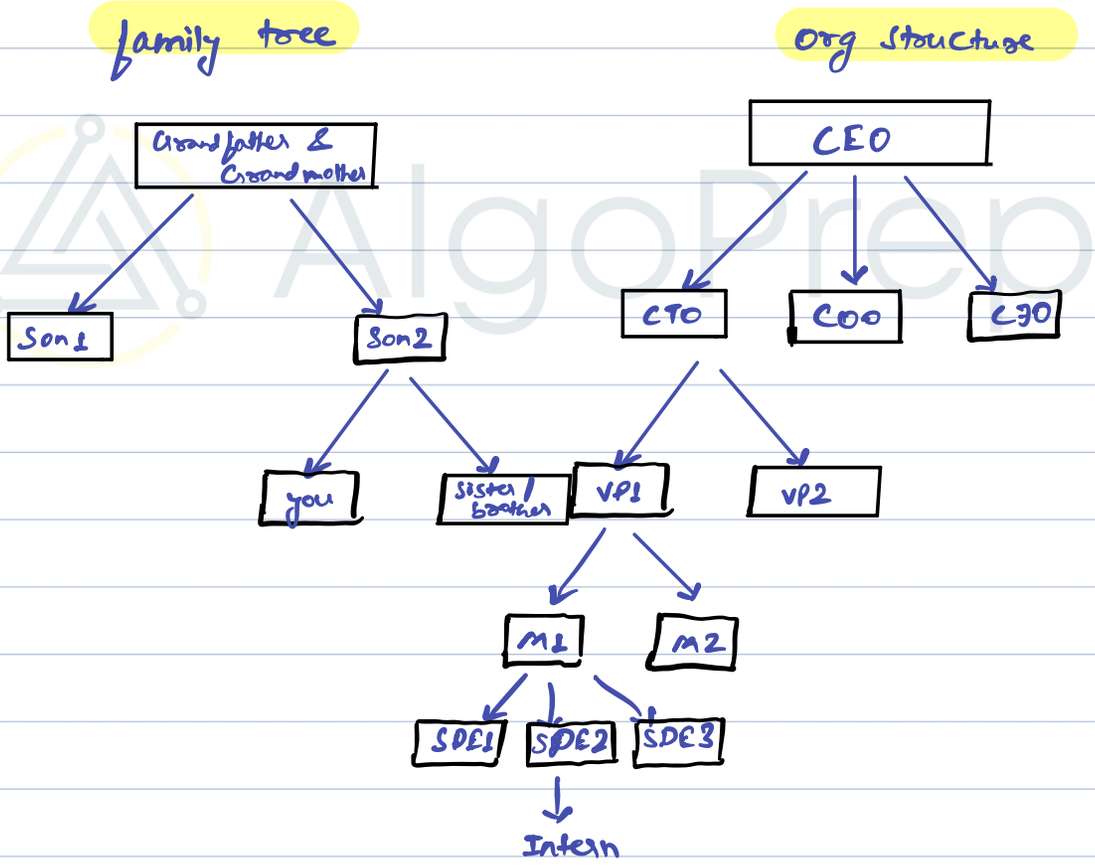
Today's agenda

- Trees Intro
- Naming convention
- Tree traversal
- Basic tree problems

## Linear

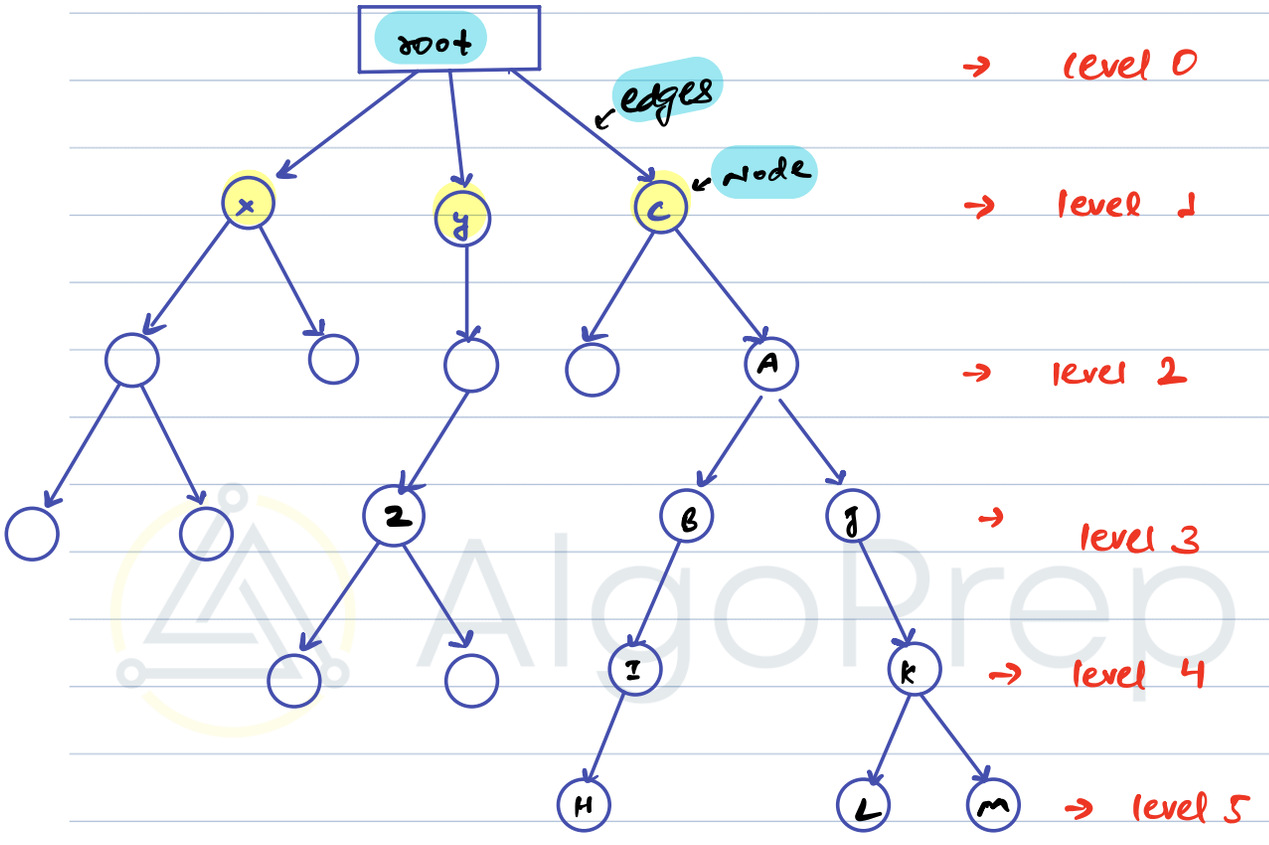
Arrays →  , LL , Hashmap, Stack, queues.

## Hierarchical



// Tree

↳ Naming Convention






Parent & child : A is Parent of B, B is child of A

Ancestors : All the nodes in the Path from root node to that node.

Descendents : All the nodes below the given node.

leaf nodes : Nodes without children

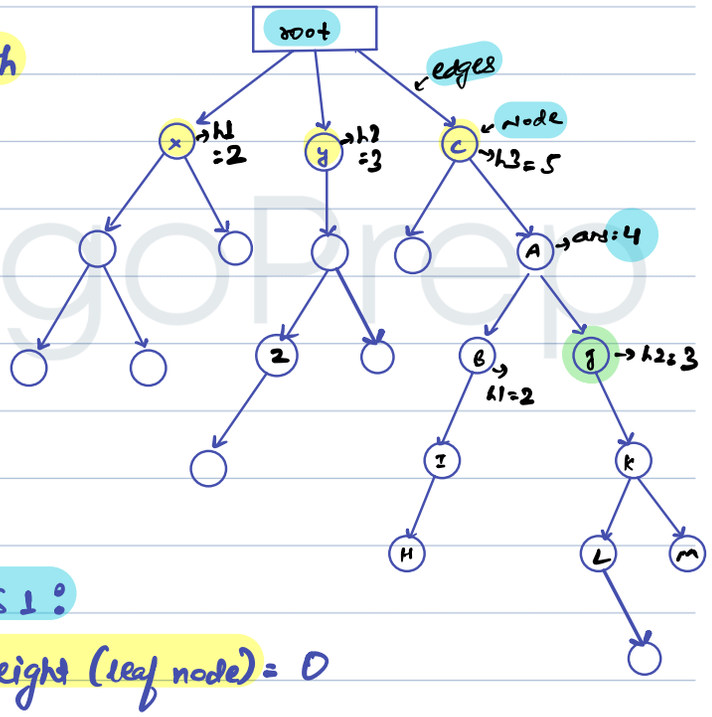
Siblings : Nodes with Same Parent

* Property of tree:

↳ we will have only 1 root Node.
↳ For every Node, there can be only 1 Parent.
↳ Cycle Not allowed

* Height (Node)

↳ length of longest Path from Node to any of its descendent leaf Nodes.




Height (j) = 2

Height (y) = 3

Height (A) = 4

Obs 1:

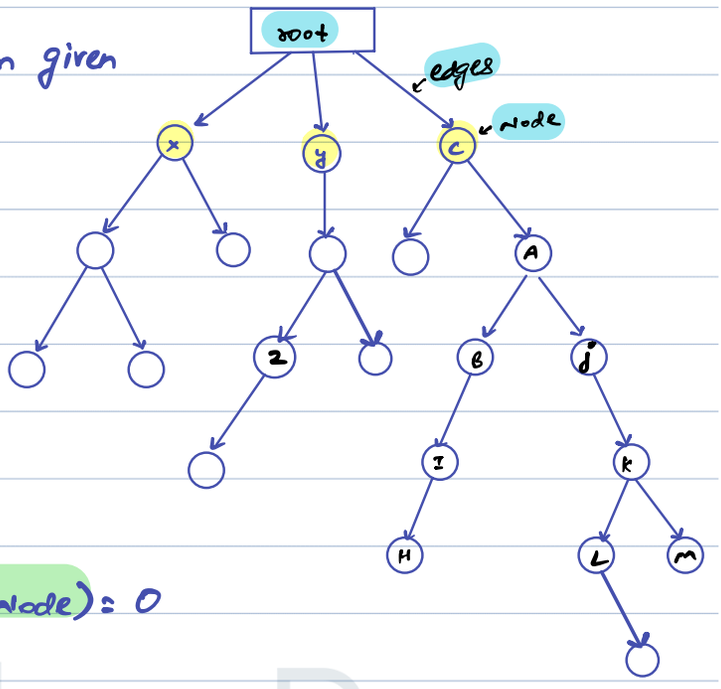
Height (leaf node) = 0

Obs 2:

H(Node) = max(Height of Child nodes) + 1

* Depth (node)

↳ length of Path from given node to root node.



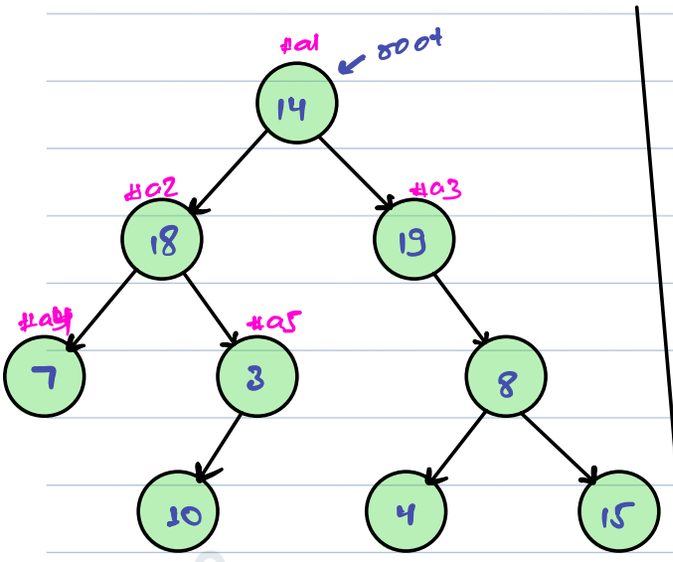

Depth (j) = 3

Depth (H) = 5

Obs 1: depth (root node) = 0

Obs 2: depth (node) = depth (Parent node) + 1

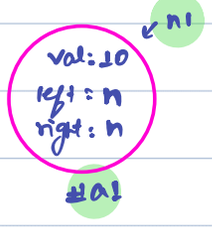
**Binary tree:** Every node can have atmax 2 child Nodes.



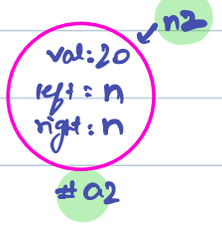

```
Class Node {
    int val;
    Node left;
    Node right;
    Node (int n) {
        val = n;
    }
}
```



`Node n1 = new Node (10);`

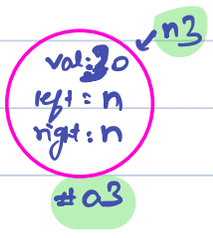


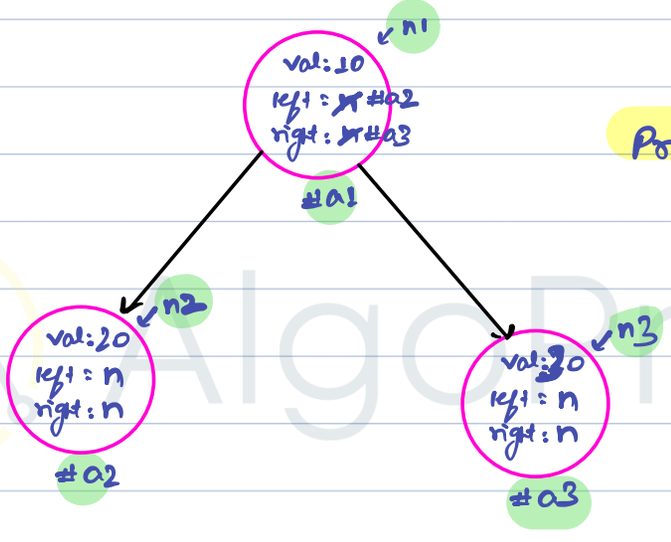

`Node n2 = new Node (20);`




```
Class Node {
    int val;
    Node left;
    Node right;
    Node (int n) {
        val = n;
    }
}
```

Node n3 = new Node (30);




Print (n1.left.val) → 20

n1.left = n2;
n1.right = n3;

→ Don't worry about tree Construction. (serialization & deserialization)

↓↓

Solve the Problem for constructed tree.

// Tree traversal

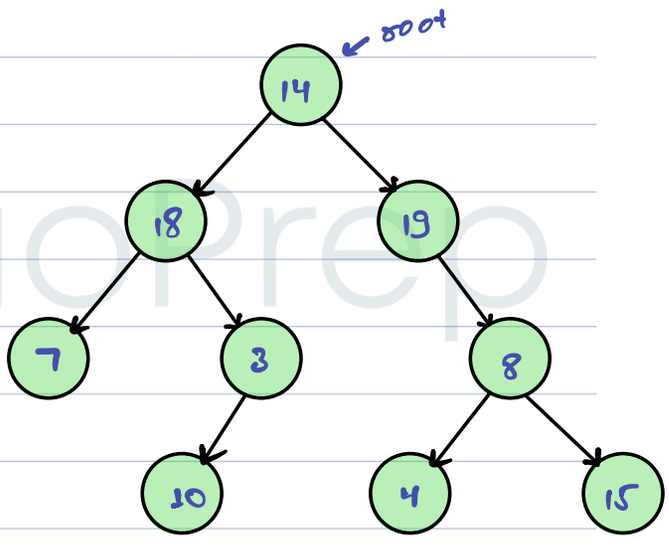
↳ level order traversal (bfs) ← breadth first search

↳ Recursion traversal

↳ Pre/Post/In (dfs)

↓ depth first search

```
void traversal (Node root){
    if (root == null){
        return;
    }
    traversal (root.left);
    traversal (root.right);
}
```

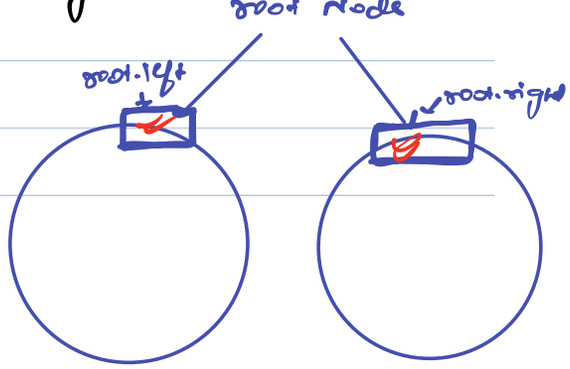



Faith: Given root node, travel every descendent of root node.

