Binary Search Tree → Questions.

M T W T F S S | Page No.: | Date: | YOUVA

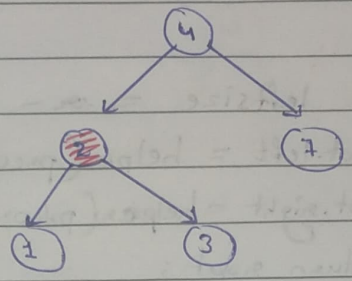
(1) LeetCode Q.No. (700) { Search in a Binary Search Tree }

→ If we search the entire tree, then T.C is O(n)

→ In BST, we do not need to search the entire tree.

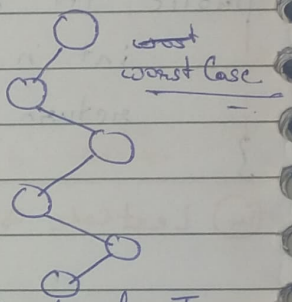
=> Code :-

```
public TN searchBST(TN root, int val){
    if(root == null) return null;
    if(root.val == val) return root;
    if(val < root.val){
        return searchBST(root.left, val);
    }
    else {
        return searchBST(root.right, val);
    }
}
```



T.C => O(log n).

Best Case :- O(log n) / O(h)
[Balanced Binary Tree]



Avg. & Worst Case :- O(h) → height / levels of a Tree.
↓
O(n)

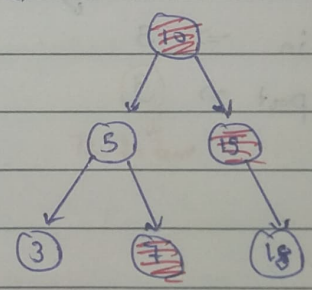
---

(2) LeetCode Q.No. (938) { Range Sum of BST }

high => 15
low => 7

=>Code :-

```
public int rangeSumBST(TN root, int low, int high){
    if(root == null) return 0;
    int sum = 0;
    if(root.val > low){
        sum += rangeSumBST(root.left, low, high);
    if(root.val < high)
        sum += rangeSumBST(root.right, low, high);
    if(root.val <= high && root.val >= low)
        sum += root.val;
    return sum;
}
```



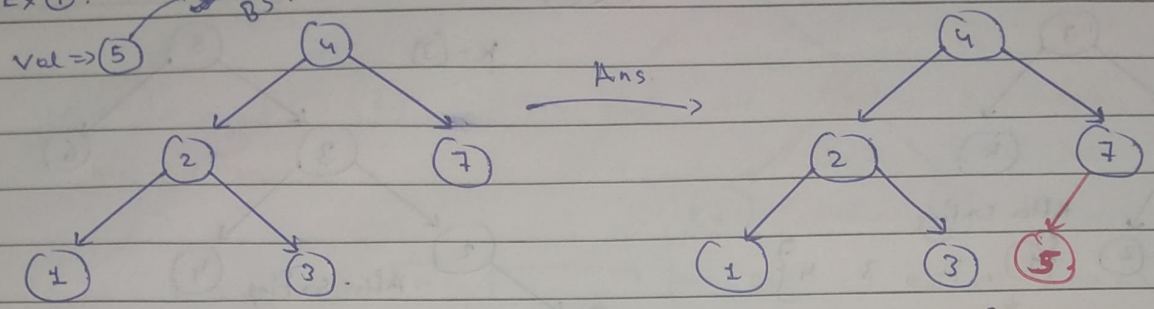
Ans => (32)

10 + 15 + 7 => 32 //

③ Leetcode Q.No. 701 { Insert Into A Binary Tree.

