No.	Title	Page No.	Date
	DATA STRUCTURE		
1.	Linear Search Using Unsorted Data	37	27/11/19
2.	Linear Search Using Sorted Data.	39	27/11/19
3.	Binary Search	41	4/12/19
4.	Bubble Sort	42	4/12/19
5.	Stack	43	
6.	Circular Queue	44	
7.	Queue, Add and Delete	45	
8.	Linked List	46	15/1/19
9.	Postfix Expression	47	15/1/19

```
print("Name:Vikram Pandit \nRoll No.:1729")
i = 0
arr = [1,2,6,14,5,85,26,42,16,48,73,54,46]
print(arr)
search = int(input("Enter No in Search :"))
for j in range(len(arr)):
   if(search == arr[j]):
      print("No. found at :",j)
      print("No is :",arr[j])
   i += 1
   break
if(i != 1):
   print("No not found")
```

Output :

```
Name:Vikram Pandit
Roll No.:1729
[1, 2, 6, 14, 5, 85, 26, 42, 16, 48, 73, 54, 46]
Enter No in Search :85
No. found at : 5
No is : 85
```

Practical - 1

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

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 Unsorted 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 or descending order. They are arranged in random manner that is what it calls Unsorted linear search.

Unsorted Linear Search :-

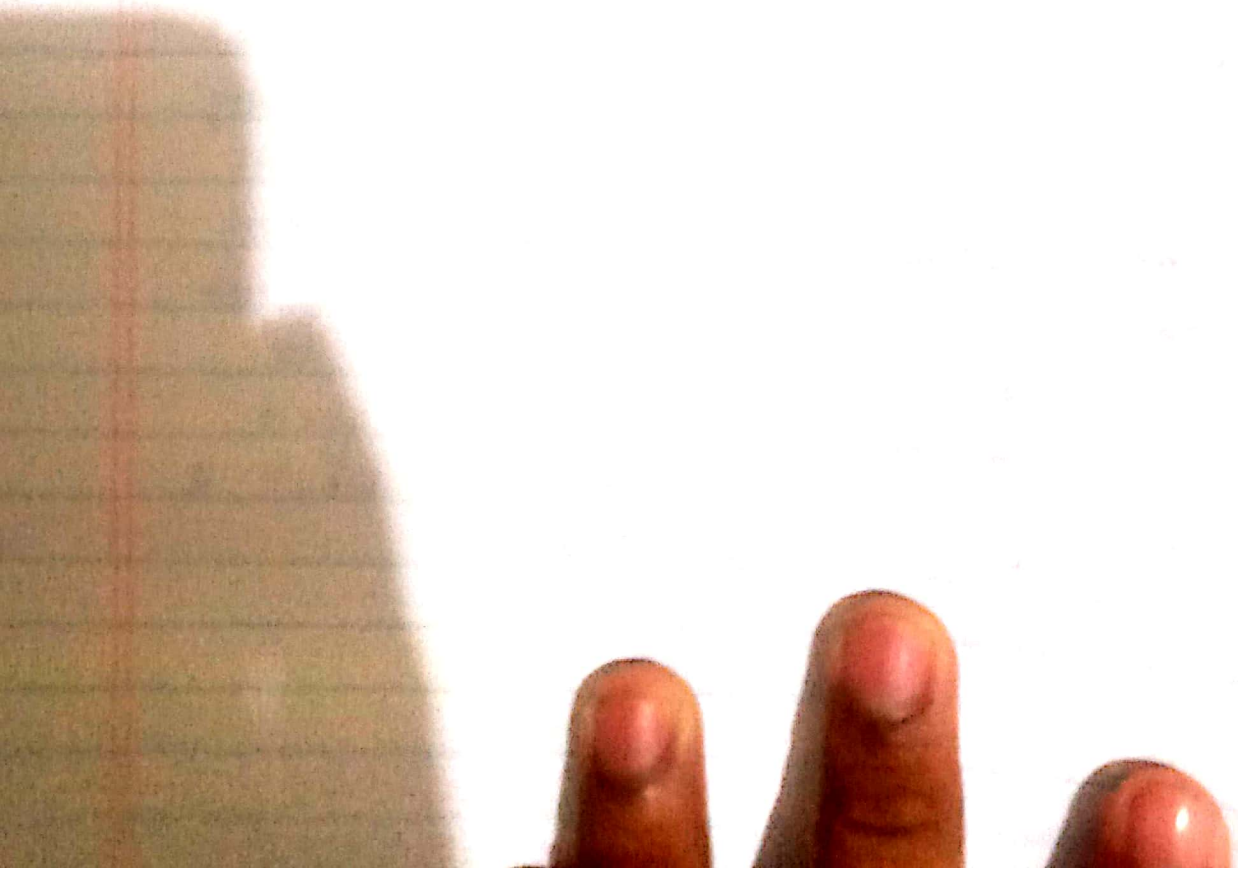
i] The data is entered in random manner

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 and also the number

iv] Increment the initial value i as +1 as data is stored from location zero

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



Aim :

Theory

```
# Sorted Search
print("Name:Vikram Pandit \nRoll No.:1729")
i = 0
arr = [4,6,8,12,15,19,21,61,75,89,99]
print(arr)
search = int(input("Enter No in Search :"))
if(search<arr[0] or search>arr[10]):
    print("Number does not exist")
for j in range(len(arr)):
    if(search == arr[j]):
        print("No. found at :",j)
        print("No. is :",arr[j])
    i += 1
    break
if(i == 1):
    print("No. not found")
```

Output:

```
Name:Vikram Pandit
Roll No.:1729
[4, 6, 8, 12, 15, 19, 21, 61, 75, 89, 99]
Enter No in Search :25
No. not found
```

Practical - 11

To search a number from the list using linear sorted method.

:- Searching and Sorting are two different modes or types of data structures

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, the data is arranged in ascending to descending or descending to ascending i.e. all what is meant by searching through sorted that is will arrange the data.

