```
found=False
a=[6,213,17,1,8,213,54]
search=int(input("Enter a number to be searched: "))
for i in range(len(a)):
  if (search==a[i]):
print("Number found at place ",i+1)
    found=True
    break
if (found==False):
print("Number not found!")
print("Sachin Kanojiya 1744")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Admin/Documents/temp2.py ==========
Enter a number to be searched: 11
Number not found!
Sachin Kanojiya 1744
>>>
========== RESTART: C:/Users/Admin/Documents/temp2.py ==========
Enter a number to be searched: 213
Number found at place  2
Sachin Kanojiya 1744
>>>
```


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 type of search

i) Linear search

ii) Binary search

The linear search further classified, as

a) Sorted

b) Unsorted

Hence we will look on the unsorted linear search. it is 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 assending or dessending in random manor. That is what it called unsorted linear searched.

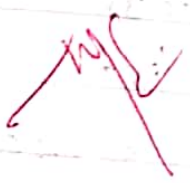
Unsorted linear search

i] The data is entered in random manner

ii] User needs to specify the element to be searched in the entered list

iii] Check the condition that whether the entered number matches if it matches then display the location plus increament 1 as data is stored from location zero.

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



```
found=False
a=[1,4,6,7,8,84,154,213]
search=int(input("Enter a number to be searched: "))
if (search<a[0] or search>a[len(a)-1]):
print("Sorry! Number does not exist!")
else:
  for i in range(len(a)):
    if (search==a[i]):
print("Number found at place ",i+1)
      found=True
      break
  if (found==False):
print("Number does not exist!")
print("Sachin Kanojiya 1744")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:\Users\Admin\Documents\Practice\Sorted_Linear.py =======
Enter a number to be searched: 500
Sorry! Number does not exist!
Sachin Kanojiya 1744
>>>
======= RESTART: C:\Users\Admin\Documents\Practice\Sorted_Linear.py =======
Enter a number to be searched: 84
Number found at place  6
Sachin Kanojiya 1744
>>
====== RESTART: C:\Users\Admin\Documents\Practice\Sorted_Linear.py =======
nter a number to be searched: 80
mber not found!
chin Kanojiya 1744
>
```


Practical = 2

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

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

sorting - To basically sort the input data in ascending or decending manner.

Searching - To search elements and to display the same.

In searching that too in linear sorted search the data is arranged in assending or decending that is all what it meant by Searching. Through 'sorted' that is well arranged data.

Sorted Linear Search

i) The user is supposed to enter data in sorted manner

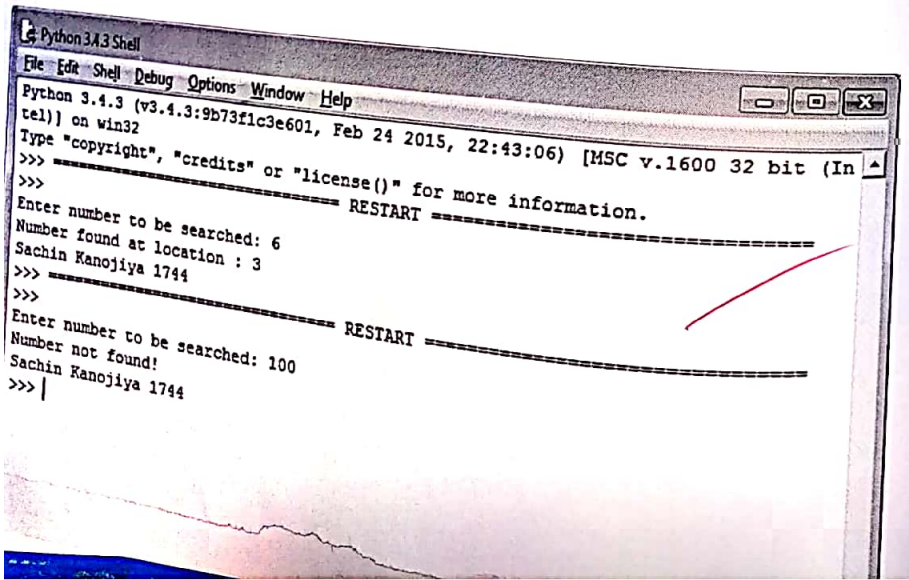
ii) User has to given an element for Searching through sorted list.

