No.

1] Unsorted :

Algorithm :

Step 1 : Create an empty list and assign it to a variable

Step 2 : Accept the total no of elements to be inserted into the list from the user : say 'n'.

Step 3 : Use for loop adding the elements into the list.

Step 4 : Print the new list

Step 5 : Accept an element from the user that to be sorted in the list

Step 6 : Use for loop in a range from '0' to the total no of elements to search the elements from the list

Step 7 : Use if loop that the elements in the list is equal to the element accepted from user.

Step 8 : If the element is found then print the statement that the element is found along with the element's position


```
Print ("linear Search")
a = []
n = int(input ("Enter a number"))
for s in range (0,n):
    s = int (input("Enter a number"))
    a.append (s)
    a.sort()
    print(a)
c = int(input("Enter a number to be searched"))
for i in range (0,n):
    if (a[i] == c):
        print("found at position", i)
        break
    else:
        print("not found")
```

Output :-

```
linear Search
Enter a range : 4
Enter a number : 1
[1]
Enter a number : 3
[1, 3]
Enter a number : 2
[1, 2, 3]
Enter a number : 4
[1, 2, 3, 4]
Enter a number to be searched 1
    found at position 0
```

Step 9 : Use another if loop to print that the element is not found if the element which is accepted from user is not their in the list.

Step 10 : Draw the output of given algorithm

2] Sorted Linear Search :

Sorting means to arrange the element in increasing or decreasing order

- Algorithm :

Step 1 : Create empty list and assign it to a variable

Step 2 : Accept total no. of element to be inserted into the list from user, say 'n'.

Step 3 : Use for loop for using append() method to add the element in the list

Step 4 : Use sort() method to sort the accepted element and assign in increasing order the list then print the list.

Step 5 : Use if statement to give the range in which element element is not found in given range then display "Element not found"

Step 6: Then use else statement if element is not found in range then satisfy the given conditions

Step 7: Use for loop in range from 0 to the total no of element's to be searched before doing using Input statement

Step 8: Use if loop that the elements in the list is equal to the element accepted from users

Step 9: If the element is found then print the statement that the element is found along with the element position.

Step 10: Use another if loop to print that the element is not found if the element which is accepter from user is not their in the list

Step 11: Attach the input and output of above algorithm.


```
Print ("linear Search")
a= []
n= int(input ("Enter a range:"))
for S in range (0,n):
    S= int (input (" Enter anumber"))
    a. append(s)
    print(a)
C= int (input ("Enter anumber to be searched"))
for i in range (0,n):
    if (a[i] ==c)
        Print (" found at position",i)
        break
else
    Print (" not found")
```

Output

```
linear search
enter search:3
enter a number: 1
[1]
enter a number 3
[1, 3]
enter anumber 4
[1, 3, 4]
enter a number to be searched 3
found at position 1
```

Practical - 2

Aim : Implement Binary Search to find an searched no in the list

Theory :

Binary Search

Binary search is also known as Half-interval search, logarithmic search or binary position of a target value within a sorted array. If you are looking for the number which is at the end of the list then you need to search entire list in linear search, which is time consuming. This can be avoided by using Binary fashion search.

Algorithm :

Step 1 : Bubble sort algorithm start by comparing the first two elements of an array and swapping if necessary

Step 2 : If we 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 : Use for loop, add elements in list using append() method

Step 5: Use sort() method to sort the accepted element and assign it in increasing ordered list print the list after sorting.

Step 6: Use if loop to give the range in which element is found in given range, then display a message "Element not found".

Step 7: Then use else statement if statement is not found in range then satisfy the below condition.

Step 7: Accept an argument & key of the element that element has to be searched

Step 8: Initialize first to 0 and last to last element of the list as array from 0 hence it is initialized 1 less than the total count.

Step 9: Use for loop & assign the given range.

Step 10: If statement in list and it will the element to be searched is not found then find the middle element (m) the middle term then it will be searched if still the mid


```
print(" Linear Search")
a = []
n = int(input("Enter range:"))
for i in range(0,n):
    i = int(input("Enter number"))
    a.append(i)
    a.sort()
    print(a)
s = int(input("Enter a number to be searched"))
if (s < a[0] or s > a[n-1]):
    Print("Element not found")
else:
    f = 0
    l = n-1
    for i in range(0,n):
        m = int((f+l)/2)
        print(m)
        if (s == a[m]):
            Print("Element found at :",m)
            break
    else if (s < a[m]):
        l = m-1
    else
        f = m+1
Output: Linear search
Enter range : 3
Enter number :1
[1]
Enter a number : 2
[1,2]
Enter a number :8
Enter
```

Enter a number to be searched to element not found.