Sorted linear Search :-

i] The user is supposed to enter data in stored manner.

ii] User has to given an element for searching through sated list

iii] If element is found display with an updation as value is stored from location 0

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

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

SOURCE CODE:

```
print("Name : vikram pandit\nRoll No. : 1729 \n")
A=[12,21,24,25,34,67,78,79,85,88,93,99]
print(A)
search=int(input("Enter the number to be search : "))
l=0
h=len(A)-1
m=(l+h)//2
if (search<A[l]) or (search>A[h]):
    print("Number does not exist!")
elif search==A[l]:
    print("Number found at location ",l)
elif search==A[h]:
    print("Number found at location ",h)
else:
    while (l!=h):
        if search==A[m]:
            print("Number found at location ",m)
            break
        else:
            if search<A[m]:
                h=m
                m=(l+h)//2
            else:
                l=m
                m=(l+h)//2
    if (l==h):
        print("Number found at location ",l)
```

OUTPUT:

```
========== RESTART: D:\Data Structure\prac3.py
==========
Name : vikram pandit
Roll No. : 1729

[12, 21, 24, 25, 34, 67, 78, 79, 85, 88, 93, 99]
Enter the number to be search : 78
Number found at location  6
>>>
========== RESTART: D:\Data Structure\prac3.py
==========
Name : vikram pandit
Roll No. : 1729

[12, 21, 24, 25, 34, 67, 78, 79, 85, 88, 93, 99]
Enter the number to be search : 100
Number does not exist!
```

Practical - 11

To search a number from the given sorted list using binary search

:- A binary search also known as a half erval search, is an algorithm used in computer ence to locate a specified value (key) ithin an array. For the search to be linear, ie array must be sorted in either ascending descending order.

At each step of the algorithm, a comparison is made and the procedure branches into one or two direction.

Specifically, the key value is compared to the middle element of the array. If the key value is less than or greater than the middle element, The algorithm knows which half of the array to continue searching in because the array is sorted.

