Unsorted

```
s= int (input ("Enter the requested number"))
a= [10, 12, 9, 14, 17, 9]
for i in range (len(a))
        if (a[i] ==5):
            print "Required number is position ", break
if((s) = a[i]):
        print " The required number not fourd"
```

O/P

Enter the required number 4

Required found in position 2.

PRACTICAL No. 1

Aim-

Implement Linear Search to find on item in the list

Theory-

Linear Search.

It is one of the simpliest searching algorithm. In which targetted searching algorithm in which each item in the last

It is worst searching algorithm. with case item. Complexity, it is a fare approach. On other hand incase of ordered list, instead of searching the list in sequence. A binary search is used which is sort by examing the middle term.

LS is a technique. to compare each and every element with the key element, tb be found it, both of them. The algorithm that element found and its position is abo found.

Unsorted -

Algorithm-

step 1 - Create an empty list and assign it to be unsorted.

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 element in to list.

step 4 - Print the news list.

step 5 - Accept on element from user that the searched in the list.

step 6 - Use for loop in a range from '0' to total no. of elements to search the elements from the list.

step 7- Use if loop that the element in the list is equal to the element accepted from user.

step 8- If the element is found the print the statement that the element is found along with the element position.

step 9- Use another if loop to print that the element found it the element which is not found if the element their in the list

step 9- If the element is found the print the statement that the element is found along with elements position

step 10- Draw the output of given algorithm.

2) Started-

Algo

step 1 - Create empty list & design it to a variable accept total no. of elements to be inserted into the list from user say 'n'

step 2- Use for loop for using append() method to add the elements in the list. Use sorted method to sort the accepted element assign in increasing order the list.

step 3- Use if statement to give the range in which elements is found in given range then display the "element not found"

```
Sorted
s = list (input ("Enter the element"))
s = sort()
print 's'
a = int (input ("Enter the number to be searched"))
for i in range (len(s));
        if (a==s[i]):
          print "Number found! .";i
          break
else,
  print "No. not found'.
```

O/P

Enter element : 9,6,7,5,2,12,8 :

[5,6,7,8,9,12]

Enter the no. to be searched : 8

Number found : 8 . In position 3

Step 4: Then use also statement, it element is not found in range then satisfy the given value.

Step 5- Use for loop in range from 0 to total no of elements of to be searched before doing this accept on search from user using impact statements

Step 6- Use if loop that the element in the list is equal to element accepted from user. If the element is found then print the statement that the element is found algorithm the element position.

Step 7- Use another if loop to print the it to element which is their in the list

Step 8-

Attach the report and output of above algorithum.

PRACTICAL NO. 2.

Aim-

Implement Binary search to find an searches number in the list

Theory-

Binary Search is also known as half envolved search algorithmic search or binary in a search algorithm that finds the position of a target value within a sorted array. If your looking for the number which is at the end of list. then you need to search line list in linear turns you.

This can be avoided by using binary fashion search.

Algo.

1) Create empty list and assign it to variable
2) Using input methods, accept the range of gives list.
3) Use for loop, add elements in list using append methods.
4) Use sort() method to start the accepted element and assign in ordered list. Print the list often sorting.
5) Use if loop to give range in which element is found, in given range, then display "Element not found"
6) Then use else statement of statement is not found in range then satisfy the below.
7) Accept an argument & key of the element that element has to be searched.

```
a=[]
n=int(input("Enter the range")
for b in range(0,n):
 b=int(input("Enter no"))
   a.append(b)
   a.sort()
   print(a)
s=int(input("Enter the number of search"))
    if (s<[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", m)
       break.
       else.
           if (s<a[m]):
                l=m-1
        else:
            f=m+1
```

O/P

>>> Enter the range : 3

Enter the no. 2

[2]

>>> Enter number : 1

[1, 2]

Enter the number to search : 3

Element not found.

```
a=[]
b = int (input ("Enter no. of element"))
for x in range (0, b) =
        s = int (input ("Enter the element")
        a.append (e)
        print (a)
for 'i' in range (0, len (a)):
        for j in range (len (a)-1):
                if a[i] < a[j]
                temp = a[j]
                a[j] = a[i]
                a[i] = temp
print (" Element after sorting ", a)
```

O/P

>>> Bubble sort algo,

Enter no : 3

Enter element : 8

[8]

Enter element : 9

[8, 9]

8) Initialize first 0 & last element at the list as array is starting from 0, hence it is initiated less than the total correct.
9) Use for loop and assign the given arrange.
10) If statement in list and skill the element to be searched not found then find the middle element (m).
11) Else if the item to be searched is still less than middle term than initialize last(h) = (m)-1

Else
Initialize first (l) - mid (m) 1

Repeat the form 2 th e element strike the input and o/p of above algorithm.

