SEM II

TE

# Practical No 1

**Aim** - Linear Search.

**Sorted Array :-**

**Step1** - Define a function with two parameters for conditional statement with range i.e length of array to find index.

**Step2** - Now use if conditional statement to check whethere the given no by user is equal to the elements in the array.

**Step3** - If the condition in step2 satisfies return the index no of the given array. If the condition doesn't satisfies then get out of loop

**Step4** - Now initialize a variable to enter elements in the arrays from user. Now use split() method & to split the values.

Theory :-

Linear search is one of the simplest searching algorithm in which each item is sequentially method each item in the list. It is worst searching algorithm with worst case time complexity. It is a force approach. A binary search is used which will start by examining the middle term linear search is a technique to compare each & every element with the very element to be found is both as item matches found and is position is also found.

# Sorted array:

```
def linear(arr, n):
    for i in range(len(arr)):
        if arr[i] == n:
            return i
inp = input("Enter element in array:").split()
array = []
for ind in inp:
    array.append(int(ind))
print("elements in array are:", array)
array.sort()
n1 = int(input("enter element to be searched:"))
n2 = linear(array, n1)
if n2 == n1:
    print("Element found at position", n2)
else:
    print("Element not found").
```

>>> Elements in array: 1 2 3 4 5
>>> Elements to be searched: 2
The element is found at position 2
>>> Enter element to be searched: 6
Element not found.

Step 5 - Now initialize a variable as empty array.

Step 6 - Now use for conditional statement to append the elements given as input by user in the empty array.

Step 7 - Now again initialize another variable to ask user to find element in array.

Step 8 - Again initialize a variable to call the defined function.

B] Unsorted Array:

Algo - Step1 Define a function with two parameters used for conditional statement with range.

Step2 - Now use if conditional statement to check whether the given statement is equal to elements in array.

Step3 - If the conditional in step 2 satisfies return the index no of the given array. If the conditional doesn't satisfies then get out of loop.

Step 4 - Now initialize a variable to enter elements from user. Now use split() method to split values.

Step5 - Now initialize the variable as array i.e empty.

Step6 - Now use for conditional statement to append elements given as i/p by user in empty array.

Step7 - Now again initialize another variable to ask user to find element in array.

Step8 - Again initialize to call defined function

```
#unsorted
def linear(arr, n):
    for i in range(len(arr)):
        if arr[i] == n:
            return i
inp = input("Enter elements in array:").split()
array = []
for ind in inp:
    array.append(int(ind))
print("Elements in array are:", array)
n1 = int(input("Enter the elements to be searched:"))
n2 = linear(array, n1)
if n2 == n:
    print("Element found at location:", n2)
else:
    print("Element not found")
```

>>> Enter elements in array: 3 2 4 5 1

>>> Element to be searched: 4

The element is found at location 2

```
# Binary Search
def binary(arr, key):
    start = 0
    end = len(arr)
    while start < end:
        mid = (start + end) // 2
        if arr[mid] > key:
            end = mid
        elif arr[mid] < key:
            start = mid + 1
        else:
            return mid
    return -1
arr = input("Enter the sorted list as no's: ").split()
arr = [ ]
for int in array:
    arr.append(int(ind))
key = int(input("Element to be searched: "))
index = binary(arr, key)
if index < 0:
    print("Element not found")
else:
    print("Element found at index", index)
```

```
>>> Enter element in array : 3 5 10 12 15 20
>>> Element to be searched : 12
>>> Element found at index : 3
```

# PRACTICAL-2

Aim - Binary Search

Algorithm

Step1 - Define a function with two parameters Now initialize variable with 0. Use while conditional statement to find mid value.

Step2 - Use if conditional statement to determine at which position the mid value should point

Step3 - If the condition doesn't satisfies the return -1

Step4 - Now initialize a variable to enter the elements in the array.

Step5 - Use for conditional statement to append the elements in empty array.