Initialize last (h) = mid (m) - 1
Else
Initialize first (l) = mid (m) - 1

Step 12 :- Repeat till you found the element stick the input & output of above algorithm.

Practical - 3

Bubble Sort

- Aim: Implementation of Bubble sort program on given list

- Theory: Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements and then swapping their position if they exist in the wrong order. This is the simplest form of sorting available. In this, we sort the given element in ascending or descending order by comparing two adjacent elements at a time.

Algorithm:

Step1: Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary.

Step2: If we want to sort the elements of array in ascending order then first element is greater than second then we need to swap the element.

Step3: If the first element is smaller than second then we do not swap the element.


```
print (" bubble sort")
a=[]
b= int (input (" Enter number of elements "))
for s in range (0, b):
    s= int (input (" Enter element ! "))
    a. append (s)
    print (a)
n=len(a)
for i in range (0,n):
    for j in range (n-1):
        if [j] < a [j]
            tmp = a[j]
        a[j] = a [i]
        a [i] = tmp
print (" Element after sorting:", a]
```

Output:

```
Bubble sort
Enter the number of elements : 5
    element element : 7
     [7]
    element element : 3
     [7, 3]
    enter element : 1
    [7, 3, 1]
    enter element: 8
    [7, 3, 1, 8]
element after sorting [1, 3, 7, 8]
```

Step 4: Again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped.

Step 5: There are n elements to be sorted then the process mentioned above should be repeated n-1 times to get the required result.

Step 6: Stick the output and input of above algorithm of bubble sort stepwise.

Practical ~ 4

Quick sort

Aim: Implement quick sort to sort the given list.

Theory: The quick sort is a Recursive algorithm based on the divide and conquer technique

Algorithm:

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

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

Step 3: Partitioning begins by locating two position markers - let's call them leftmark & right mark. at the beginning and end of remaining items in the list. The goal of the partition process is to move items that are on wrong side with respect to pivot value.

Code:-

```
def quicksort(alist)
    help(alist, 0, len(alist)-1)

def help(alist, first, last):
    if first < last
        split = part(alist, first, last)
        help(alist, first, split-1)
        help(alist, split+1, last)

def part(alist, first, last):
    piviot = alist[first]
    l = first+1
    r = last
    done = false
    while not done
        while l<=r and list[l]<=pivot:
            l = l+1
        while a list[r]> = pivot and r>=1:
            r = r-1
        if r<1:
            done = True
        else:
            temp = alist[1]
            alist[1] = alist[r]
            alist[r] = temp
    temp = alist[first]
    alist[first] = alist[r]
    alist[r] = temp
    return r
```

```
x = int (input ("Enter a range of list"))
alist []
for i in range [0, x]:
    G = int (input (" Enter element"))
    alist. append (i)
    n = len (alist
```

Output:

Enter range for the list : 4
Enter the element : 6
Enter the element : 8
Enter the element : 4
Enter the element : 9

[4, 6, 8, 9]

4] we begain by incremening leftmark until the located valve which is greater than the pivot we then decrement rightmark until the we find value that is lesser than the pivot values. At this point we have descrese to eventual split point

5] At the point where rightmark becomes less than leftmark we stop. The position of rightmark is the split point

6] The pivot value can be exchanged with the context of split point and pivot is now in place

7] In addition, all the items to left of split are less than PV all the items to the right of split point are greater than PV. The list can now be dividend at split point

8] The quicksort function invokes a recurrision function, quicksort

9] Quicksort help begins with some base conde as the merge sort

10] IS it is greater of the list is less than or eqval is one, it is called, it is already sorted

DA

Practical-5

Aim - Implementation of stack using Python List

Theory - A stack is a linear data structure that can be represented in the real world in the form of a physical stack or a pile. The elements in the stack are added or removed only from one position, i.e the topmost position. Thus the stack works on the LIFO [Last In First Out] principle as the element that was inserted last will be removed first. A stack can be implemented using array as well linkind list stack has three basic operation: push, pop, peek. The generation of adding and remove the elements is known as Push & POP

Algorithm:

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

Step 2: Define the init method with self argument and initialize the initial value and then initialize to an empty list

Step 3: Define methods push and pop under the class stack

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

```
### stack ###
print (" Rugved Pednekar")
class stack:
    global tos
    def __init__ (self):
        self.l = [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.l[self.tos]=0
            self.tos = self.tos-1

    def peek (self):
        if self.tos<0:
                print (" stack empty")
        else:
            p = self.l[self.tos]
            print (" Top Element = ", p)

s= stack()
```

Output

Rugved Pednekar

```
>>> s.push(10)
>>> s.push(20)
>>> s.lpush
[10, 20, 0, 0, 0]
>>> s.peek()
Top Element = 20
>>>
```

s.pop() ?