PRACTICAL No. 3

Bubble sort

Aim- Implementation of bubble sort program on given list

Theory-

Bubble sort is based on idea of repeatedly comparing pairs of adjacent elements and swapping their position if they exist in wrong order. This is the we sort the given in ascending or descending order by comparing two adjacents elements at time.

Algorithm

1) Bubble sort algorithm stand by comparing first two elements of an array and swapping if necessary.
2) If we want to sort the element of array in ascending order than first element is greater the second then we need to swap the element.
3) If the element is smaller than second then we do not swap the element.

4) Again second and third elements are compared and swapped if it is necessary and that process go on until the last and seconds last element is compared and swapped.

Enter the element : 1

[8,9,]

Enter the no. after sorting :[1,8,9]

5) There are 'n' elements .to be sorters than the process .mentioned above Should be repeat (n-1) to get the required result.

6) Hick the output and input of above algorithum .of bubble sort stepwise.

PRACTICAL No.4

Quick Sort.

Aim-

Implement Quick sort in a recursive algorithm. based on the devices.

Theory -

The quick sort is a recursive platform algorithm. based on the device and conquer technique.

Algorithm-

1) Quick sort first select a value which is called pivot value, first element serve as fist pivit value since we known that last.

2) The position partition process will happen next. It will find the split point at the same time move other items other items to co-operate size of the visit either less than a greater than pivot value.

3) Partionting begins by locating two partition markers call them left mark right mark at the same speed and end of remaining items in the list

The good of position is to move items that on the wrong sides with respect to pivot value while also convering on the spirit value.

```
def Quick (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)
    def part (alist, first, last)
        pivot = alist (first)
        l = first+1
        r = last
        done = false
        while not some:
            while l <= alist [l] <= pivot:
                l = l+1
            while alist [r] >= pivot & r >= l
                r = r-1
        if r < l:
            done = True.
    else:
        t = alist [l]
        alist [l] = alist [r]
        alist [r] = t
        t = alist [first]
        alist = [first] = alist [r]
        alist [r] = t
            return r
    x = int(input "Enter range :"))
        alist = [ ]
    for b in range (0, x):
        b = int(input ("Enter element"))
```

```
        alist.append(b)
        n = len(alist)
    quick(alist)
    print(alist)
```

O/P

```
Enter range = 5
Enter element = 4
Enter element = 3
Enter element = 5
Enter element = 8
Enter element = 1
[1,3,4,5,8]
```

4) We begin by increasing leftmark, while we locate a value that is greater than greater than pivot then the decrement right mark until we find value that.

5) The point where right mark become less than leftmark we become less stop. The position of right work mark is how the split point.

6) The pilot value can be exchanged with the content of staff count and pivot value is not in place.

7) The quick sort function involves a recursive function, quick sort helper.

8) The quick sort helper logins with some base as the merge sort.

9) The length of list is to the less then equal on it already sorted.

10) If it is growth, then it can be parton on recursively sorted.

11) The partition function implemented the process described earlier.

12) Display and stick the coding and o/p of above algorithm.

PRACTICAL No. 5.

Aim-

Implementation of stack using python list

Theory-

A stack is linear data structure that can be represented in real world in the form of physical stack or a pike. The element in stack or a pile. The element in stack are added or removed from position i.e the temporary position. Thus, the stack works on LIFO principal as element that was inserted last will be removed first. A black seat has implemented by array as well as linked list. Stack, have these basic operation. Thus pop, peek. The operation of abbing and receiving the element is known as push and pop.