iii) If element is found display with an updation as value is stored from location 'o'.

iv) If data or element not found print the same.

v) In sorted order list of elements we can check the entered condition that whether the element is stored from starting point till the last element if not point then we can say Processing in the list. without any number

```
a=[1,4,6,7,8,84,154,213]
search=int(input("Enter number to be searched: "))
l=0
r=len(a)-1
while(True):
  m=(l+r)//2
  if(l>r):
    print("Number not found!")
    break
  if(search==a[m]):
    print("Number found at location :",m+1)
    break
  elif(search<a[m]):
    r=m-1
  else:
    l=m+1
print("Sachin Kanojiya 1744")
```



Practical = 3

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

Theory:

A ~~linear~~ binary search also known as a half interval search is an algorith used in computer science to locate a specified value with in an array for the search to be binary. the array must be sorted in either assending or descending order.

At each step of the algorithm a comparision is made and the procedure branches into one of two directi

Specifically the key value is compared to the ~~key value~~ is middle elemen of the array.

If the key value is less Th or greater Than This middle element the algorithm knowns which half of the array to continues 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 divids the array size in half a binary search will complete successfully in logarithmic time.

```
##stack##
print("sachin 1744")
class stack:
 global tos
 def __init__(self):
  self.l=[0,0,0,0,0,0,0,0]
  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)
```


Practical = 8

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 principle 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 Last out).

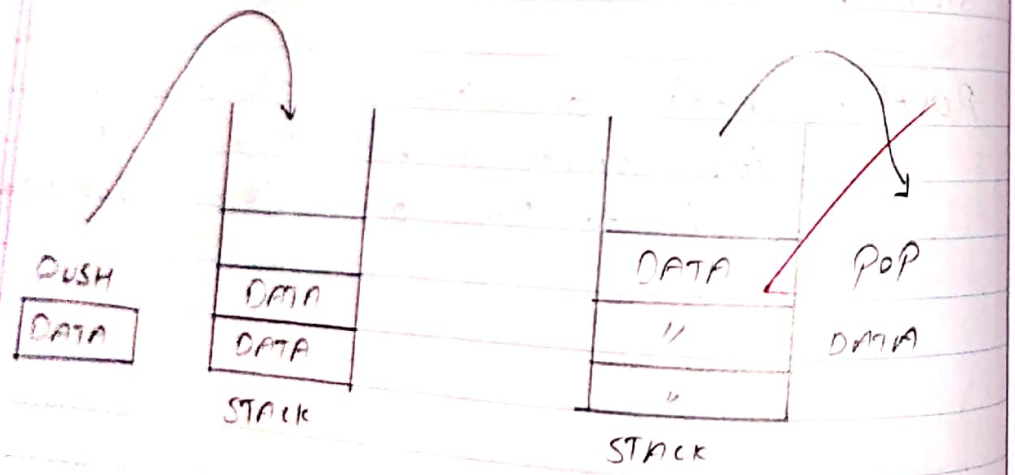
The Basic operations are performed in the stack.

PUSH :- Adds an item in the stack if stack is full then it is said to the over flow condition

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

Peek or Top: Returns top element of stack.

isempty: Return true id stack is empty else false.



Last in First Out

```
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()
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MS
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
sachin 1744
STACK IS FULL
data= 70
data= 60
data= 50
data= 40
data= 30
data= 20
data= 10
STACK EMPTY
```

```
## Queue add and Delete ##
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)
```


Practical 6

Aim :- To determine 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 point 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 and the other is used to delet data because queue is open at both of its ends.

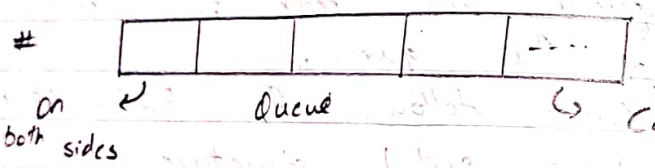
inqueue() can be termed as add() in queue i.e adding a element is queue.