Step6 - Now initialize variable to find element in array.

Step7 - Now initialize a variable to call defined function.

Step8 - Now use if condition to determine the index & print the index value

Theory

Binary search is also known as half interval search logarithmic search or binary is a search algorithm they the position of a target value within a sorted array. If you are looking for number which is at the end of list is linear search which is time consuming

```
#Bubble Sort
inp= input("Enter number: ").split()
arr = [ ]
for ind in inp:
    arr.append(int(ind))
print("Elements of array before sorting them:",
      arr)
n = len(arr)
for i in range(0,n):
    for j in range(0,n):
        if arr[i] < arr[j]:
            temp = arr[j]
            arr[j] = arr[i]
            arr[i] = temp
print("Element of array after bubble
      sort:", arr)
```

>>> Enter elements : 2 3 6 1

>>> Elements before sorting : [2 3 6 1]

>>> Elements after sorting: [1 2 3 6]

# Practical NO-3

**Aim** - Implementation of Bubble sort program on given list.

**Theory** - Bubble Sort is based on the idea of repeatedly comparing pair of adjacent elements and then swapping their position if the simpliest form of sorting available. In this, we sort the given element in ascending or descending order by comparing two adjacent element at a time

**Algorithm**

**Step 1** - Bubble sort algorithm starts by comparing first two elements of an array & swapping if necessary.

**Step 2** - If we use want to sort the elements of array in ascending order then first element is greater than second then we need to swap the element

**Step 3** - If the first element is smaller than second element then we donot swap the element

**Step 4** - Again second & third element are compared & swapped if it is necessary & process go on until last, second last is compared & swapped.

**Step 5** - If there are n elements to be sorted then process mentioned n-1 above should be mentioned to get required result.

# Practical No 4

**AIM:** Implementation of Stacks Using Python List.

**Theory:** A Stack is a Linear data structure that can be represented in the real world from by physical stack or pile. The element in the stack or the top most position. Thus it works in the LIFO principle. It has 3 basic operations namely: push, pop, peek

**Algorithm:**

**Step 1:** Create a class stack with instance variable items.

**Step 2:** Define the int method with self argument and initialize the initial value and the initialize to an empty list.

**Step 3:** Define the method push and pop under the class stack

**Step 4:** Use if conditional statement to give the condition that if length of given list is greater than the range of list then print stack is full.

```
class stack:
    def __init__(self):
        self.tos = -1
    def push(self, data):
        n = len(self.l)
        if self.tos == n-1
            print("The stack is full")
        else:
            self.tos = 1
            self.l[self.tos] = data
    def pop(self):
        if self.tos < 0:
            print("The stack is empty")
            self.l[self.tos] = 0
            self.tos == 1
x.stack()
    def peek(self):
```

## output

```
>>> x.push(10)
>>> x.push(7)
>>> x.push(8)
>>> x.push(79)
>>> x.push(72)
>>> x.push(69)
   The stack is full
>>> x.pop()
    72
>>> x.l
  [10,7,8,79,72,69]
```

Step5 : Use the else statement to print a statement as input the element into the Stack and intlatialize the value.

Step6: Push method is used to insert element but pop method is used to delete the element from stack at top most position..

Step7: If in pop method, value is less 1 return is empty, or else delete the element from stack at topmost position.

Step8: Assign the element values in push method and print the given value.

Step 9: Attach the input and output of above diagram.

Step10: First condition checks whether number of elements are zero while second case whether top is assigned any value If top is not assigned any value then print that the stack is empty.

09/01/20

Practical No. 5

Aim: Implement Quick sort to sort the given list

Algorithm:

Step 1: Quick sort first selects a value, which is called pivot value element serve as our first pivot value since we know that first will eventually end up as last in that list.

Step 2: The position process will happen next. It will find the split point and at the same time move other items to appropriate side of the list either less than or grater than pivot value.