M T W T F S S | Page No.: | Date: | YOUVA

Ex ①:- Need to insert with following BST Rules



M-1

=> Code :-

```
p v insert(TN root, int val){
    if(val < root.val){ // Attach to Left.
        if(root.left == null){
            root.left = new TreeNode(val);
            return;
        }
        else{
            insert(root.left, val);
        }
    }
    else if(val > root.val){ // Attach to Right.
        if(root.right == null){
            root.right = new TreeNode(val);
            return;
        }
        else
            insert(root.right, val);
    }
}
```

```
p TN insertIntoBST(TN root, int val){
    if(root==null) return new TN(val);
    insert(root, val);
    return root;
}
```

M-2

=> Code :-

```
p TN insertIntoBST(TreeNode root, int val){
    if(root == null) return new TreeNode(val);
    if(val < root.val)
        root.left = insertIntoBST(root.left, val);
    else
        root.right = insertIntoBST(root.right, val);
    return root;
}
```

(4) LeetCode 9.No. (230) { Kth Smallest Elements in a BST

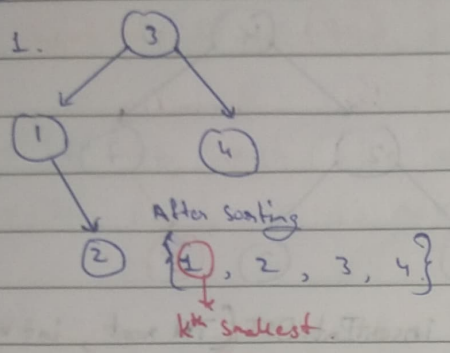
~~0-based~~ 1 based indexing.

Hint :- Use Inorder Traversal

Page No.: Date: YOUVA

Ex (1) :-

K => 1.

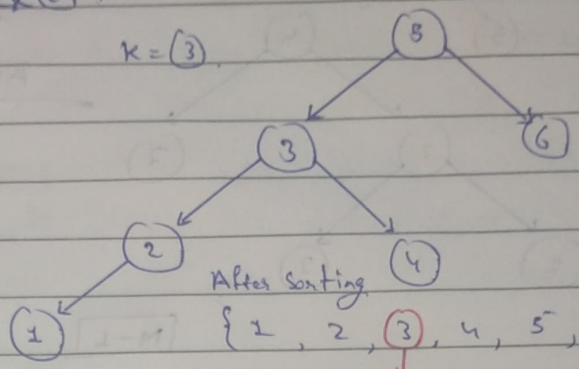


After sorting { 1 , 2 , 3 , 4 }

1 kth Smallest.

Ex (2) :-

K = (3).



After Sorting { 1 , 2 , (3) , 4 , 5 , 6 }

kth Smallest

M-1

=> Code :-

```
public int KthSmallest (TN root , int k){
    Stack<TN> st = new Stack<>();
    TN temp = root;
    while (true){
        if (temp != null){
            st.push (temp);
            temp = temp.left;
        }
        else { // temp == null.
            if (st.size() == 0) break;
            TN top = st.pop();
            if (k==1) return top.val;
            else k--;
            temp = top.right;
        }
    }
    return 0;
}
```

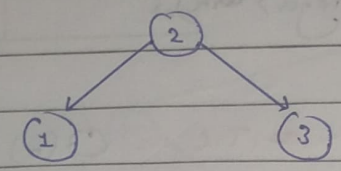
M-2

```
public int KthSmallest(TN root , int k){
    List<Integer> ans = helper(root , k , new ArrayList<>());
    return ans.get(k-1);
}

public List<Integer> helper(TN root , int k , List<Integer> ans){
    if(root == null) return ans;
    helper (root.left , k , ans);
    ans.add (root.val);
    helper(root.right , k , ans);
    return ans;
}
```

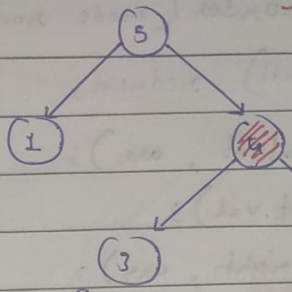
(5) LeetCode Q.No. (98) { Validate Binary Search Tree. }

Ex (1)



Ans :- True.

Ex (2)



Ans :- False.

