PRATICAL-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
(i) Linear search
(ii) Binary search.
The Linear search is further classified as
(i) SORTED (ii) UNSORTED.
Here we will look on the UN-SORTED. 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 elements to be searched are not specifically arranged in ascending order ~~or~~ or descending order. They are arranged in random manner. That is what it calls unsorted linear search.

* UNSORTED LINEAR SEARCH:

① The data is entered in random manner.

② The user needs to specify the element to be searched in the entered list.

③ Check the condition that whether the entered number matches if it matches then display the location for increment 1 as data is stored from location zero.

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

Source code:

*ds1.py - C:\Users\DELL\AppData\Local\Programs\Python\Pyt...

File Edit Format Run Options Window Help

```
print("Sachit Pandey \n 1734")
a=[3,33,4,45,23,79,38]
j=0
print(a)
search=int(input("Enter no to be searched: "))
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!")
```

Ln: 12 Col: 30

Output:

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit
 (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\DELL\AppData\Local\Programs\Python\Python37-32\ds1.py
 ==
Sachit Pandey
 1734
[3, 33, 4, 45, 23, 79, 38]
Enter no to be searched: 23
Number found at:  5
>>>
== RESTART: C:\Users\DELL\AppData\Local\Programs\Python\Python37-32\ds1.py
 ==
Sachit Pandey
 1734
[3, 33, 4, 45, 23, 79, 38]
Enter no to be searched: 10
```

Ln: 17 Col: 4

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

Theory :- Searching and sorting.
are different modes or types of data-structure
SORTING- To basically SORT the inputed data in ascending or descending manner.
SEARCHING:- To search elements and to display the same.
In searching that too in LINEAR SORTED search ascending to descending or descending to ascending that is all what it meant by searching through 'sorted' that is well arranged data.

SORTED LINEAR SEARCH :-

① The user is supposed to enter data in sorted manner.
② user that has to give an element for searching through sorted list
③ If element is found display with an updation as value is stored from location '0'.
④ If data or element not found print the same
⑤ In sorted order list of elements we can check the condition that whether the entered number lies from starting point till the last element if not then without any process we can say number not in the list.

Source code:

*ds2.py - C:\Users\DELL\AppData\Local\Programs\Python\Python37-32\ds...

File Edit Format Run Options Window Help

```
print("Sachit Pandey \n1734")
a=[3,10,21,59,69,74,80]
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!")
```

Ln: 15 Col: 33

Output:

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Inte
l)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\DELL\AppData\Local\Programs\Python\Python37-32\ds2.py ==
Sachit Pandey
1734
[3, 10, 21, 59, 69, 74, 80]
Enter the number to be searched: 99
Number doesnt exist!
>>>
== RESTART: C:\Users\DELL\AppData\Local\Programs\Python\Python37-32\ds2.py ==
Sachit Pandey
1734
[3, 10, 21, 59, 69, 74, 80]
Enter the number to be searched: 21
Number found at:  3
>>>
== RESTART: C:\Users\DELL\AppData\Local\Programs\Python\Python37-32\ds2.py ==
Sachit Pandey
1734
[3, 10, 21, 59, 69, 74, 80]
Enter the number to be searched: 11
Number NOT found!
>>>
```

Ln: 24 Col: 4

Protical - 3

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

Theory:

① 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 decending order.

② At each step of algorithm a comparison is made and the procedure branches into one of two directions

③ Specifically, the key value is compared to the middle element of the array.

④ If the key value is less than or greater than this middle element, the algorithm knows which half of array to continue searching. This process is repeated on progressively smaller segments of the array until the value is located

⑤ Because each step in algorithm divides the array size in half, a binary search will complete successifre in logrithmic time.

Source code:

```
print("name is sachit  \n roll no is 1734")
a=[12,6,48,22,62,14,34]
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 list")
      break
```

Output:

Case1:

```
name is sachit
roll no is 1734
[12,6,48,22,62,14,34]
Enter number to be searched from the list:6
Number found at location: 1
```

Case2:

```
name is sachit
roll no is 1734
[12,6,48,22,62,14,34]
Enter number to be searched from the list:88
Number not in range
```

Case3:

```
name is sachit
roll no is 1734
[12,6,48,22,62,14,34]
Enter number to be searched from the list:7
Number not in  list
```

Pratical: 4

Aim: To sort given random data by using Bubble sort

Theory:- ① Sorting is type in which any random data is sorted i.e. arranged in asc. or descending order.