Algo

1) Create a class with instance variable items.
2) Define the init method with self argument and initialize the initial value and initial than to the empty list
3) Define method, push and pop. define under the class stack.
4) Use it condition to give the condition that length in given list is greater than the range list than print "Stack is Full"
5) OR else part statement an insert the element but pop from stack.

```
class stack :
    global tos :
    def __int__ .self
    staff .l = [0,0,0,0]
    staff .loss = 1
    dat push (self , data)
        n = len (self.l)
        if slf tos == n-1
            print ("stalk is full")
        else,
            staff.l[self .tos] = data
    def pop(self)
        if self tos<0:
            print ("Empty")
        else.
            t=self l [self.to]
            print ("Data = ", k)
        stack .l [self.tos] = 0
        self.tos = self.t:.
            if self tos<0
                print ("Ready")
            else,
                a-self 1 [slf tos]
                print ["data = ", a]
```

```
>s.push[10]
>s.push[20]
>s.push[30]
>s.push[40]
>s.push[50]
>>>s
[10,20,30,40,50]
data = 50
>s.pop()
Data = 40
>>>s.pop[]
Data = 30
>>>s.pop()
Data = 20
>>>s.pop()
Data = 20
>>>s.pop()
Data = 10
>>>s
[0,0,0,0,0]
s.peek()
Stack is empty.
```

6) Push method used to demand but pop method used to delete element from stack

7) If is pop method, value is less than 1, then return the stack is empty. or disc delete the element from top most position of stack.

8) First condition checks whether the no:of element and zero with the second case checks, whether to is asigned any value, then we can be sure ~~foreach~~ stack is empty.

9) Assign the element value in push method to print given value is ~~p~~ ~~soap~~ or not (pepped).

10) Attach the case & output the above.

PRACTICAL No 6.

Aim- Implementing a Queue using Python list.

Theary-

Queue is a linear data structure which has 2 reference front and rear implementing a queue using python list is simpliest as python the specified operation of queue. It based on principal that new element is inserted after rear and element of queue is deleted which is at front. In simple terms a queue can be described as data structure based on first in first out FIFO principal

Queue(): Creates a new empty queue

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

Dequeue(): Returns the element which was at front, the front moved to successive element. An dequeue operation cannot remove element if the queue is empty.

```
class queue:
    global r
    global n
    def __init__(self):
        self.r = 0
        self.f = 0
        self.l = [0,0,0,0]
    def enqueue(self, data):
        n = len(self.l)
        if self.r < n:
            self.l[self.r] = data
            self.r = self.r + 1
            print("Element inserted :-", data)
        else:
            print("Queue is full")
            self.n = 0
    def dequeue(self):
        n = len(self.l)
        if self.f < n:
            print(self.l[self.f])
            self.l[self.f] = 0
            print("Element deleted..")
            self.f = self.f + 1
        else:
            print("Queue is empty")
q = queue()
```

O/P

~~Element inserted 60.~~

q.add(10)

Element inserted :10

q.add(20)

Element inserted : 20

q.add(30)

Element inserted :30

Queue is full

q.remove()

20

element deleted :.

Algorithm :-

Step 1 - Define a class Queue and assign global variance item define init() method the init() value with the help of self arrangement

Step 2 - Define a empty list & define .enQueue() method with 2 arrangement 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 automatically incremented by 1.

Step 4 - Define .deQueue() with .self arguement under this, 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 that element from front side and increment it by 1

Step 5 - Now call the Queue() function and give element that ha to added in the empty list .by using .enQueue() and print the list after adding .and same for deleting and display the list after deleting the element from list.

PRACTICAL NO:7

Aim-
Program on evaluation of given string by using stack in python environment i.e. postfix

Theory-
The postfix expression is free of any parenthesis. further we took care of priorities of operation in the program. A given postfix expression can easily be evaluated using stack reading that expression is always from left to right in postfix

Algorithm-

Step 1- Define element as function the create a empty stack in python.

Step 2- Convert the string to list by using the string method split

Step 3- Calculate the length of string and print it

Step 4- Use for loop to assign the range of string the gare condition using if statement.

