Exam Seat No. _____

THAKUR COLLEGE OF SCIENCE & COMMERCE

NAAC Accredited with Grade "A" (3rd Cycle)

THAKUR TRUSTS

ISO 9001 : 2015 Certified

Degree College

Computer Journal

CERTIFICATE

SEMESTER II **UID No.** _____

Class FYBSc-CS **Roll No.** 1864 **Year** I

This is to certify that the work entered in this journal is the work of Mst. / Ms. Akshatha SWAMY

who has worked for the year 2019-2020 in the Computer Laboratory.

Teacher In-Charge

Head of Department

Date : _____

Examiner

Practical NO-1

Aim - Implement linear Search to find an Item in the list.

Theory -

Linear Search

Linear Search is one of the simpliest searching algorithm in which targeted 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 in sequence. A binary search is used which will start by examining 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.

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 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 the 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.


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

```
>>> Enter element in array: 3 2 4 5 1
>>> Element in array are: [3 2 4 5 1]
>>> Element to be searched: 4
    Element found at location 2
```

SORTED ARRAY
Algorithm

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

Step 2 - Now use if conditional statement to check whether the given number by user is equal to the elements in the array.

Step3 - If the conditional in ~~se~~ step 2 satisfies, return the inden 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 and to split the values.

Step5 - Now initialize a variable as empty array

Step6 - Now use for conditional statement to append the elements given as input by user in empty array.

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

Steps - Again initialize a variable to save the defined function.

Step 9 - Use if conditional statement to check if the variable in step 8 matches with the element you want to find then print the index corresponding to the element. If the condition doesn't satisfie then print that element is not found.

```
def linear(arr, n):
    for i in range(len(arr)):
        if arr[i] == n:
            return i
inp = (input("Enter elements in arrange:")).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")
```

```
>>> Enter element in array: 1 2 3 5
>> Elements in array: [1 2 3 5]
>> enter element to be
```

```
a = list(input("Enter the list of elements:"))
a.sort()
n = len(a)
s = int(input("Enter the number to be searched:"))
if (s > a[n-1] or s < a[0]):
    print("Element not found")
else:
    f = 0
    l = n-1
    for i in range(0,n):
        m = int((l+f)/2)
        if s == a[m]:
            print("The element is found at:", m)
            break
        else:
            if s < a[m]:
                l = m-1
            else:
                f = m+1
```

```
>>> Enter the list of elents : 15 24
Enter the number to be searched : 5
The element is found at: 1
```

PRACTICAL NO - 2 041

AIM : Binary 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 for loop, add elements in list using append() method.

Step 4 - Use sort() method to sort the accepted element and assign it in increasing ordered list print the list after sorting.

Step 5 - Use the if loop to give the range in which element is found then display a message "Element not found".

Step 6 - Then use else statement, if statement is not found in range then satisfy the below condition.

Step 7 - Accept an argument and key the element has to be searched.

Step 8 - Initialize first to 0 and last element of the list as array is satisfying from 0 hence it is initialized 1 less than the total const

Step 9 - Use for loop and assign the given range.

Step 10 - If statement is list and still the element to be searched is not found then find the middle element (m).

step 11 - Else if the item to searched is still less that the middle ~~team~~ term then initialize mid(m)-1 Else initialize f(l) = mid(m).

Theory -

Binary search is also known as half interval search logarithmic search binary chop is a search algorithm that finds the position of a target 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.

```
inp = input("enter a element: ").split()
a = []
for ind in inp:
    a.append(int(ind))
print("element before sorting", a)
n = len(a)
for i in range(0,n):
    for j in range(n-1):
        if a[i] < a[j]:
            tmp = a[j]
            a[j] = a[i]
            a[i] = tmp
print("element after sort", a)
```

```
>>> enter a element; 2 5 8 6
elements before sorting [2, 5, 8, 6]
elements after sort [2, 5, 6, 8]
```

```
#code for stack
print ("akshatha swamy")
class stack:
    global tos
    def __init__(self):
        self.l =[0, 0, 0, 0, 0, 0, 0, 0]
        self.tos = -1
    def push(self, data):
        n = len(self.l)
        if self.tos == n-1:
            print ("stack is full")
        else:
            self.tos= 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
s=stack.
```

output

2/1/20

PRACTICAL- 4

- **Aim** : Implementation of stacks 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. 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 as linked list. Stack has three basic operation: push, pop, peek. The operations of adding and removing 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 4 - Use if statement to give the condition that if length of the given list is greater than the range of list then print stack is full.

Step 5 - on else print statement as insert the ~~else~~ element into the stack & initialize the value.

Step 6 - Push multed 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 whethere the no of elements are zero while the second case whethere tos is assigned any value. If top is not assigned any value, then can be sure that stack is empty.