M-1 ⟶ { min(RST) > root.val > max(LST) }

=> Code :-

```
ps long min(Node root){
    if(root == null) return Long.MAX_VALUE;
    Long a = root.val
    long minLST = min(root.left);
    long minRST = min(root.right);
    return Math.min(a , Math.min(minLST , minRST));
}.

ps long max(Node root){
    if(root == null) return Long.MIN_VALUE;
    Long a = root.val;
    Long maxLST = max(root.left);
    Long maxRST = max(root.right);
    return Math.max(a , Math.max(maxLST , MaxRST));
}.

ps boolean isValidBST(Node root){
    if(root == = null) return true;
    if(root.val <= max(root.left)) return false;
    if(root.val >= min(root.right)) return false;
    return isValidBST(root.left) && isValidBST(root.right);
}
```

T.C => O(n.log n) } ⟶ Best Case

T.C => O(n²) } ⟶ Worst Case

M-2 → {Using Inorder Traversal}

M T W T F S S
Page No.:
YOUVA

⇒ Code :- ps void inOrder(Node root, List<Integer> arr){
    if(root == null) return;
    inOrder(root.left, arr);
    arr.add(root.val);
    inOrder(root.right, arr);
}

* T.C :- O(n)
* S.C :- O(n).

ps boolean isValidBST(Node root){
    List<Integer> arr = new ArrayList<>();
    inOrder(root, arr);
    for(int i = 1; i < arr.size(); i++){
        if(arr.get(i) <= arr.get(i-1)) return false;
    }
    return true;
}

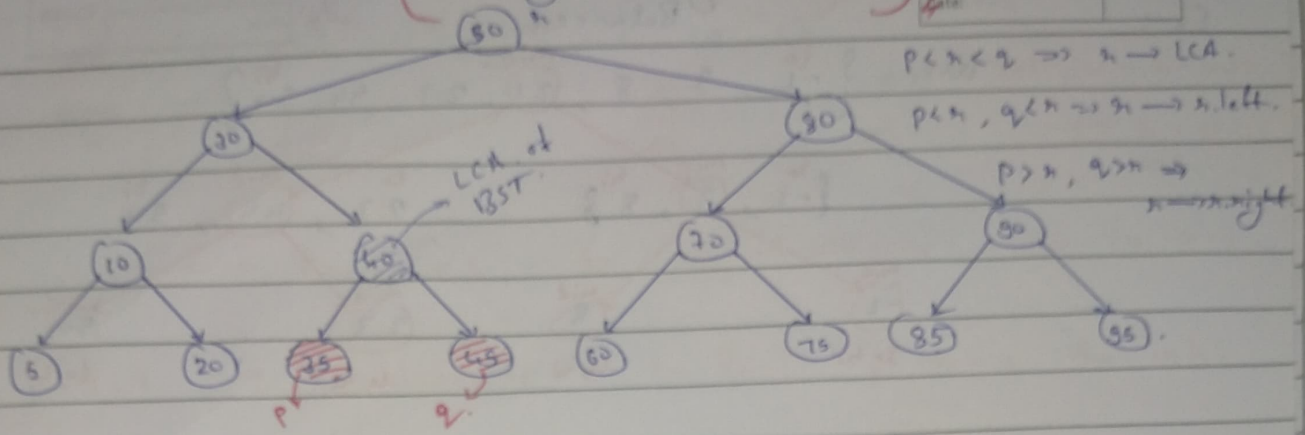
M-3 → Similar to M-1 → { Global variable flag = true }

⇒ Code :- static boolean flag;
ps long min(Node root){
    if(root == null) return Long.MAX_VALUE;
    long a = root.val;
    long minLST = min(root.left);
    long minRST = min(root.right);
    if(minRST <= root.val) flag = false;
    return Math.min(a, Math.min(minLST, minRST));
}

ps long max(Node root){
    if(root == null) return Long.MIN_VALUE;
    long a = root.val;
    long maxLST = max(root.left);
    if(maxLST = max(root.left)) flag = false;
    long maxRST = max(root.right);
    return Math.max(a, Math.max(maxLST, maxRST));
}

ps boolean isValidBST(Node root){
    flag = true;
    max(root);
    min(root);
    return flag;
}