Step 5- Scan the token list from left to right token is an

```
def evaluate(s):
    k=s.split()
    n=len(k)
    stack=[]
    for i in range(n):
        if(k[i].isdigit(n):
            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 evaluated value:",r)
```

21

O/P

The evaluated value : $120

integer and push the value into y

step6- If the token is operator *, /, +, -, a, it will need to operand. pop the 'p' twice. The first pop the op and second pop in first operand.

step7- perform the arithmatic operation, push result back on the m

step8- when the O/P expression has been, completely processed, the result is on stack. pop the p and return the value.

step9- print the result of string after evaluation of postfix

step10:- Attach the operand O/P and I/P of above algorithm.

20/01/2021

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 stores the element in store the element in a note in a linear fashion but not necessarily contigous. The individual element of linked list called a node.

Node comprises of 2 parts.

1. Data
2. Next Data stores all information w.r.t, address, etc. In case of larger list, if we add/remove only element from the list all elements of list has to adjust itself every time we add it is very tedious task so linked list in used to solving this type of problems.

Algorithm-

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

2. The entire linked list can be accessed the first node of linked list. The first node of linked list in turn is referred by head pointer of the linked list.

3. Now that we known that we can transverse the entire linked list.

```
Class node
  global data.
  global next.
  def __init__(self, item):
    self.data = item.
    self.next = None.
  def __init__(self):
    self.s = NONE.
    new.node = node(item)
    def add(self, item):
      newnode = node(item)
    if self.s == None.
        self.s = newnode
    else:
      head = self.s
      while head.next != None.
        head = next.head.
      head.next = newnode
      if self.s == none
       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("Hi")
```

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

Now that we know that we can transversed using the node which is referred by the head pointer of linked list.
We should not use the head pointers to transverse the entired linked list because the head pointer is only, reference to 1st node in the linked list, modifying the reference of load pointer can lead to changes which is cannot revert back.
We may late the refernce of 1st node in our linked list, hence most of our linked list. so in order avoid making same unwanted changes to the 1st node, we will tempoorary node to transverse to entire linked list.
We will use this temporary node as a copy of currently transversing Since we are making temporary node a copy of current node the datatype of tempoorary node should also be node.
Now that current is referring to first node if we can to access 2nd node of list, we conprefer is at the node of 1st node.
But 1st node is refered by current, so we can transverse to 2nd node is h=h.nxt.
Similarly we can transverse rest of nodes in the linked list. Since the last node of linked list does not have any next node with value in the next field of the last node in None.
Our concern now is the linked list is refered by the fail of linked list, since the last node of linked list does not have any next node, the value in next field of last node in NONE.
Our concern nbw is to fixed find something terminating condition for while loop.
So we can now refer the last node of linked list self s=none.
We have to now see how to start conversion transversing the linked list & how to identify whether we reached

searched the last node of linked list or not

15. Attach the coding on input and output of above algorithm.

```
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 += 1
else,
    arr[k] = R[j]
      j += 1
      k += 1
  while i < n1 and j < n2;
        if L[i] <= R[j]:
            i += 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)
```

Practical No. 9.

Merge Sort.

Implementation of merge sort.

Theory - Like Quicksort, mergesort in a divide and conquer algorithm it divides input array in two halves, calls itself for the two halves and then merge the two sorted halves. The merge() function is used for merging two halves. The merge (arr, l, m, r) is key process that assumes that arr [l, m] and arr [m+1, r] are sorted and merges the two sorted sub-arrays into one

Algorithm

Merge sort is one of the most efficient sorting algorithm. It works on the principal of Divide and conquer. Merge sort repeatedly breaks down a list into several sublist until each sublist consists of single element and merging those sublist in a manner that results into sorted list.

Let us consider to understand the approach better:

1. Divide the unsorted list into 'n' sublist.
2. Each comprising of 1 elements
3. A list of 1 elements is supposed sorted implementing the merge sort
4. Repeatedly merge sublist to produce newly sorted sublist until there is only 1 subject remaining.

