PRACTICAL NO. 1

Aim :- 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

i) Linear Search
ii) Binary Search

The linear Search is further classified as UNSORTED

Here will look on the UNSORTED Linear Search

Linear Search, also know as Sequential Search, also is, a process that check every element in the list sequentially until the desired element is found.

when the element we searched are not specifically arranged in ascending or arranged order. They are arranged in random. that is what it calls unsorted linear search.

Unsorted linear search

The data is entered in random manner.
user needs specifed the element to the Searched in this entered list
check the condition that wehether the entered

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

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

Program :

Unsort number:-

```
found=False
a=[10,5,6,4,9,7,8]
print("chandresh 1747")
search=int(input("enter the number search"))
for i in range(len(a)):
if(search==a[i]):
print("number found at")
found=True
break
if(found==False):
print("number not found")
```

>>> output :-

```
chandresh 1747
enter the number search11
number not found
>>> ========== RESTART ========
>>>
chandresh 1747
enter the number search10
number found at
>>>
```

PRACTICAL NO. 2

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

Theory :- Searching and Sorting are different modes or type of data Structure

SORTING - To basically SORT the inputed data in ascending or descending manner.

SEARCHING - To search element and to display the same.

In Searching that too in linear Sorted search. The data is arranged in ascending and descend that is all what it meant by Searching through Sorted that is well arranged data.

Sorted Linear Search

The user is supposed to enter data in Sorted manner

user has to given an element for Searching through Sorted list.

If element is found display with an updation of value is stored from location '0'

If data or element not found print the same.

Program :

Sort number:-

```
found=False
a=[11,12,13,14,15,16,17,67,68,70]
print("chandresh 1747")
search=int(input("enter the number search"))
if(search<a[0] or search>a[len(a)-1]):
print("number dose not exist")
else:
for i in range(len(a)):
if(search==a[i]):
print("number found at",i)
found=True
break
if(found==False):
print("number dose not exist")
```

>>> output :-

```
chandresh 1747
enter the number search17
number found at 6
>>> ========== RESTART ==========
>>>
chandresh 1747
enter the number search10
number dose not exist
```

PRACTICAL - 3

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

Theory :- A linear search also know as a half-interval search. is an algorithm used in computer science to locate. for the search to the linary. the array must list sorted in either accending or discending order.
At each step of the algorithm a comparision is mode and the procedure branches into one of two directions.
Specifically. the key value is compared to the middle element of the array.
If the key value is less than or greater than this middle element, the algorithm know which half of the array to continues searching in because the array is sorted.
This process is represented on propressively smeller segments of the array until the value is located.
Because each step in the algorithm divides the array size is half a linarly search will complete sucessfully in logarithic time.

Program :-

Binary number :-

```
a=[21,25,39,49,58]
print("chandresh 1747")
search=int(input("enter the number search"))
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 is found",m,"index number")
break
else:
if(search<a[m]):
      r=m-1
else:
     r=m+1
```

output :-

```
chandresh 1747
enter the number search25
number is found 2
>>> ================================ RESTART ================================
>>>
chandresh 1747
enter the number search15
number not found
>>>
```