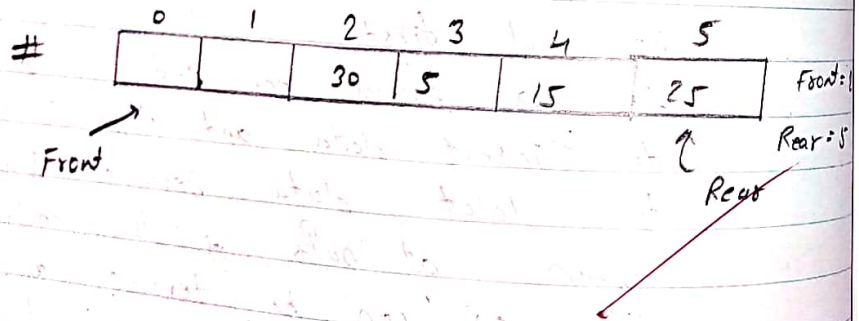
Dequeue() can be termed as delete or Remove.

i.e. deleting or removing of elements.

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

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






```
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:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06)
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART
>>>
Queue is full
30
40
50
60
70
Queue is empty
```

```
#(circular queue)
print("sachin 1744")
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:
```

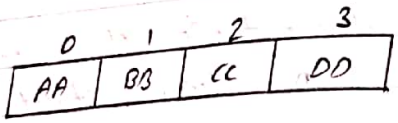

Practical = 7

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

THEORY :-

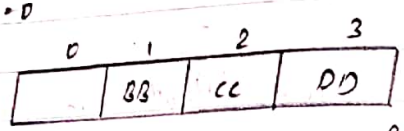
The queue that are we implement using an array suffer from one limitation. In that implementation there is a possibility that the queue is reported as full, even though is actually there might be empty slots at the beginning of the queue. To overcome this limitation we can implement queue as circular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



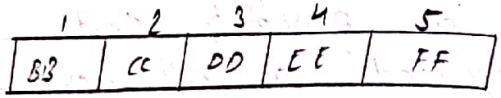
Rear = 3

Front = 0



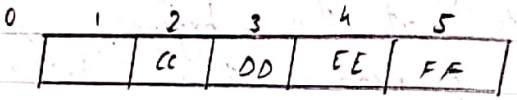
Rear = 3

Front = 1



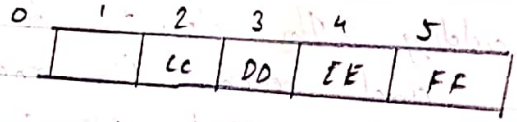
Front = 1

Rear = 5



Front = 2

Rear = 5



Front = 2

Rear = 5

```
        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)
```

```
File Edit Shell Debug Options Windows Help
Python 3.4.3 (v3.4.3:9b73f1e3e601, Feb 24 2015, 22:43:06) [MS
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ===
>>>
sachin 1744
data added: 44
data added: 55
data added: 66
data added: 77
data added: 88
data added: 99
data removed: 44
```

program:

```
class node:
global data
global next
def __init__(self,item):
self.data=item
self.next=None
classlinkedlist:
global s
def __init__(self):
self.s=None
defaddL(self,item):
newnode=node(item)
ifself.s==None:
self.s=newnode
else:
head=self.s
whilehead.next!=None:
head=head.next
head.next=newnode
defaddB(self,item):
newnode=node(item)
ifself.s==None:
self.s=newnode
else:
newnode.next=self.s
```


Practical = 8

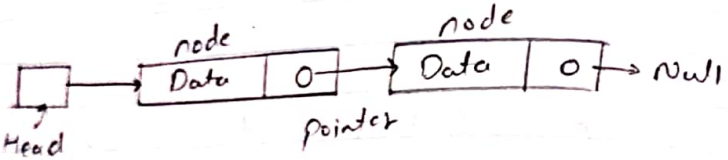
Aim :- To demonstrate the use of Linked list in data structure

THEORY :-

A Linked list is a sequence of data structure Linked list is a sequence of Linked which contains item. Each link contains a connection to another Link.