(6) LeetCode Q. No. 235 { Lowest Common Ancestor of a Binary Search Tree }



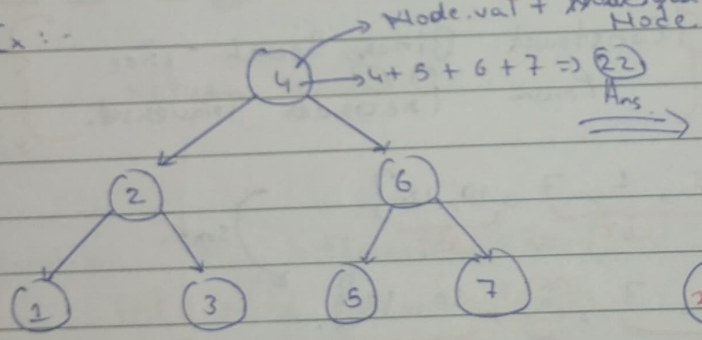
=> Code :-

```
p TN lowestCommonAncestor(TN root, TN p, TN q){
    if(root == null) return null;
    if(p.val < root.val && q.val < root.val)
        return lowestCommonAncestor(root.left, p, q);
    if(p.val > root.val && q.val > root.val)
        return lowestCommonAncestor(root.right, p, q);
    return root;
}
```

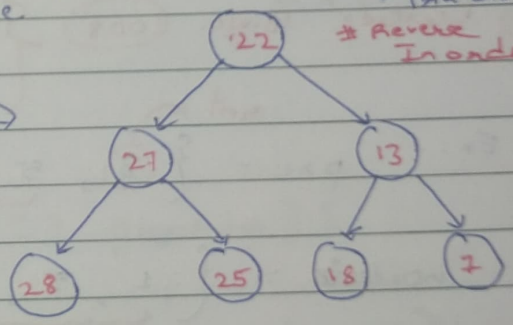
---

(7) LeetCode Q. No 1038 { Binary Search Tree To Greater Sum Tree. }

Ex :-



Use Inorder Traversal.



=> Code :-

```
Static int sum;
public void reverseInorder(TN root){
    if(root == null) return;
    reverseInorder(root.right);
    root.val += sum;
    sum = root.val;
    reverseInorder(root.left);
}
```

Tc :- O(n)

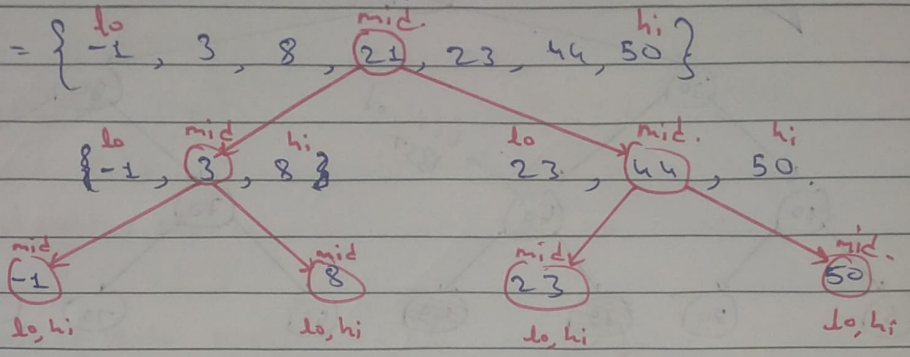
Sc :- O(h) (h is level / height of Tree)

```
p TN bstToGst(TN root){
    sum = 0;
    reverseInorder(root);
    return root;
}
```

(7) Leetcode Q.No. (108) { Convert Sorted Array to Binary Search Tree. Balanced. }

Ex :-

arr = { -1, 3, 8, 21, 23, 44, 50 }



=> Code :-

```
TN helper(int[] arr, int lo, int hi){
    if(lo > hi) return null;
    int mid = (lo + hi)/2;
    TN root = new TN(arr[mid]);
    root.left = helper(arr, lo, mid-1);
    root.right = helper(arr, mid+1, hi);
    return root;
}

TN sortedArrayToBST(int[] nums){
    return helper(nums, 0, nums.length-1);
}
```

---

(8) Leetcode Q.No. (1008) { Construct Binary Search Tree From Preorder Traversal. }