PRACTICAL - 4

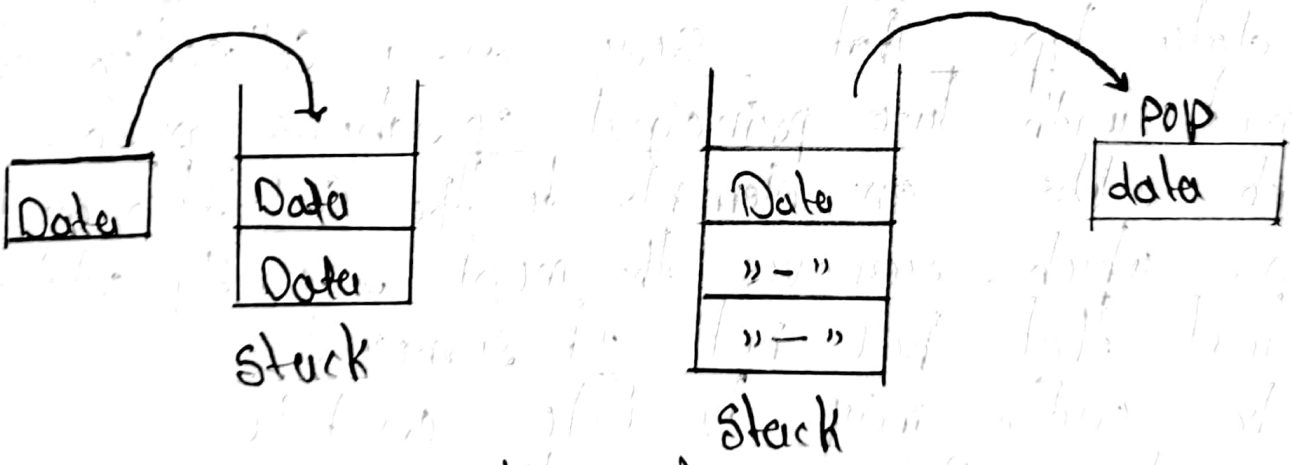
Aim :- To demonstrate the use of stack

Theory :- In Computer science, a stack is an data type that some of - a Collection of element with two principal _ operations push, which adds an elements to the Collection and pop, which removes the most recently added element that has not yet removed.
The order may be LIFO or FILO.
There Basic operation are performed in the Stack.

- PUSH : Adds an item in the if the stack is full that it is said the over flow Condition
- pop : Remove an item from the stack. The items are popped in the removed orders in which they are pushed. if the stack is empty, then it is said to be underflow Condition.

P.8;;

- peek on Top :- Returns top elements of Stack
- is Empty :- Return true if Stack is empty else false