Step 9 - Assign the element value in push method to and print the given value in pop or not

Step 10 - Attach the input & output of above algorithm.

output

Akshatha Swamy

046

```
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s.push(50)
>>> s.push(60)
>>> s.push(70)
>>> s.l
>>> s.l
[10, 20, 30, 40, 50, 60, 70]
>>> s.pop()
data = 70
>>> s.l
[10, 20, 30, 40, 50, 60]
```

PRACTICAL-5

Aim → Implement Quick Sort to Sort the given list.

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

Algorithm

STEP1 - Quick Sort first select a value, which is called pivot value first value element. Serve as our first pivot value since we know that first will eventually end up as last in that list

STEP2 - Partitioning begins by locating two positions

STEP2 - The partition process will happen next. It will find the split point & at the same time move other items to the appropriate side of the list either lets then or greater than pivot value.

STEP3 - Partitioning begins by locating two position makes lets call them left mark & right mark at the begining & end of remaining items in the list. The goal of the partition process is to more items then are on to the wrong side with repeat to first value while use converging on the split point

STEP4 - We begin by increasing leftmark until use locate is value that is greater than p.v value, we then decrement value (right mark) until we find value that is less than the p.v. At this point we have discovered two items

```
s.push(70)
s.push(80)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```

OUTPUT

stack is full
data = 70
data = 60
data = 50
data = 40
data = 30
data = 20
data = 10
Stack is empty

23/01/2020

CODE

```
Class Queue:
    global r
    global f
    def __init__(self):
        self.r = 0
        self.f = 0
        self.l = [0, 0, 0, 0, 0, 0]
    def __add__(self, data):
        n = len(self.l)
        if self.r < n-1:
            self.l[self.r] = data
            self.r = 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-6

Aim - Implementing a Queue using python list.

Theory - Queue is a linear data-structure which has reference front & rear. Implementing a queue using python list is the simplest. Because it provides inbuilt functions to perform the specified operations of the queue. It is based on the principal that a 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 a data-structure based on FIFO (first in first out).

- Queue() : Creates a new empty queue
- Enqueue() : Insert an element at the real of queue & similar to that of insertion of linked list using tail.
- Dequeue() : Return element which was at the front. The front element is moved to the successive element. It cannot remove element if queue is empty

Algorithm:

Step1 - Define a class queue and assign global variable the def init() method with self argument. Assign or initialize the first value with help of self argument.

Step2 - Define a empty list and define enqueue() method with 2 arguements assign length of empty list.

Step3 - Use if statement that len is equal to rear that Queue is full or else insert the element in empty list or display that Queue element added successfully & increment by 1.

Step4 - Define dequeue with self argument under the use if statement that front is equal to len of the list then display Queue is empty or else give front is at zero & using that delete the element from front side & increment it by 1.

Step5 - Now call the queue() funn & give the element that has to be added in empty list by using enqueue() & print the list after adding & same for deleting & displaying the list after deleting element from list.

#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()
evaluate(s)
print("The evaluated value is :", s)
```


PRACTICAL-7

Evaluation of POSTFIX EXPRESSION

AIM - Program on evaluation of given string by using stack in python i.e POSTFIX.

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

ALGORITHM.

Step1 - Define variable to evaluate function then create empty list in python.

Step2 - Convert the string to a list by using string method 'split'

Step3 - Calculate and print the length of string

Step4 - Use for loop to assign the range of string then give condition using if statement.

Step5 - Scan the taken list from left to right. If taken is on operand, convert it from string to using int & push value into 'p'

Step6 - If the token is an operator *, +, /, -, it will need two operands. Pop the 'p' twice. The first pop is 2nd operand and the second pop is 1st operand

Step7 - Perform the Arithmetic operation - push the value II back to 'm'

Step8 - When the input expression has been completely processed the result on the stack is pop is 'p' & return value.

Step9 - Find the result of string after the evaluation of postfix

Step10 - Attach output & input of above algorithm.

Output
The evaluated value is 62

code -