06/01/22

Step 5: OR, else print statement as insert the element into the stack and initialize the value.

Step 6: Push method used to insert the element but pop method used to delete the element from the stack.

Step 7: If in pop method, value is less than 1 then return the stack is empty, or else delete the element from stack at topmost position.

Step 8: First condition checks whether the no. of element are zero while the record case whether tos is assigned only value. If tos is not assigned any value. Then can be sure that stack is empty.

Step 9: Assign the element values in push method to add and print the given value is poped

Step 10: Attach the input and output of above algorithm.

06/01/22

Practical-6

Aim- Implementing a queue using Python list

Theory: Queue is a linear data structure which has 2 references front and rear. Implementing a queue using Python list is the simplest as the Python list provides in build function to perform the specified operation of the queue. It is based on the principle that a new element is inserted after rear and element of queue is decided. Which is at front. In simple terms, a queue can be described as a data structure based on first in First out FIFO principle.

- Queue (): Creates a new empty queue
- Enqueue (): 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:

step 1: Define a class Queue and assign global and able then define init() method with self argument in init(), assign or initialize the initial value with the help of self argument

step 2: Define a empty list and define equeue() method with 2 argument assign the length of empty list

step 3: Use if statement that length is equal to rear then Queue is full or else insert the element in empty list or display that Queue element added successfully and increment by 1.

step 4: Define deQueue() with self argument under this, use if statement that front is equal to length of list then display Queue is Empty or else, give that front is at zero and using that delete the element from front side and increment it by 1.

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

Practical - 7

Evalution of a Postfix Expression

Aim : Program on Evalution of given string by using, stack in Python Enviroment i.e Postfix

Theory :

The postfix expression is free of any parenthesis. Further we took care of the priorities of the operators in the program. A given postfix expression can easily be evaluated using stacks. Reading the expression is always from left to right in postfix.

Algorithm

1) Define evaluate as function then create a empty stack in python
2) Convert the string to a list by using the string method split
3) Calculate the length of string and print
4) Use for loop to assign the range of string to
5) Scan the token list from left to right. If token is an operand, convert it from a string

code :

```
def evalute ():
    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 = "8 6 9 * +"
r = evaluate(s)
print("The evaluted value is",r)
print("U T")
```

Practical - 8

Aim: Implementation of single linked list by adding the nodes from last position

Theory: A linked list is a linear data structure which stores the elements in a node in a linear fashion but not necessarily contiguous. The individual element of the linked list called node. Node comprises of 2 parts i) Data ii) Next. Data stores all the information with respect to the element. for eg. roll no, name, address etc. Whereas next refers to the next node. In case of larger list if we add/remove any element from the list, all the elements of list has to adjust itself every time we add it is a very tedious task so linked list is used to solve this type of problems.

Algorithm:

Step 1: Transversing of a linked list means visiting all the nodes in the linked list in order to perform some operation on them.

Step 2: The entire linked list can be accessed with the first node of the linked list. The first node of the linked list in turn is referred by the Head pointer of the linked list.

#code

Step 3: Thus, the entire linked list can be transversed using the node which is refered by the head pointer of linked list.

Step 4: Now, that we know we can transverse the entire liked list using the head pointer, we should only use it to refer the first node of list only.

Step 5: We should not use the head pointer to transverse the entire linked list because the head pointer is our only reference to the 1st node in the linked list. Modifying the reference of the head pointer can lead to change which we cannot revert back

Step 6: We may lose the reference to the 1st node in out linked list. So, in order to avoid making some unwanted changes to the 1st node, we will use a tempory node to tranverse the entire linked list.

Step 7: We will use this temperory node as a copy of the node we are currently tranversing. since we are making temporary made a copy of current node, The datatype of the temporary node should also be node.

Step 8: Now that current is refering to the first node, if we want to accept and node of list, we can refer it as the next node of the 1st node.

Step 9: But the 1st node is refered by current. So we can transverse to 2nd nodes as h = h.next

Step 10: Similarly, we can transverse rest of nodes in the linked list using some method by while loop

Step 11: Our concern now is to find terminating condition for the while loop

Step 12: The last node in the linked list is referred by the tail of linked list. Since the last of node of linked list does not have any next node, the value in the next field of the last node is none.

Step 13: So we can refere the last node of linked list as self.s = none

Step 14: We now have to see how to start transversing the linked list and how to identify whether we have reached the last node of linked list or not.

Step 15: Attach the coding (input and output of the above algorithm.

