Prac: 01

Aim: Implement linear search to find are item in the list

Theory:

linear search.

linear search is one of the simplest searching algorithm in which integrated item is sequentially matched with each item in the list.

It is worst searching algorithm with worst case time complexity. It is a force approach on the other hand. incase of an ordered list instead of searching the list in sequence A binary search is used which will start by examing the middle term.

Linear search is a technique to compare each and every element with the key element to be found, if both of them matches, the algorithm returns that element found and its position is also found.


```
# sorted array.
# code
def linear (arr, x)
    for i in range (len (arr))
        if arr [i] == x
            return i
input = (input ("Enter elements in array:").
        split ( )
array : [ ]
for ind in inp:
    array. append (int (ind))
print ("elements in array are :", array)
array sort ( )
x1 = int (input ("enter element to be
        searched" ))
x2 = linear (array, x1)
if x2 == x1
    print ("Elements found
        at position", x2)
else :
    Print ("Element not found")
```

>>> Enter elements in array 1 2 3
>>> Elements in array are : [1, 2, 3,
>>> Enter Element to be searched : 2
the element is found at position 1

unsorted array.

```
def linear (arr, x)
    for i in range (len(arr)):
        if arr[i] == x:
            return i
inp = input ("Enter elements in array:") .split()
array = [ ]
for ina in inp
    array.append (int (ina))
print ("elements in array are", array)
x1 = int (input ("Enter the elements to be searched:"))
x2 = linear (array, x1)
if x2 == x1:
    print ("element found at location", x2)
else:
    print ("element not found")
```

>>> Enter element in array? 3 2 4 5

>>> elements in array are: [3 2 4 5

>>> element to be searched: 4

element found at location 2

unsorted:

Algo

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 for adding the elements into the list.

Step 4: print the new list

Step 5: Accept an element from the user that to be searched 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.

Scanned with CamScanner

Scanned with CamScanner

e] Sorted linear search :

sorting means to arrange the element in increasing or decreasing order.

- Algorithm :

Step 1 : create empty list and assign it to variable

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

Step 3 : use for loop for using append() method to add the elements in the list.

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

Step 5 : use if statement, if 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 cond^n.

Scanned with CamScanner

Step 7: use for loop in range from 0 to the total no. of elements to be searched before doing this accept on search no from user using input statement

Step 8: use if loop that the elements in the list is equal to the element accepted from the user.

~~Step 9. If the el~~

Step 9: If the element is found then pr 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 accep from user is not their in the list.

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

19/12/19

Scanned with CamScanner

```
# binary search
# ~~Mode~~

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

array = input ("Enter the sorted list of numbers").split( )
arr = [ ]
for ind in array;
    arr. append (int (ind))
key = int (input("Enter element to search:"))
index = binary (arr, key)
if index < 0:
    print ("Element not found")
else:
    print ("Element found at index", index).
```


- **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 chop is a search algorithm that finds the 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: Create empty list and assign it to a variable.

Step 2: Using input method, accept the range of given list

Step 3: Use sort ( ) method to sort the accepted elements and assign it in increasing ordered list print the list after sorting

Step4 : use sort () method to sort the a accept element and assign it increasing ordered list. Print the list after sorting

Step5 : use if loop to give the range in which element is found as given range then display a message "Element not found".

Step6 : Then use else statment, if statment is not found in range then satisfy the below condition

Step7 : Accept an anguement & key of the element that element has to be searched

Step8 : Initialize first to 0 and last to last element of the list as array is starting from 0 hence it is initialised less than the total count

Step 9 : else for loop and assign range.

>>> Enter the element in array: 42
3 5 10 12 15 20

>>> Element to be search : 12
12 element was found at index 3

>>> Enter the elements in array:
3 6 1 5 6 7 8

>>> Elements to be search : 2
Element was not found

Scanned with CamScanner

Step 10 : If Statement in list and skill the element to be searched

Step 11 : Else if the item to be searched is still is less than the middle term then.
Initilize last (n) – mid (m):)
Else
Initiaze first (1) = mid (m)-1 .