This process is repeated on progressively smaller segments of the array until the value is located. because each step in the algorithm divides the array size in half, a binary search will complete successfully in logarithmic

A

Practical - IV

Aim: To sort given random data by using bubble sort

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

Bubble sort sometimes referred to as sinking sort. It is a simple sorting algorithm that repeatedly steps through the lists, compares adjacent elements and swaps them if they are in wrong order

The pass through the list is repeated until the list is sorted. The algorithm which is a comparison sort is named for the way smaller or larger elements "bubble" to the top of the list.

Although the algorithm is simple, it is too slow as compares one elements and checks, and if condition fails then only swap otherwise goes on.

source code:

```
print("name:vikram pandit \n roll no:1729")
a=[7,3,8,9,5,6]
for p in range(len(a)-1):
    for c in range(len(a)-1-p):
        if (a[c]>a[c+1]):
            t=a[c]
            a[c]=a[c+1]
            a[c+1]=t
print a
```

output:

```
>>>
========== RESTART: C:\Users\vikram\Documents\ds practical 4.py
==========
name:vikram pandit
 roll no:1729
[3, 5, 6, 7, 8, 9]
>>>
```

```
#Stack
print("Name:Vikram Pandit \nRoll No:1729")
class Stack():
        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 += 1
        self.l[self.tos] = data
        def pop(self):
        if self.tos < 0:
        print("Stack is Empty")
        else:
        k = self.l[self.tos]
        print("data = ",k)
        self.tos -= 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:
Name: Vikram Pandit
RollNo:1729
Stack is Full
data = 70
data = 60
data = 50
data = 40
data = 30
data = 20
data = 10
Stack is Empty
```

PRACTICAL - V

Demonstrate the use of stack ()

:- 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 may be LIFO (Last In First Out) or FIFO (First In First Out).

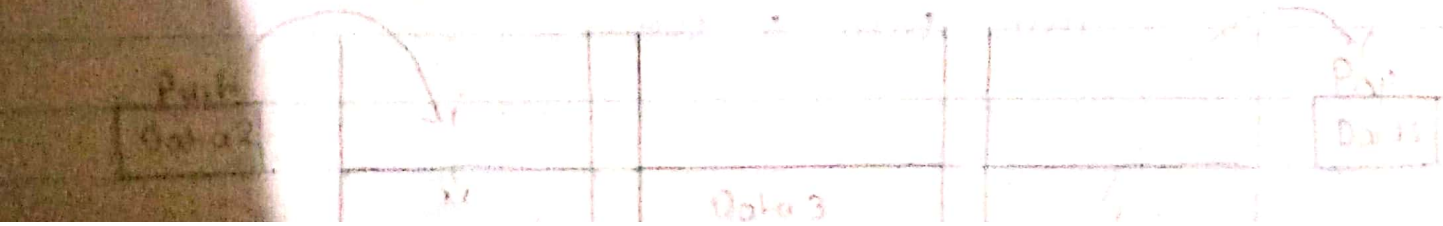
There are 3 basic operations performed in the stack

i] Push:- Adds an item in the stack. if the stack is full then it is said to be over flow condition.

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

iii] Peak or Top :- Returns Top element of Stack.

iv] isEmpty :- Returns True if stack is empty else false.



PRACTICAL – VI

Aim:- Demonstrate the use of Queue and delete.

Theory:- 1] 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.

2] Front point to the beginning of the Queue and Rear point to the end of the Queue.

3] Queue follows the FIFO (First In First- Out) Structure

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

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

6] enqueue() can be termed as add() in queue i.e. adding an element in queue and dequeue() can be termed as delete or remove i.e. deleting or removing the element

7] Front is used to get the front data item from a queue. and Rear is used to get the last item from a queue.