Practical - 9

Aim: Implementation of merge sort

Theory: Like Quicksort, merge sort is a divide and conquer algorithm it divides into array in two halves calls itself for two halfves & then merges the two sorted halves. The merge() function is used for two merging two halves

Algorithm:

1) Define the Sort
2) Stores the starting position of both parts in temporary variable
3) Check if first part comes to an end or not
4) Check if second part comes to an end or not
5) Check which parts has smaller element
6) Now the veal array has elment in sorted manner including both parts.

7) Defines the correct array in 2 parts :

8) Sort the 1st part of array

9) Sort the 2nd part of array

Merge both parts by comparing elements of both the parts.


```
Def sort [arr, l, m, r]
    n1 = m - l + 1
    n2 = r - m
    L = [0] * [n1]
    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 & j < n2
        if L[i] <= R[j]
            arr[k] = L[i]
            i += 1
        else :
            arr[k] = R[j]
            j += 1
            k += 1
        while l < n1 :
            arr[k] = L[i]
            i += 1
            k += 1
        while j < n2 :
            arr[k] = R[i]
            j += 1
            k += 1
```

```
def mergsort (arr, l, r):
    if (l < r :
        m = int ((l + r)) /x )
        merge sort (arr, l, m)
    merge sort (arr, l, m)
    sort (arr, l, m, r)
    print (arr)
    n = tn (ars)
    mergesort (ars, 0, n-1)
    print (ars)
```

Output :-

[12 11 13 5 6 7]

[5 6 7 11 12 13]

#code

```
Print(" Rugved Padnekar")
set 1 = set()
set 2 = set()
for i in range (8,15):
        set 1.add(i)
for i in range (1,12):
        set 2.add(i)
print (" set 1:", set 1)
print (" set 2:", set 2)
print ("\n")
set 3 = set 1 | set 2
print (" Union of set 1 and set 2 : set 3 ", set 3)
set 4 =  set 1 & set 2
print (" Intersection of set 1 and set 2 : set 4", set 4)
print ("\n")
if set 3 > set 4:
    print (" set 3 is subset of set 4")
elif set 3 < set 4 :
    print (" set 3 is subset of set 4")
else :
   print (" set 3 is same as set 4")
   if set 4 < set 3:
       print (" set 4 is subset of set 3")
       print ("\n")
set 5 = set 3 - set 4
print (" Element in set 3 and not in set 4 : set 5: ", set 5)
print ("\n")
if set 4.isdisjoint (set 5):
```

Practical ~ 10

57

Aim: Implemention of set in Python

Algorithm:

Step 1: Define two empty set as set1 and set2 now use for statement providing the range of above 2 sets.

Step 2: Now add() method used for adding the element according to given range then print the sets after adding.

Step 3: Find the union and intersection of above 2 sets by using & (and), : (or) method. print the set after adding and intersection as set 3 & set 4

Step 4: If statement to find out the subset and superset of set 3 and set 4. Display the above set

Step 5: Display that element in set 3 is not in set 4 using mathematical operation.

Step 6: Use isdisjoint() to check the anything is common element is present or not. If not then display that it is mutually Exclusing event.

Step 7: Use clear() to remove or delete the set and print the set after clearing the element present in the set.


```
print("set 4 and set 5 are Mutually Exclusive \n")
set5.clear()
print("After applying clear, set 5 is empty set:")
  print("set 5=",set 5)
```

Output :-

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 and set 2 : set {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

Intersection of set 1 and set 2 : set 4 {8, 9, 10, 11}

set 3 is support of set 4

set 4 is subset of set 3

element in set 3 and not in set 4 : set 5 {1, 2, 3, 5, 6, 7, 12, 13, 14}

sets and set 5 are mutually exclusive

after applying clear, set 5 is empty set

set 5. set()

#code

```
class Node:
    global v
    global l
    global data
    def __init__(self):
        self.l = None
        self.data = l
        self = 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:
                    else:
                        h.l = newnode
                        print(newnode.data, "added on left", h.data)
                        break
                else:
                    if h.r != None
                        h = h.v
else:
    h.r = newnode
    print(newnode.data, "added on right of", h.data)
    break
```

Practical ~ 11

Aim: Program based on Binary search tree by implementing Inorder, Preorder & Postorder, Transversal

Theory: Binary Tree = is a tree which supports maximum of 2 children for any node within the Tree. Thus any particular node can have either 0 or 1 or 2 children. Here it is for any binary tree that it is ordered such that one child is identified as left child and other as right child.

Inorder: (i) Transverse the left subtree, the left subtree inturn might have left and right subtrees.
ii) Visit the root node
iii) Transverse the right subtree and repeat it.