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

Scanned with CamScanner

Bubble Sort.

- **Aim:-** To implement bubble sort program on given list.

- **Theory:-** Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements & 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:-**

Step 1:- Bubble sort algorithm starts 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:- If the first element is smaller than second then we do not swap the element.

Step 4:- Again second and third elements are compared and swapped if its necessary and this process go on

CODE

```
inp = input ("Enter elements:").split ( )
arr = [ ]
for ind in inp :
    arr - append (int (ind))
print ("Elements of array before sorting them:",array
n = len(arr)
for i in range (0,n):
    for j in range (n-1):
        if arr[i] < arr[i] :
            temp = arr [i]
            arr [i] = arr [i]
            arr [i] = temp.
print ("Element of array after bubble
        sort =", arr)
```

24/12/24

>>> Enter elements: 2 3 1 , 5

>>> Elements of array before sorting: [2,3,1

>>> Elements of array after sorting: [1,2,3,5

Scanned with CamScanner

until last and second last element is compared and swapped.

(5) If there are n elements to be sorted then the process mentioned above should be repeated n-1 times to get the required result.

24/12/14

Scanned with CamScanner

Practical: 04

Aim : Implementation of stack using Python list

Theory : A stack is linear data structure that can be represented in a real world form by physical stack or pile. The elements in the form stack are the top and position. This is works in the lifo principle (last in first out). It has 3 basic operations namely ; push pop peek.

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 the initial value and then initialize to an empty list

Step 3. Define method push and pop under the class stack.


```
class stack :
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 (" 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
def peek(self):
x = stack()
```

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, 0]
```

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.

Step 5. Else stack to print a statement- or ~~ago~~ insert the element into the stack and initize 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 is empty, or else delete the element from stack at topmost position.

Step 8. Assign the element values, in Push method, and print the given value

Step 9. Attach the input an output of above algorithm.

09/10

Scanned with CamScanner

Step 10: Test condition checks whether the no. of elements are zero which is second case whether top is assigned any value. If top is not assigned any value then it means that the stack is empty

Scanned with CamScanner

Practical : 05

Aim : Implement quick sort to sort the given list.

Theory : The quick sort is a recursive algorithm then based on the divide and conquer technique.

Algorithm :

Step 1 : quick sort first selects a value, which is called pivot value first value element. solve as our first pivot value since, we know that the 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 the same time move other items to the appropriate side of the list either lets then one greater than pivot value.

Scanned with CamScanner

Step 3: partitioning begins by locating two position mark lets call them leftmark and right mark at the begning and end of remaining items in the list. The goal of the partition process is to move items that are on a wrong side with respect to ~~pivot~~ pivot value while also converging on the split point.

Step 4: we begin by increasing left mark until we locate a value that is greater than the p.v we then decrement right mark until we find value that is less than the p.v. At this point we have discovered two items that are out of place with respect to eventual split point

Step 5: At the point where rightmark becomes less than left mark we stop. The position of right mark is now the split point.

code