```
class node:
    global data
    global next
    def __init__(self, item):
        self.data = item
        self.next = None
class linked list:
    global s
    def __init__(self):
        self.s = None
    def add(self, item):
        newnode = node(item)
        if self.s == None:
            self.s = newnode
        else:
            head = self.s
            while head.next != None:
                head = head.next
            head.next = newnode
    def addB(self, item):
        newnode = node(item)
        if self.s == None:
            self.s == newnode
        else:
            newnode.next = self.s
            self.s = newnode
    def display(self):
        head = self.s
        while head.next != None:
            print(head.data)
            head = head.next
            print(head.data)
```


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 element in the node in a linear fashion but not necessarily contigous. The individual element of the linked list called a node. Node comprises of 2 parts ① Data ② Nent. Data stores all info with the element for eg- roll no, name, address etc. whereas the nent refers to the nent 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 very tedious task.

Algorithm

Step 1 - Traversing of linked list means using all the nodes in the list in order to perform some operation on them.

Step 2 - The entire linked list means can be accessed from the first node of the linked list. The 1st node of the linked list is returned by the nent pointer of the linked list.

Step 3 - Thus the entire linked list can be traversed using the node which is referred by the head pointer of linked list.

Step 4 - Thus the entire linked list can be transversed using the ~~head pointer~~ node which is referred by the head pointer of the L.L

Step 5 - We should not use the

start = linked list ( )

OUTPUT -

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

~~20~~
30
40
50
60
70
80

13/02/2020.

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


PRACTICAL-09

Merge Sort

Aim - Implementation of Merge Sort by python list.

Theory - Merge sort is a divide & conquer algo. It divides i/p array in two halves, calls itself for the two halves and the merges 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 [l...m] and arr [m+1...r] are sorted & merges the two sorted sub-array. The key algo first moves from top to bottom, divides the list into smaller & smaller parts until only the separate element remain.

Algo

Step1 - The list is divided into left & right in each recursive call until two adjacent elements are obtained.

Step2 - Now begins the sorting process. The i & j iterators traverse, the two halves in each call. The k iterator traverses the whole list & makes change.

Step 3 - If the value at i is smaller then the value at j, L[i] is assigned to the arr[l+i] slot and i is incremented. If not the R[j] is chosen.

Step 4: This way the values being assigned through [l+i] are all sorted.

```
while j < n2:
    arr[k] = R[j]
    j += 1
    k += 1
def mergesort(arr, l, r):
    if l < r:
        m = int((l+(r-1))/2)
        mergesort(arr, l, m)
        mergesort(arr, m+1, r)
        sort(arr, l, m, r)
arr = [12, 23, 34, 56, 78, 45, 86, 98, 42]
print(arr)
n = len(arr)
mergesort(arr, 0, n-1)
print(arr)
```

output :-

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

056

CODE

```
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 superset of set 4")
elif set 3 < set 4:
    print ("set 3 is subset of set 4")
else:
    print ("set 3 is same of set 4")
if set 4 < set 3:
    print ("set 4 is subset of set 3")
    print ("\n")
```

057

Practical - 10

Aim - Implementation of Sets Using Python.

Algo -

Step 1 - Define two empty set as set 1 & set 2 now use for statement providing the range of above for 2 sets.

Step 2 - Now add() method used for addition the elements according to given range then print the sets After addition.

Step 3 - Find the union and intersection of above 2 sets by using (and) & ! (or) method. 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 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 is disjoint() to check that any thi common or element is present or not. If not ther display that it is mutually exclusive event.

&

set 3 = set 4
print("Elements in set 3 and not in set 4:
set 5", set 5)
print("\n")
if set 4.isdisjoint(set 5):
print("set 4 and set 5 are mutually
enclusive \n")
set 5.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 3 {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 superset of set 4
elements often set 3 and not in set 4: set 5 {1, 2, 3, 4, 6, 6, 7, 12, 13, 14}
set 4 and set 5 are mutually enclusive
after applying clear, set 5 is empty set:
set 5 = set()

CODE

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

class BST:
    def __init__(self):
        self.root = None
    def add(self, value):
        p = node(self, value):
        if self.root == None:
            self.root = p
            print("Root is added successfully", p.val)
        else:
            ll = self.root
            while True:
                if h.left == None:
                    h.left = p
                    print(p.val, "Node is added to left side successfully")
                    break
                else:
                    h = h.left
            else:
                if h.right == None:
                    h.right = p
                    print(p.val, "Node is added to right side successfully")
                    break
```


PRACTICAL - 11

Aim - Implementation of binary search tree - BINARY SEARCH TREE using python & Inorder, preorder & post order, Traversal

Theory - Binary Tree is a tree which supports max of 2 children for any node within the tree. Thus any particular node can have 0, 1 either 2 children. There is another identity of binary tree that is ordered search that one child as identified as left child as other as right child.

1) Inorder : i) Transverse the left sub-tree. The left sub-tree inform might have left & right sub-tree.
ii) Visit the root node.
iii) Transverse the right sub-tree & repeat it

2) Pre-order : i) Visit the root node
ii) Transverse the left sub tree.
iii) Transverse the right sub-tree.

3) Post order : i) Transverse the left sub-tree The left subtree in turn might h left & right sub-tree
ii) Transverse the right sub-tree
iii) Insert the root node.