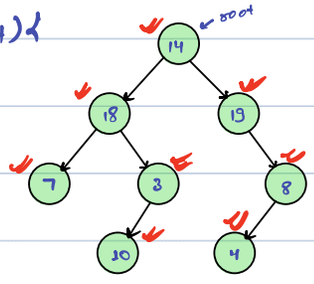
basecase:

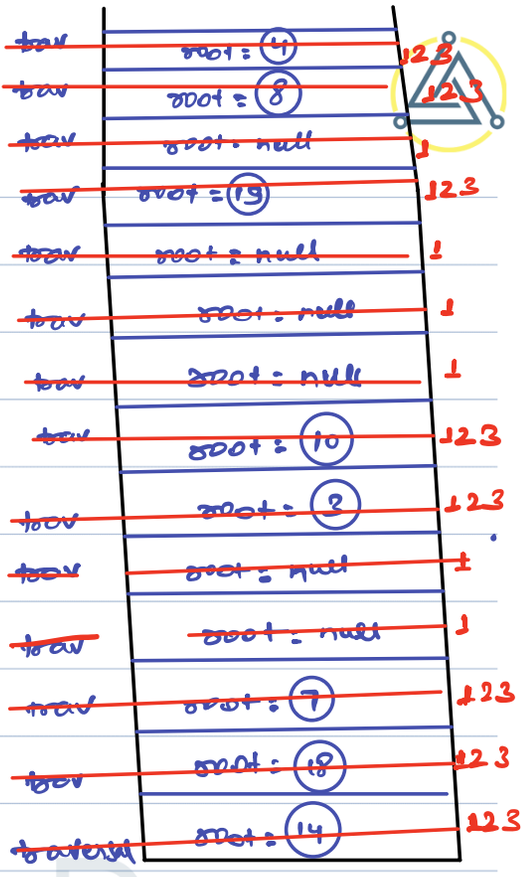
```
if (node == null){
    return;
}
```

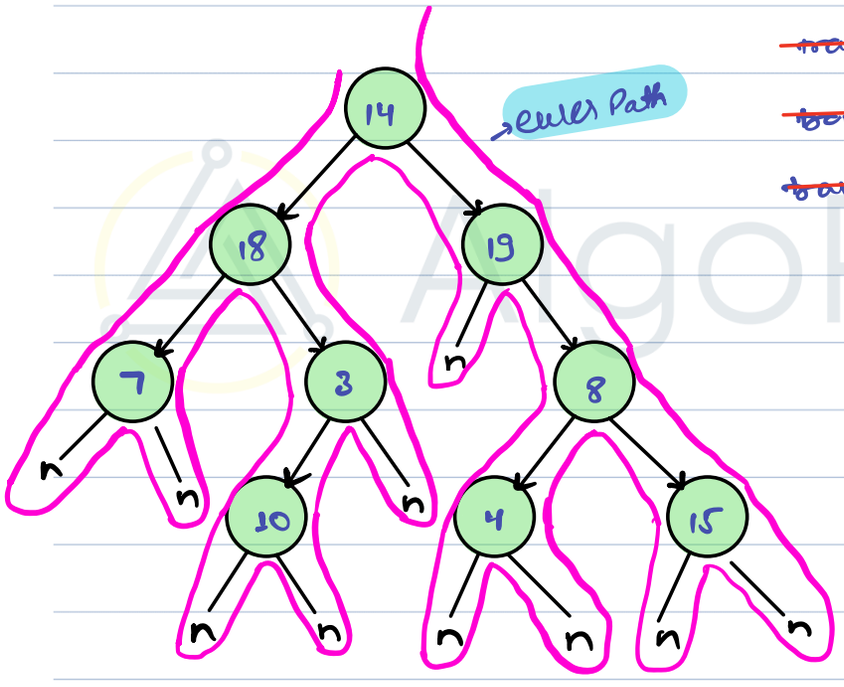
main logic:



```
void traversal (Node root){
    1  if (root == null){
           return;
       }
    2  traversal (root.left);
    2  traversal (root.right);
}
```




↳ How many times you visited a single node?

↳ 3 times

T.C: $O(3*N) \approx O(N)$

Break till 10:05 PM

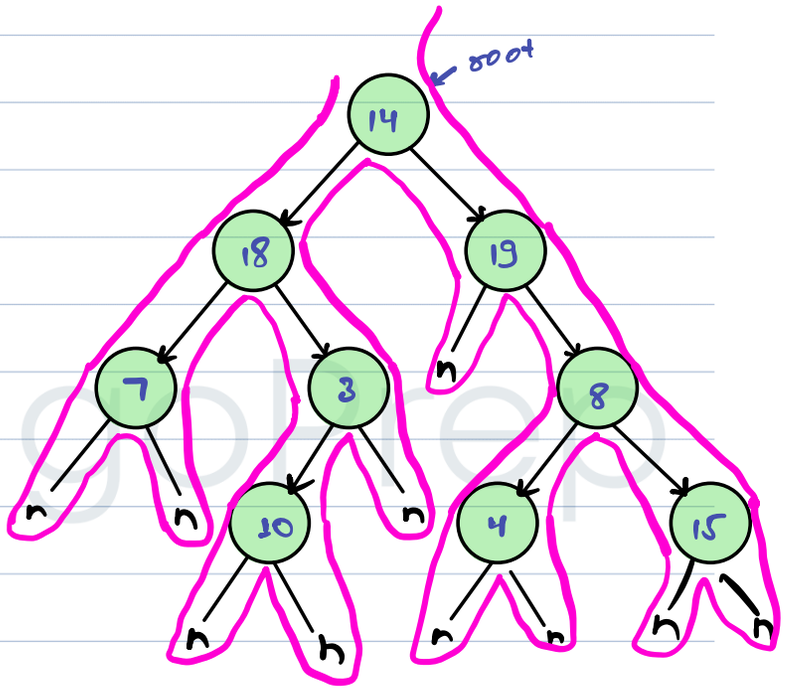
# Traversal

- Preorder traversal → NLR
- Inorder traversal → LNR
- Postorder traversal → LRN

**Preorder traversal**

```
void Preorder(Node root){
    if(root == null){
        return;
    }

    System.out.print(root.val);
    traversal(root.left);
    traversal(root.right);
}
```

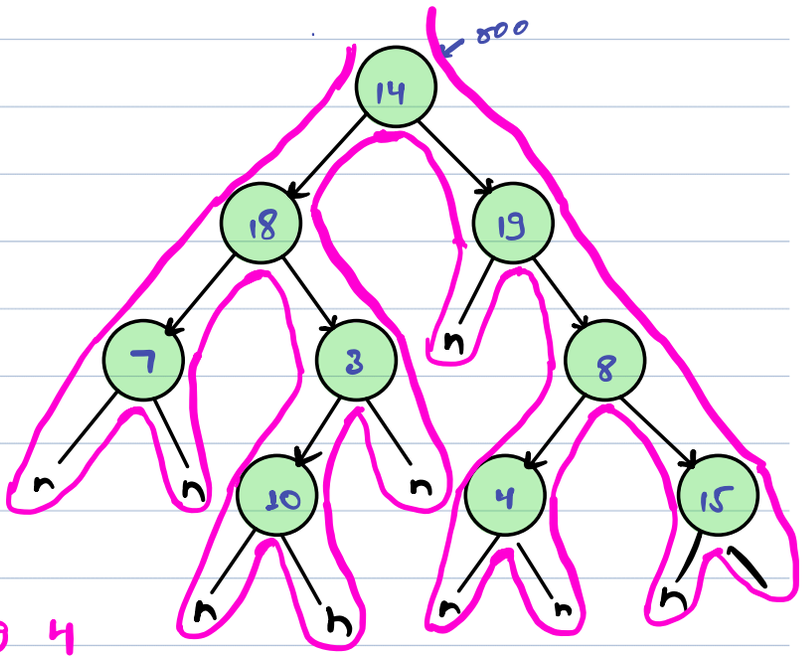
14 18 7 3 10 19 8 4 15



## Inorder traversal

↳ LNR

```
void Inorder (Node root){
    if (root == null){
        return;
    }
    traversal (root.left);
    System.out.print (root.val);
    traversal (root.right);
}
```

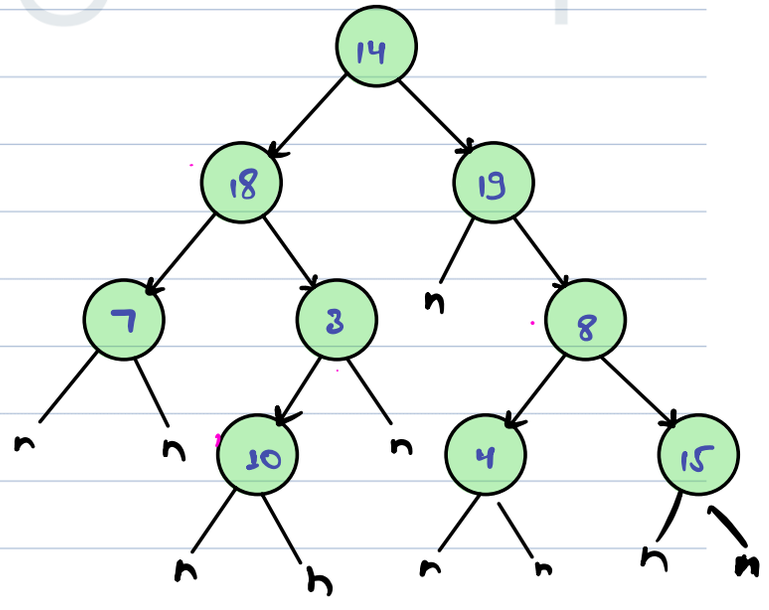


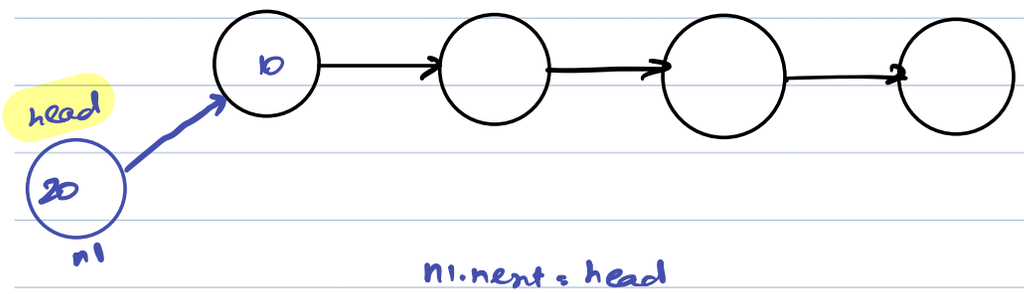

7 18 10 3 14 19 4
8 15

## Postorder traversal

↳ LRN

```
void Postorder(Node root){
    if (root == null){
        return;
    }
    traversal (root.left);
    traversal (root.right);
    System.out.print (root.val);
}
```






n1.next = head

head = n1;

# PreOrder Traversal

## Java Code:

```java
class Solution {
public List<Integer> preorderTraversal(TreeNode root) {
List<Integer> pre = new ArrayList<>();
preHelper(root,pre);
return pre;
}
public void preHelper(TreeNode root, List<Integer> pre) {
if(root==null) return;
pre.add(root.val);
preHelper(root.left,pre);
preHelper(root.right,pre);
}
}
```

## C++ Code:

```cpp
#include <iostream>
#include <vector>

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    std::vector<int> preorderTraversal(TreeNode* root) {
        std::vector<int> pre;
        preHelper(root, pre);
        return pre;
    }

    void preHelper(TreeNode* root, std::vector<int>& pre) {
        if (root == nullptr) return;
        pre.push_back(root->val);
```

```
        preHelper(root->left, pre);
        preHelper(root->right, pre);
    }
};

int main() {
    // You need to create the binary tree and call the function here.
    return 0;
}
```

## Python Code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def preorderTraversal(self, root: TreeNode):
        pre = []
        self.pre_helper(root, pre)
        return pre

    def pre_helper(self, root: TreeNode, pre):
        if root is None:
            return
        pre.append(root.val)
        self.pre_helper(root.left, pre)
        self.pre_helper(root.right, pre)

# Example usage:
# You need to create the binary tree and call the function here.
```

# Inorder Traversal

## Java Code:

```java
class Solution {
public List<Integer> inorderTraversal(TreeNode root) {
List<Integer> pre = new ArrayList<>();
preHelper(root,pre);
return pre;
}
public void preHelper(TreeNode root, List<Integer> pre) {
if(root==null) return;
preHelper(root.left,pre);
pre.add(root.val);
preHelper(root.right,pre);
}
}
```

## C++ Code:

```cpp
#include <iostream>
#include <vector>

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    std::vector<int> inorderTraversal(TreeNode* root) {
        std::vector<int> in;
        inorderHelper(root, in);
        return in;
    }

    void inorderHelper(TreeNode* root, std::vector<int>& in) {
```

```
        if (root == nullptr) return;
        inorderHelper(root->left, in);
        in.push_back(root->val);
        inorderHelper(root->right, in);
    }
};

int main() {
    // You need to create the binary tree and call the function here.
    return 0;
}
```

## Python Code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def inorderTraversal(self, root: TreeNode):
        in_order = []
        self.inorder_helper(root, in_order)
        return in_order

    def inorder_helper(self, root: TreeNode, in_order):
        if root is None:
            return
        self.inorder_helper(root.left, in_order)
        in_order.append(root.val)
        self.inorder_helper(root.right, in_order)

# Example usage:
# You need to create the binary tree and call the function here.
```

# PostOrder Traversal

## Java Code:

```java
class Solution {
public List<Integer> postorderTraversal(TreeNode root) {
List<Integer> pre = new ArrayList<>();
preHelper(root,pre);
return pre;
}
public void preHelper(TreeNode root, List<Integer> pre) {
if(root==null) return;
preHelper(root.left,pre);
preHelper(root.right,pre);
pre.add(root.val);
}
}
```

## C++ Code:

```cpp
#include <iostream>
#include <vector>

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    std::vector<int> postorderTraversal(TreeNode* root) {
        std::vector<int> post;
        postorderHelper(root, post);
        return post;
    }

    void postorderHelper(TreeNode* root, std::vector<int>& post) {
        if (root == nullptr) return;
        postorderHelper(root->left, post);
        postorderHelper(root->right, post);
```

```
        post.push_back(root->val);
    }
};

int main() {
    // You need to create the binary tree and call the function here.
    return 0;
}
```

## Python Code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def postorderTraversal(self, root: TreeNode):
        post_order = []
        self.postorderHelper(root, post_order)
        return post_order

    def postorderHelper(self, root: TreeNode, post_order):
        if root is None:
            return
        self.postorderHelper(root.left, post_order)
        self.postorderHelper(root.right, post_order)
        post_order.append(root.val)

# Example usage:
# You need to create the binary tree and call the function here.
```

Today's agenda

- ArrayList
- Size of the tree
- Sum of all nodes
- level order
- reverse level order

Arrays: `int[] arr = new int[10];`

→ in background is nothing but array

ArrayList: → dynamic array.

`List<Integer> ls = new ArrayList<>();`

ls

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 |

`ls.get(3);` → 40

`ls.size();` → 5

`ls.add(10);` → O(1)
`ls.add(20);`
`ls.add(30);`
`ls.add(40);`
`ls.add(50);`

`ls.remove(idx)` → O(1) ← last idx
O(n) ← other idx

iterate →

```
for(int i=0; i<ls.size(); i++){

}
```

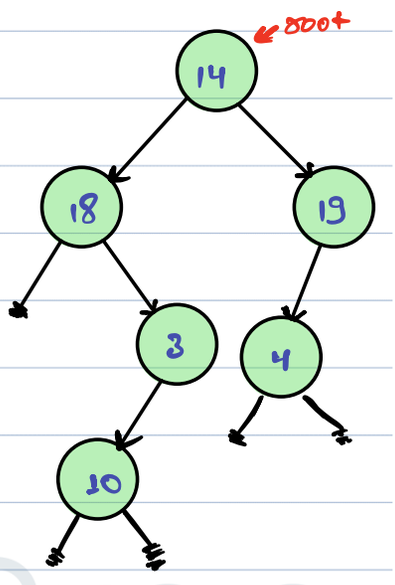
## Q) Size of a tree

↳ Given a tree, Calculate no. of Nodes in it.

```
int size (Node root) {
    if (root == null) {return 0;}
    int x = size (root.left);
    int y = size (root.right);
    return x+y+1;
}
```

T.C: O(N)

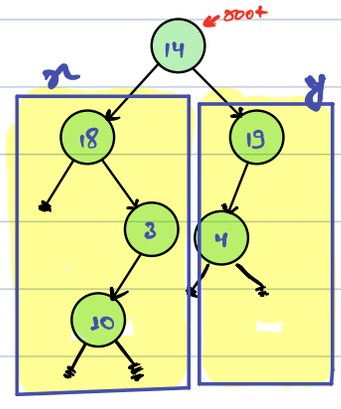
S.C: O(N)






**Faith:** Given root Node, find & return no. of Node in that tree.

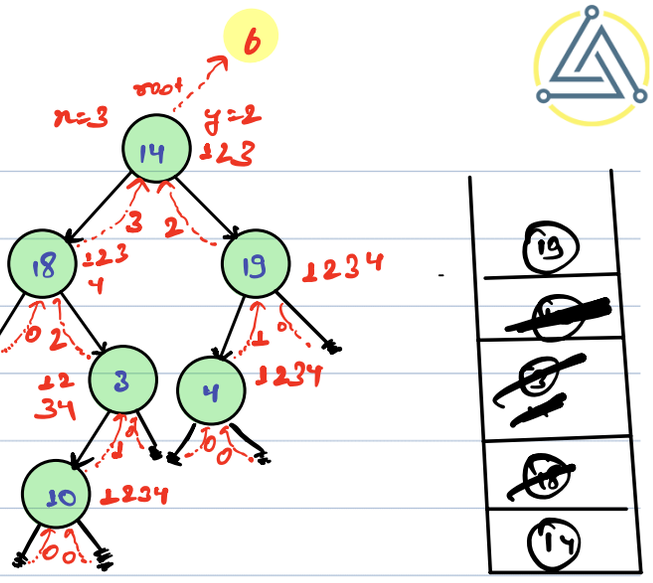
**Main logic:**




→ x+y+1

**base case:**

```
int size (node root) {
1   if (root == null) {return 0;}

2 int x = size (root.left);
3 int y = size (root.right);
4     return x+y+1;

}
```

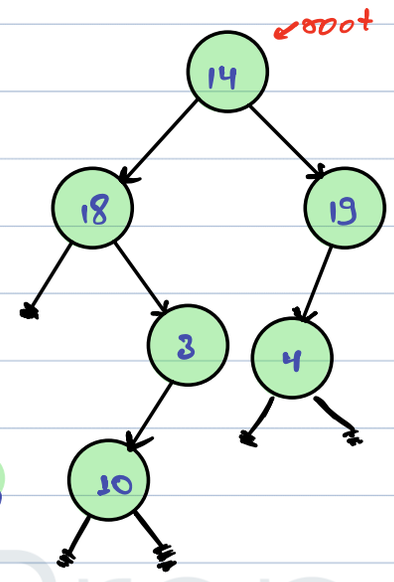



S.C: O(No. of Levels) ≈ O(N)

## Q) Sum of a tree

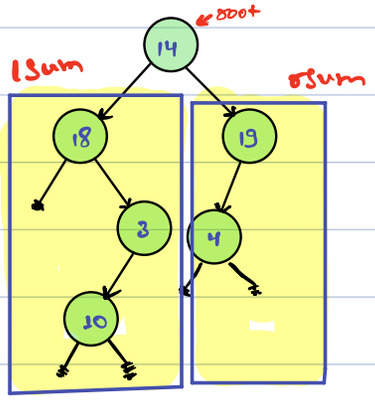
↳ Given a tree, calculate sum of all Nodes data in it.