```
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 [j], arr [i]
    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)
        quicksort (arr, pi+1, high)
x1 = input (" enter element in the list ").split()
alist1 = [ ]
for b in x1:
    alist1.append (int (b))
print (" element in list are:", alist )
n = len (alist)
quicksort (alist, 0, n-1)
print (" elements after quick sort are", alist1)
```

Output

Quick sort

enter elements in list 21 22 30 40 50 6

elements in list are [21, 22, 20, 30, 40, 50, 24, 56]

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

Step 6 : The pv can be exchanged with the content of split point and p.v is now in place

Step 7. In addition all the items to left of split point is than p.v and all the items to left to right of split point are greater than p.v. The list can now be divided at split point and quick sort can be invoked recursively on the two values.

Step 8 : The quickest function invokes a recursive function quick sort helper.

~~Step 9 : quick~~

Step 9 : quick sort helper, begins with same base as the merge sort.

Step 10 : If length of the list is less than 0 or equal one list is already sorted

Step 11: If it is greater than it can be partitional and recursive function.

Step 12: The partition function implements the process described earlier

Step 13: Display and state the coding and output of algorithm

MM
23/01/2020

Scanned with CamScanner

Practical 6 Queue

Title: Implementing a queue using python list

Theory: Queue is a linear data structure which has 2 reference front and rear, Implementing a queue using python list is the simplest as the specific operations of the queue.

Queue(): creates a new empty queue

Queue(): Insert an element at the rear of the queue and similar to that of insertion of linked list

~~DeQueue~~(): Returns the element which was at front is moved to successive element

Algorithm

Step 1: Define a class queue and assign global variables then define (__init__) method with self arguments unit() assign as initialize the initial value with the help of self arguement.

Step 2: Define a empty list and define enqueue () method 2 arguements assign the length of empty line

Step 3: use of statement that length is equal to rear is full or else insert element which is empty list and increment by 1

Step 4: Define queue() with self argument that front is equal then display turning the value the element from front side and increment it by 1.

Step 5: Now call the queue() fn and give the element that has to be added in the empty list and print after adding.


```
#code
class queue:
    global r
    global f
    def __init__(self):
        self.l = 0
        self.b = 0
        self.l = [0, 0, 0, 0, 0]
    def add (self.data):
        n = len (self.l)
        if self.r < n - 1:
        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
```

Scanned with CamScanner

code

```
def evaluate(t):
    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 evaluated value is", r)
```


practical no. 07

Aim: Program on evalution of given string by using stack in python environment: postfix

Thory: The postfix expression is free of any parathesis further the work case by the priority of the expression is always from left to right in postfix

Algorithm:

Step 1: Define evalute 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 and print it

Step 4: Scan the token left to right

Step 5: If token is an operand convert it from string to integer and push the value onto the 'p'

Step 7: If token is an operator +, /, -, * it will need two operands. Pop the 'P' twice. The first pop is second operand and second pop is the first operand

Step 8: Perform the arthmetic oper.n push the result back on 'm'

Step 9: When the input expression has been completely processed

Step 10: Print output & input of above algorithm.

output

the evaluate value is 6

Scanned with CamScanner

H code

```
class node:
    global data
    global next
    def __init__(self, item):
        self.data = item
        self.next = None

class linked list:
    global .6
    def add c(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 add B(self, item)
        newnode = node = node(item)
        if self.s = = None:
            self.s = Newnode
        else:
            newnode.next = self.s
            self.s = newnode
```


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 is storing the elements in a node in a linear fashion but not sequence of the linked list.

① Data ② Next. data stores all the next refers to the next node

~~Algorithm~~

Step 1: Traversing of a linked list means positioning all the nodes in the linked list to perform same operation on them

Step 2: The entire list may be accessed as 1st node.

Step 3: The entire linked list can be transversed using node by head pointer.

Step 4: Now that we know, that we transverse 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 som reference to the 1st node.

Step 6: we will use temp node as a copy of node. we currently transversing.

Step 7: Since, we are making temp node a copy of current node the data type of temp node should also be node.

Step 8: if we want to access 2nd node of list we need to say the next 1st node.

Step 9: But the 1st node is referred by current so we can transverse to 2nd node as n = x.next

Step 10: Similarly, we can transverse rest of nodes in linked list