② Bubble sort sometimes reffered to as sinking sort

③ It is a simple sorting algorithm that repeatdly steps through list, compares adjacent elements and swap them if they are in wrong order.

Although the algorithm is simple it is slow as compare one element then if condition fails then only swap otherwise goes on

Example

(6 2 1 8 9) ⟶ (2 6 1 8 9) . first pass

(2 6 1 8 9) ⟶ (2 1 6 8 9)

(2 1 6 8 9) ⟶ (2 1 6 8 9)

Second pass:

(2 1 6 8 9) ⟶ (1 2 6 8 9) swap as 2>1

Third pass:

(1 2 6 8 9) it checks and gives the data sorted order.

Source code:

```
print("name sachit pandey \n roll no 1734")
a=[10,4,6,26,8,9]
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:

```
name sachit pandey
roll no 1734
Before BUBBLE SORT elemets list:
 [10,4,6,26,8,9]
After BUBBLE SORT elemnts list:
 [4,6,8,9,10,26]
```

PRATICAL - 5

Aim:- To demonstrate the use of stack.

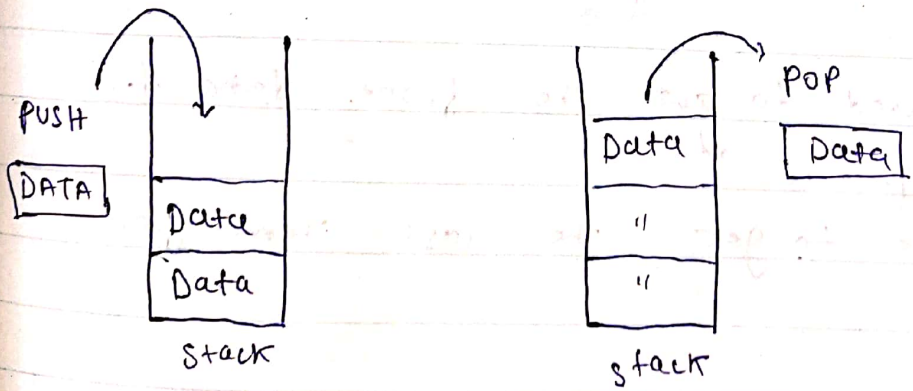
Theory :- In computer science, a stack is an abstract data type that serves as a collection of elements with two principal operations Push, which adds an element to the collection and Pop, which removes the most recently added element that was not yet removed

The order maybe LIFO (Last IN first out) or FILO (First In Last out)

Three Basic operations are performed in the stack

- PUSH: Adds an item if the steack is full then it is said to be overflow condition
- POP: Removes an item from the stack. the items are popped in the reserved 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.



Last-in-first-out

```
# Stack #
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()
print("Sachit pandey\n1734")
```

OUTPUT:

```
>>>stack is full
data= 70
data= 60
data= 50
data= 40
data= 30
data= 20
data= 10
stack empty
Sachit pandey
1734
```

PRATICAL - 6

Aim: To demonstrate Queue add and delete.

Theory: Queue is a linear data structure where the first element is ins inserted from one end called REAR and deleted from other end called as FRONT.

Front points to the beginning of the Queue and Rear points to the end of the queue

Queue follows the FIFO (First-in-First-out) Structure.

According to its FIFO structure, element inserted first will also be removed first

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

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

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

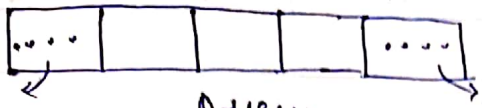
rear is used to get the last item from a queue