Step 3:- Partioning begins by locating 2 positions markers let call new left mark at the beging of remaining items in the list and right mark at the end. The goal of the partition process is the more items that are on wrong line with repeat to first value while

```
Print ("Quick Sort")
def partition (arr, low, high)
    i = low - 1
    Pivot = arr[high]
    for j in range (low, high):
        if arr[j] <= pivot:
            i = i+1
            arr[i], arr[j] = arr[i], arr[j]
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return i+1
def quicksort (arr, low, high)
    if low < high:
        pi = partition (arr, low, high)
        quick sort (arr, low, pi-1)
        quick sort (arr, pi+1, high)
I1 = input["Enter elements in the list"). split()
list1 = []
for b in I1 :
    a list1. oppered (int(b))
Printf("Elements in list are :", alls|1)
n = len (alist1)
quicksort (alist1, 0, n-1)
Print ("Element after quick sort are", alist)
```

1.A. Output:

Quick sort

Enter Element in the list 21 22 20 30 24 56

Element in list are = [21, 22, 30, 30, 24, 56]

Elements after quick sort are [20, 21, 22, 24, 30, 56]

also converging on the split solid.

**Step4:** We begin by increasing leftmark until we locate a value that is greater than the p.u.j we the decrement rightmark until we find value that is less than the P.v at this point we have discovered two items that are dot of plan with respect to eventual split point.

**Step5:** At the point where right mark becomes less than leftmark we stop. The position of right mark in move the split point.

**Step6:** The P.V can be exchanged with the Content of split point and p.v is now in place.

**Step7:** In addition all the items to left of split round are less than p.v and all the items to left to the right of split point are greater than pv and all the items to the left to the right of split point are greater than pv. The list can now be divided at split point are greater than pv

The list can now be divided at split point are less than L.V & quick ~~point~~ sort can be invoked recursively on the two values.

Step8: The quickest function invokes a recursive function quick sort.

Step9: Quick sort helper, begins with same base as the message sort.

Step10: If length of the test is less than 0 are equal one it is ~~not~~ already sorted.

Step11: If it is greater than it can be partioned and recursive function.

Step12: The to partition function the process determined earlier coding

Step13: Display and stick the and at above algorithm.

3A.

#CODE :

```
class queue:
    global r
    global f
    def __init__(self):
        ref r = 0
        ref f = 0
        ref i = [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()
```

# Practical No. 6

TITLE : Implementing a Queue using python List.

THEORY : Queue is a Linear Data structure which has 2 reference front & rear implementing a queue using python list is the simplest as the python list is the simplest as the python list provides inbuilt functions to perform the specified operations of the queue At is based on the principle that a It is on new element is inserted after rear and element of queue is dected which is at front In Simple term a queue is can be described as a data structure based on first in first out.

Queue () : Creates a new empty queue

En queue () : Insert an element at the rear of the queue and similar to that of insertion of linked using tail.

Dequeue : Returns the element which was at the front, the front is moved to the Successive element. A dequeue operation cannot remove element if the queue is empty -

ALGORITHM:

Step1: Define a class queue and assign global variables then define init() method with self argument or in init(), assign initialize the initial value with the help of self argument in init(), assign the

Step2: Define a empty list and define ensure de() method with arguments assign the length of empty list

Step3: Use if statement that length of q is equal to rear then queue is full or else insert the element added sucessfully by it. increment

Step4: Define Queue() with self argument use if statement that front is equal to length of list then display Queue is at 0 & using the element from front side increment it by 1. delete

Output:

```
>> q.add(30)
>> q.add(40)
>> q.add(50)
>> q.add(60)
>> q.add(70)
>> q.add(80)
>> q.add(90)
Queue is full
>> q.remove()
30
>> q.remove()
40
>> q.remove()
50
>> q.remove()
60
>> q.remove()
70
>> q.remove()
80
>> q.remove()
Queue is empty.
```

Step 5: Now all the Queue() function & give the element that has to be added in the empty list by using enqueue() & print list by ~~using~~ after adding & same for deleting

# Practical No. 7

AIM: Program on Evaluation of given solving string by using stack in python. environment ie. postfix.