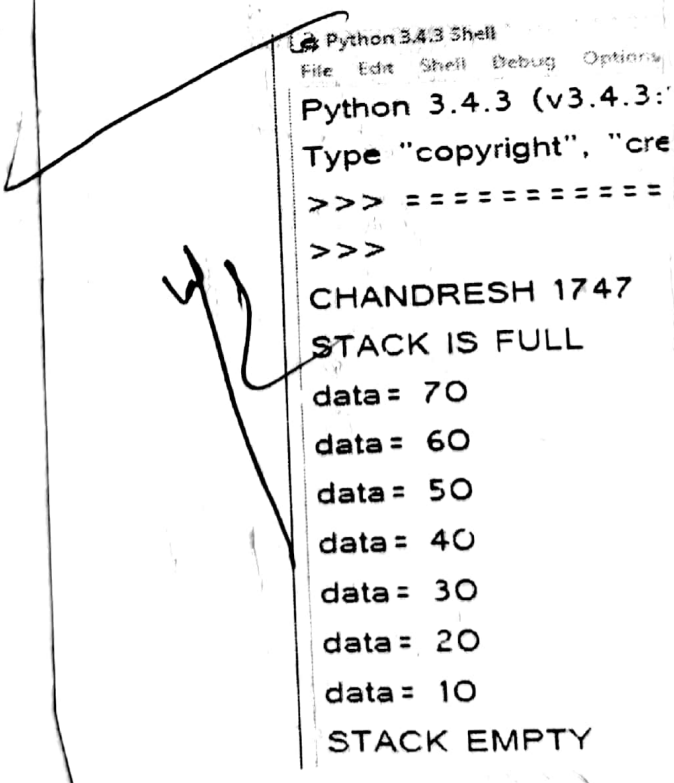
Last - in - first - out

PROGRAM:

```
##stack##
print("CHANDRESH 1747")
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:



PRACTICAL - 5

Aim :- To demonstrate Queue add and delete

Theory :-

Queue is linear data structure where the first element is inserted from one end called REAR and delet from the other end called as FRONT Front points to the beginning of the queue and rear point to the end of the queue.

Queue follows the FIFO structure element inserted first will also be removed first.

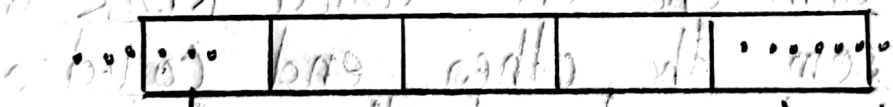
In a queue, one end is always used the insterted data queue and the other is used to delete data (dequeue) because queue is open at both of its ends

enqueue() can be termed as add() in queue i.e adding a elements in queue.

Dequeue() can be termed as a delet or Remove i.e deleting or removing of elements

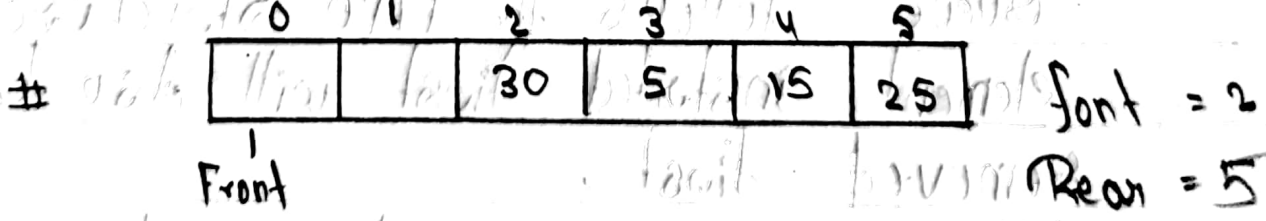
Fount is used to get front data item from a queue.

Rear is used to get the last item from a queue



Queue

Can have both ends on side



PROGRAM:

```
## Queue add and Delete ##
class Queue:
  print("CHANDRESH GUPTA 1747")
  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()
```

OUTPUT:

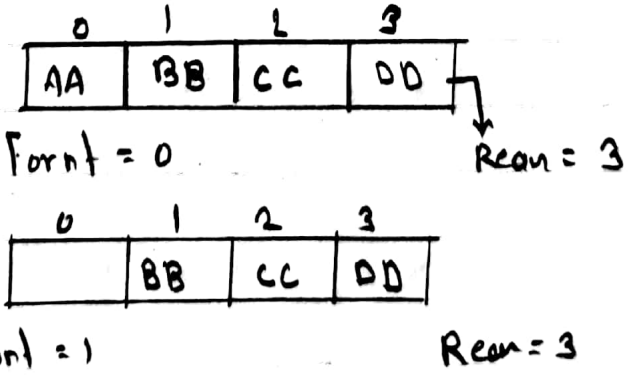
```
Python 3.4.3 Shell
File  Edit  Shell  Debug  Options  Windows  Help
Python 3.4.3 (v3.4.3:9b73f1c3e60
Type "copyright", "credits" or "lice
>>> ====================
>>>
CHANDRESH GUPTA 1747
Queue is full
30
40
50
60
70
Queue is empty
```

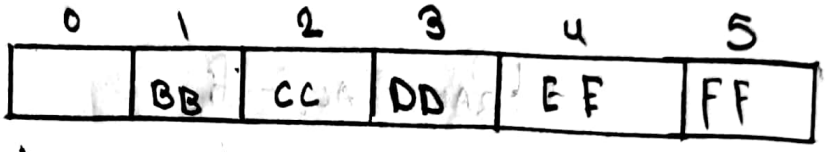
PRACTICAL - 6

Aim :- To demonstrate the use of Circular queue in data - Structure

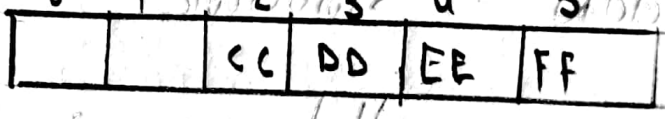
Theory :- The queue that use implement using an array Suffer from one limitation. In that implementation there is a possibility that the queue is actually there might be empty state at the beginning of the queue. To overcome this limitation we can implement queue as Circular queue. In Circular queue we go can on adding the element to the queue and reach the end of the array. The next element is stored in this first slot of this array.

Example :-

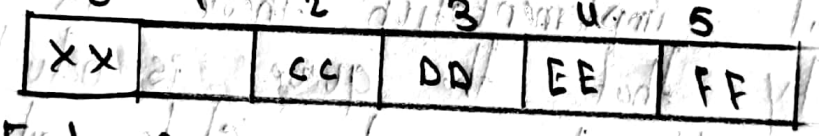




Front = 1 Rear = 5



Front = 2 Rear = 5



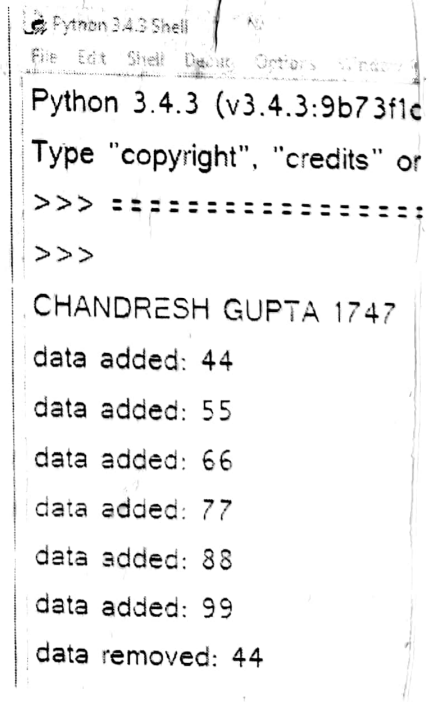
Front = 2 Rear = 5

PROGRAM:

```
#(circular queue)
print("CHANDRESH GUPTA 1747")
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:



PRACTICAL : 7

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

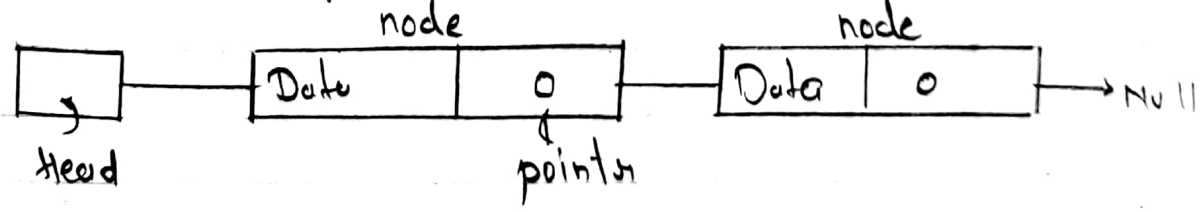
Theory :- A linked list is a Sequence of data Structures. Linked list is a Sequence of links which Contains items Each link contains a Connection to another link

- LINK - Each link of a linked list Can store to the next link Called NEXT.

- NEXT - Each link of a linked list Contains a data called an element

- LINKED LIST - A linked list Contains the Connection link to the first link Called First.

LINKED LIST Representation



Types of Linked list :

1) Simple
2) Doubely
3) Circular

Basic Operation

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

Program: # link list #

```
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
```

```
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(80)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
print("chnadresh 1747")
```

OUTPUT:

```
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1
tel)] on win32
Type "copyright", "credits"
>>> ======================
>>>
20
30
40
50
60
70
80
chnadresh 1747
```

PRACTICAL - 8

Aim :- To evaluate postfix expression using Stack

Theory :- Stack is an (ADT) and works on LIFO (Last-in-first-out) i.e push & pop operation

A postfix expression is a collection of operator is placed after the operands

Step to we followed :-

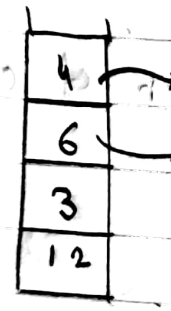
1. Read all the Symbol one by one from left to right in the given postfix expension.
2. If the reading Symbol is operand then push it on to the stack.
3. If the reading Symbol is operator (+, -, *, etc) Symbol from perform Two pop opeenration and store the two popped operand on two different variable (operand 1 & operand 2) Then perform reading Symbol operator using operand 1 & operand 2 and push result back on the stack
4. finally perform a pop operation and display the popped value as final result

5.8

value of postfix expression:

5 12 3 6 4 - * +

Stack:



b - a = 6 - 4 = 2

b + a = 3 + 2 = 5

b * a = 12 * 5 = 60

PROGRAM: # postfix#

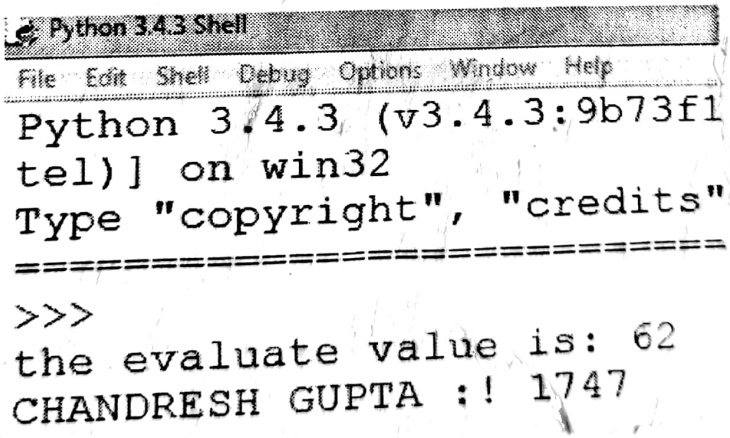
```
def evaluate(s):
    k=s.split()
    n=len(k)
stack=[]
for i in range(n):
if k[i].isdigit():
stack.append(int(k[i]))
elif k[i]=='+':
            a=stack.pop()
            b=stack.pop()
stack.append(int(b)+int(a))
elif k[i]=='-':
            a=stack.pop()
            b=stack.pop()
stack.append(int(b)-int(a))
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))
returnstack.pop()
s="8 6 9 * +"
```

```
r=evaluate(s)

print("the evaluate value is:",r)


print("CHANDRESH GUPTA :! 1747")
```

OUTPUT:



PRACTICAL-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 accesending on desending order
BUBBLE sort Sometime referred is as Simple Sorting algorithm that repeated step throughth The list they are is wrong order The pass unit the list is sorted the algoritham which is named for the way Smaler on largest element "bubble" to the top of the list.
Although the algorithm is simple the algorithm is Simple it is too slow as it Compares are element checks it Condition fails then only Swaps otherwise goes on

Example :-
First pass
(5 1 4 2 8) → (5 4 2 8) here algoritham Compere the first two element and Swaps 5>1

Secound :
(4 2 5 8) → (1 4 2 5 8)
(1 4 2 5 8) → (1 2 4 5 8) Swap sin 4>2

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
4854,85]
print("bubble shorted 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\nchandresh 1747")
```

output:

```
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
bubble shorted number is:

[2, 1, 6, 5, 85, 4, 6, 74, 8, 0, 96, 4, 2, 9, 8, 54, 22, 35, 33, 56
 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
 74, 85, 85, 96, 99, 4854]



chandresh 1747
>>>
```

PRACTICAL - 10

Aim :- To evaluate i.e to sort the given data in 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 version 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 -
pick median of pivot

example:

Consider following aways :

50 23 9 18 61 32

Program:-

```
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=[42,54,45,67,62,97,10,4,0,89,66,55,80,100]
quickSort(alist)
print(alist)
print("chandresh gupta1747")
```

output:

```
tel)] on win32
Type "copyright", "credits" or "license()" for more inform
>>> ========== RESTART ==========
>>>
[0, 4, 10, 42, 45, 54, 55, 62, 66, 67, 80, 89, 97, 100]
chandresh gupta1747
```

PRACTICAL - 11

Aim :- To evaluate to sort the given by Selection Sort

THEORY :- Selection Sort is an in-place Comparise Sorting algorithm. It has an an time complexity, which make it inefficient on large lists, and generally pe worse than the similar insertion sort. Th Smallest element is selected from the uns array and swapped with the leftmost elemen and that element because a part of the sorte array. This process continues moving unsorted arrays boundary by one element to the ri

Program:

```
a=[11,23,14,35,34,60]
print(a)
print("chandresh 1747")
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(" number after selection srot:\n",a)
```

output:

```
tel)] on win32
Type "copyright", "credits" or "licens
>>> ================================ R
>>>
[11, 23, 14, 35, 34, 60]
chandresh 1747
 number after selection srot:
 [11, 14, 23, 34, 35, 60]
>>>
```

PRACTICAL-12

Binary tree

THEORY :- Binary tree is a tree data structure. tree combines the advantage of two other structures an order arrays and a linked list. You can search a tree quickly, as can an order arrays and a linked an orders. and you can also insert and date items quickly can with a linked list.

If is every node in a tree can have at most two children, the tree is called a binary. it even node will focus on binary trees because they are the simple, the most common, and in many situations the most frequetly used.

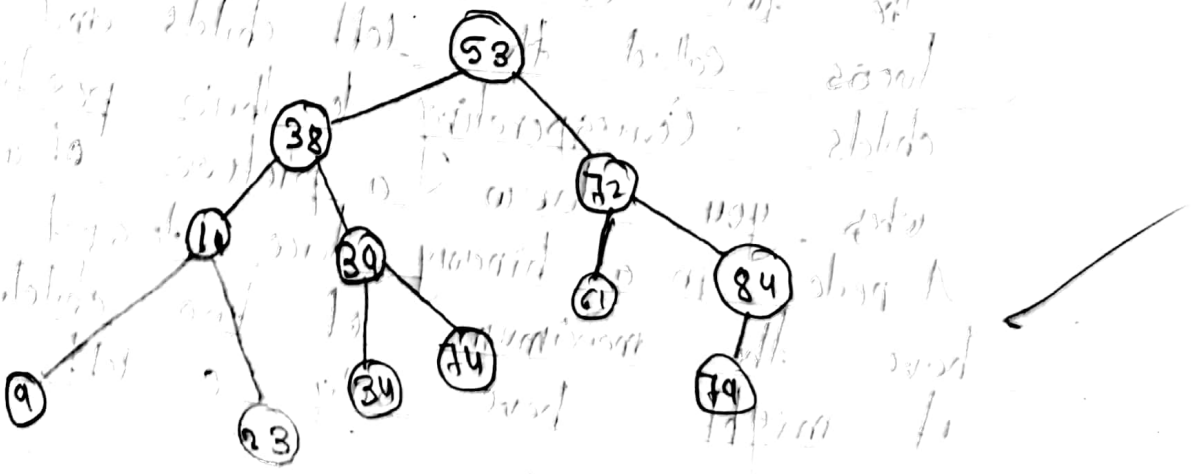
The two children of each node in a binary tress called the left childs and right childs. Corresponding to their position when you drow a picture of a tree. A node in a binary tree doesn't neccessarily have the maximum of two children it might have only a left child.

or only can right child, or it can have not children at all (which means it's leaf)

The king of binary the tree we'll be with this in discussion it technically called the tree child a binary search tree

The define characteristic of a binary Search tree is this; A node's left node child must be have a key key than it parent, and a node's right child must have a key greater on equal to it parent, and a node's right child must have a key greater than or equal to its parent.

Now that we've learned how describe the part of tree, let's look at trees structure.



Program:

```
class Node:

print("chandresh 1747 : -->>")

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:
ifh.l!=None:
                    h=h.l
else:
h.l=newnode
print(newnode.data,"addedleft of",h.data)
break
else:
ifh.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)
T=Tree()
T.add(400)
T.add(11)
T.add(17)
T.add(50)
T.add(30)
T.add(99)
T.add(69)
T.add(200)
T.add(19)
T.add(47)
print("preoeder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
```

OUTPUT:-

```
Python 3.4.3 (v3.4.3:9b73f1c3e601,
Type "copyright", "credits" or "li
>>> ================================
>>>
chandresh 1747 : -->>
11 added on left of 400
17 added on right of 11
50 added on right of 17
30 added on left of 50
99 added on right of 50
69 added on left of 99
200 added on right of 99
19 added on left of 30
47 added on right of 30
preoeder
400
11
17
50
30
19
47
99
69
200
inorder
11
17
19
30
47
50
69
99
200
400
postorder
19
47
30
69
200
99
50
17
11
400
>>>
```

PRACTICAL - 13

Topic :- Merge sort

THEORY :- Merge sort uses the divide and Conquer techniques. In merge sort we divide the list into nearly equal sublist each of which are then again divides into two sublists we continues this procedure until there is only one element in the individual sublist.

one we have individual elements in the sublist, we consider there sublist sub problems, which can sorted. so now we merge the sub problems.

Now while merging the sub problms, we Compare the 2 sublist and Creats the combined list by comparing the element of the sublist.

Program:

```
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)
 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)
```

```
>>> ================================ RESTART
>>>
chendresh 1747
[124, 236, 354, 556, 785, 465, 866, 968, 426
[124, 236, 556, 556, 426, 465, 785, 866, 96
>>>
```