```
def display (self):
    head = self.s
    while head.next != None
        print (head.data)
        head = head.next
    print (head.next)
start = linklist()
```

~~output~~

>>> start.add(180)
>>> start.add(80)
>>> start.add(60)
>>> start.add(50)
>>> start.add(40)
>>> start.add(20)
>>> start.add(20)
>>> start.display()

Scanned with CamScanner

Step 11 - Our concern now to find terminating condition for while loop.

Step 12 : The last node in the linked list is accessed by tail

Step 13: We can refer to last node of linked list.

Scanned with CamScanner

Practical - 9

Aim: Implementation of merge sort using Python.

Theory: Merge sort is a divide and conquer algo. It divides the array into halves and the merge two sorted halves. The merge() function is used for merging two halves.

(*) Algorithm

Step 1: The list is divided into left and right in each recursive call until adjacent element.

Step 2: Now begin the sorting process. The i & it iterators transverse the two halves in each all.

Step 3: ~~Now begin~~ If the value of i is smaller then the value at j is assigned (i+1) sort and is incremented

Step 4: In this way the values being assigned through h[k] arr will be sorted

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 [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] = L [i]
            j += 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)
```

```
merge sort (arr, l, m)
merge sort (arr, m+1, r)
sort (arr, l, m, r)

arr = [12, 23, 34, 56, 56, 78, 45, 86, 98, 42]
print (arr)
n = len (arr)
merge sort (arr, 0, n-1)
Print (arr)
```

Output

[12, 23, 34, 56, 48, 45, 86, 98, 42]
[12, 2, 3, 56, 56, 42, 45, 48, 86, 98]

STEP 6: After the sorting, the numbers are placed in particular order. Thus the merge has been implemented

Practical 10 :-

Aim: Implementation of sets using Python.

Algorithm:

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

Step 2: Now use add () method for add" of elements acc to given range then print set for add".

Step 3: Find the union and intersection of above 2 sets by using method then print the sets of union & intersection of set 3.

Step 4: use if statement to find out the subset and superset of set 3 and 4. display sets.

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

code

```
set 1 = set ()
set 2 = set ()
for i in range (0,15):
    set 1.add(i)
for i in range (1,12):
    set 2.add(i)
print ("set 1 :", set1)
print ("set 2 :", set2)
print ("\n")
set 3 = set 1 | set 2
print ("union of set 1 and 2 :", set3)
set 4 = set 1 & set 2
print ("intersection of set & 2 :", set4, sets)
print ("\n")
if set 3 > set 4:
    print ("set 3 is superset of set 4")
elif set 3 < set 4:
    print ("set 3 is same as set 4")
if set 4 < set 3:
    print ("set 4 is subset of sets")
    print ("\n")
set 5 = set 3 - set 4
print ("Elements in set 3 are not in set 4 : set 5", set 5)
print ("\n")
```

Scanned with CamScanner

if set — is disjoint (set5):
print ("set 4 and 5 are mutually exclusive")

set.clear()
print ("After applying clear set is empty set(")
print ("set 5:", set5)

Output

set 1 : {8, 9, 10, 11, 2, 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, 13, 14}

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

set 3 is superset of set 4
elements in set 3 are not in set 4.
set 5 : {1, 2, 3, 4, 5, 6, 7, 12, 13, 14}
set 4 and 5 are mutually exclusive
After applying clear, set 5 is empty set.
set 5 = set()

Practical : 11

Aim : Program based on binary search tree by implementation, inorder, preorder & post-order traversal.

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. There is another identity of binary tree that it is ordered such that one child is identified as left child

Inorder : Traverse the left subtree, the left subtree root node might have left and right subtree.

Preorder : visit the root node traverse the left subtree and right subtree.