Ex :-

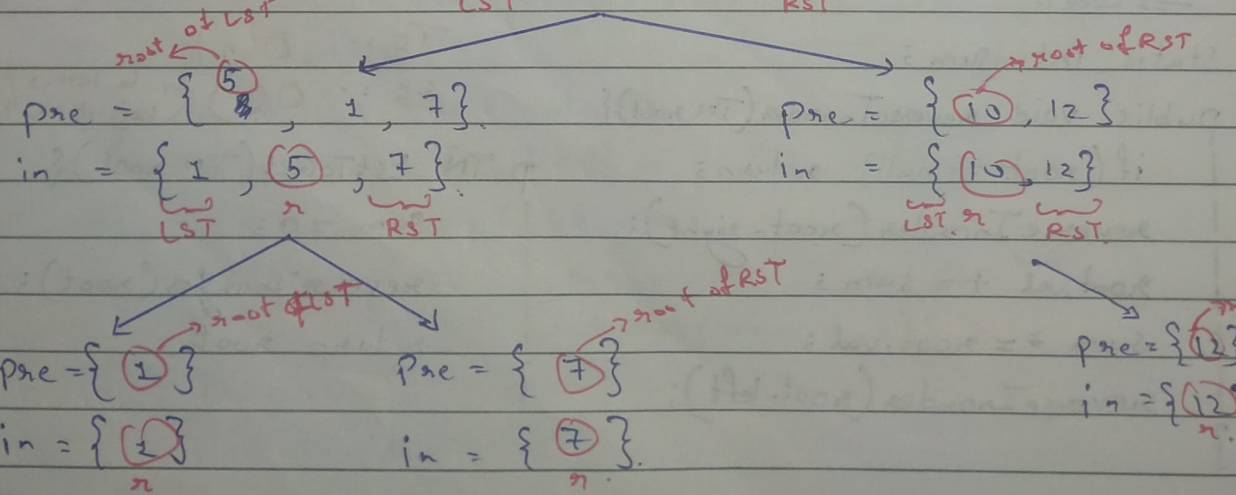
pre = { 8, 5, 1, 7, 10, 12 }

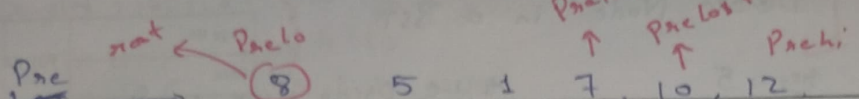
LST, RST

Sort.

inorder = { 1, 5, 7, 8, 10, 12 }

r, LST, RST





Leftsize = r - inlo.

=> 3 - 0

=> 3.

LST ⟶ Prelo + 1 to Prelo + leftsize , inlo to r-1.

RST ⟶ Prelo + leftsize + 1 . to Prehi , r + 1 to inhi

=> Code :-

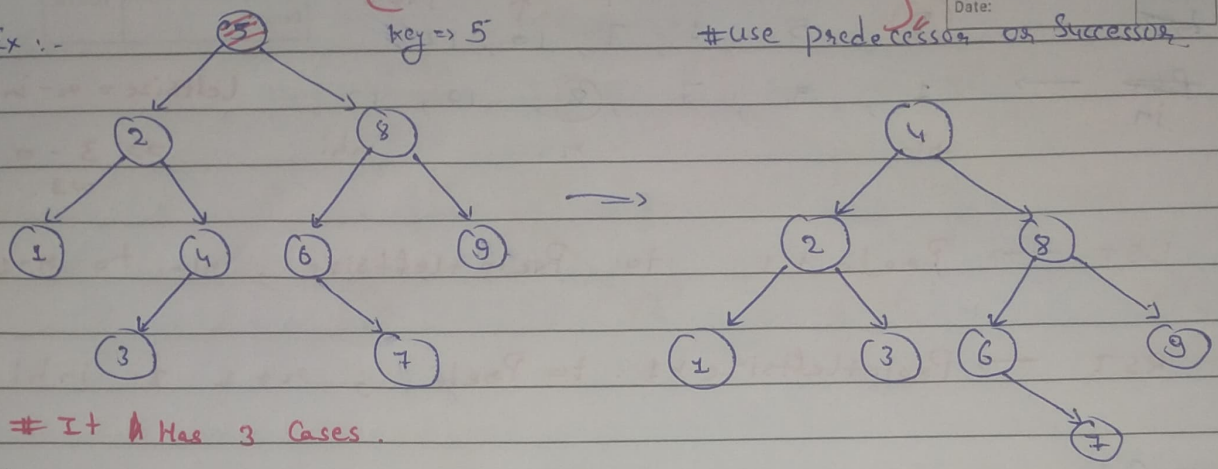
```
TN helper(int[] preorder, int[] inorder, int prelo, int prehi,
                                          int inlo, int inhi){
    if(prelo > prehi || inlo > inhi) return null;
    TN root = new TN(preorder[prelo]);
    int r = 0;
    while(inorder[r] != preorder[prelo]) r++;
    int leftsize = r - inlo;
    root.left = helper(preorder, inorder, prelo+1, prelo+leftsize, inlo,
                                                          r-1);
    root.right = helper(preorder, inorder, prelo+leftsize+1, prehi,
                                                  r+1, inhi);
    return root;
}.

public TN bstFromPreorder(int[] preorder){
    int n = preorder.length;
    int[] inorder = Arrays.copyOf(preorder, n);
    Arrays.sort(inorder);
    return helper(preorder, inorder, 0, n-1, 0, n-1);
}.
```