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

LINKED LIST Representation :



Type of LINKED List :

- Simple
- Doubly
- Circular

Basic operation :

- Insertion
- Deletion
- Display
- SearCh
- Delete

```
self.s=newnode
def display(self):
head=self.s
whilehead.next!=None:
print(head.data)
head=head.next
print(head.data)
start=linkedlist()
start.addL(50)
start.addL(60)
start.addL(70)
start.addL(10)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
print("sachinkanojiya 1744")
```

Output:

```
>>> ========== RESTART ==========
>>>
20
30
40
50
60
70
80
sachin kanojiya 1744
>>> ========== RESTART ==========
>>>
```

Program:

```
# postfixEvalution#


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 :
```


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 operators and operands in which the operator is placed after the operands.

Steps to the lie followed:

1: Read all the symbols one by one from left the 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 poped operand in two different variables (operand 1 & operand 2) Then perform reading symbol operation using operand 1 & 2 and push

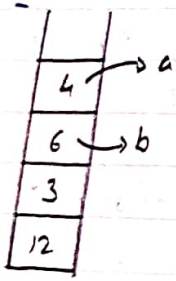
result back on to the stack.

4. Finally I perform a pop operation and display the poped as final result.

Value of postfix expression;

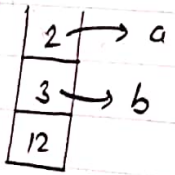
S= 12 3 6 4 - + *

Stack =



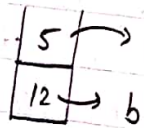
b-a = 6-4 = 2

// store again in stack



b+a = 3+2 = 5

// store result in stack



b*a = 12 * 5 = 60


```
        a=stack.pop()
        b=stack.pop()
stack.append(int(b)/int(a))
returnstack.pop()
s="8 6 9 * + "
r=evaluate(s)
print("The Evaluated value is:",r)
print("sachin 1744")
```

output:

```
>>> ================================ RESTART ================================
>>>
The Evaluated value is: 62
sachin 1744
>>>
```

Program:

```
a=[2,1,6,5,85,4,6,74,8,0,96,4,2,9,8,54,22,35,33,56,11,54,55,56,7,57,99,
4854,85]
print("unshorted number is:\n")
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("shorted number is :\n",a)
print("\n\n\nsachinkanojiya 1744")
```

output:

```
>>> ========== RESTART ==========
>>>
unshorted number is:

[2, 1, 6, 5, 85, 4, 6, 74, 8, 0, 96, 4, 2, 9, 8, 54, 22, 35, 33, 56, 11, 54, 55,
 56, 7, 57, 99, 4854, 85]
shorted number is :
 [0, 1, 2, 2, 4, 4, 5, 6, 6, 7, 8, 8, 9, 11, 22, 33, 35, 54, 54, 55, 56, 56, 57,
 74, 85, 85, 96, 99, 4854]


sachin kanojiya 1744
>>>
```


Practical = 10

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

THEORY :- SHORTING is type in which any an random data is sorted ie. arranged in descending or ascending order.

BUBBLE Sort sometime referred it as sinking sort. It a simple sorting algorithm That repeatedly steps through the lists, compares adjacent elements and swa them if They are in wrong order The pass through the list is repeated until the list is sorted the algorith which is a comparision sort is named dor The way smodwler or larger elements " " to The top of The list.

Although the algorithm is simple it is too slow ass. if compares one element cheks if cor fails Then only sweaps otherwise goes on.

Example :

first pass
(5 1 4 2 8) → (1 5 4 2 8) Here algorithm compares the first two element 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
(1 2 4 5 8) → (1 2 4 5 8)

Third pass :

(1 2 4 5 8) It checks and gives the data is sorted order

Program:

```
defquickSort(alist):
quickSortHelper(alist,0,len(alist)-1)
defquickSortHelper(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:
whileleftmark<=rightmark and alist[leftmark]<=pivotvalue:
leftmark=leftmark+1
whilealist[rightmark]>=pivotvalue and rightmark>=leftmark:
rightmark=rightmark-1
ifrightmark<leftmark:
done=True
```


Practical = 11

Aim : To "evaluate" i.e to sort the given data is quick sort.