Theory: The postfix expression is free of any parenthesis of the operations in can easily be evaluated using stack Reading the expression is always from left to right in postfix.

ALGORITHM:

Step 1: Define evaluate as function then Create a empty stack in python

Step 2: Convert the string to a list by using the string method 'split'

Step 3: Calculate the length of string & print it

Step 4: Use for loop to assign the range of string then give condition using if statement.

Step 5: We should not use the head point to traverse the entire linked list because the head pointer is just only references to 1st node

# Code:

```
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 = "86 9* +"
r = evaluate(s)
print("The evaluated value is:", r)
```

output:

The evaluate value is :62.

Step 6: We may lose the reference to 1st Node on our linked list & hence most of our linked list so in order to avoid making some unwanted changes to the node we will use temprorary node to terminate the entire linked list

Step 7: We will use this temprorary node as a copy of the node we are currently temproray node a copy of Currently traversing. Since we are making temprorary node a copy of Current node the dt datatype of temprorary node should also be node.

Step 8: Now that Current is referring to the first node if we wont to access 2ND NODE of list we need to refer it as next Node of 1st Node.

# Practical No. 8

AIM: Implementation of Single linked list by adding the nodes from Last position.

Theory: A linked list is a linear data structure which is storing the elements in a ~~not~~ node In a linear fashion but no necessary continues. The individual element of the linked list called as a node Compries of 2 parts (1) Data (2) New data ~~stored~~ stores all the information wrt the element whereas next refers to the next Node.

ALGORITHM:

Step1: Traversing of a linked list means positioning all the nodes in the list in order to ~~per~~ perform some operation on them.

Step2: The entire ~~list~~ linked list means can be accessed as the first node of the linked list.

Step3: Thus the entire linked list can be traversed using the node which is ~~repared~~ referred by the head pointer of linked list.

# CODE

```
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 say.s == none:
            say.s = newnode
        else:
            head = self.s
            while head.next != None:
                head = head.next
            headnext = newnode
    def addB(self, item):
        newnode = node(item)
        if self.s == none:
            self.s = newnode
        else:
            newnode.next = self.s
            self.s = new node
    def display(self):
        head = self.s
        while head.next != None:
            print(head.data)
            head = head.next
        print(head.data)
```

start = LinkedList()

# Output:

```
>>> start.addL(80)
>>> start.addL(70)
>>> start.add(60)
>>> start.add(50)
>>> start.add(40)
>>> start.add(30)
>>> start.add(20)
>>> start.display()
```

Step4: Now that we know that we can traverse the entire linked list using the head pointer we should only we it to refer the first node of list only.

Step5: We should not use the head pointer to traverse the entire linked list because the head pointer is our only references to 1st node.

Step6: We may lose the referance to 1st Node in our linked list & hence most of our linked list so. In order to avoid making some unwanted changes to the 1st Node will use temporary node to terminate the entire linked list.

Step7: We will use this temprorary node as a copy of the node we are currently traverssing node a copy of current node the datatype of temporary node should also be node.

Step8: Now that Current is referring to the first node if we wont to access 2ND NODE of list we need to refer it it as next node of 1st node.

Step9: But the 1st node is referred by current so we can traverse to 2ND Nodes as n = n.next.

Step10: Similarly we can traverse most of nodes in the linked list.

Step11: Our concern now is to find terminating condition for the while loop.

Step12: The last Node in the linked list is referred by the tail of linked list.

Step13: So we can refer to last node of linked list.

Step14: We have to now see how to start traversing the linked list & now to indentify whether the we have reached the last node.

Step15: Attach the coding of input & output of above algorithm.