Preorder:
(i) Visit the root node
ii) Transverse the left subtree. The left subtree inturn might have left and right subtree.
iii) Transverse the right subtree. repeat it

Postorder:
(i) Transverse the left subtree. the left subtre inturn might have left and right subtrees
ii) Transverse the right subtree)
iii) Visit the root node.

Algorithm:

step 1: Define class node and define init() method with 2 argument. Initialize the value in this method.

step 2: Again, Define a class BST that is Binary Search Tree. with init() method with self argument and assign the root is None.

step 3: Define add() method for adding the node. Define a variable p that p = node (value)

step 4: Use if statement for checking the condition that root is none then use else statement for if node is less than the main node then put or arrange that in Left side.

step 5: Use while loop for checking the node is less than or greater than the main node and break the loop if it is not satisfying.

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

step 7: After this, leftsubtree and right subtree, repeat this method to arrange the node according to Binary Search Tree

```
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)
    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(7)
T.add(5)
T.add(12)
T.add(6)
T.add(70)
T.add(85)
T.add(90)
T.add(17)
T.add(13)
print("preorder")
T.preorder(T.root
print("inorder")
T.inorder(T.root
print("postorder")
T.postorder
print("Ruyved")
```

Output :

80 added on left of 100
70 added on left of 80
85 added on left of 80
10 added on left of 70
78 added on left of 70
60 added on left of 10
88 added on left of 85
15 added on left of 65
12 added on left of 15
preorder
100
80
70
10
60
15
12
78
85
88
inorder
10
12
15
60
70
80
85
88
100
postorder
10
12
15
60
70
78
80
85
100

step8 : Define Inorder(), Preorder() and Postorder() with root argument and use if statement that root is none and return that in all.

step9 : In Inorder, else statement used for giving that condition first left, root and then right node

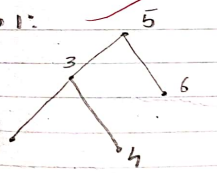
step10 : for Preorder, We have to give condition in else that first root, left and then right node.

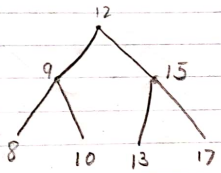
step11 : For Postorder, In else part, assign left right and then go for root node

step12 : Display the output and input.

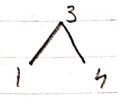
• Inorder : (LVR)

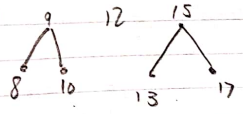
step 1:





step 2:

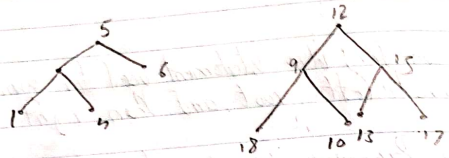




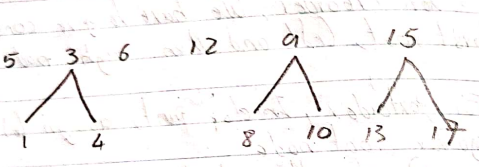
step 3 : 1 3 4 5 6 7 8 9 10 12 13 15 17

- Preorder : (VLR)

step 1 : 7



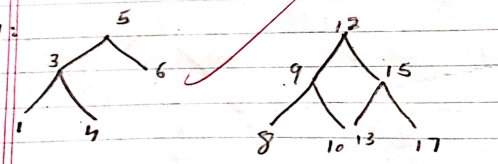
step 2 : 7 5 3 6 12 9 15



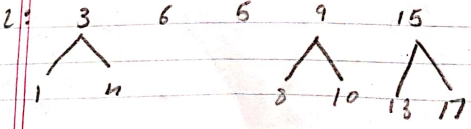
step 3 : 7 5 3 1 4 6 12 9 8 10 13 15 17

- Postorder : (LRV)

step 1 :

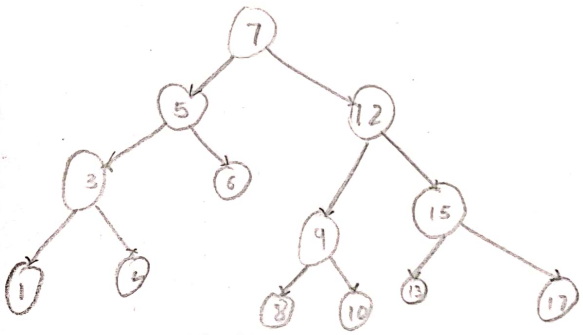


step 2 : 3 6 5 9 15 12 7



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

* Binary Search Tree



10/02/2022