THEORY :

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

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

The key process is quicksort is partition(). Target of partitions given as pivot, put it at its correct position is sorted array and put all smaller elements

(smaller than x) before x, & put all greater element (greter than x) after x All this should be done in linear time.

```
else:
temp=alist[leftmark]
alist[leftmark]=alist[rightmark]
alist[rightmark]=temp
temp=alist[first]
alist[first]=alist[rightmark]
alist[rightmark]=temp
returnrightmark
alist=[42,54,45,67,62,97,10,4,0,89,66,55,80,100]
quickSort(alist)
print(alist)
print("sachinkanojiya 1744")
```

output:

```
>>> ========== RESTART ==========
>>>
[0, 4, 10, 42, 45, 54, 55, 62, 66, 67, 80, 89, 97, 100]
sachin kanojiya 1744
>>>
>>> ========== RESTART ==========
>>>
[0, 4, 10, 42, 45, 54, 55, 62, 66, 67, 80, 89, 97, 100]
sachin kanojiya 1744
>>>
```

Practical = 12

Topic : To demonstrate the selection sort method.

Theory :

The sorting is based on selection hence is known as selection sort. It is also known as placing shorting algorithm.

select the first largest element & place it at end of array, select the second largest element & so on.

eg : [12, 5, 16, 1, 23, 9, 30, 4, 45]

[5, 12, 16, 1, 23, 9, 30, 4, 45]

find largest & replace with last & it same.

[5, 12, 16, 1, 23, 9, 4, 30, 45]

[5, 12, 16, 1, 9, 4, 23, 30, 45]

[5, 1, 9, 4, 12, 16, 23, 30, 45]

[5, 1, 4, 9, 12, 14, 16, 23, 30, 45]

[1, 4, 5, 9, 12, 14, 16, 23, 30, 45]

Program:

```
a=[2,1,6,5,85,4,6,74,8,0,96,4,2,85]
print("unshorted number is:\n")
print(a)
for i in range(len(a)-1):
for j in range(len(a)-1):
if (a[j]>a[i+1]):
a[j],a[i+1]=a[i+1],a[j]
print("shorted number is :\n",a)
print("\n\nsachinkanojiya 1744")
```

output:

```
>>> ========== RESTART ==
>>>
unshorted number is:

[2, 1, 6, 5, 85, 4, 6, 74, 8, 0, 96, 4, 2, 85]
shorted number is :
 [0, 1, 2, 2, 4, 4, 5, 6, 6, 8, 74, 85, 85, 96


sachin kanojiya 1744
>>>
```

Program:

```
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):
ifself.root==None:
self.root=Node(val)
else:
newnode=Node(val)
h=self.root
while True:
ifnewnode.data<h.data:
```


Practical = 14

Topic = Binary Tree

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

Lead node : Nodes which do not have any children

Internal node : Nodes which are non-lead nodes.