5. This will be the sorted list.
6. The top-down merge sort is approach is a methodology which uses recursion mechanism.
7. It starts at top and proceeds downwards, with each recursive turn to sort the array.
8. merge functions take 2 intervals.
9. One from start and one from mid
10. start and end are the starting and ending index of the current interval of Arr.

```
    sort (arr, l, m, r)
arr = [12, 23, 24, 78, 45, 86, 98, 42]
print (arr)
n = len(arr)
mergesort (arr, 0, n-1)
print (arr)
```

```
>>> [12, 23, 34, 56, 78, 45, 86, 98, 42]
    [12, 23, 56, 56, 42, 45, 86, 98, 42]
>>>
```

```
print("8k,1774")
set1 = set()
set2 = set()
for i in range(8,15):
    set1.add(i)
for i in range(1,12):
    set2.add(i)
print("set 1:", set1)
print("set2:", set2)
print("\n")
set3 = set1 | set2
print("Union of set1 & set2: set3", set3)
set4 = set1 & set2
print("Intersection of set1 and set2: set4", set4)
print("\n")
if set3 > set4:
    print("set3 is suppest of set4")
elif set3 < set4:
    print("set3 is subset of set4")
else:
    print("set 3 is same as set4")
if set4 < set3:
    print("set is subset of set3")
    print("\n")
set5 = set3 - set4
print("Elements in set3 and not in set4: set5", set5)
print("\n")
if set4.isdisjoint(set5):
    print("set4 and set5 are Mutually
                Exclusive \n")
```

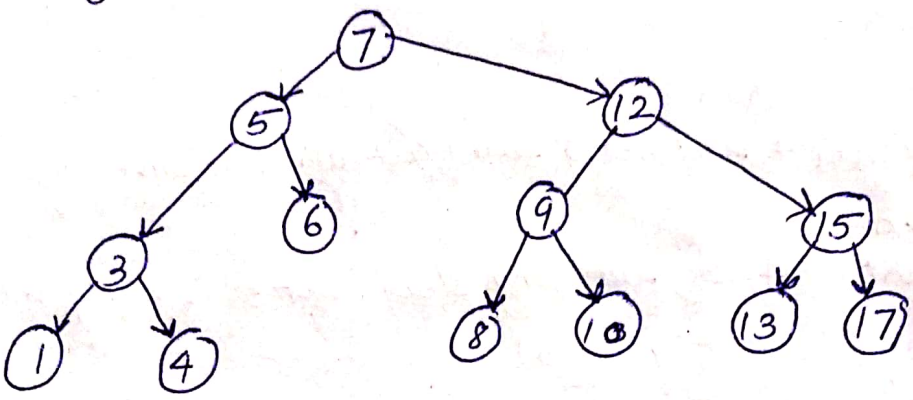
3-2-2020

Practical No. 10.

1. Define two empty set as set1 and set2 now use for statement providing the range of above 2 sets.
2. Now add() method for adding the element accordingly to given range then find sets after reading.
3. Find the union and intersection of above 2 sets by using & (and) i(or) method. point the set of set 3 and set 4 . display . the above set.
4. Ux if statement to find out the subset and superset of set 3 and set 4 . display the above set.
5. Display the element in set 3 is not in set 4 using mathematical operation.
6. Ux isdisjoint () to check that anything in element is present or not If not then display that it is mutually exclusive event.
7. Ux clear() to remove or delete the set, print the set after clearing the element present in the set.

```
print("set4 and set5 are Mutually Exclusive \n")
set5.clear()
  print("After applying clear ; set5 is empty set :")
  print("set5=", set5)
```

Binary Search Tree




```
class node:
    global r
    global l
    global data
    def __init__(self):
        self.l = None
        self.data = l
        self.r = None
class Tree
    global root.
    def __init__(self)
        self.root = name
    def add(self, val):
        if self.root == Name
            self.root == None(val)
        h = self.root
        while True:
            if newnode.data < h.data
```

Practical No. 11

Aim:- Program based on binary search tree by implementing, Inorder, preorder and postorder transversal

Theory- Binary Tree is a tree in which supports maximum of 2 children for any node can have either 0 or 1 or 2 particular there is another identity binary tree that is ordered such that one child is identified as left child and other a right child.

Inorder (i) Transverse the left subtree, The left sub tree inform 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 intro might have left and right subtree
(iii) Transverse the right subtree repeat it.