```
int Sum (Node root) {
    if (root == null) { return 0; }

    int lsum = Sum (root.left);
    int rsum = Sum (root.right);
    return lsum + rsum + root.val;
}
```




**Faith:** Given root Node, find & return sum of Node in that tree.

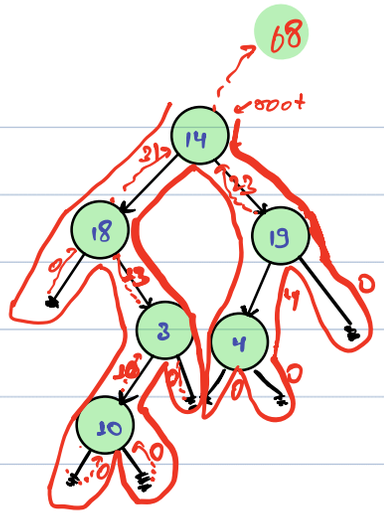
**Main logic:**




→ lsum + rsum + root.val

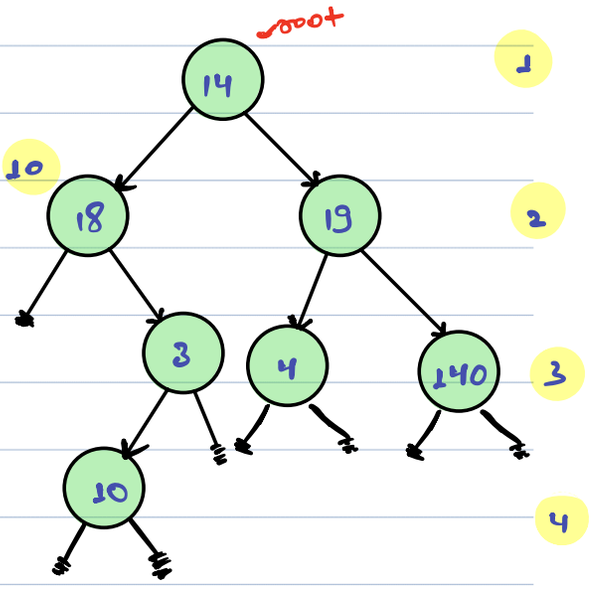
**base case:**

```
int Sum (Node root) {
    if (root == null) { return 0; }

    int lsum = Sum (root.left);
    int rsum = Sum (root.right);
        return lsum + rsum + root.val;
}
```



Preorder ↙ NLR  Inorder ↓ LNR  Postorder ↘ LRN

Q) Level order of tree




level order: 14 18 19 3 4 140 10

(node)

Queue:

Output:

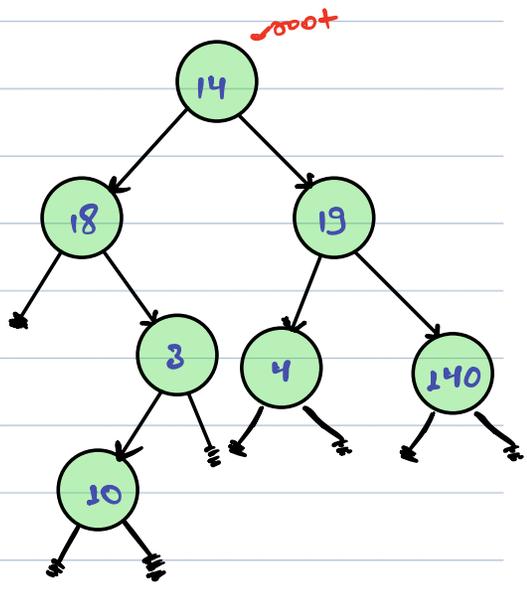
14 18 19 3 4 140 10

```
//Pseudo code

void levelorder (Node root){
    Queue<Node> q;
    q.add(root);

    while (q.size()>0) {
        Node temp = q.remove();
        System.out.print(temp.val);
        if (temp.left!=null){
            q.add(temp.left);
        }
        if (temp.right!=null){
            q.add(temp.right);
        }
    }
}
```

T.C: O(N)

S.C: O(N)

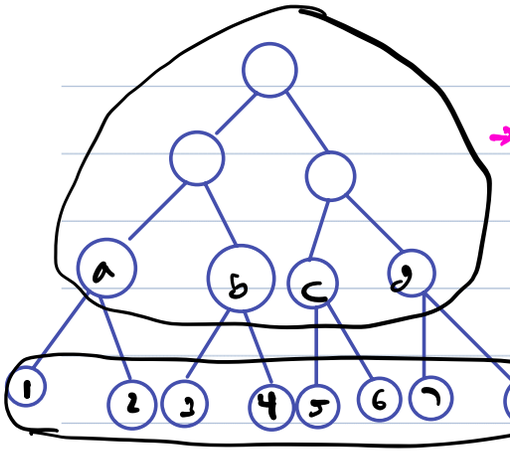
```
void levelorder (Node root){
    Queue<Node> q;
    q.add(root);

    while (q.size()>0) {
        Node temp = q.removel();
        System.out.print(temp.val);
        if (temp.left!=null){
            q.add(temp.left);
        }
        if (temp.right!=null) {
            q.add(temp.right);
        }
    }
}
```




q: ~~14~~ ~~18~~ ~~19~~ 3 4 140

temp = 19

14 18 19




→ O(N)

→ N-1 nodes

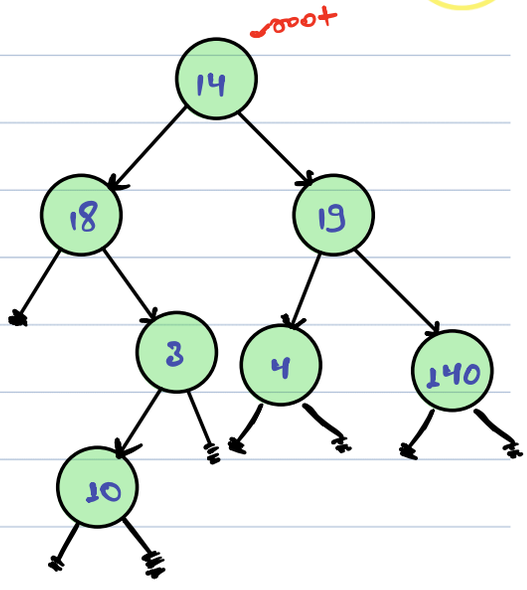
Total no. nodes : 2N-1 → O(N)

→ why? → HW

→ N nodes + O(N)

Break till 9:37 PM

## Q) Level order of tree 2




14

18 19

3 4 140

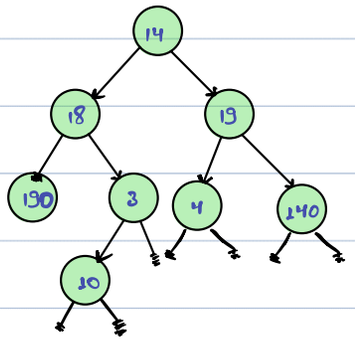
10

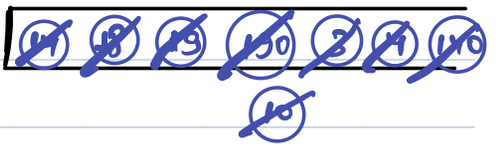
```
void levelorder (Node root){
    Queue < Node > q;
    q.add (root);
    while ( q.size () > 0 ) {
        int n = q.size ();
        for(int i=1; i<=n; i++) {
            Node temp = q.remove();
            System.out.print (temp.val);
            if ( temp.left != null){
                q.add (temp.left);
            }
            if ( temp.right != null) {
                q.add (temp.right);
            }
        }
        System.out.println();
    }
}
```

T.C: O(N)

S.C: O(N)

```
while ( q.size() > 0 ) {
    int n = q.size();
    for (int i = 1; i <= n; i++) {
        Node temp = q.removel();
        System.out.print (temp.val);
        if ( temp.left != null) {
            q.add (temp.left);
        }
        if ( temp.right != null) {
            q.add (temp.right);
        }
    }
    System.out.println();
}
```




q 

n = 1

14

18 19

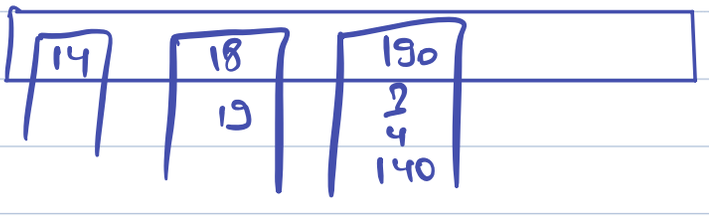
190 3 4 140

10

→

→ List < List < Integer > > ans;

ans:

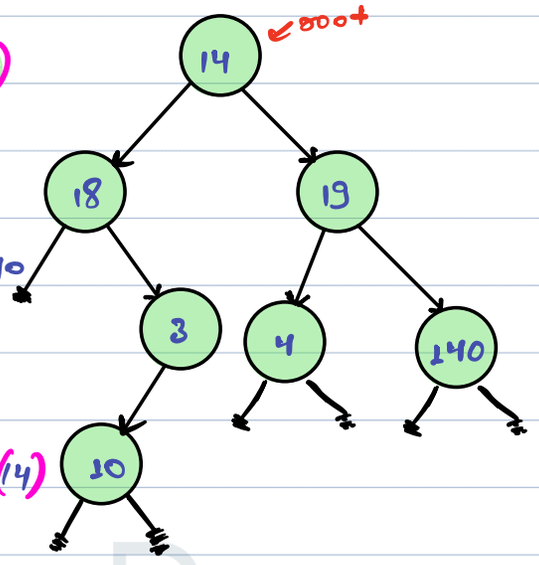


```
List<List<Integer>> levelOrder(Node root){
    Queue<Node> q;
    q.add(root);
    List<List<Integer>> ans = new ArrayList<>();
    while(q.size() > 0){
        int n = q.size();
        List<Integer> levelAns = new ArrayList<>();
        for(int i=1; i<=n; i++){
            Node temp = q.removeL();
            levelAns.add(temp.val);
            if(temp.left != null){
                q.add(temp.left);
            }
            if(temp.right != null){
                q.add(temp.right);
            }
        }
        ans.add(levelAns);
    }
    return ans;
}
```

## Q) Reverse level order

↳ Print the level order from last level to first.

Output: (10)(3 4 140)(18 19)(14)




↳ level order: 14 18 19 3 4 140 10

↓

(10) (140 4 3)(19 18)(14)

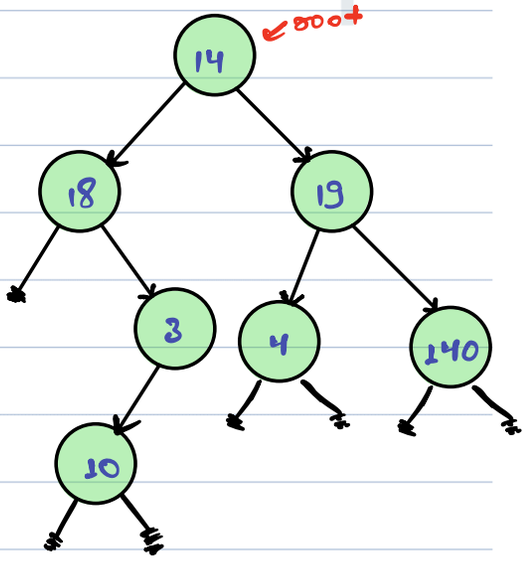
R-L

level order:

↳ 14 19 18 140 4 3 10

↓ reverse

(10) (3 4 140)(18 19)(14)



IIPSuedo Code

↳ HW

# Size of Tree

## Java Code:

```
class Tree
{
        public static int getSize(Node root)
        {
      if (root == null)
        return 0;


        int a = getSize(root.left);
        int b = getSize(root.right);

        return a+b+1;
   }
}
```

## C++ Code:

```
#include <iostream>

class Node {
public:
   int data;
   Node* left;
   Node* right;

   Node(int value) : data(value), left(nullptr), right(nullptr) {}
};

class Tree {
public:
   static int getSize(Node* root) {
      if (root == nullptr)
         return 0;

      int a = getSize(root->left);
      int b = getSize(root->right);

      return a + b + 1;
   }
```

```
};

int main() {
    // You can create a sample binary tree and test the getSize function here
    return 0;
}
```

## Python Code:

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

class Tree:
    @staticmethod
    def get_size(root):
        if root is None:
            return 0

        a = Tree.get_size(root.left)
        b = Tree.get_size(root.right)

        return a + b + 1
```

# Sum of Tree

## Java Code:

```
class BinaryTree
{
    static int sumBT(Node head){
        if (head == null)
            return 0;


            int a = sumBT(head.left);
            int b = sumBT(head.right);

            return a+b+head.data;
    }
}
```

## C++ Code:

```
#include <iostream>

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int value) : data(value), left(nullptr), right(nullptr) {}
};

class BinaryTree {
public:
    static int sumBT(Node* head) {
        if (head == nullptr)
            return 0;
```

```
        int a = sumBT(head->left);
        int b = sumBT(head->right);

        return a + b + head->data;
    }
};

int main() {
    // You can create a sample binary tree and test the sumBT function here
    return 0;
}
```

## Python Code:

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

class BinaryTree:
    @staticmethod
    def sumBT(head):
        if head is None:
            return 0

        a = BinaryTree.sumBT(head.left)
        b = BinaryTree.sumBT(head.right)

        return a + b + head.data
```

# LevelOrder of Tree

## Java Code:

Java

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        List<List<Integer>> wrapList = new
LinkedList<List<Integer>>();

        if(root == null) return wrapList;

        queue.offer(root);
        while(!queue.isEmpty()){
            int size = queue.size();
            List<Integer> subList = new LinkedList<Integer>();
            for(int i=1; i <= size; i++) {
                if(queue.peek().left != null)
queue.offer(queue.peek().left);
                if(queue.peek().right != null)
queue.offer(queue.peek().right);
                subList.add(queue.poll().val);
            }
            wrapList.add(subList);
        }
        return wrapList;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> wrapList;

        if (root == nullptr) return wrapList;

        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            int size = q.size();
            vector<int> subList;

            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();

                subList.push_back(node->val);

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }

            wrapList.push_back(subList);
        }

        return wrapList;
    }
};
```

## Python Code:

```
from collections import deque

# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        wrapList = []

        if not root:
            return wrapList

        queue = deque()
        queue.append(root)

        while queue:
            size = len(queue)
            subList = []

            for _ in range(size):
                node = queue.popleft()
                subList.append(node.val)

                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

            wrapList.append(subList)

        return wrapList
```

# Size of Tree

## Java Code:

```
class Tree
{
    public ArrayList<Integer> reverseLevelOrder(Node node)
    {
        // code here
        Stack<Node> s = new Stack<Node>();

        Queue<Node> q = new LinkedList<Node>();

            ArrayList<Integer> t = new ArrayList<Integer>();
            if(node != null){
                q.add(node);
                s.push(node);
            }

            while(q.size() > 0){
              Node temp = q.remove();
              //t.Add(temp.data);

            if(temp.right != null){
                  q.add(temp.right);
                  s.push(temp.right);
              }
              if(temp.left != null){
                  q.add(temp.left);
                  s.push(temp.left);
              }



            }

            while(s.size()>0){
                Node temp = s.pop();
                t.add(temp.data);
            }

            return t;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>

using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Tree {
public:
    vector<int> reverseLevelOrder(TreeNode* node) {
        vector<int> t;
        if (!node)
            return t;

        stack<TreeNode*> s;
        queue<TreeNode*> q;

        q.push(node);
        s.push(node);

        while (!q.empty()) {
            TreeNode* temp = q.front();
            q.pop();

            if (temp->right) {
                q.push(temp->right);
                s.push(temp->right);
            }
            if (temp->left) {
                q.push(temp->left);
                s.push(temp->left);
            }
        }

        while (!s.empty()) {
            TreeNode* temp = s.top();
            s.pop();
```

```
            t.push_back(temp->val);
        }

        return t;
    }
};
```

## Python Code:

```
from collections import deque

# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Tree:
    def reverseLevelOrder(self, node: TreeNode) -> list[int]:
        t = []
        if not node:
            return t

        stack = []
        queue = deque()

        queue.append(node)
        stack.append(node)

        while queue:
            temp = queue.popleft()

            if temp.right:
                queue.append(temp.right)
                stack.append(temp.right)
            if temp.left:
                queue.append(temp.left)
                stack.append(temp.left)

        while stack:
            temp = stack.pop()
            t.append(temp.val)
```

return t

## Zig zag Tree Traversal_HW

### Solution vid:

https://youtu.be/vWBSRdtUVsQ

### Java Code:

## Maximum Level Sum_HW

### Solution vid:

https://youtu.be/fVSkSm5NLP4

### Java Code:

## Size of Tree_HW

### Solution vid:

https://youtu.be/96V1RT7CvOU

### Java Code:

# SMaximum Depth of Tree_HW

## Solution vid:

https://youtu.be/zFarfV2UFk0

## Java Code:

Today's agenda

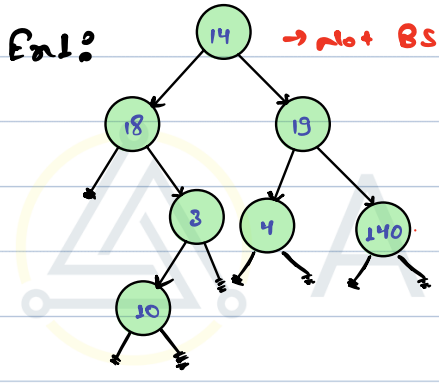
- BST
- Search in BST
- isBST
- insert in BST
- Recover BST

BST: Binary Search tree

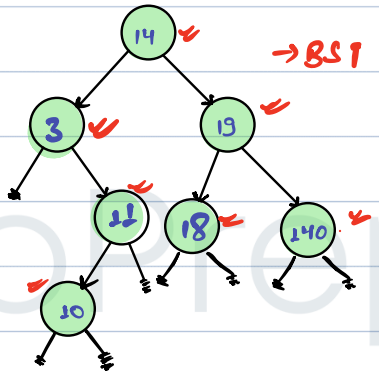
↳ A Binary Tree is a BST if:

for all nodes: All the nodes in the left subtree < node.val < All the nodes in the right subtree

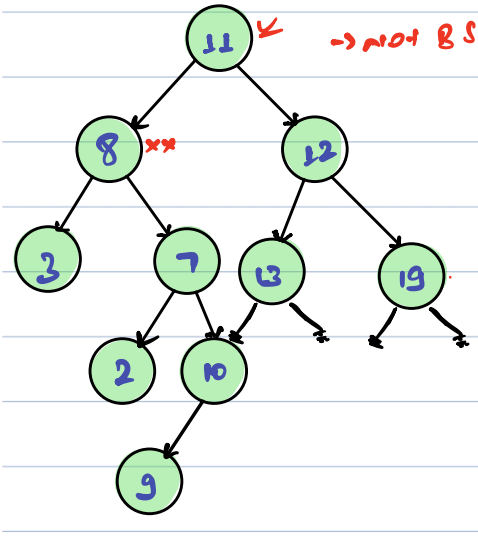
Ex1:



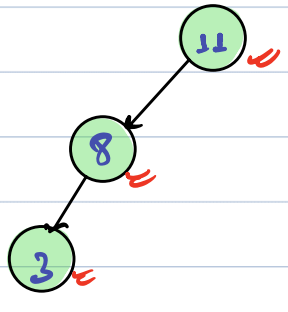

→ not BST




→ BST

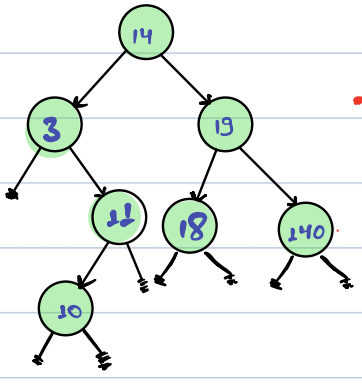



→ not BST



Property :

↳ Inorder in BST is always sorted.