Queue

Can have end:

Front

```
#Queue and Delete
print("Name:Vikram Pandit \nRoll No.:1729")
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 += 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 += 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:
Name: Vikram Pandit
Roll No.:1729
Queue is Full
30
40
50
60
70
is Empty
```

```
#Circular Queue
print("Name:Vikram Pandit \nRoll No.:1729")
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 += 1
        else:
        s = self.r
        self.r = 0
        if self.r < self.f:
        self.l[self.r] = data
        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 += 1
        else:
        s = self.f
        self.f = 0
        if self.f < self.r:
        print(self.l[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)
```

```
Name: Vikram Pandit
Roll No.:1729
data added : 44
data added : 55
data added : 66
data added : 77
data added : 88
data added : 99
data removed : 44
```

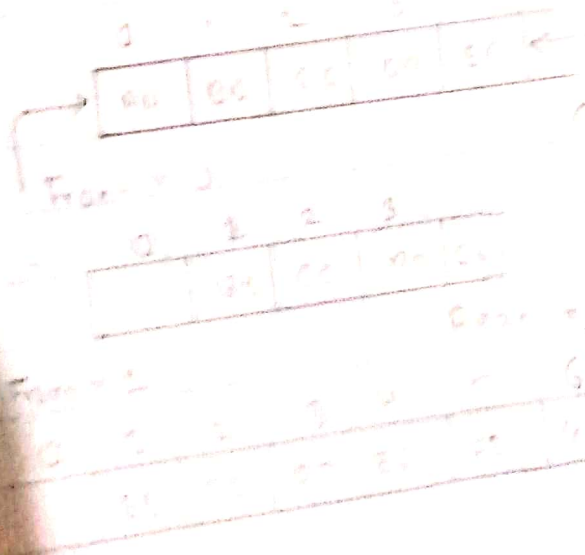
Practical - XII

m :- Demonstrate the use of Circular Queue in Data Structure.

eory :- 1) The queue that are implemented 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.

2) To overcome the limitation, we can implement queue as Circular queue.

3) In Circular Queue, we go on adding the element to the queue and reach the end of the array and the next element is stored in the first slot of the array.



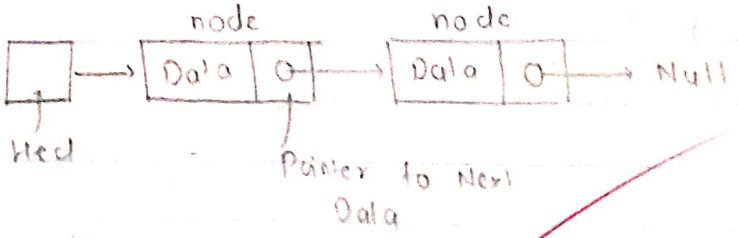
PRACTICAL - VIII

Aim:- Demonstrate the use of Linked Lists in Data Structure

Theory:- 1] A linked list is a sequence of data structures. Linked List is a sequence of links which contain items and 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] Doubly 3] Circular

Basic Operation:-

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

```
#LinkedList
print("Name: Vikram Pandit \nRoll No: 1729")
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()

#Output
Name: Vikram Pandit
Roll No: 1729
20
30
40
50
60
70
80
```

```
#PostFix Evaluation
print("Name: Vikram Pandit \nRoll No.:1729")
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 = "5 6 4 2 * - +"
r = evaluate(s)
print("The Evaluated Value is ",r)

#Output
Name: Vikram Pandit
Roll No.:1729
The Evaluated Value is  3
```

Aim:- Evaluation of Postfix Expression. using Stack.

Theory:- Stack is an ADT and works on LIFO (Last in first Out) i.e. Push and 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 evaluation.

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 popped operands ~~on~~ in two different variables (operand 1 and operand2) Then perform reading symbol operator using operand 1 and operand 2 and push result back on to the stack.

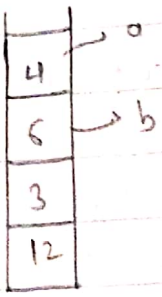
4] Finally. Perform a pop operation and display the popped value as final result.