Postorder (i) Tranverse the left subtree. The left subtree intrum might have left and right subtree.
(ii) Transverse the right subtree
(iii) Visit the root node

Algorithm -

1. Define class node and define init() method with 2 arguments Initialize the value in this method.
2. Again define o class BT that is binary search tree with init() with self argument and assign the root node.
3. Define add() for adding the node. Define a variable p that p = node (value)
4. Use if statement for checking the condition the node is less than or greater than the main mode and break
5. Use while loop for checking the node is less than or greater than main gate and break the logic the loop if it is not satisfying.
6. Use if statement within than else statement for checking the node is greater main root put it into right side.
7. After this, left tree and right subtree, repeat this method to arrange the node accordingly in Binary Search Tree.
8. Define Inorder(), Preorder() and postorder() with root argument and use if statement the root in name and return that in all.
9. In order, else statement used for giving that condition first left, root & right node.
10. For ~~pre~~ postorder, we have to give information in else that first root, left and the right node.

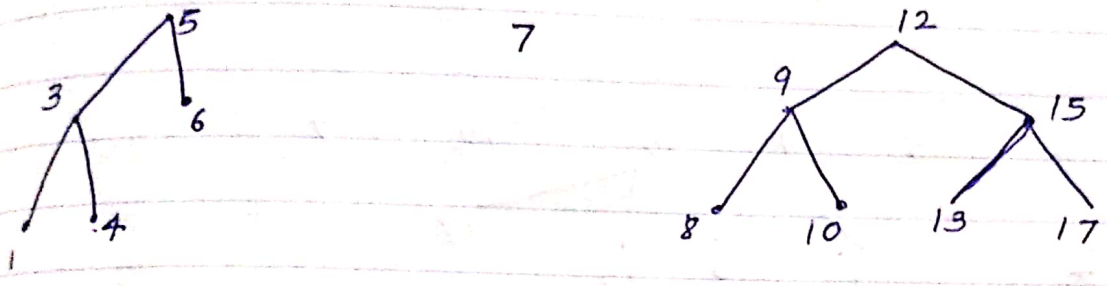
```
        if newnode.data < h.data
            if h.l != None :
                h = h.l
            else :
                h.l = newnode
            print (newnode.data, " added on left of ", h.data)
            break
        else :
            if h.r != None :
                h = h.r
        else :
            h.r = newnode
            print (newnode.data, " added of right . h.data)
            break
    def preorder (self, start) :
        if start != none
            print (start.data)
            self.preorder (self, start)
            if start = None
                self.inorder (start.l)
                print (start.data)
                self.inorder (start.r)
                    if start != None;
                        self.inorder (start.l)
                        self.inorder (start.r)
                        print (start.data)
```

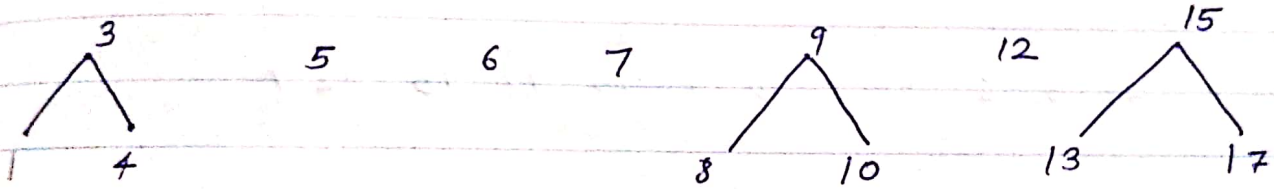
```
print(start.data)
T= tree ()
T.add (7)
T.add (5)
T.add (12)
T.add (3)
T.add (6)
T.add (9)
T.add (15)
T.add (1)
T.add (4)
T.add (8)
T.add (10)
T.add (13)
T.add (17)
print("preorder")
T.preorder (T.root)
print ("inorder")
T.inorder (T.root)
print ("postorder")
T.postorder(T.root)
print ("ok)
```

For postorder, in else part, assign left then, right and then go to root node.

Display the output and input of above algorithm.