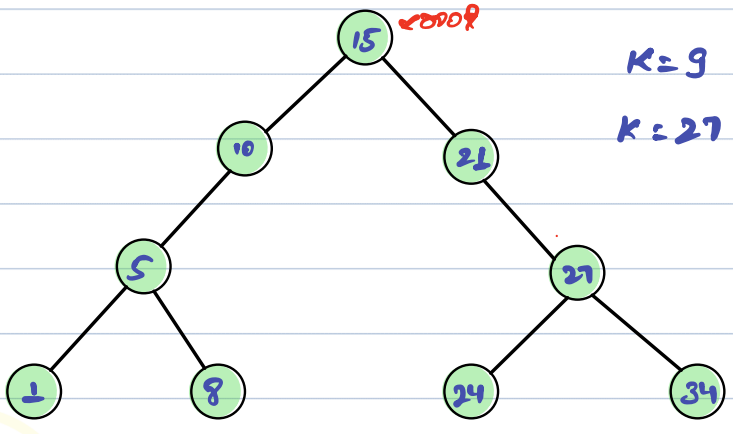

→ 3 10 11 14 18 19 140

↳ LNR

note: BST in general Can't have duplicate values.

Q) Search K in BST

↳ Given a BST and a value K, return true if a node with value K is present else false.




K = 9 → false

K = 27 → true

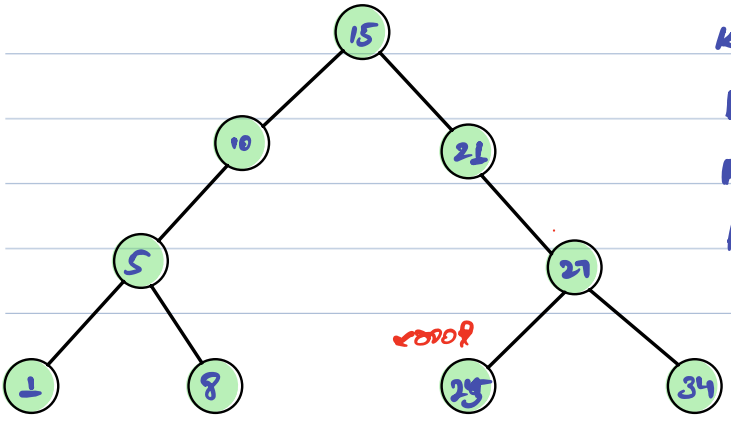


**Idea1**

↳ Write any traversal and check if K is present?

T.C: O(N) S.C: O(N)

**Idea2**




K = 24

K > 15 → go to right

K > 21 → go to right

K < 27 → go to left

K == 24 → return true

//Pseudo code

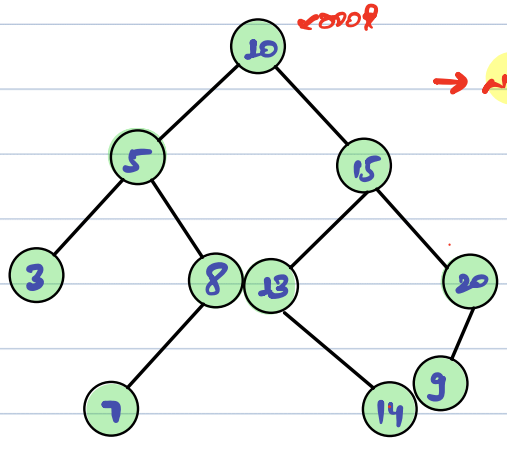
```
boolean search (Node root, int K){

    while (root != null) {
        if (root.val == K) {
            return true;
        }
        else if (root.val > K){
            root = root.left;
        }
        else {
            root = root.right;
        }
    }

    return false;
}
```

T.C: O(H) → O(N) / O(logN)

S.C: O(1)

Q) isBST?

↳ Given BT, Check if it is a BST or Not?




→ not BST

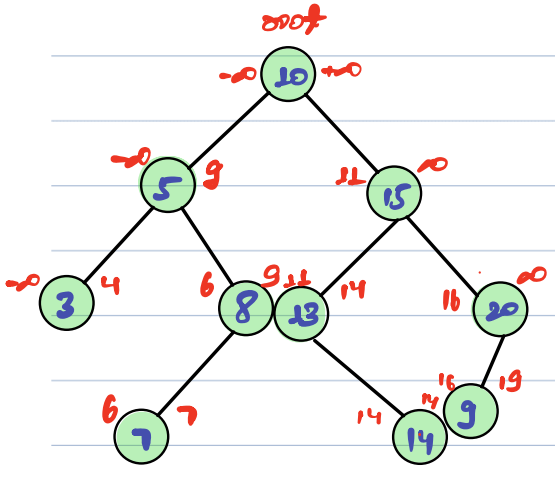


**Idea1**

↳ (i) Do inorder and store all the elements in array.

↳ (ii) Check whether array is sorted or not?

**Idea2**




min = -∞ max = +∞

going to left : (par.min, par.val − 1)

going to right : (par.val + 1, par.max)

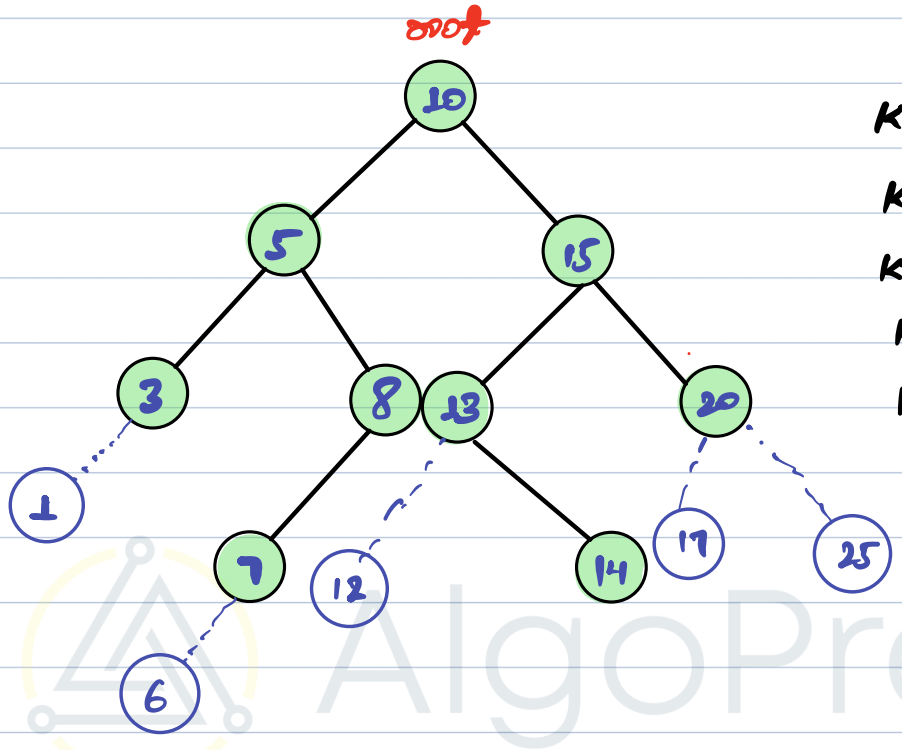
//Pseudo Code

```
boolean isBST (Node root, int min, int max){
    if (root == null){ return true;}
    if (root.val < min || root.val > max){
        return false;
    }
    boolean l = isBST (root.left, min, root.val-1);
    boolean r = isBST (root.right, root.val+1, max);
    if (l == false || r == false){
        return false;
    }
    return true;
}
```

T.C : O(N)

S.C : O(H)

Q) insert a node with value K in a BST.




K = 1

K = 25

K = 6

K = 12

K = 17

↳ you can always add as a child node of a leaf node.

//Pseudo code

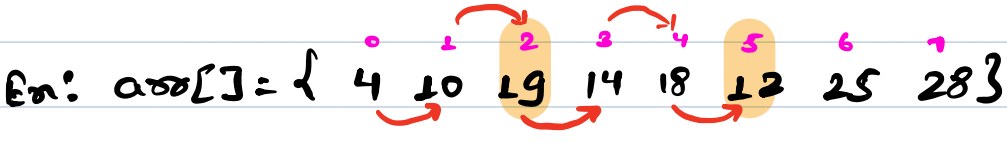
```
void insert (node root, int k){

    node n = new node (k);

    while (true) {
        if(root.val > k){
            if (root.left == null){
                root.left = n;
                return;
            }else {
                root= root.left;
            }
        }
        else{
            if (root.right == null){
                root.right = n;
                return;
            }
            else{
                root = root.right;
            }
        }
    }
}
```

T.C: O(H)

S.C: O(1)

## Q) Recover sorted arr[]

↳ Given an arr[] which is formed by swapping 2 distinct index in a sorted (inc) arr: {1 swap}



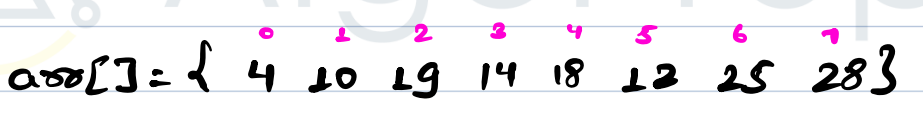

→ Compare Consecutive indexes.

↳ 1st failed condition: ith index is the faulty one.

↳ 2nd failed condition: (i+1)th index is the faulty one.



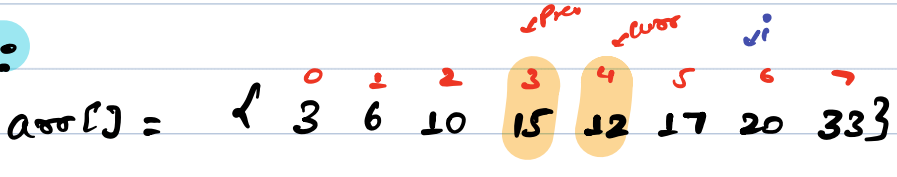
### Algo




ansIdx1 = ~~-1~~ 2

ansIdx2 = ~~-1~~ 5

### edge case:



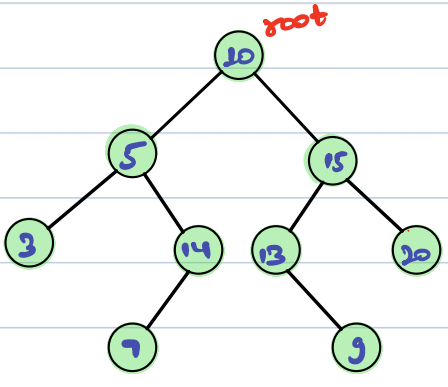

ansIdx1 = ~~-1~~ 3

ansIdx2 = ~~-1~~ 4

Q) Recover BST

↳ Given a BST, which is formed by swapping 2 distinct nodes, recover original BST.

↳ inorder of BST is sorted.




```
Node f;
Node s;
Node Prev;

void inorder (Node Curr){
    if (Curr == null) {return;}

    inorder (Curr.left);
    if (Prev != null && Curr.val < Prev.val && f == null){
        f = Prev;
        s = Curr;
    }
    else if (Prev != null && Curr.val < Prev.val && f != null){
        s = Curr;
    }
    Prev = Curr;
    inorder (Curr.right);
}
```

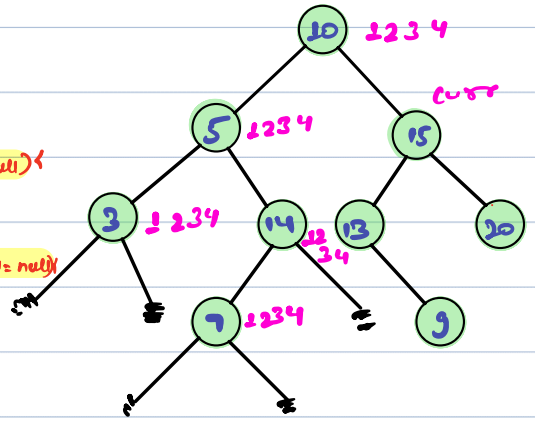
T.C: O(N)

S.C: O(H)

Prev: null 3 5 7 14 10

f = null (14)  s: null 10

```
void inorder (Node Curr){
1 if (Curr == null) {return;}
2 inorder (Curr.left);
3 if (Prev != null && Curr.val < Prev.val && f == null){
      f = Prev;
      s = Curr;
  }
  else if (Prev != null && Curr.val < Prev.val && f != null){
      s = Curr;
  }
  Prev = Curr;
4 inorder (Curr.right);
}
```




Prev Curr

3 5 7 14 10

# Left View of Binary Tree

## Java Code:

```
class Tree
{
    //Function to return list containing elements of left view of binary tree.
    ArrayList<Integer> leftView(Node root)
    {
      ArrayList<Integer> ans = new ArrayList<>();
         Queue<Node> q = new LinkedList<>();
        if(root == null) return ans;
         q.add(root);
         while(q.size() >0){
             int n = q.size();
             for(int i = 1; i<=n; i++){
                 Node temp = q.remove();
                 if(i==1){
                   ans.add(temp.data);
                 }
                 if(temp.left != null){
                     q.add(temp.left);
                 }
                 if(temp.right != null){
                     q.add(temp.right);
                 }
             }
         }
         return ans;
    }
}
```

## C++ Code:

```
#include <vector>
#include <queue>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
```

```
};

vector<int> leftView(Node* root) {
    vector<int> ans;
    if (!root) return ans;

    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        int n = q.size();
        for (int i = 1; i <= n; i++) {
            Node* temp = q.front();
            q.pop();

            if (i == 1) {
                ans.push_back(temp->data);
            }

            if (temp->left) {
                q.push(temp->left);
            }
            if (temp->right) {
                q.push(temp->right);
            }
        }
    }

    return ans;
}
```

## Python Code:

```
from collections import deque

class Node:
    def __init__(self, val):
        self.data = val
        self.left = None
        self.right = None

def left_view(root):
    ans = []
    if not root:
        return ans
```

```
    q = deque()
    q.append(root)

    while q:
        n = len(q)
        for i in range(1, n+1):
            temp = q.popleft()

            if i == 1:
                ans.append(temp.data)

            if temp.left:
                q.append(temp.left)
            if temp.right:
                q.append(temp.right)

    return ans
```

# Right View of Binary Tree

## Java Code:

```
class Solution{
    //Function to return list containing elements of right view of binary tree.
    ArrayList<Integer> rightView(Node node) {
      ArrayList<Integer> ans = new ArrayList<>();
         Queue<Node> q = new LinkedList<>();
        if(node == null) return ans;
         q.add(node);
         while(q.size() >0){
             int n = q.size();
             for(int i = 1; i<=n; i++){
                 Node temp = q.remove();
                 if(i==n){
                   ans.add(temp.data);
                 }
                 if(temp.left != null){
                     q.add(temp.left);
                 }
                 if(temp.right != null){
                     q.add(temp.right);
                 }
```

```
            }
        }
        return ans;
    }
}
```

## C++ Code:

```
#include <vector>
#include <queue>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

vector<int> rightView(Node* node) {
    vector<int> ans;
    if (!node) return ans;

    queue<Node*> q;
    q.push(node);

    while (!q.empty()) {
        int n = q.size();
        for (int i = 1; i <= n; i++) {
            Node* temp = q.front();
            q.pop();

            if (i == n) {
                ans.push_back(temp->data);
            }

            if (temp->left) {
                q.push(temp->left);
            }
            if (temp->right) {
                q.push(temp->right);
            }
        }
    }

    return ans;
```

}

## Python Code:

```
from collections import deque

class Node:
    def __init__(self, val):
        self.data = val
        self.left = None
        self.right = None

def right_view(node):
    ans = []
    if not node:
        return ans

    q = deque()
    q.append(node)

    while q:
        n = len(q)
        for i in range(1, n+1):
            temp = q.popleft()

            if i == n:
                ans.append(temp.data)

            if temp.left:
                q.append(temp.left)
            if temp.right:
                q.append(temp.right)

    return ans
```

# Top View of Binary Tree

## Java Code:

```
 class Pair{
    Node n;
```

```
    int vtlevel;
    Pair(Node n1 , int x){
        n = n1;
        vtlevel = x;
    }
}
class Solution
{
    //Function to return a list of nodes visible from the top view
    //from left to right in Binary Tree.
    static ArrayList<Integer> topView(Node root)
    {
        ArrayList<Integer> ans = new ArrayList<>();
        Queue<Pair> q = new LinkedList<>();
        HashMap<Integer , Integer> map = new HashMap<>();
        if(root == null) return ans;
        int min = Integer.MAX_VALUE,max = 0;
        q.add(new Pair(root,0));
        while(q.size() > 0){
            Pair rem = q.remove();
            Node remn = rem.n;
            int remvl = rem.vtlevel;
            min = Math.min(min , remvl);
            max = Math.max(max , remvl);
            if(map.containsKey(remvl) == false){
                map.put(remvl , remn.data);
            }
            if(remn.left != null){
                q.add(new Pair(remn.left , remvl-1));
            }
            if(remn.right != null){
                q.add(new Pair(remn.right ,remvl+1));
            }
        }
        for(int i = min; i<=max; i++){
            ans.add(map.get(i));
        }
        return ans;

    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <queue>
```

```
#include <unordered_map>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class Pair {
public:
    Node* n;
    int vtlevel;
    Pair(Node* n1, int x) : n(n1), vtlevel(x) {}
};

vector<int> topView(Node* root) {
    vector<int> ans;
    queue<Pair> q;
    unordered_map<int, int> map;
    if (!root) return ans;
    int minVal = INT_MAX, maxVal = 0;
    q.push(Pair(root, 0));

    while (!q.empty()) {
        Pair rem = q.front();
        q.pop();
        Node* remn = rem.n;
        int remvl = rem.vtlevel;
        minVal = min(minVal, remvl);
        maxVal = max(maxVal, remvl);

        if (map.find(remvl) == map.end()) {
            map[remvl] = remn->data;
        }

        if (remn->left) {
            q.push(Pair(remn->left, remvl - 1));
        }
        if (remn->right) {
            q.push(Pair(remn->right, remvl + 1));
        }
    }

    for (int i = minVal; i <= maxVal; i++) {
        ans.push_back(map[i]);
```

```
    }

    return ans;
}
```