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

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

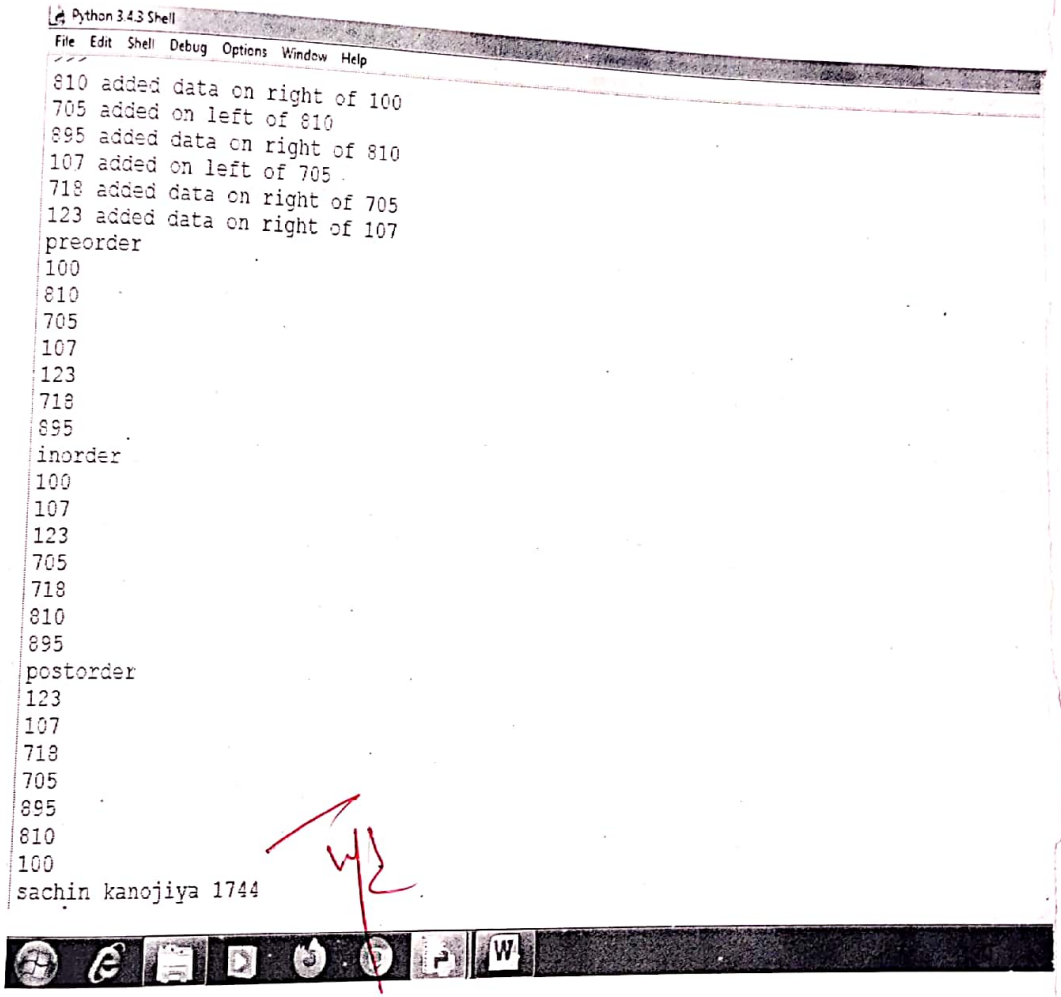
Inorder :- i) visit the root nodes
ii) Traverse the left subtree. The left subtree inturn might have left and right subtrees
iii) Traverse the right subtree the right subtree inturn might have left and right subtree

Postorder :- i) Traverse the left subtree
The left subtree inturn might have left and right subtrees.
ii) Traverse the right subtree. The right subtree inturn might have left and right subtrees.
iii) visit the root nodes.

```
ifh.l!=None:
                h=h.l
else:
h.l=newnode
print(newnode.data,"added on left of",h.data)
break
else:
ifh.r!=None:
                h=h.r
else:
h.r=newnode
print(newnode.data,"added 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)
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)
T=Tree()
T.add(100)
T.add(810)
T.add(705)
T.add(895)
T.add(107)
T.add(718)
T.add(123)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
print("sachinkanojiya 1744")
```

out put:



```
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]
```


Practical = 14

Topic : Merge sort

Theory : Merge sort uses the divide and conquer technique. In merge sort we divide the list into nearly equal sublist each of which are then again divided into two individual.

- We continue this procedure until there is only one element in the individual sublist.
- One we have individual elements in the sublists we consider these sublist as sub problems which can be solved. Since, the list is already sorted. So now we merge the sub problems.
- Now while merging the sub problems we compare the 2 sublist and create the combined list by comparing the element of the sublist. After comparison we place the element in their correct position in the original sublists.

```
            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)
 print("sachin\n1744\nmergesort")
 arr=[12,23,34,56,78,45,86,98,42]
 print(arr)
 n=len(arr)
 mergesort(arr,0,n-1)
 print(arr)
```