Algorithm

Step1 - Define class node & define init() method with 2 argument. Initialize the value in this method.

Step2 - Again define a class BST that is "Binary Search Tree" with init() method with self argument & assign the root is node.

Step3 - Define add() method for adding the node.

Step4 - Use if statement for checking the condition that root is none than we use else. For if value is less than the main then put on. For if value arrange that in left side.

Step 5 - Use while loop for checking the node is less than or greater than the main node & break the loop if it is not satisfying.

Step 6 - Use if statement within that else statement for choosing that node is greater than main root then put it into right side.

Step 7 - After this, left subtree & right & type repeat this method to arrange the node according to Binary Search Tree.


```
        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:
            postorder(root.left)
            Postorder(root.right)
            print(root.val)

t = BST()
t.add(1)
t.add(2)
t.add(3)
t.add(4)
```

OUTPUT

```
>>> print("\n Inorder form of tree ", Inorder(t.root))
   1
   2
   3
   4
   5
  Inorder form of tree None
>>> print("Preorder form of tree", Preorder(t.root))
   1
   2
   3
   4
   5
  Preorder form of tree None
>>>print("Postorder form of tree", Postorder(t.root))
   3
   5
   4
   2
   1
Postorder form of tree None
```


step 8 - Define Inorder(), Preorder() or Postorder() with root argument & use if statement that root is none & return That in all

step 9 - Inorder, else statement used for giving the condition first left root & then right root

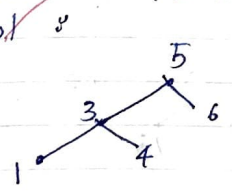
step 10 - In preorder, we have to give condition in else that first root, left & then right node.

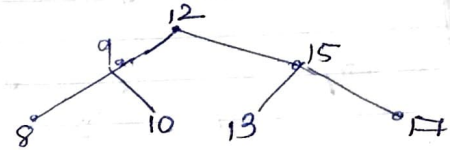
step 11 - For postorder, in case post, assign left then right & then go for root node.

step 12 - Display the output & input.

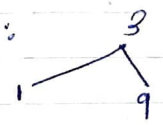
Inorder : (L V R)

Step 1 :

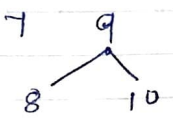




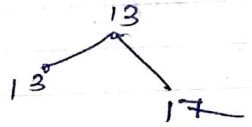
Step 2 :



5 6 7



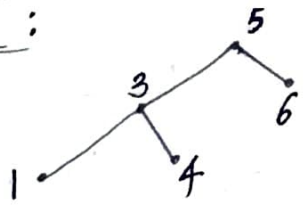
12

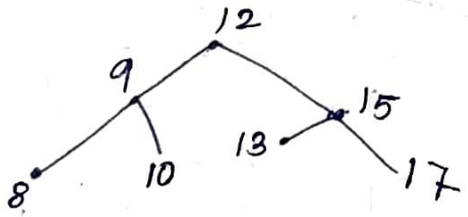


Step 3 = 1 3 4 5 6 7 8 9 10 12 13 15

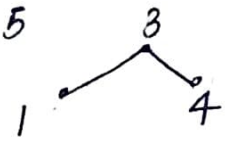
- Pre-order : (V L R)

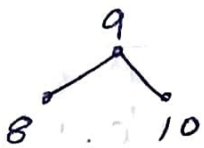
Step 1 :

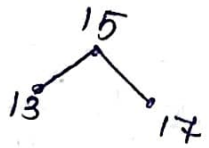




Step 2 :

7 5 

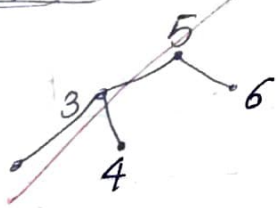
6 12 

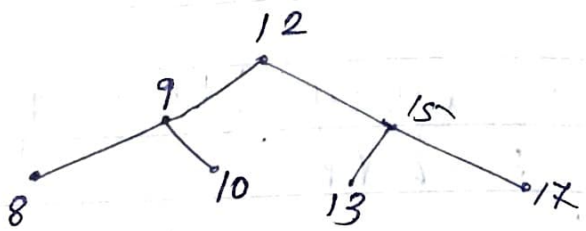


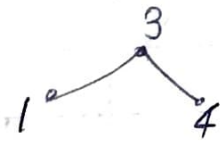
Step 3 : 7 5 3 1 4 6 12 9 10 15 13 17

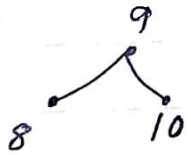
- POST ORDER : (L R V)

Step 1 :





Step 2 : 

6 5 

12 7

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