## Python Code:

```
from collections import deque

class Node:
    def __init__(self, val):
        self.data = val
        self.left = None
        self.right = None

class Pair:
    def __init__(self, n1, x):
        self.n = n1
        self.vtlevel = x

def top_view(root):
    ans = []
    q = deque()
    map = {}
    if not root:
        return ans
    min_val, max_val = float('inf'), 0
    q.append(Pair(root, 0))

    while q:
        rem = q.popleft()
        remn = rem.n
        remvl = rem.vtlevel
        min_val = min(min_val, remvl)
        max_val = max(max_val, remvl)

        if remvl not in map:
            map[remvl] = remn.data

        if remn.left:
            q.append(Pair(remn.left, remvl - 1))
        if remn.right:
            q.append(Pair(remn.right, remvl + 1))

    for i in range(min_val, max_val + 1):
        ans.append(map[i])
```

```
    return ans
```

# Bottom View of Binary Tree

## Java Code:

```
class Pair{
    Node n;
    int vtlevel;
    Pair(Node n1 , int x){
        n = n1;
        vtlevel = x;
    }
}

class Solution
{
    //Function to return a list containing the bottom view of the given tree.
    public ArrayList <Integer> bottomView(Node root)
    {
        ArrayList<Integer> ans = new ArrayList<>();
        Queue<Pair> q = new LinkedList<>();
        HashMap<Integer , Integer> map = new HashMap<>();
        if(root == null) return ans;
        int min = Integer.MAX_VALUE,max = 0;
        q.add(new Pair(root,0));
        while(q.size() > 0){
            Pair rem = q.remove();
            Node remn = rem.n;
            int remvl = rem.vtlevel;
            min = Math.min(min , remvl);
            max = Math.max(max , remvl);

            map.put(remvl , remn.data);

            if(remn.left != null){
                q.add(new Pair(remn.left , remvl-1));
            }
            if(remn.right != null){
                q.add(new Pair(remn.right ,remvl+1));
            }
```

```
        }
        for(int i = min; i<=max; i++){
            ans.add(map.get(i));
        }
        return ans;

    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class Pair {
public:
    Node* n;
    int vtlevel;
    Pair(Node* n1, int x) : n(n1), vtlevel(x) {}
};

vector<int> bottomView(Node* root) {
    vector<int> ans;
    queue<Pair> q;
    unordered_map<int, int> map;
    if (!root) return ans;
    int minVal = INT_MAX, maxVal = 0;
    q.push(Pair(root, 0));

    while (!q.empty()) {
        Pair rem = q.front();
        q.pop();
        Node* remn = rem.n;
        int remvl = rem.vtlevel;
        minVal = min(minVal, remvl);
        maxVal = max(maxVal, remvl);
```

```
        map[remvl] = remn->data;

        if (remn->left) {
            q.push(Pair(remn->left, remvl - 1));
        }
        if (remn->right) {
            q.push(Pair(remn->right, remvl + 1));
        }
    }

    for (int i = minVal; i <= maxVal; i++) {
        ans.push_back(map[i]);
    }

    return ans;
}
```

## Python Code:

```
from collections import deque

class Node:
    def __init__(self, val):
        self.data = val
        self.left = None
        self.right = None

class Pair:
    def __init__(self, n1, x):
        self.n = n1
        self.vtlevel = x

def bottom_view(root):
    ans = []
    q = deque()
    map = {}
    if not root:
        return ans
    min_val, max_val = float('inf'), 0
    q.append(Pair(root, 0))

    while q:
        rem = q.popleft()
        remn = rem.n
        remvl = rem.vtlevel
```

```
        min_val = min(min_val, remvl)
        max_val = max(max_val, remvl)

        map[remvl] = remn.data

        if remn.left:
            q.append(Pair(remn.left, remvl - 1))
        if remn.right:
            q.append(Pair(remn.right, remvl + 1))

    for i in range(min_val, max_val + 1):
        ans.append(map[i])

    return ans
```

# Vertical Traversal of Binary Tree

## Java Code:

```
class Pair{
    Node n;
    int vtlevel;
    Pair(Node n1 , int x){
        n = n1;
        vtlevel = x;
    }
}
class Solution
{
    //Function to find the vertical order traversal of Binary Tree.
    static ArrayList <Integer> verticalOrder(Node root)
    {

        ArrayList<Integer> ans = new ArrayList<>();
        Queue<Pair> q = new LinkedList<>();
        HashMap<Integer , ArrayList<Integer>> map = new HashMap<>();
        if(root == null) return ans;
        int min = Integer.MAX_VALUE,max = 0;
        q.add(new Pair(root,0));
        while(q.size() > 0){
                Pair rem = q.remove();
                Node remn = rem.n;
                int remvl = rem.vtlevel;
```

```
                min = Math.min(min , remvl);
                max = Math.max(max , remvl);
                if(map.containsKey(remvl) == false){
                    map.put(remvl , new ArrayList<>());
                }
                map.get(remvl).add(remn.data);

                if(remn.left != null){
                    q.add(new Pair(remn.left , remvl-1));
                }
                if(remn.right != null){
                    q.add(new Pair(remn.right ,remvl+1));
                }
            }

        for(int i = min; i<=max; i++){
            ArrayList<Integer> temp = map.get(i);
            for(int ele : temp) ans.add(ele);
        }
        return ans;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class Pair {
public:
    Node* n;
    int vtlevel;
    Pair(Node* n1, int x) : n(n1), vtlevel(x) {}
};

vector<int> verticalOrder(Node* root) {
```

```
    vector<int> ans;
    queue<Pair> q;
    unordered_map<int, vector<int>> map;
    if (!root) return ans;
    int minVal = INT_MAX, maxVal = 0;
    q.push(Pair(root, 0));

    while (!q.empty()) {
        Pair rem = q.front();
        q.pop();
        Node* remn = rem.n;
        int remvl = rem.vtlevel;
        minVal = min(minVal, remvl);
        maxVal = max(maxVal, remvl);

        if (map.find(remvl) == map.end()) {
            map[remvl] = vector<int>();
        }
        map[remvl].push_back(remn->data);

        if (remn->left) {
            q.push(Pair(remn->left, remvl - 1));
        }
        if (remn->right) {
            q.push(Pair(remn->right, remvl + 1));
        }
    }

    for (int i = minVal; i <= maxVal; i++) {
        vector<int>& temp = map[i];
        for (int ele : temp) {
            ans.push_back(ele);
        }
    }

    return ans;
}
```

## Python Code:

```
from collections import deque

class Node:
    def __init__(self, val):
        self.data = val
        self.left = None
```

```
        self.right = None

class Pair:
    def __init__(self, n1, x):
        self.n = n1
        self.vtlevel = x

def vertical_order(root):
    ans = []
    q = deque()
    map = {}
    if not root:
        return ans
    min_val, max_val = float('inf'), 0
    q.append(Pair(root, 0))

    while q:
        rem = q.popleft()
        remn = rem.n
        remvl = rem.vtlevel
        min_val = min(min_val, remvl)
        max_val = max(max_val, remvl)

        if remvl not in map:
            map[remvl] = []

        map[remvl].append(remn.data)

        if remn.left:
            q.append(Pair(remn.left, remvl - 1))
        if remn.right:
            q.append(Pair(remn.right, remvl + 1))

    for i in range(min_val, max_val + 1):
        temp = map[i]
        ans.extend(temp)

    return ans
```

# Construct Binary Tree from Preorder and Inorder Traversal

## Java Code:

```
class Solution {
    HashMap<Integer, Integer> map;
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        int n = inorder.length;
        map = new HashMap<>();
        for(int i = 0; i<n; i++){
            map.put(inorder[i] , i);
        }
        return construct(preorder , 0 , n-1 , inorder , 0,n-1);
    }
    public TreeNode construct(int[] pre , int ps,int pe , int[] in , int is, int ie){
        if(ps>pe || is>ie) return null;

        TreeNode root = new TreeNode(pre[ps]);
        int idx = map.get(pre[ps]);

        int count = idx-is;
        root.left = construct(pre ,ps+1,ps+count, in , is , idx-1);
        root.right = construct(pre ,ps+count+1,pe, in , idx+1 , ie);

        return root;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
class Solution {
private:
    unordered_map<int, int> map;

public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        int n = inorder.size();
        map.clear();
        for (int i = 0; i < n; i++) {
            map[inorder[i]] = i;
        }
        return construct(preorder, 0, n - 1, inorder, 0, n - 1);
    }

    TreeNode* construct(vector<int>& pre, int ps, int pe, vector<int>& in, int is, int ie) {
        if (ps > pe || is > ie) return nullptr;

        TreeNode* root = new TreeNode(pre[ps]);
        int idx = map[pre[ps]];

        int count = idx - is;
        root->left = construct(pre, ps + 1, ps + count, in, is, idx - 1);
        root->right = construct(pre, ps + count + 1, pe, in, idx + 1, ie);

        return root;
    }
};
```

# Python Code:

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def __init__(self):
        self.map = {}

    def buildTree(self, preorder, inorder):
        n = len(inorder)
        self.map.clear()
        for i in range(n):
            self.map[inorder[i]] = i
        return self.construct(preorder, 0, n - 1, inorder, 0, n - 1)
```

```
def construct(self, pre, ps, pe, ino, is, ie):
    if ps > pe or is > ie:
        return None

    root = TreeNode(pre[ps])
    idx = self.map[pre[ps]]

    count = idx - is
    root.left = self.construct(pre, ps + 1, ps + count, ino, is, idx - 1)
    root.right = self.construct(pre, ps + count + 1, pe, ino, idx + 1, ie)

    return root
```

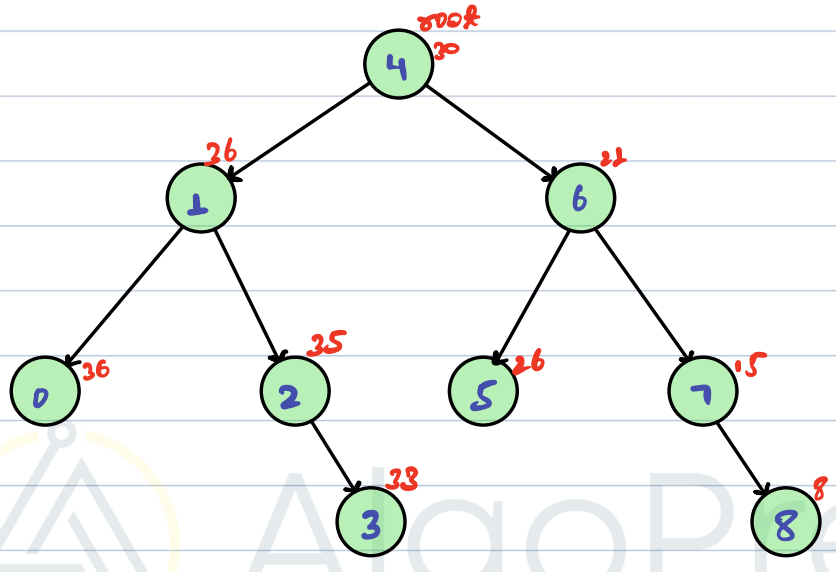
Today's agenda

- BST to greater sum
- Inorder successor in BST
- Populate next Pointer on right.
- Path from root to given Node.
- LCA in Binary tree
- LCA in BST.

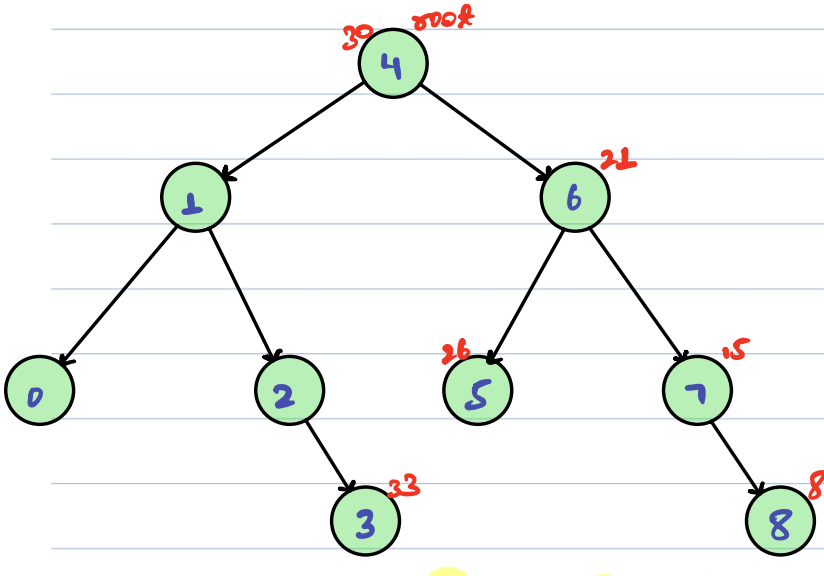
{Leetcode 1038}

Q) BST to greater sum

↳ Convert every node of BST to the original key plus the sum of all keys greater than the original key in BST.

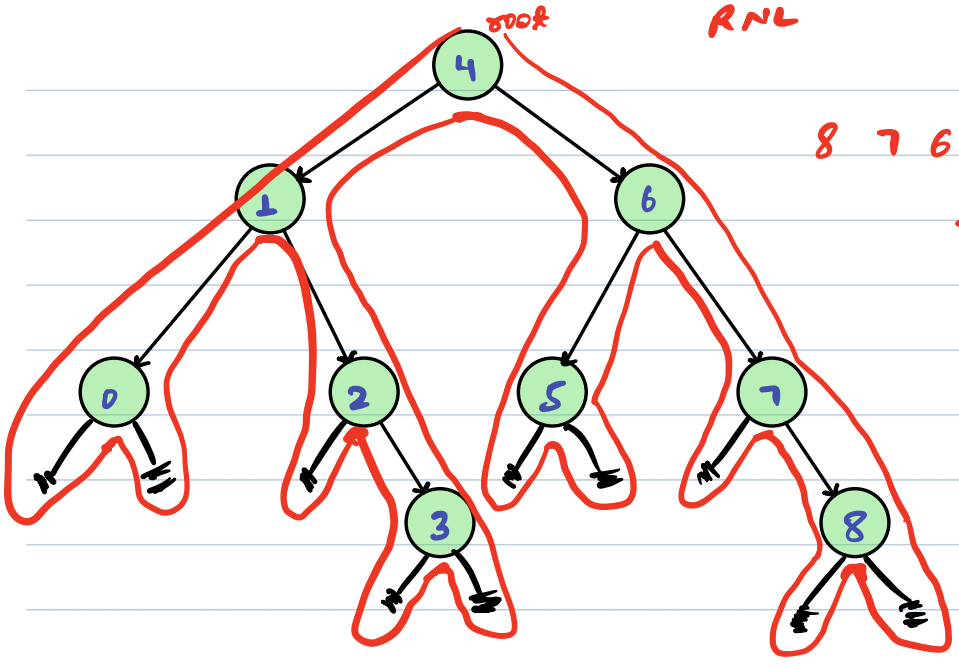



→ inorder: Sorted in inc. order → LNR




greater = 0+8+7 = 15+6 = 21
= 21+5 = 26+4 = 30+3
= 33

→ Reverse inorder: Sorted in dec. order → RNL






```
//Pseudo code
Public Node SumtoGreater (Node root){
    helper(root);
    return root;
}
int greater=0;
Public void helper (Node root){
    if (root==null){return;}