* Value of Postfix Evaluation:-

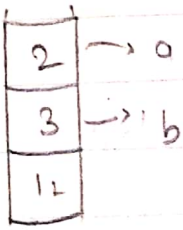
S = 12 3 6 4 ⊖ + *

Q.1

Stack :-



b - a = 6 - 4 = 2



b + a = 3 + 2 = 5

```
print("Name:Vikram Pandit  \nRoll No.:1729")

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 += 1
    while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
        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:Vikram Pandit
Roll No.:1729
[42, 45, 54, 55, 66, 89, 67, 80, 100]
```

rithm:- Sorting the Numbers Using Quick Sort

Theory:- Quicksort is a fast sorting algorithm, which is used not only for educational purposes, but widely applied in practice. On the average it has O(n log n) complexity, making quicksort suitable for sorting big data volumes. The idea of the algorithm is quite simple and once you realize it, you can write quicksort as fast as bubble sort.

Algorithm:
The divide-and-conquer strategy is used in quicksort. Below the recursion step is described:

1. Choose a pivot value:- We take the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.

2. Partition:- Rearrange elements in such a way; that all elements which are lesser than the pivot go to the left part of the array and all element greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.

3. Sort both parts:- Apply quicksort algorithm recursively to the left and right parts

PRACTICAL - XI

Aim:- Merge sort

Theory :- Merge sort is the the algorithm which follo divide and conquer approach. consider an array of n no. of elements. The algorithm processes the element in 3 steps.

1]. If it contains 0 or 1 elements then it is alre sorted. otherwise, Divide it into two sub-arra of equal no. of elements.

2] Conquer means sort the two sub-arrays recursive using the merge sort

3] Combine the sub-arrays to form a single final sorted array maintaining the ordering of the array.

The main idea behind merge sort is that, the short list takes less time to be sorted.

Complexity	Best Case	Average Case	Worst Case
Time Complexity	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Space Complexity	—	—	$O(n)$

```
#Merge Sort
print "Name: Vikram Pandit \nRoll No.:1729")
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: Vikram Pandit
Roll No.:1729
[12, 23, 34, 56, 42, 45, 78, 86, 98]
```

```
#Binary Tree and Traversal
print("Name: Vikram Pandit \nRoll No.:1729")
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)
```

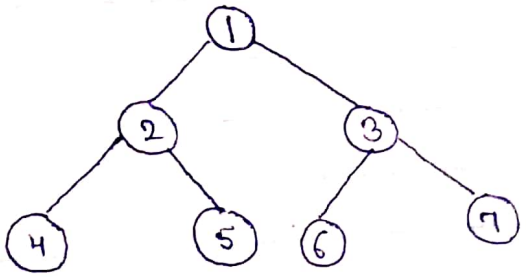
PRACTICAL - XII.

n: Binary Tree and Traversals.

eory:-

Binary Tree:-

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary can have only 2 children, we typically name them the left and right child.



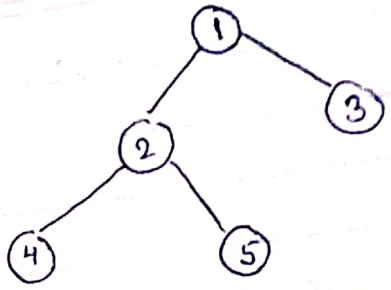
A Binary Tree node contains following parts :-

1] Data
2] Pointer to left child
3] Pointer to right child.

* Binary Tree Traversals :-

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

2



Traversals:-

a] Inorder (Left, Root, Right) : 4 2 5 1 3
b] Preorder (Root, Left, Right) : 1 2 4 5 3
c] Postorder (Left, Right, Root) : 4 5 2 3 1

* Inorder Traversal :-

Algorithm Inorder (Tree)
1] Traverse the left subtree
2] Visit the root.
3] Traverse the right subtree

* Preorder Traversal :-

Algorithm Preorder (Tree)
1] Visit the root.
2] Traverse the left subtree.
3] Traverse the right subtree

* Postorder Traversal :-

Algorithm Postorder (Tree)
1] Traverse the left subtree
2] Traverse the right subtree
3] Visit the root.

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

T=Tree()
T.add(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: Vikram Pandit
Roll No.:1729
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
```