⑨ LeetCode Q.No. 450 { Delete Node in a BST }

M T W T F S S | Page No.: | Date: | YOUVA

Ex :- key => 5



#use predecessor or Successor.

It Has 3 Cases.

=> Code :-

```
public TN deleteNode(TN root, int key){
    if(root == null) return null;
    if(root.val == key) {  // Deletion.
        if(root.left == null && root.right == pred)   } Case ① : 0 Child Nodes.
            return null;
        else if(root.left == null || root.right == null){
            if(root.left == null)
                return root.right;
            else
                return root.left;
        }
        else{
            TN pred = iop(root);
            TN predParent = parent(root, pred);
            if(root == predParent){
                pred.right = root.right;
                return pred;
            }
            predParent.right = pred.left;
            pred.left = root.left;
            pred.right = root.right;
            return pred;
        }
    }
    else if(root.val > key)   root.left = deleteNode(root.left, key);
    else   root.right = deleteNode(root.right, key);
    return root;
```

Case ② : 1 Child Nodes.

Case ③ : 2 Child Nodes.

```
p TN iop (TN root){
    TN temp = root.left;
    while (temp.right != null)
        temp = temp.right;
    return temp;
}

p TN parent(TN root, TN pred){
    if (root.left == pred || root.right == pred)  return root;
    TN temp = root.left;
    while (temp.right != pred){
        temp = temp.right;
    return temp;
}
```

---

(10) Leetcode (94). { Binary Tree Inorder Traversal } # Using Morris Traversal.

↳ Left Root Right.

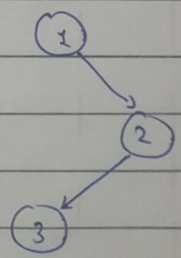
=> Code :-

```
p List<Integer> inorderTraversal (TN root){
    List<Integer> ans = new Arraylist<>();
    TN curr = root;
    while (curr != null){
        if(curr.left != null){
            TN pred = curr.left;
            while (pred.right != null && pred.right != curr)
                pred = pred.right;
            if(pred.right == null){
                pred.right = curr;
                curr = curr.left;
            }
            else{
                ans.add(curr.val);
                curr = curr.right;
                pred.right = null;
            }
        }
        else{
            ans.add(curr.val);
            curr = curr.right;
        }
    return ans;
}
```

⇒ Linking

⇒ Unlinking

Ex :-



Ans = {1, 3, 2}

(11) LeetCode Q.No. (98) { Validate Binary Search Tree }.