```
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()
print("Sachit pandey \n 1734")
```

OUTPUT:

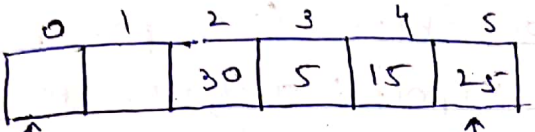
```
>>>Queue is full
30
40
50
60
70
Queue is empty
Sachit pandey
 1734
```



Queue

on both sides

Can have ends



front = 2

Rear = 5

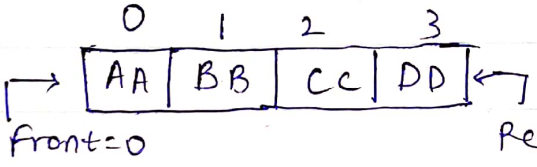
7A

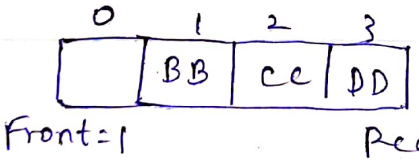
PRATICAL-7

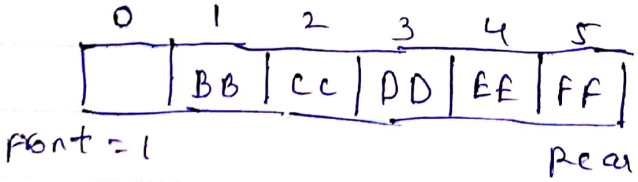
Aim: To demonstrate the use of circular queue in data structure.

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:-



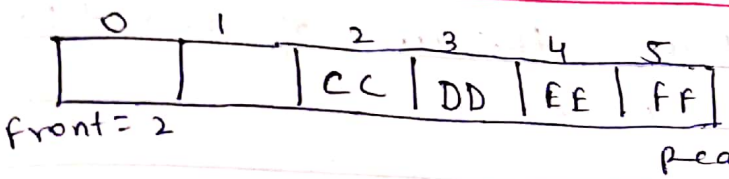




```
#circular queue
class Queue:
    global r
    global f
    def __init__(self):
        self.r=0
        self.f=0
        self.l=[0,0,0,0,0,0]
    def add(self,data):
        n=len(self.l)
        if self.r<=n-1:
            self.l[self.r]=data
            print("data added:",data)
            self.r=self.r+1
        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)
print("Sachit pandey\n 1734")
```

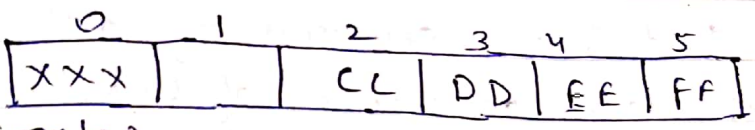
OUTPUT:

```
>>>data added: 44
data added: 55
data added: 66
data added: 77
data added: 88
data added: 99
data removed: 44
Sachit pandey
 1734
```



front = 2

rear = 5



front = 2

rear = 0.

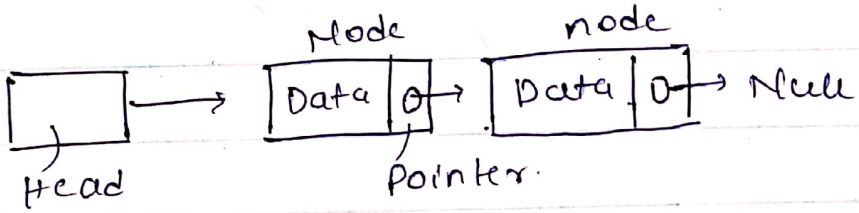
PRATICAL - 8

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:

1. Simple
2. Dubl Doubley
3. Circular

BASIC OPERATIONS:

1. Insertion
2. Deletion
3. Display
4. Search
5. Delete.

```
#linked list
class node:
  global data
  global next
  def __init__(self,item):
    self.data=item
    self.next=None
class linkedlist:
  global s
  def __init__(self):
    self.s=None
  def addL(self,item):
    newnode=node(item)
    if self.s==None:
      self.s=newnode
    else:
     head=self.s
     while head.next!=None:
       head=head.next
     head.next=newnode
  def addB(self,item):
    newnode=node(item)
    if self.s==None:
      self.s=newnode
    else:
      newnode.next=self.s
      self.s=newnode
  def display(self):
     head=self.s
     while head.next!=None:
       print (head.data)
       head=head.next
     print (head.data)
start=linkedlist()
start.addL(50)
start.addL(60)
start.addL(70)
start.addL(80)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
print("Sachit pandey\n1734")
OUTPUT:
>>>20
30
40
50
60
70
80
Sachit pandey
1734
```

PRATICAL- 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 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 the stack.
3) If the reading symbol is operator (+, -, * etc.) then perform Two pop operation in two different variables (operand1 & operand2). Then perform reading symbol operators using operands 1 & operand 2 and push result back on to the stack.
4) Finall! perform a pop operation and display the popped value as final result

value of postfix expression.

~~S = 13 5 3 7 - + *~~

S = 13 5 3 7 - + * "