~~outp~~

```
>> 20
   30
   40
   50
   60
   70
   80
```

#code

```
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[i + l]
    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 = i + 1
        else:
            arr[k] = R[j]
            j = j + 1
        k = k + 1
    while i < n1:
        arr[k] = L[i]
        i = i + 1
        k = k + 1
```

## Practical No. 9

Aim: Implementation of merges out by using Python

Theory: Merge sort is a divide and conquer algorithm. It divides input array into two halves calls itself for the two halves & the merge function used for merging two halves.

Algorithm:

step1: The list is divided into left & right in each recursive call until two adjacent elements all obtained.

step2: Now begins the sorting process. The i & j iterators traverse the two halves in each call. The k iterators traverse the whole lists & makes changes along the way.

step3: If the value at i is smaller than the value at j L(i) is assigned to the arr [i+1] slot & is incre to the arr [i+1] slot & i is incremented. If not then R[j] is chosen.

Step 4: This way, the value being assigned through n[i+1] are all sorted.

Step 5: At the end of this loop one of the ~~half~~ halves may ~~be~~ not have been traversed completely remaining slots in the list

Step 6: Thus, the merge sort has been implemented.

```
while  j < n2 :
    arr[k] = R[j]
    j+ = 1
    k = k+1
def merge sort (arr, i, u) :
    if  i < u :
        m = int ((l + (r-1)) /2)
        merge sort (arr, l, m)
        merge sort (arr, m+1, r)
        sort (arr, l, m, r)
arr = [12, 23, 34, 56, 48, 56, 98, 42]
print (arr)
n = len (arr)
merge sort (arr, 0, n-1)
print (arr)
```

output : [12, 23, 34, 56, 78, 45, 86, 98, 42]
[12, 23, 56, 56, 42, 45, 78, 80, 98]

#CODE

```
set1 = set()
set2 = set()
For i in range (8,15):
    set1.add(i)
For i in range (1,12):
    set2.add(i)
print ("set1 : " , set1)
print ("set2 : ", set2)
print ("\n")
set 3 = set 1, set2
Print ("Union of set1 & set2 : set3", set3)
set4 = set1 & set2
print (" Intersection of set1 & set2 : ",
                                    set4)
print("\n")
if set 3 > set4 :
  print ("set 3 in subset of set4")
else :
    print ("set 3 is same as set 4")
if set 4 < set3 :
    print ("set 4 is subset of set 3")
    print("\n")
Set 5 = set3 - set4
Print ("Elements in set 3 & not in set4:")
                                    set5
Print ("\n")
if set 4 is disjoint (set5) :
    print ("set 4 & set 5 are mutually
                exclusively")
set . clear()
Print ("After applying clear, set 5 is empty:")
print ("set 5", set 5)
```

# Practical No. 10

Aim: Implementation of sets using python.

Algorithm:

Step1: Define 2 empty sets 1 & sets2 Now use for statements providing the range of above 2 sets.

Step2: Now add() is used for addition of elements in the given range then print sets for additions.

Step3: find the union & inter section of above 2 sets by using AND or method print sets of union & interset as set3.

Step4: Using if statement find out subset & superset of set3 & set4. Display the above set.

Step5: Display the elements in set3 is not in set 4 using mathematical op.

Step6: Use clear() to remove or delete the sets & print set after clearing the element present in the set()

set 1 : { 8, 9, 10, 11, 12, 13, 14 }
set 2 : { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 }
Union of set 1 & set 2. set 3
{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }
Intersection of set 1 & set 2 & 8, 9, 10, 11 }
set 3 is super ~~set~~ of set 4
~~set~~ Elements in set 3 & not in set 4.
~~Elemnt~~
{ 1, 2, 3, 4, 5, 6, 7, 12, 13, 14 }
set 4 & set 5 are mutualy exclusive
after applying clean set 5 is repeated
set 5 = set ()