    helper (root.right);
    greater = greater + root.val;
    root.val = greater;
    helper (root.left);
}
```

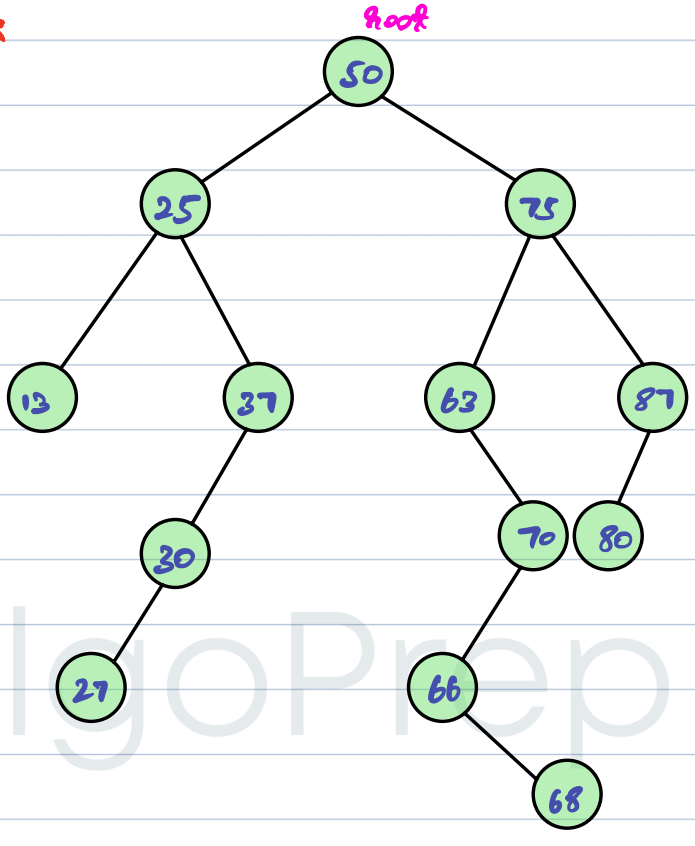
T.C: O(N)

S.C: O(H)

Q) Inorder successor of K in BST

↓

Just greater to K




IS(68) : 70

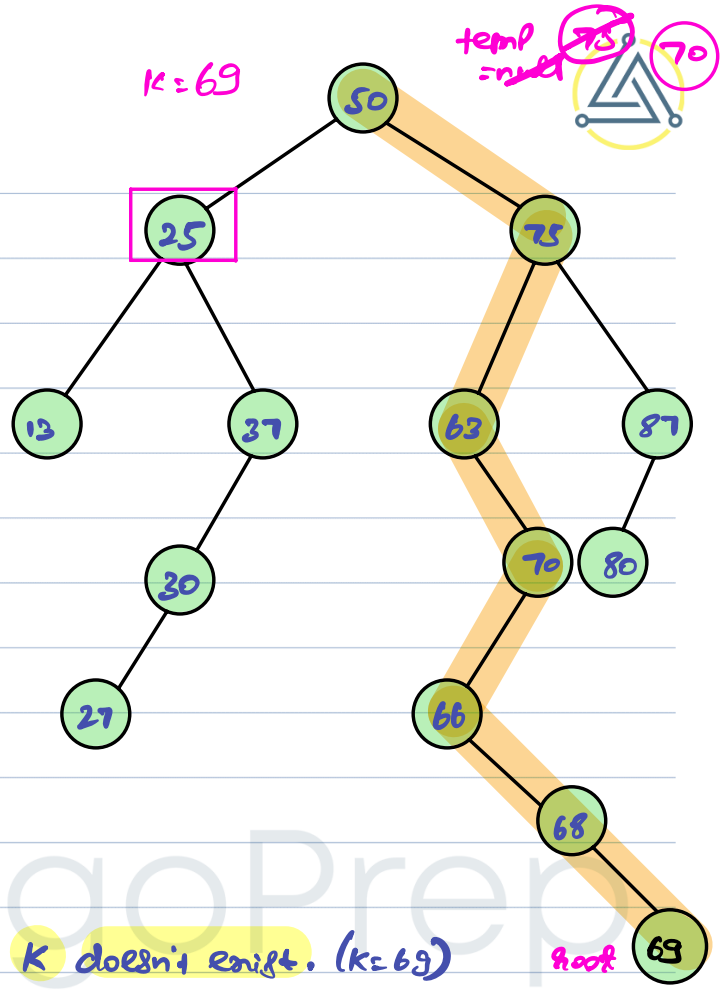


//idea 1

↳ inorder traversal → Sorted value

↳ The first val >K is your ans.

T.C: O(N)




**Obs 1:**

① if right subtree of K exists.

⇓

ans = leftmost node of that subtree.

**Obs 2:** if rightSubtree of K doesn't exist. (K=69)

⇓

ans = last node from where you taking left turn.

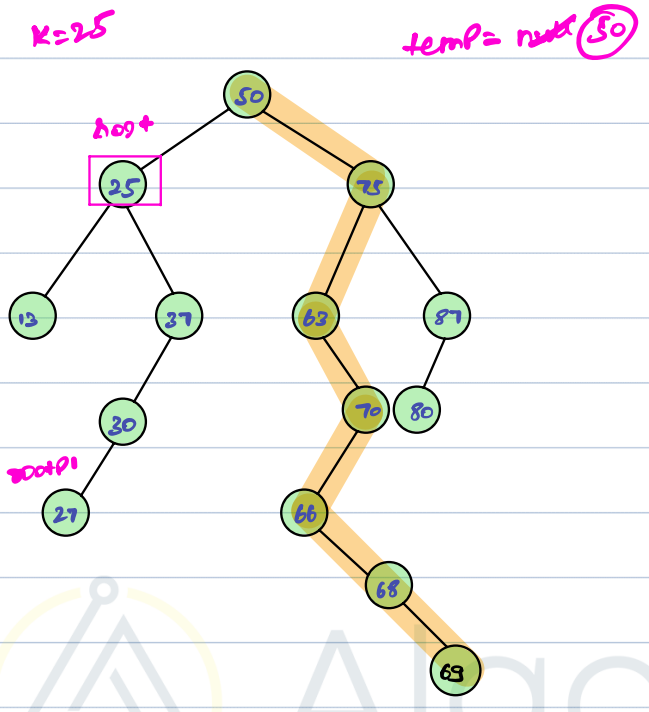
//Pseudo code

```
Node inorderSuccessor (Node root, int k){
    Node temp = null;

    while (true) {
        if (root.val > k){
            temp = root;
            root = root.left;
        }
        else if (root.val < k){
            root = root.right;
        }
        else
            break;
    }
    if (root.right == null){
        return temp;
    }
    Node rootPl = root.right;
    while (rootPl.left != null){
        rootPl = rootPl.left;
    }
    return rootPl;
}
```

T.C: O(H)

S.C: O(1)

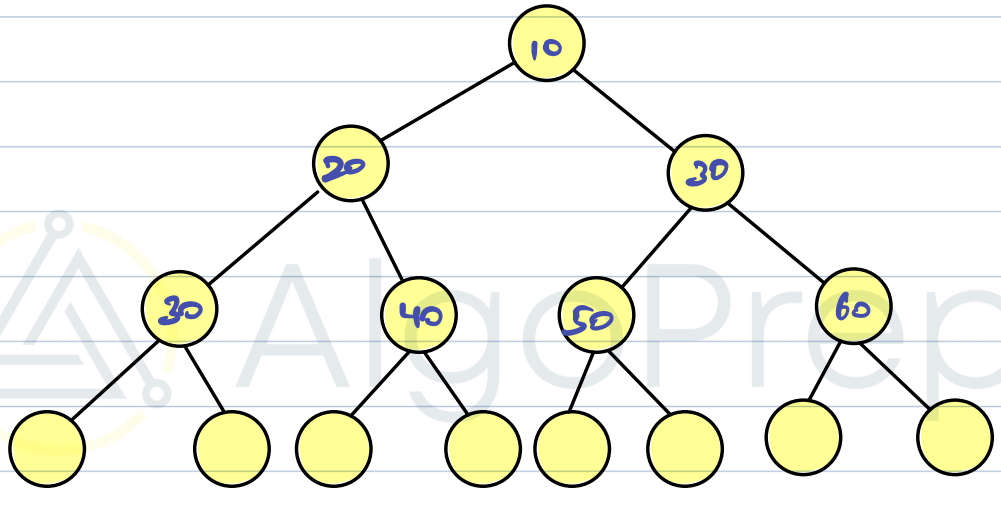
Tracing



* Perfect binary tree

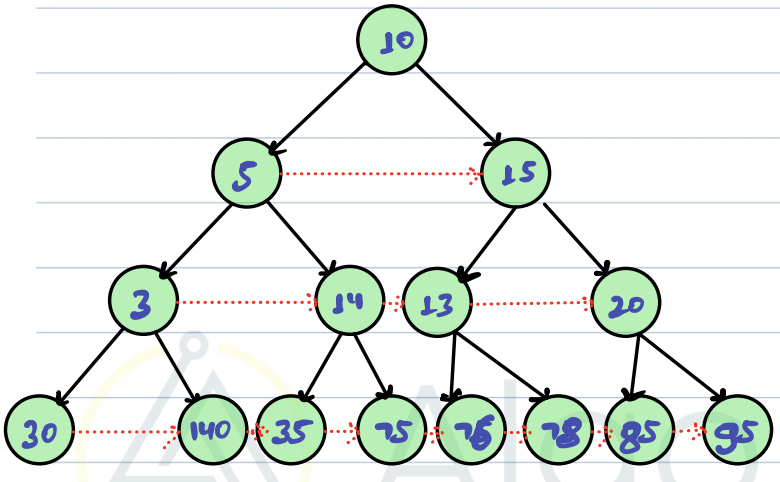
(i) Each node can have 0 or 2 child nodes.

(ii) The child nodes will be added in order.



Q) Populate next Pointer

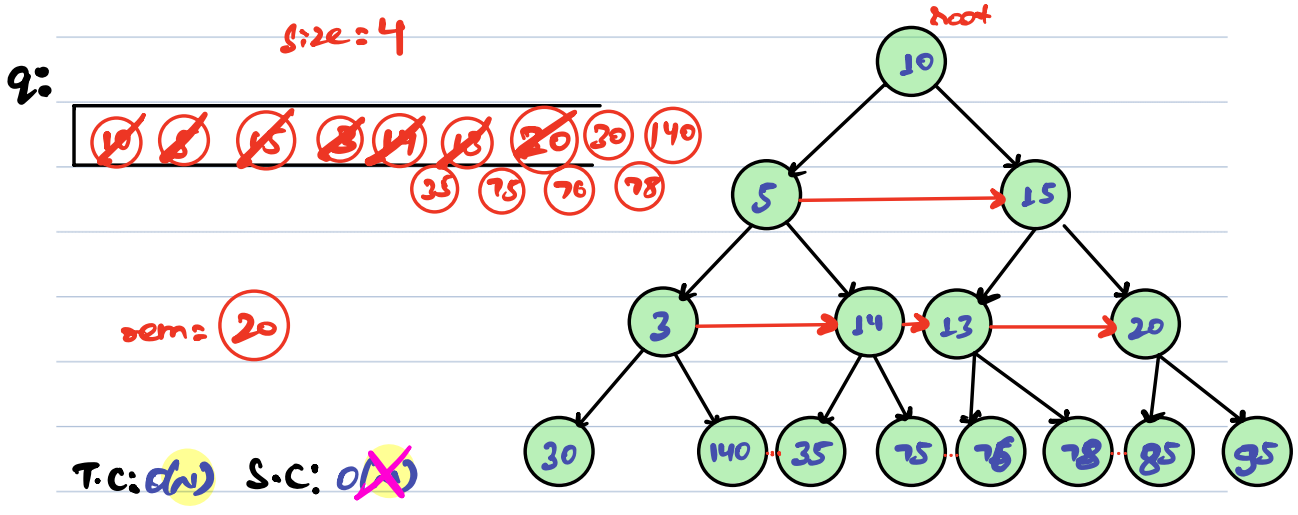
↳ Given a Perfect binary tree, Point next Pointer to the node on right side.




```
class Node {
    int val;
    Node left;
    Node right;
    Node next;
}
```

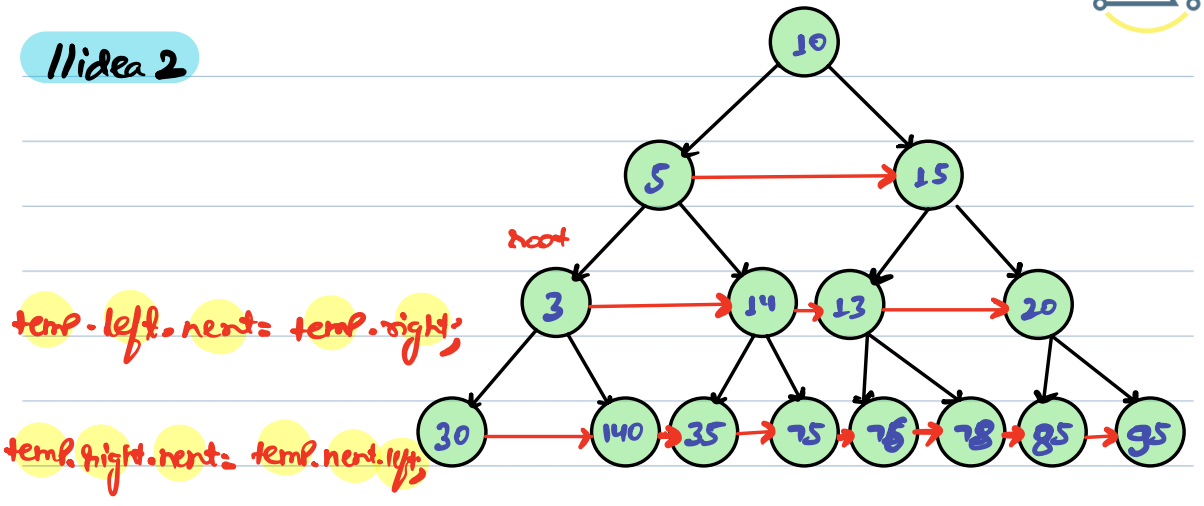
Idea 1

↳ do level Order and Set the next Pointers.




T.C: O(N) S.C: O(N)

//Idea 2




temp = temp.next;

//Pseudo code

```
void NextPointers(Node root){
    while(root.left != null){
        Node temp = root;
        while (temp != null) {
            temp.left.next = temp.right;
            if(temp.next != null){
                temp.right.next = temp.next.left;
            }
            temp = temp.next;
        }
        root = root.left;
    }
}
```

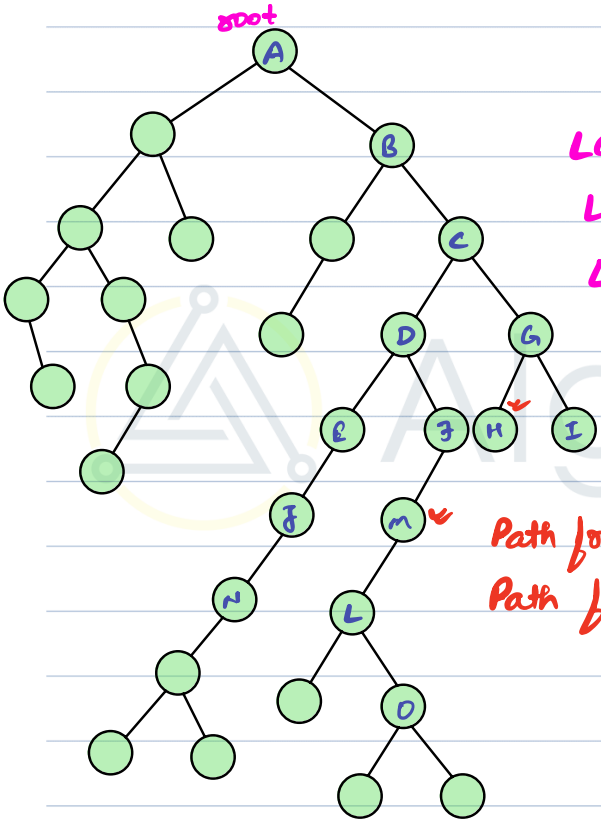
T.C: O(N)

S.C: O(1)

Q) LCA: Lowest Common Ancestor

↳ LCA is defined between 2 nodes P and Q as the lowest node in tree, which is an ancestor to both P and Q.

Note: we allow a node to be ancestor to itself.




LCA(N,O): D

LCA(J,I): C

LCA(L,D): D

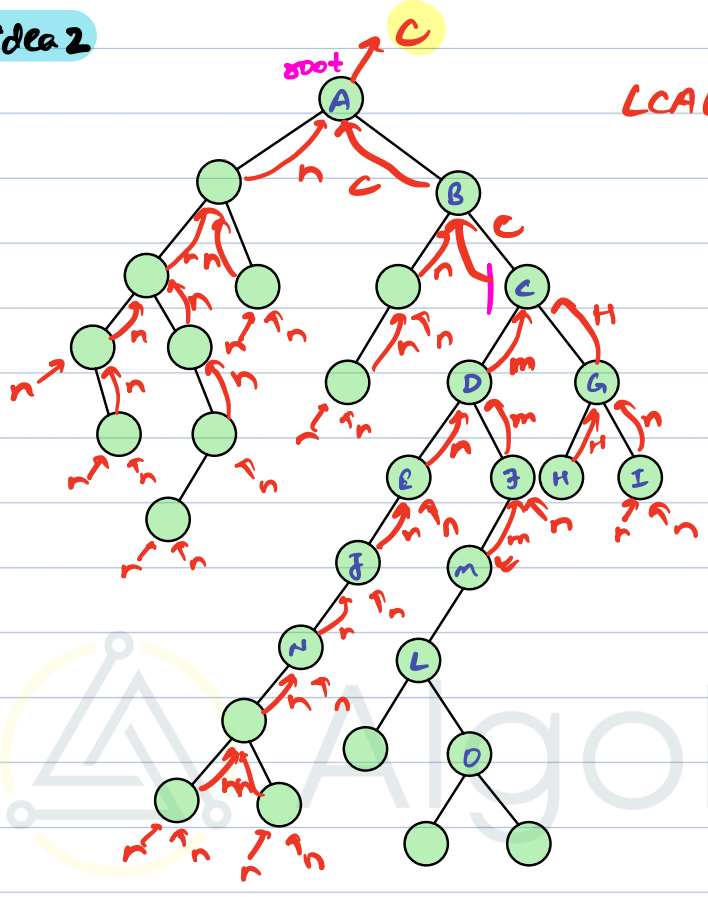
Ideal

↳ LCA(M,H): C

Path from root to M: A B C D J M

Path from root to H: A B C G H

## Idea 2

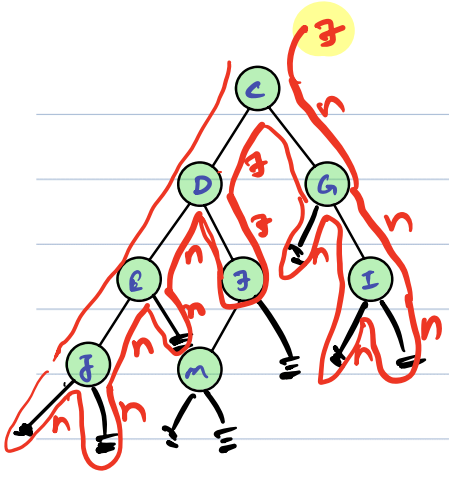


## //Pseudo Code

```
Node LCA (Node root, int n1, int n2){
    if (root == null) { return null; }
    if (root.val == n1 || root.val == n2){
        return root;
    }
    Node l = LCA(root.left, n1, n2);
    Node r = LCA(root.right, n1, n2);
    if (l != null && r != null){
        return root;
    }
    if (l != null && r == null) { return l; }
    if (r != null && l == null) { return r; }
    return null;
}
```

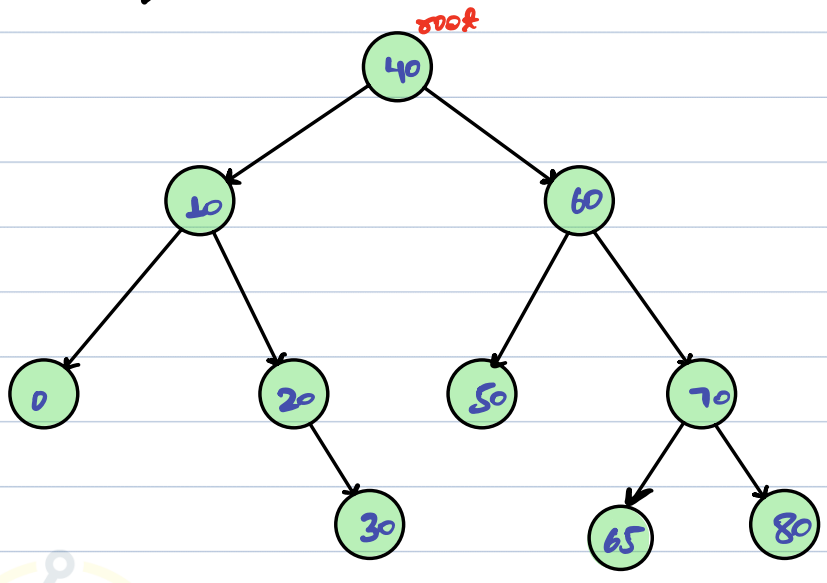
T.C: O(N)

S.C: O(H)




LCA(J, M) : F

Q) LCA of BST



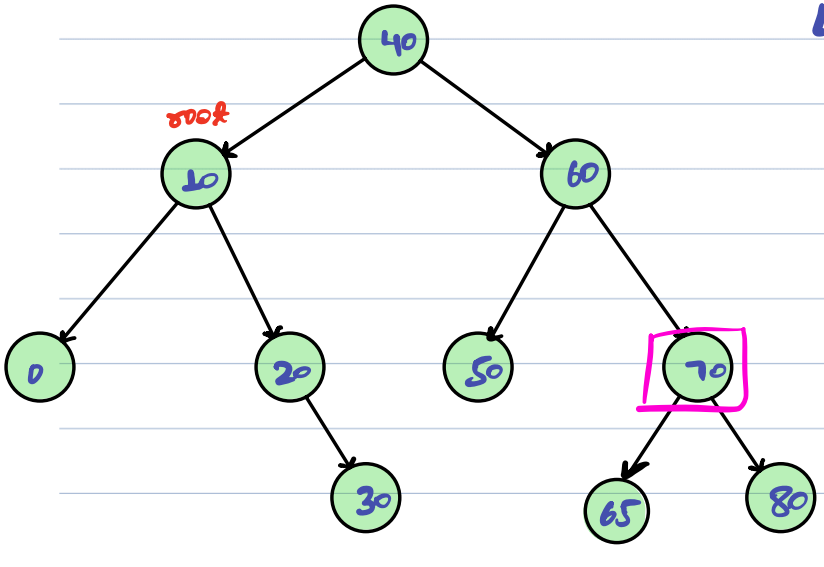



Idea 1

↳ Run the same code of BT.

T.C: O(N) S.C: O(H)

Idea 2




LCA (65, 80)

↓

70

LCA (10, 30)

↓

10

//Pseudo code