Postorder: Transverse the left subtree the left subtree in return might have left and right substance

Algorithm:

1) define the class node and init () which has 2 argument initialization the value is ship method.
2) Again define a class BST that is binary search Tree with init () with self argument and assign the root is None.
3) Define add() for adding node define a variable 'p' and p.node (value).
4) use if statement for checking cond that root is None then use else statement for if node is less than main node and break the loop if it is not satisfying.
5) use while loop for checking the node is less then or greater
else if statement within that if statement for checking if greater then main out

code


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

def Post order (root):
    if root == None:
        return
    else:
        Print (root.val)
        Postorder (root.left)
        Postorder (root.right)

t = BST()
```


1) After this for left or right tree and right tree, this method to arrange the node according to binary tree.

2) Define inorder(), Preorder(), postorder() as argument with root. None and that root is inorder, use else return that's all for giving that condition first left node and then right root.

3) For Preorder, you have to give condition else that first left root and right node.

4) For Postorder in else part assign left and right root so display input & output.

Output

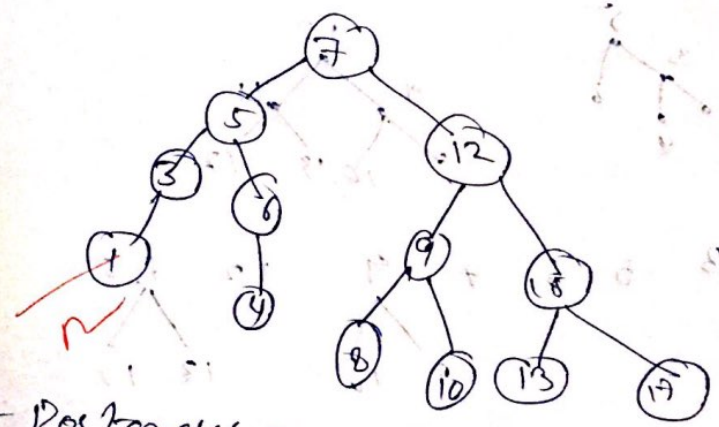
>>> t.add(1)
root is added successfully
>>> t.add(2)
node is added side
>>> t.add(4)
node is added to right side
>>> t.add(6)
node is added to left side
>>> t.add(3)
node is added to right side
>>> print("\n Inorder:", Inorder(t.root))
Inorder:
1
2
3
4
5

Inorder: None.

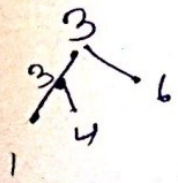
>>> print("\n preorder:", preorder(t.root))
1
2
3
4
5

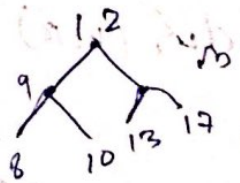
>>> print("Post order:", Post order(t.root))
Postorder:
3
5
4
3
2
1

Post order: None

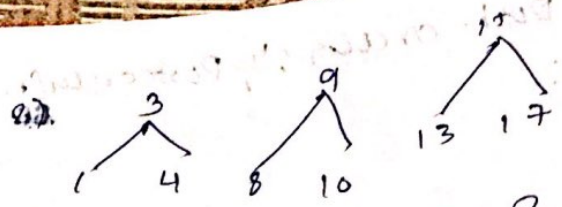


Postorder(root,





3 . 6 5 9 15 12 7

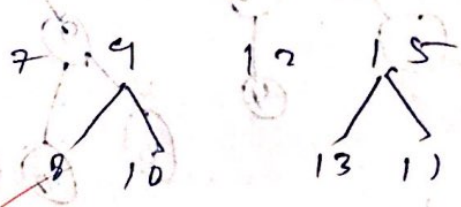


3) 1 4 3 6 5 8 10 9 13 7 15

Inorder (LVR)

1)

2) 3 5 6 7 9 12 15

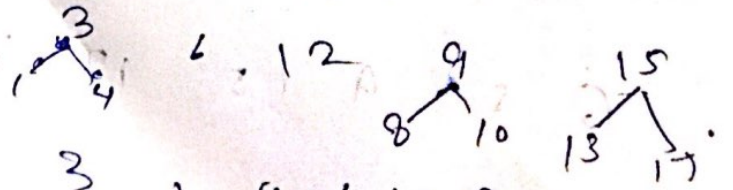


3) 1 3 4 5 6 7 8 9 10 12 13 15 17

Preorder (VLR)

1)

2) 7 5



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