# CODE

```
class node:
    def __init__(self, value):
        self.left = None
        self.val = value
        self.right = None

class BST:
    def __init__(self):
        self.root = None
    def add(self, value)
        p = node(value)
        if self.root == none
            self.root = p
            print("Root is added
                  successfully", p.val)
    else:
        h = self.root
        if p.val < h.val:
            if h.val = None:
                h.left = p
                print(p.val "None is added
                      Successfully")
                break:
            else,
            h = h.left
    else: if h.right = None:
```

# Practical 11

**AIM:** Program based on binary search tree by implementing inorder, preorder & postorder traversal.

**Theory:** BINARY tree is a tree which supports maximum of 2 children for any node have either 0 & 1 or 2 children there is another identity as left child & other as rightchild.

**Inorder:** Traverse the left subtree. The left subtree inform might have left and right subtrees.

**Preorder:** Visit the root node traverse the left subtree and right subtree. traverse the right subtree.

**Postorder:** Traverse the left subtree the left subtree inturn might have left & right subtrees.

ALGORITHM:

Step1: Define class node & define init() with 2 arguments Initialize the value in this arguments.

Step2: Again, define a class BST that is binary search with self argument & assign the root is none.

Step3: Define add() for adding the node Define a variable p that p=node(values)

Step4: Use if statement for checking the conditional that root is none then use else statement for if node is less than the main node then put arrangement in left side.

Step5: Use while loop for checking the node is less than or greater than the main node & break the loop if it is not satisfying.

```
        added to right side successfully")
        break
    else:
        h = h.right
def inorder(root):
    if root = None:
        return
    else:
        inorder(root.left)
        print(root.val)
        inorder(root.right)
def preorder(root):
    if root = None
        return
    else: print(root.val)
        preorder(root.left)
        preorder(root.left)
        preorder(root.right)
def postorder(root):
    if root == None:
        return
    else: print(root.val)
        preorder(root.left)
        preorder(root.right)
def postorder(root):
    if root == None:
        return
```

```
        else:
            postorder(root.left)
            postorder(root.right)
            print(root.val)
    t = BST()
# output:
    >> t.add(1)
    root is added sucessfully
    >>> t.add(2)
2 node is added to right side successfully
>>> t.add(4)
4 node is added to right side
  successfully.
>>> t.add(5)
5 node is added to right side successfully
>>> t.add(3)
3 node is added to left side successfully
>>> print("Inorder:", Inorder(t.root))
    Inorder:
    1
    2
    3
    4
    5
    Inorder: None.
```

step 6: Use if statement within that else statement for checking that node is greater than main root then main root then put it into right side.

step 7: After this, left side tree & right subtree repeat this method to binary search tree repeat.

step 8: Define Inorder(), preorder() & postorder with root as argument & use if statement that root is none & return that all.

step 9: Inorder, else statement used for giving that condition if first left root & then right.

step 10: for preorder we have to give condition in else that first root left, right & root.

step 11: For postorder in else part assign left right & root.

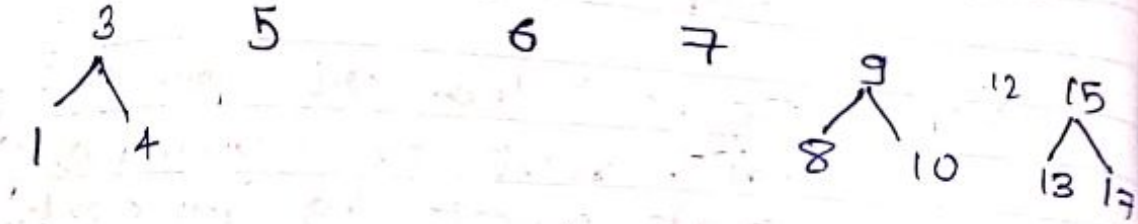
step 12: Display the output & input.

2.