```
int LCAinBST (Node root, int n1, int n2){
    while (true){
        if (n1 < root.val && n2 < root.val){
            root = root.left;
        }
        else if (n1 > root.val && n2 > root.val){
            root = root.right;
        }
        else{
            return root.val;
        }
    }
}
```

T.C: O(H)

S.C: O(1)

# Search in a Binary Search Tree

## Java Code:

```
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {
        while(root!=null){
            if(root.val == val){
                return root;
            } else if(root.val > val){
                root = root.left;
            } else {
                root = root.right;
            }
        }
        return null;
    }
}
```

## C++ Code:

```
#include <iostream>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    TreeNode* searchBST(TreeNode* root, int val) {
        while (root != nullptr) {
            if (root->val == val) {
                return root;
            } else if (root->val > val) {
                root = root->left;
            } else {
                root = root->right;
            }
        }
```

```
        return nullptr;
    }
};
```

## Python Code:

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def searchBST(self, root, val):
        while root:
            if root.val == val:
                return root
            elif root.val > val:
                root = root.left
            else:
                root = root.right
        return None
```

# Validate Binary Search Tree

## Java Code:

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isBST(root , Long.MIN_VALUE , Long.MAX_VALUE);
    }
    public boolean isBST(TreeNode root , long min , long max){
        if(root == null) return true;
        long rval = root.val;
        if(rval<min || rval > max) return false;

        boolean l = isBST(root.left ,min , rval-1);
        boolean r = isBST(root.right ,rval+1 , max);

        if(l==false || r == false){
            return false;
```

```
        }
        return true;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <climits>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return isBST(root, LLONG_MIN, LLONG_MAX);
    }

    bool isBST(TreeNode* root, long long min, long long max) {
        if (root == nullptr) return true;

        long long rval = root->val;
        if (rval < min || rval > max) return false;

        bool l = isBST(root->left, min, rval - 1);
        bool r = isBST(root->right, rval + 1, max);

        return l && r;
    }
};
```

## Python Code:

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
```

```
        self.right = None

class Solution:
    def isValidBST(self, root):
        return self.isBST(root, float('-inf'), float('inf'))

    def isBST(self, root, min_val, max_val):
        if not root:
            return True

        rval = root.val
        if rval < min_val or rval > max_val:
            return False

        l = self.isBST(root.left, min_val, rval - 1)
        r = self.isBST(root.right, rval + 1, max_val)

        return l and r
```

# Insert into a Binary Search Tree

## Java Code:

```
class Solution {
    public TreeNode insertIntoBST(TreeNode root, int val) {
        TreeNode n = new TreeNode(val);
        if(root == null) return n;
        TreeNode ans = root;
        while(true){
            if(root.val > val){
              if(root.left == null) {
                 root.left = n;
                 return ans;
              } else{
                 root = root.left;
              }
            } else {
               if(root.right == null){
                  root.right = n;
                  return ans;
               } else {
                  root = root.right;
               }
```

```
            }
        }
    }
}
```

## C++ Code:

```
#include <iostream>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    TreeNode* insertIntoBST(TreeNode* root, int val) {
        TreeNode* n = new TreeNode(val);
        if (root == nullptr) {
            return n;
        }

        TreeNode* ans = root;
        while (true) {
            if (root->val > val) {
                if (root->left == nullptr) {
                    root->left = n;
                    return ans;
                } else {
                    root = root->left;
                }
            } else {
                if (root->right == nullptr) {
                    root->right = n;
                    return ans;
                } else {
                    root = root->right;
                }
            }
        }
    }
};
```

## Python Code:

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def insertIntoBST(self, root, val):
        n = TreeNode(val)
        if root is None:
            return n

        ans = root
        while True:
            if root.val > val:
                if root.left is None:
                    root.left = n
                    return ans
                else:
                    root = root.left
            else:
                if root.right is None:
                    root.right = n
                    return ans
                else:
                    root = root.right
```

# Recover Binary Search Tree

## Java Code:

```
class Solution {
    TreeNode f,s,prev;
    public void recoverTree(TreeNode root) {
        f = s = prev = null;
        inorder(root);
        int t = f.val;
        f.val = s.val;
```

```
        s.val = t;
    }
    public void inorder(TreeNode curr){
        if(curr == null) return;

        inorder(curr.left);
        if(prev != null && curr.val < prev.val &&  f == null){
            f = prev;
            s = curr;
        }
        else if(prev != null && curr.val < prev.val &&  f != null) s = curr;
        prev = curr;
        inorder(curr.right);
    }
}
```

## C++ Code:

```
#include <iostream>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    TreeNode *f, *s, *prev;

    void recoverTree(TreeNode* root) {
        f = s = prev = nullptr;
        inorder(root);
        int t = f->val;
        f->val = s->val;
        s->val = t;
    }

    void inorder(TreeNode* curr) {
        if (curr == nullptr) return;

        inorder(curr->left);
        if (prev != nullptr && curr->val < prev->val && f == nullptr) {
            f = prev;
```

```
            s = curr;
        } else if (prev != nullptr && curr->val < prev->val && f != nullptr) {
            s = curr;
        }
        prev = curr;
        inorder(curr->right);
    }
};
```

## Python Code:

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def __init__(self):
        self.f = None
        self.s = None
        self.prev = None

    def recoverTree(self, root):
        self.f = self.s = self.prev = None
        self.inorder(root)
        t = self.f.val
        self.f.val = self.s.val
        self.s.val = t

    def inorder(self, curr):
        if curr is None:
            return

        self.inorder(curr.left)
        if self.prev is not None and curr.val < self.prev.val and self.f is None:
            self.f = self.prev
            self.s = curr
        elif self.prev is not None and curr.val < self.prev.val and self.f is not None:
            self.s = curr
        self.prev = curr
        self.inorder(curr.right)
```

Today's agenda

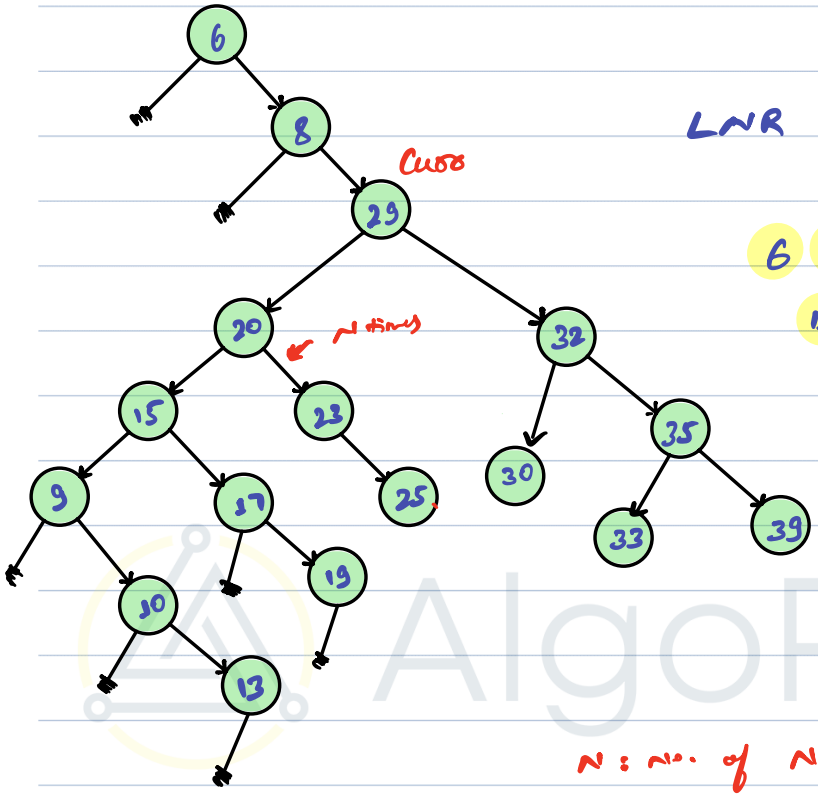
- Morris traversal
- Height of tree
- Diameter of tree

# Morris Inorder traversal

LNR

→ recursion → T.C: O(N)
S.C: O(H)

Morris: T.C: O(N) → iteration
S.C: O(1)




LNR

6 8 9 10 13 15 17 19 20

N : no. of Nodes

E = N-1

//Pseudo code

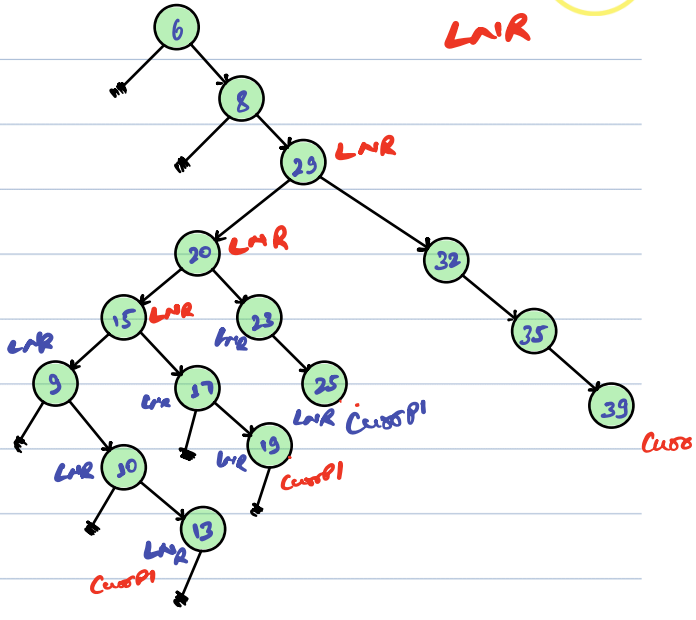
```
void Inorder(Node root){
    Node Curr = root;

    while(Curr != null){
        if(curr.left == null){
            S.O.P(Curr.val);
            Curr = Curr.right;
        }
        else{
            Node CurrPl = Curr.left;

            while(CurrPl.right != null && CurrPl.right != Curr){
                CurrPl = CurrPl.right;
            }
            if(CurrPl.right == null){
                CurrPl.right = Curr;
                Curr = Curr.left;
            }
            else{ //CurrPl.right == curr
                CurrPl.right = null;
                S.O.P(curr.val);
                Curr = Curr.right;
            }
        }
    }
}
```

T.C: 3*N to O(N)

S.C: O(1)

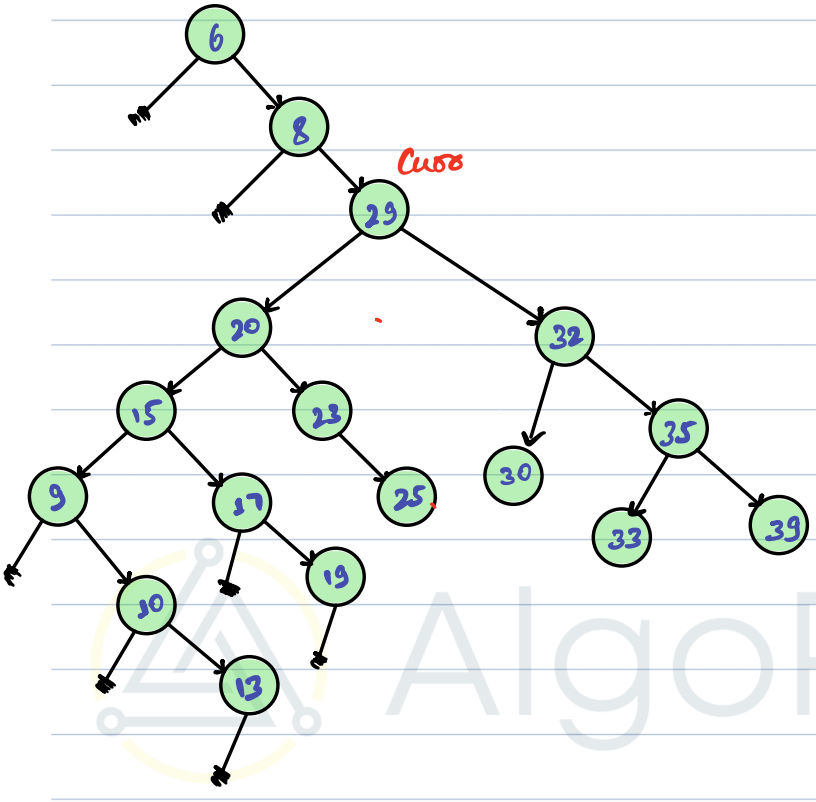
6 8 9 10 13 15 17 19 20 23
25 29 32 35 39

```
void Inorder(Node root){
Node Curr = root;

while (Curr != null) {
    if (Curr.left == null){
        S.o.p(Curr.val);
        Curr = Curr.right;
    }
    else{
        Node CurrPl = Curr.left;
        while(CurrPl.right!=null && CurrPl.right!=Curr){
            CurrPl = CurrPl.right;
        }
        if (CurrPl.right == null){
            CurrPl.right = Curr;
            Curr = Curr.left;
        }
        else{ //if(CurrPl.right == Curr
            CurrPl.right = null;
            S.o.p(Curr.val);
            Curr = Curr.right;
        }
    }
}
```



Morris Preorder traversal

N-LR




Case

//Pseudo code

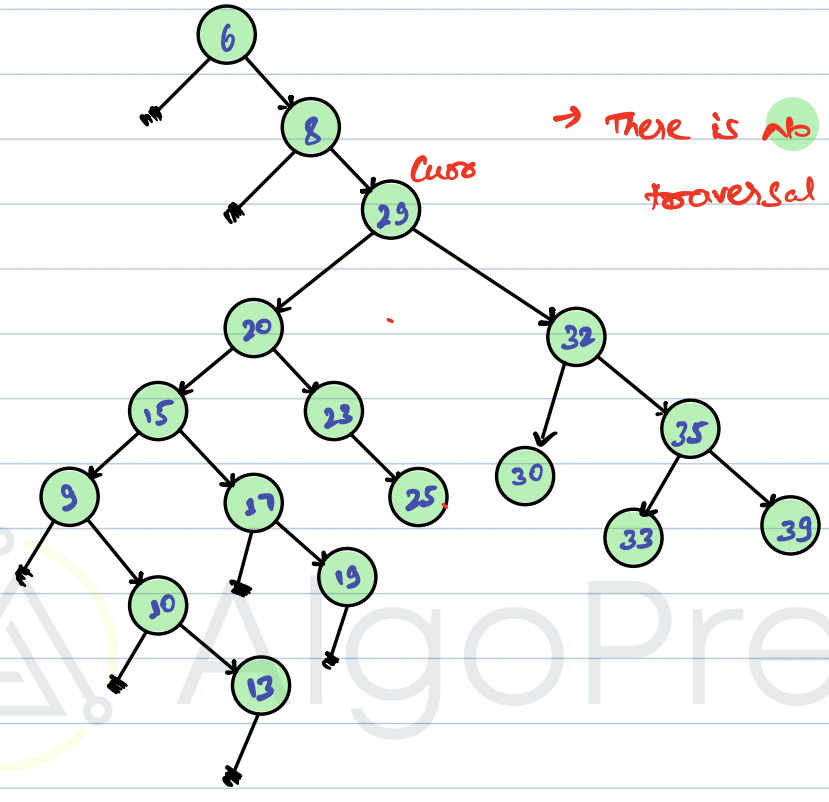
```
void Inorder(Node root){
    Node Curr = root;

    while (Curr != null){
        if (Curr.left == null){
            s.o.p(Curr.val);
            Curr = Curr.right;
        }
        else{
            Node CurrPl = Curr.left;

            while(CurrPl.right != null && CurrPl.right != Curr){
                CurrPl = CurrPl.right;
            }
            if (CurrPl.right == null){
                CurrPl.right = Curr;
                s.o.p(Curr.val);
                Curr = Curr.left;
            }
            else{ //CurrPl.right == Curr
                CurrPl.right = null;
                s.o.p(Curr.val);
                Curr = Curr.right;
            }
        }
    }
}
```

T.C: 3*O(N)

S.C: O(1)

LRN

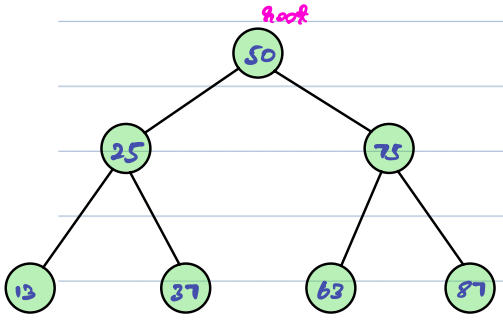
Morris Postorder traversal




Case 3

→ There is no morris traversal for Postorder.