M T W T F S S
Page No.:
Date:
YOUVA

M-4 → Morris Traversal

=> Code :-

```
public boolean isValidBST(TN root){
    TN prev = null;
    TN curr = root;
    while(curr != null){
        if(curr.left != null){
            TN pred = curr.left;
            while(pred.right != null && pred.right != curr){
                pred = pred.right;
            }
            if(pred.right == null)
                pred.right = curr;
                curr = curr.left;
            }
            else{
                if(prev != null && prev.val >= curr.val)
                    return false;
                prev = curr;
                curr = curr.right;
                pred.right = null;
            }
        }
        else{
            if(prev != null && prev.val >= curr.val)
                return false;
            prev = curr;
            curr = curr.right;
        }
    }
    return true;
}
```

(12) LeetCode Q.No. (230) { K^{th} Smallest Element in a BST }

M T W T F S S
Page No.:
Date:
YOUVA

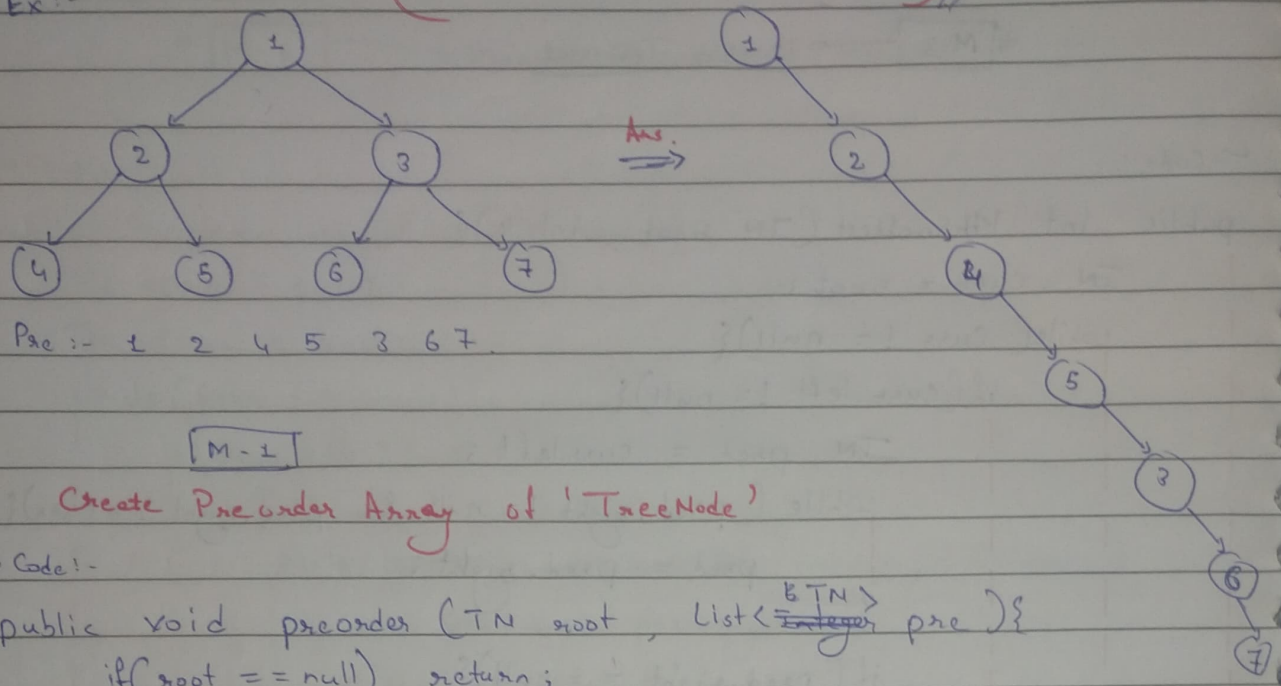
M-3 → Morris Traversal

=> Code :-

```
public int kthSmallest(TN root, int k){
    TN curr = root;
    while(curr != null){
        if(curr.left != null){
            TN pred = curr.left;
            while(pred.right != null && pred.right != curr){
                pred = pred.right;
            }
            if(pred.right == null){
                pred.right = curr;
                curr = curr.left;
            }
            else{
                if(k==1) return curr.val;
                else k--;
                curr = curr.right;
                pred.right = null;
            }
        }
        else{
            if(k==1) return curr.val;
            else k--;
            curr = curr.right;
        }
    }
    return 0;
}
```

(13) Leetcode Q.No. ~~115~~ (114) { Flatten Binary Tree to Linked List

Ex:-



Pre :- 1 2 4 5 3 6 7.