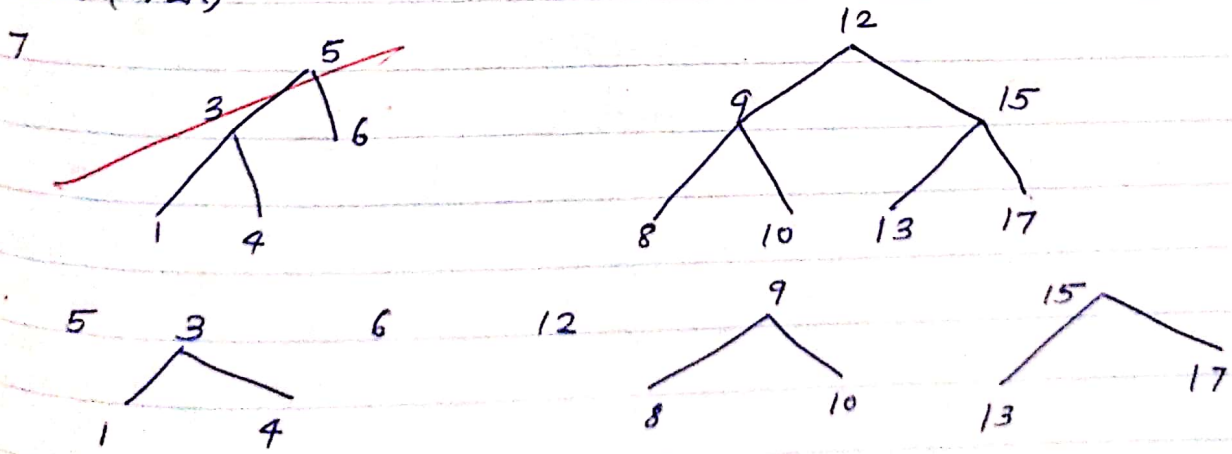
Inorder (LVR)





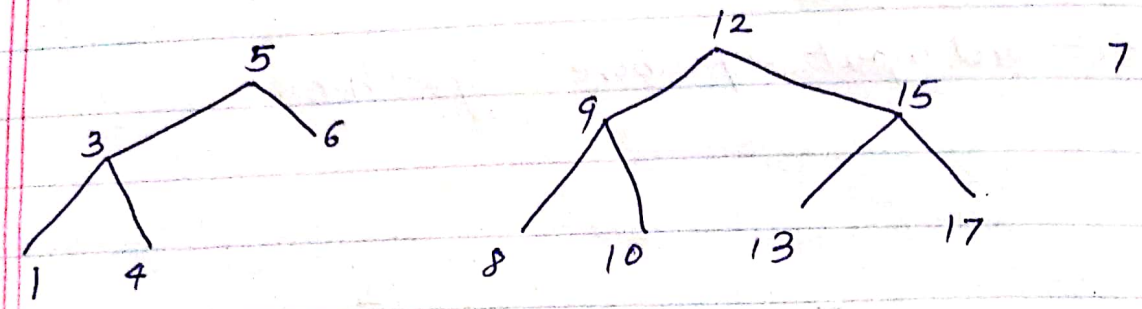
1 3 4 5 6 7 8 9 10 12 13 15 17.

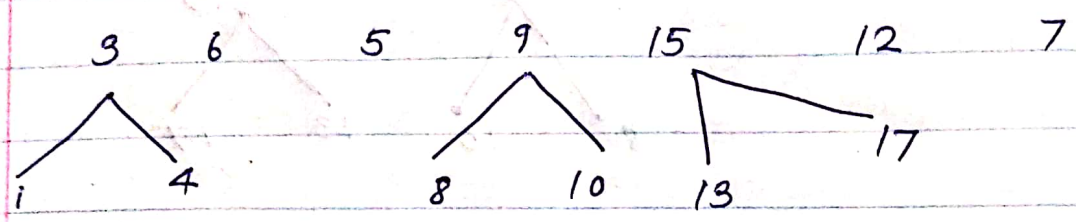
Preorder (VLR)



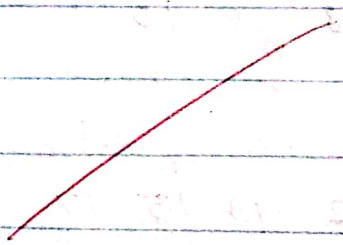
7 5 3 1 4 6 12 9 8 10 15 3 17

Postorder (LRV)





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



>77

5 added on left of 7

12 added on left of 7

3 added on right of 5

6 added on left of 5

9 added on right of 12

15 added on right of 12

1 added on left of 12

4 added on left of 3

8 added on left of 9

10 added on right of 9

13 added on left of 15

17 added on right of 15

preorder

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

inorder

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

postorder

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

8k.

m
10/02/2020.