```
#Postfix Evaluation

def evaluate(s):
    k=s.split()
    n=len(k)
    stack=[]
    for i in range(n):
        if k[i].isdigit():
            stack.append(int(k[i]))
        elif k[i]=='+':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)+int(a))
        elif k[i]=='-':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)-int(a))
        elif k[i]=='*':
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)*int(a))
        else:
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)/int(a))
    return stack.pop()
s="13 5 3 7 - + *"
r=evaluate(s)
print("The evaluated value is:",r)

print("Sachit pandey \n 1734")
```

OUTPUT:

```
The evaluated value is: 13
Sachit pandey
 1734
```

Aim:- To evaluate i.e to sort the given data in Quick sort.

Theory:- Quick sort is an efficient sorting algorithm type of a Divide and conquer algorithm It picks an element at pivot and partitions the given array around the picked pivot i.e. 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 in quicksort is partition().
Target of partitions is given an array and an element 'x' of array as pivot, put x at its correct position is sorted array and put all smaller elements (smaller than x) after x. All this should be alone in linear time.

```
print("Name: Sachit Pandey \nRoll No.:1734")

def quickSort(alist):
    quickSortHelper(alist,0,len(alist)-1)
def quickSortHelper(alist,first,last):
    if first<last:
        splitpoint=partition(alist,first,last)
        quickSortHelper(alist,first,splitpoint-1)
        quickSortHelper(alist,splitpoint+1,last)
def partition(alist,first,last):
    pivotvalue=alist[first]
    leftmark=first+1
    rightmark=last
    done=False
    while not done:
        while leftmark<=rightmark and alist[leftmark]<=pivotvalue:
            leftmark=leftmark+1
        while alist[rightmark]>=pivotvalue and rightmark>=leftmark:
            rightmark=rightmark-1
        if rightmark<leftmark:
            done=True
        else:
            temp=alist[leftmark]
            alist[leftmark]=alist[rightmark]
            alist[rightmark]=temp
    temp=alist[first]
    alist[first]=alist[rightmark]
    alist[rightmark]=temp
    return rightmark
alist=[42,54,45,67,89,66,55,80,100]
quickSort(alist)
print(alist)
```

Output:

```
Name: Sachit Pandey
Roll No.:1734
[42, 45, 54, 55, 66, 89, 67, 80, 100]
```

PRACTICAL - 11

Aim: To insert data in a binary tree and understand binary tree traversal.

Theory:

1) A Binary tree is a special type of tree in which every node or vertex has either no child node, one child node or two child nodes.
2) It is an important class of tree data structures in which a node can have at most two children.
3) The topmost node in the tree is called the root.
4) Every node is connected by directed edge from exactly one other node - This node is called as parent.
5) Every node on the other hand each node can be connected to other node called as ~~sibling~~ children.
6) Node with the same parent are called as siblings.
7) The height of a tree is the height of root.