INORDER (LVR)

step 1:

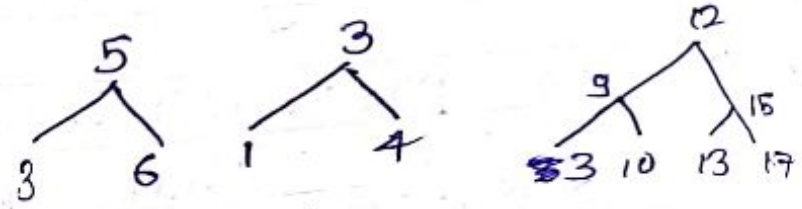

step2: 3 5 6 7



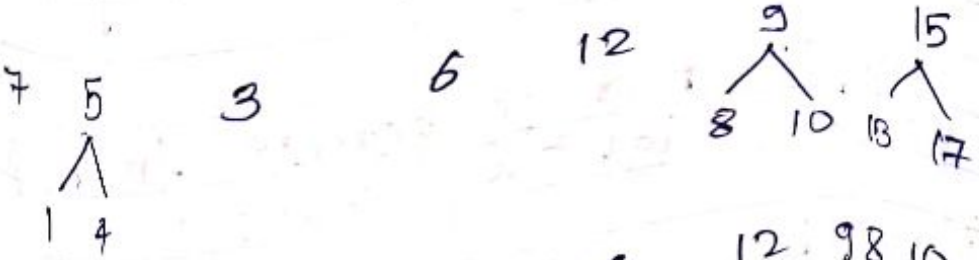

Step3: 1 3 4 5 6 7 8 9 10 12 13 15 17

PREORDER: (NLR)

step1: 7



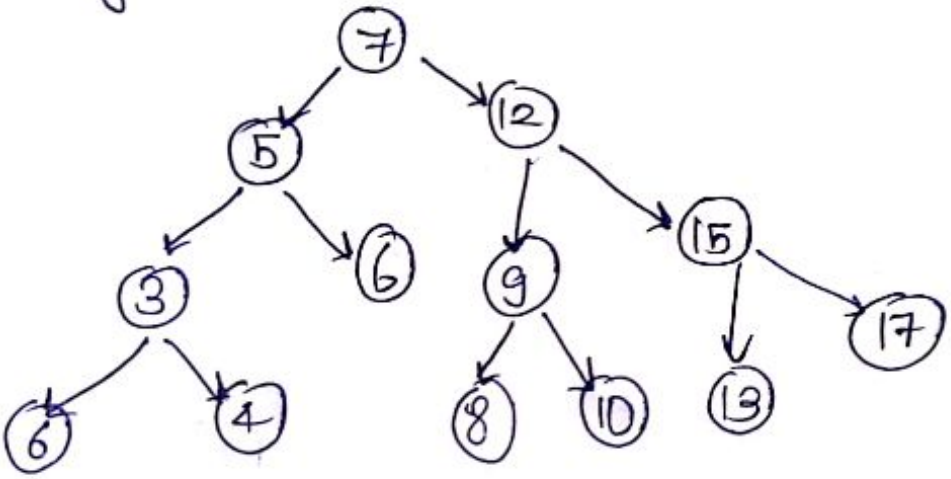

step2:




Step3: 7 5 3 1 4 6 12 9 8 10 15 13 17.

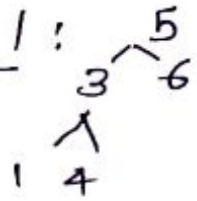
```
>> print("\n preorder:", preorder(F.root)
preorder:
1
2
3
4
5
preorder ! None
>>> print("postorder:", postorder(F.root)
3
5
4
2
1
Post order = None
search
```

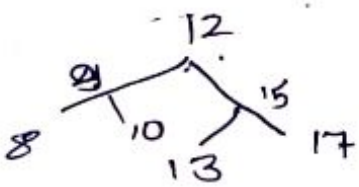
* Binary Search



(3) POSTORDER : (LRV)

Step1 :




Step2 :


6 5



12 7

Step3 : 1 4 3 6 5 8 10 9 13 17 15