if the reading Symbol is operator (+, -, *, /, etc) then perform two operator & store th two pop operator in two different variable (operand 1 & operand 2). The perform reading Symbol operator using operand 1 & operand 2 & push back on to the stack.
4) finally I perform a pop operation and displ the popped value a final result.

```
                    h.l
                else:
                h.l=newnode
                print(newnode.data,"added 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)
    definorder(self,start):
        if start!=None:
        self.inorder(start.l)
        print(start.data)
        self.inorder(start.r)
    defpostorder(self,start):
        if start!=None:
        self.postorder(start.l)
        self.postorder(start.r)
        print(start.data)
```

Output

PRACTICAL NO: 9

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 order or descending BUBBLE Sort Some time Before this as Seeking Sort is a Simple Sorting algorithm that Repeatedly step through the list Compares adjacent Element & Swap them if they are in wrong order the pass through the list is Repeatedly until the list is Repeated until the list is sorted the algorithm which is Comparison Sort is named for way Smaller or larger Element "Bubble" to the top of the list Although the Algorithm is Simple it is to slow as it Compares one Element check if condition fails then only Swaps otherwise goes on

Example!

First pass

(5 1 4 2 8) → (1 5 4 2 8) Here algorithm Compare the first two element & Swap 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 element are already in order (8>5) algorith 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 4>2

(1 2 4 5 8) → (1 2 4 5 8)

Third pass.

(1 2 4 5 8) It Checks & gives the data in sorted order.

```
print("Abhishek")
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.addB(40)
start.addB(30)
start.addB(20)
start.display()
```

OUTPUT:

```
Abhishek
20
30
40
50
60
70
```


PRACTICAL – 10

AIM:- To 'Evaluate' i.e to Sort the given data in Quick Sort

THEORY:- Quicksort is an efficiency sorting Algorithm. Type of a Divide Conquer algorithm. It picks an Element as pivot & partitions the given array around & picked pivot. There are many different version of Quick Sort that pick pivot in different ways.

1) Always pick first Element as pivote
2) Always pick last Element as pivot
3) Pick a random Element as pivot
4) pick median as pivot

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

Practical - 11

Topic : Binary tree

Theory : Binary tree is a tree which suppo
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 : Node which are non-Leaf
nodes

Traversing can be defined as a process
of visiting Every node of the tree Exact
once

preorder : i) visit root nodes
ii) Traverse the left Subtree the left
Subtree in turn might have left
& right Subtree
iii) Traverse the Right Subtree The Right
Subtree inturn might have left & right
Subtree.

Inorder : i) Traverse the left Subtree The left
Subtree inturn might have left & right subt
ii) visit the root node
iii) Traverse the right Subtree
There Subtree inturn might have

postorder:- i) Traverse the left subtree
The ~~return~~ left inturn might have
left & right subtree
ii) Traverse the Right subtree The
right subtree inturn might have
left & Right subtrees
iii) visit the root node

Topic: merge sort

Theory: merge sort use the divide & conquer technique. In merge sort we divide the list into nearly equal sublist each of which one then again dividing into two sublist

- we continue the list procedure until there is only one element in the individual sublist
- One we have individuals elements in the sublist we consider those sublist as sub problems which can be solved since there is only one element in the sublist the sublist is already sorted so now we merge the sub problems
- Now while merging the sub problems we compare the 2 sublist & create the combined list by comparing the Elements of the sublist. After comparison we place the element in their correct position in the original sublists.