```
print("Name:sachit pandey \nRoll No.:1734 ")
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.l!=None:
                                                h=h.l
                                        else:
                                                h.l=newnode
                                                print(newnode.data,"added on 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)
        def inorder(self,start):
                if start!=None:
                        self.inorder(start.l)
                        print(start.data)
                        self.inorder(start.r)
```

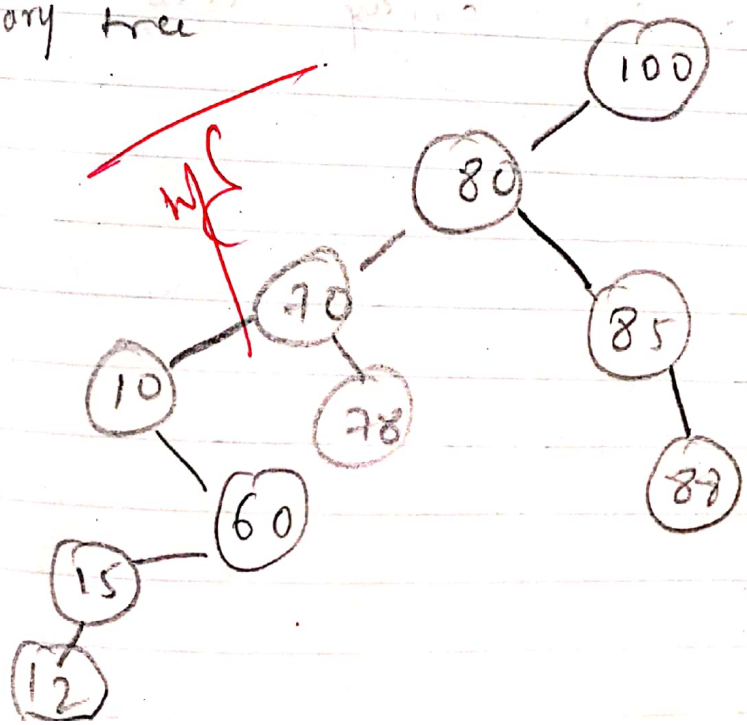
Types of binary tree

i) Full binary tree: every node other than leaf node has two child nodes. Thus, every array node has either 0 or 2 children. full binary tree is either a single vertex b) a tree whose root node has true subtrees. both of which are full binary trees.

ii) complete binary tree: In a complete binary tree. every level ~~except~~ accept posibly the last is completley filled and all nodes in the last level are as left as possible.

iii) Perfect binary tree: A perfect binary tree is a tree in which all interiors nodes have two children and all leaves have the same depth or same level.

Binary tree



```
def postorder(self,start):
    if start!=None:
        self.inorder(start.l)
        self.inorder(start.r)
        print(start.data)

T=Tree()
T.add(100)
T.add(80)
T.add(70)
T.add(85)
T.add(10)
T.add(78)
T.add(60)
T.add(88)
T.add(15)
T.add(12)
print("Preorder")
T.preorder(T.root)
print("Inorder")
T.inorder(T.root)
print("Postorder")
T.postorder(T.root)
```

Output:

Name:sachit pandey

Roll No.:1734

80 added on left of 100

70 added on left of 80

85 added on right of 80

10 added on left of 70

78 added on right of 70

60 added on right of 10

88 added on right of 85

15 added on left of 60

12 added on left of 15

Preorder

100

80

70

10

60

15

12

78

85

88

Inorder

10

12

15

60

70

78

80

85

88

100

Postorder

10

12

15

60

70

78

80

85

88

100

PRATICAL - 12

Aim: merge sort

Theory: merge sort is a sorting technique based on divide and conquer technique with worst-case time complexity being O(n log n), it one of the most repected algorithm

merge sort first divides the array in equal halves and then combines them in a sorted manner :

It divides input array in two halves, calls itself for the two halves and then merge the two sorted halves. The ~~mer~~ merge() function is used for merging halves. The merge(arr, l, m, r) is key process that assumes that arr[l...m] and arr[m+1...r] are sorted and merge the two sorted sub-array into one.

```
print("Name:sachit pandey\nRoll No.:1734 ")
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
                        j+=l
                        k+=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:
                m=int((l+(r-1))/2)
                mergesort(arr,l,m)
                mergesort(arr,m+1,r)
                sort(arr,l,m,r)
arr=[12,23,34,56,78,45,86,98,42]
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

Output:

```
Name:sachit pandey
Roll No.:1734
[12, 23, 34, 56, 42, 45, 78, 86, 98]
```

INDEX

No.	Title	Page No.	Date	Staff Member's Signature
1.	To search a number from the list using Linear unsorted	37	27/11/2019	
2.	To search a number from the list using Linear sorted method.	39	27/11/2019	
3.	To search a number from List using Binary search.	41	4/12/19	
4.	To sort given random data by using bubble sort	42	4/12/19.	
5.	stack	43		
6.	Queue add and delete	44		
7.	Circular Queue	46		
8.	Linked list.	48		
9.	Postfix Expression	50		
10.	Impetus	8.-	21/1/20	

INDEX

No.	Title	Page No.	Date	Staff Member's Signature
11.	To Evaluate i.e to sort the given data in Quick Sort			
12.	To Insert data in a binary tree and understand transversal method.	52		
13.	To Sort given random data using merge sort method.	55		