M-1

Create Preorder Array of 'TreeNode'

=> Code:-

```
public void preorder (TN root, List<~~Integer~~ TN> pre ){
    if(root == null) return;
    pre.add(root);
    preorder(root.left, pre);
    preorder(root.right, pre);
}

public void flatten (TN root){
    List<TN> pre = new ArrayList<>();
    preorder(root, pre);
    int n = pre.size();
    for(int i=0; i<n; i++){
        if(i != n-1)
            pre.get(i).right = pre.get(i+1);
        pre.get(i).left = null;
    }
}
```

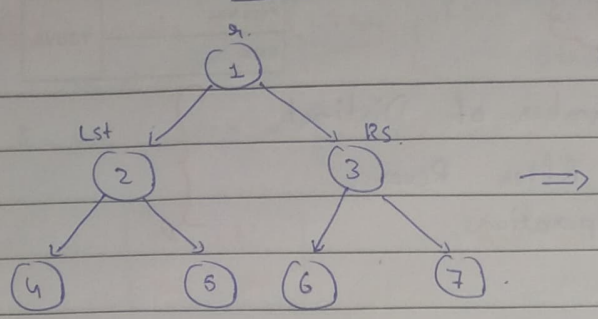
① Put Null in left of every Node.

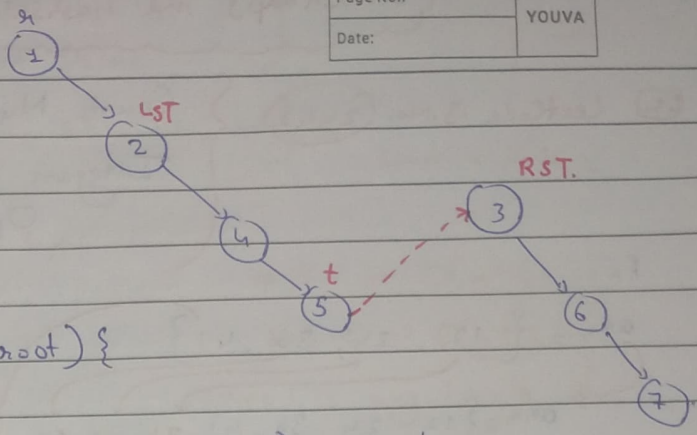
② ans[i].right = ans[i+1]

T.C = O(n)

S.C = O(n)

M-2 ⟹ Recursion





=> Code :- public void flatten (TN root) {

```
if(root == null) return;
if(root.left == null && root.right == null) return.
TN lst = root.left;
TN rst = root.right;
flatten (lst);
flatten (rst);
root.left = null;
root.right = lst;
TN temp = root;
while (temp.right != null)
        temp = temp.right;
}
temp.right = rst;
}.
```

T.C => O(n)

S.C => O(n) → Recursive stack space.

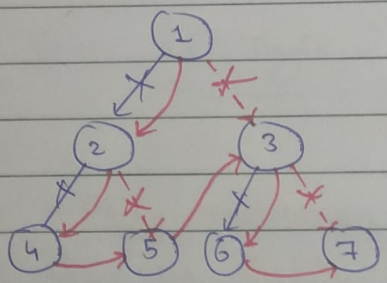
M-3 → Something Like Morris traversal.

T.C => O(n)

S.C => O(1)

=>Code :- public void flatten (TN root){

```
TN curr = *root;
while (curr != null){
    if(curr.left != null){
        TN r = curr.right;
        curr.right = curr.left;
        TN pred = curr.left;
        while(pred.right != null)
            pred = pred.right;
        pred.right = r;
        curr.left = null; // IMPORTANT.
        curr = curr.right;
    }
    else{
        curr = curr.rights;
    }
}
}
```



Bonus :- Pre/In/Post

O(h) Space Recursive Soln.

In ⇒ Morris → O(1) Space

pre ⇒ Morris → O(1) space

[But Structure of tree is ~~pre~~ permanantly Changed]