NRL

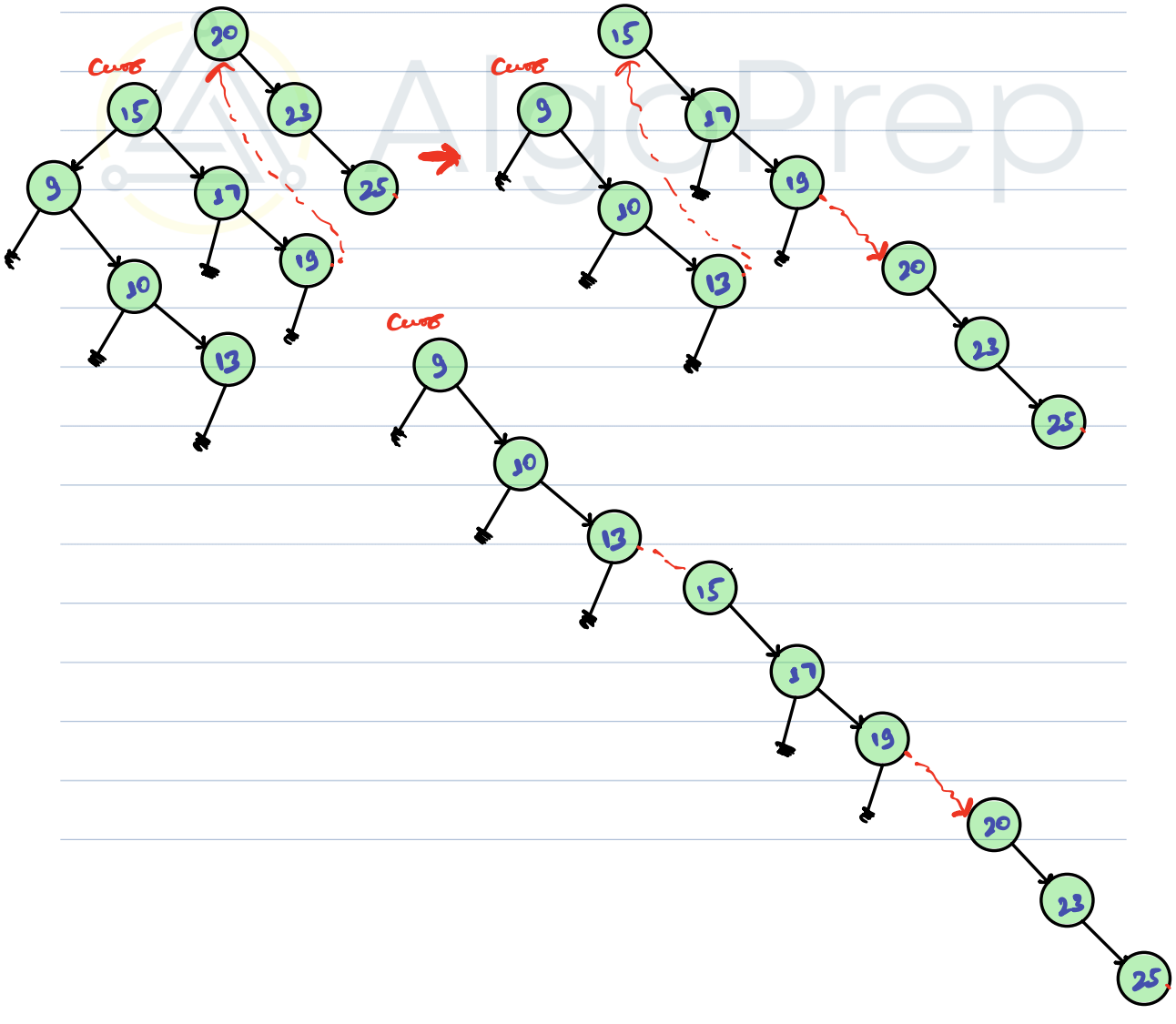



→ Postorder: 13 37 25 63 87 75 50

→ Reverse Preorder: {50 75 87 63 25 37 13}

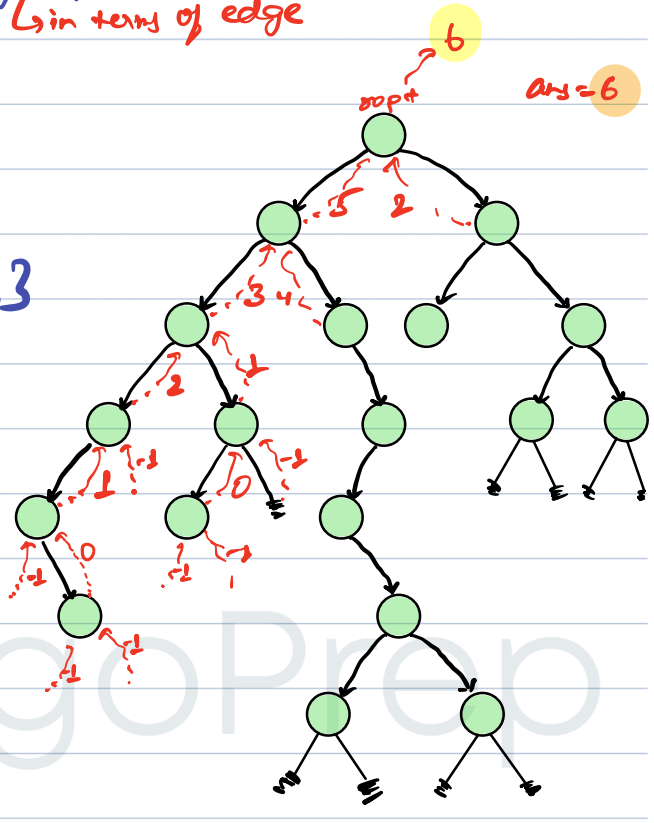
13 37 25 63 87 75 50

## Insight on morris traversal



Q) Given root calculate height of the tree.

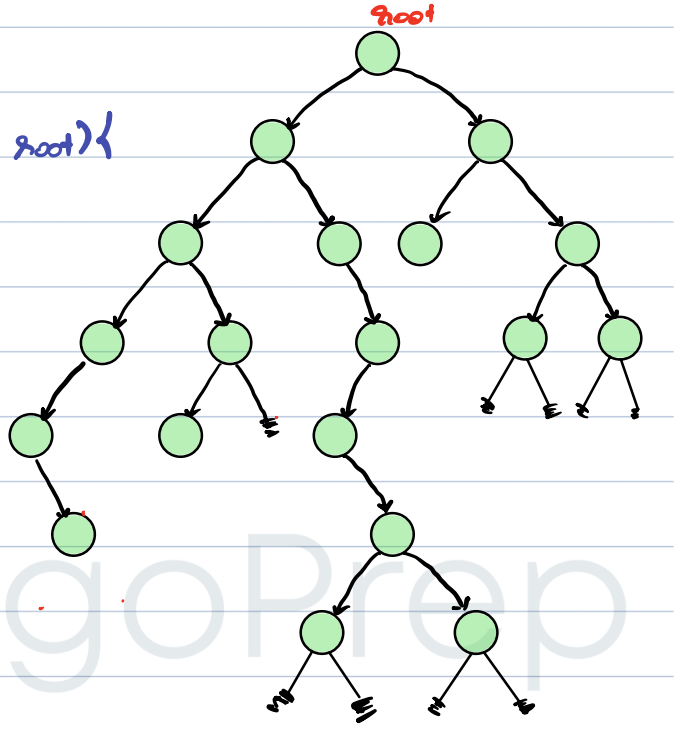
↳ in terms of edge




```
int height (Node root){
    if (root == null){return -1;}

    int lht = height (root.left);
    int rht = height (root.right);

    return max(lht,rht) +1;
}
```

T.C: O(N)  S.C: O(H)

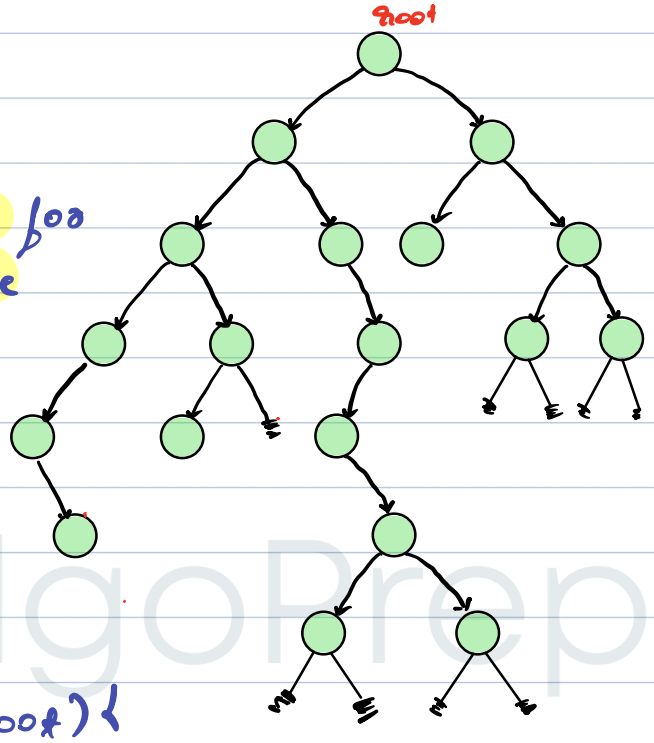
Q) Given root Calculate longest Path across root node of the tree.
(in terms of edge)




```
int manPathAcrossRoot (Node root){

    int lht = height(root.left);
    int rht = height (root.right);

    return lht + rht + 2;

}
```

T.C: O(N)  S.C: O(H)

Q) Given root Calculate longest Path across A node of the tree.
↳ in terms of edge ↳ Diameter of the tree




//idea1

↳ Call manPathAcrossRoot for every node and Pick the maximum.

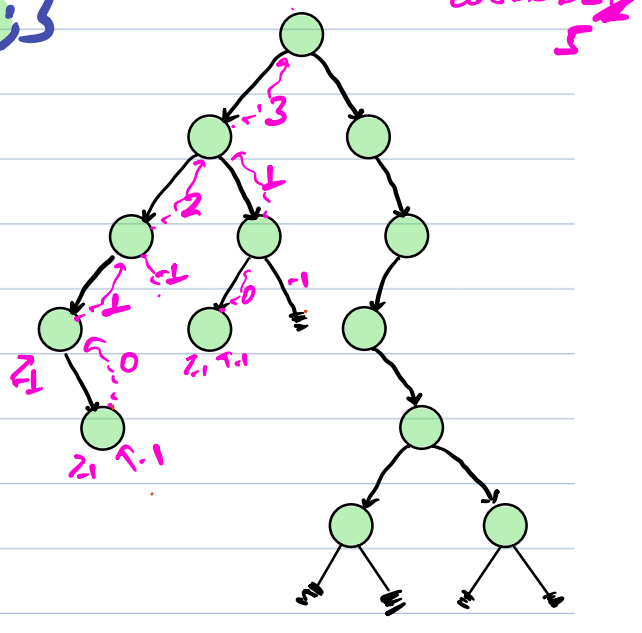
T.C: $O(N^2)$

//idea2

```
int ans = 0;
int height (Node root){
    if (root == null){return -1;}

    int lht = height (root.left);
    int rht = height (root.right);

    ans = max(ans, lht + rht + 2)
    return max(lht, rht) + 1;
}
```

T.C: O(N)
S.C: O(H)

```
int main() {
    height (root);
    return ans;
}
```



# Binary Search Tree to Greater Sum Tree

## Java Code:

```
class Solution {
    int greater=0;
    public TreeNode bstToGst(TreeNode root) {
        helper(root);
        return root;
    }
    public void helper(TreeNode root){
        if(root == null) return;
        helper(root.right);
        greater = greater + root.val;
        root.val = greater;
        helper(root.left);

    }
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
private:
    int greater;

public:
    TreeNode* bstToGst(TreeNode* root) {
        greater = 0;
        helper(root);
        return root;
    }

    void helper(TreeNode* root) {
```

```
        if (root == nullptr) return;
        helper(root->right);
        greater += root->val;
        root->val = greater;
        helper(root->left);
    }
};

int main() {
    // Example usage
    Solution solution;
    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(1);
    root->right = new TreeNode(6);
    root->left->right = new TreeNode(2);
    root->right->left = new TreeNode(5);

    TreeNode* modifiedRoot = solution.bstToGst(root);

    // Access the modified tree as needed
    // ...

    return 0;
}
```

## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def bstToGst(self, root):
        self.greater = 0
        self.helper(root)
        return root

    def helper(self, root):
        if root is None:
            return
        self.helper(root.right)
        self.greater += root.val
        root.val = self.greater
```

```
        self.helper(root.left)

# Example usage
solution = Solution()
root = TreeNode(4)
root.left = TreeNode(1)
root.right = TreeNode(6)
root.left.right = TreeNode(2)
root.right.left = TreeNode(5)

modified_root = solution.bstToGst(root)

# Access the modified tree as needed
# ...
```

# Inorder Successor in BST

## Java Code:

```
class Solution
{
    // returns the inorder successor of the Node x in BST (rooted at 'root')
          public Node inorderSuccessor(Node root,Node x)
        {
          Node temp = null;
          while(true){
              if(root.data > x.data){
                  temp = root;
                  root = root.left;
              } else if(root.data < x.data){
                  root = root.right;
              } else {
                  break;
              }
          }
          if(root.right == null) return temp;

          Node rootp1 = root.right;
          while(rootp1.left != null){
              rootp1 = rootp1.left;
          }
          return rootp1;
        }
```

```
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : data(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    TreeNode* inOrderSuccessor(TreeNode* root, TreeNode* x) {
        TreeNode* temp = nullptr;
        while (true) {
            if (root->data > x->data) {
                temp = root;
                root = root->left;
            } else if (root->data < x->data) {
                root = root->right;
            } else {
                break;
            }
        }
        if (root->right == nullptr) {
            return temp;
        }

        TreeNode* rootp1 = root->right;
        while (rootp1->left != nullptr) {
            rootp1 = rootp1->left;
        }
        return rootp1;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
```

```
    def __init__(self, x):
        self.data = x
        self.left = None
        self.right = None

class Solution:
    def inOrderSuccessor(self, root, x):
        temp = None
        while True:
            if root.data > x.data:
                temp = root
                root = root.left
            elif root.data < x.data:
                root = root.right
            else:
                break
        if root.right is None:
            return temp

        rootp1 = root.right
        while rootp1.left is not None:
            rootp1 = rootp1.left
        return rootp1
```

# Populating Next Right Pointers in Each Node

## Java Code:

```
class Solution {
    public Node connect(Node root) {
        if(root == null) return null;
        Node ans = root;

        while(root.left != null){
            Node temp = root;
            while(temp!= null){
                temp.left.next = temp.right;
                if(temp.next != null){
                    temp.right.next = temp.next.left;
                }
                temp = temp.next;

            }
```

```
            root = root.left;
        }
        return ans;
    }
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct Node {
    int val;
    Node* left;
    Node* right;
    Node* next;
    Node(int x) : val(x), left(nullptr), right(nullptr), next(nullptr) {}
};

class Solution {
public:
    Node* connect(Node* root) {
        if (root == nullptr) return nullptr;
        Node* ans = root;

        while (root->left != nullptr) {
            Node* temp = root;
            while (temp != nullptr) {
                temp->left->next = temp->right;
                if (temp->next != nullptr) {
                    temp->right->next = temp->next->left;
                }
                temp = temp->next;
            }
            root = root->left;
        }
        return ans;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class Node:
```

```
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
        self.next = None

class Solution(object):
    def connect(self, root):
        if root is None:
            return None
        ans = root

        while root.left is not None:
            temp = root
            while temp is not None:
                temp.left.next = temp.right
                if temp.next is not None:
                    temp.right.next = temp.next.left
                temp = temp.next
            root = root.left
        return ans
```

# Lowest Common Ancestor in a Binary Tree

## Java Code:

```
class Solution
{
    //Function to return the lowest common ancestor in a Binary Tree.
        Node lca(Node root, int n1,int n2)
        {
                if(root == null) return null;
                if(root.data == n1 || root.data == n2){
                    return root;
                }
                Node l = lca(root.left , n1 , n2);
                Node r = lca(root.right , n1 , n2);
                if(l!= null && r != null){
                    return root;
                }
```

```
                if(l != null && r == null) return l;
                if(r != null && l == null) return r;
                return null;
        }
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int x) : data(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    // Function to return the lowest common ancestor in a Binary Tree.
    Node* lca(Node* root, int n1, int n2) {
        if (root == nullptr) return nullptr;
        if (root->data == n1 || root->data == n2) {
            return root;
        }
        Node* l = lca(root->left, n1, n2);
        Node* r = lca(root->right, n1, n2);
        if (l != nullptr && r != nullptr) {
            return root;
        }
        if (l != nullptr && r == nullptr) return l;
        if (r != nullptr && l == nullptr) return r;
        return nullptr;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class Node:
    def __init__(self, data):
        self.data = data
```

```
        self.left = None
        self.right = None

class Solution:
    # Function to return the lowest common ancestor in a Binary Tree.
    def lca(self, root, n1, n2):
        if root is None:
            return None
        if root.data == n1 or root.data == n2:
            return root
        l = self.lca(root.left, n1, n2)
        r = self.lca(root.right, n1, n2)
        if l is not None and r is not None:
            return root
        if l is not None and r is None:
            return l
        if r is not None and l is None:
            return r
        return None
```

# Lowest Common Ancestor of a Binary Search Tree

## Java Code:

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

        while(true){
            if(p.val < root.val && q.val < root.val) {
                root  = root.left;
            } else if(p.val > root.val && q.val > root.val){
                root = root.right;
            } else {
                return root;
            }
        }

    }
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        while (true) {
            if (p->val < root->val && q->val < root->val) {
                root = root->left;
            } else if (p->val > root->val && q->val > root->val) {
                root = root->right;
            } else {
                return root;
            }
        }
    }
};
```

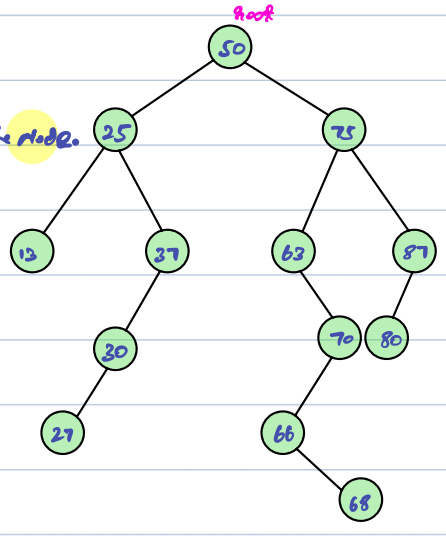
## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def lowestCommonAncestor(self, root, p, q):
        while True:
            if p.val < root.val and q.val < root.val:
                root = root.left
            elif p.val > root.val and q.val > root.val:
                root = root.right
            else:
                return root
```

Today's agenda

- Remove k from BST
- Serialization
- Deserialization
- Binary tree Cameras → HW

Q) Given K, Remove the Node from BST.

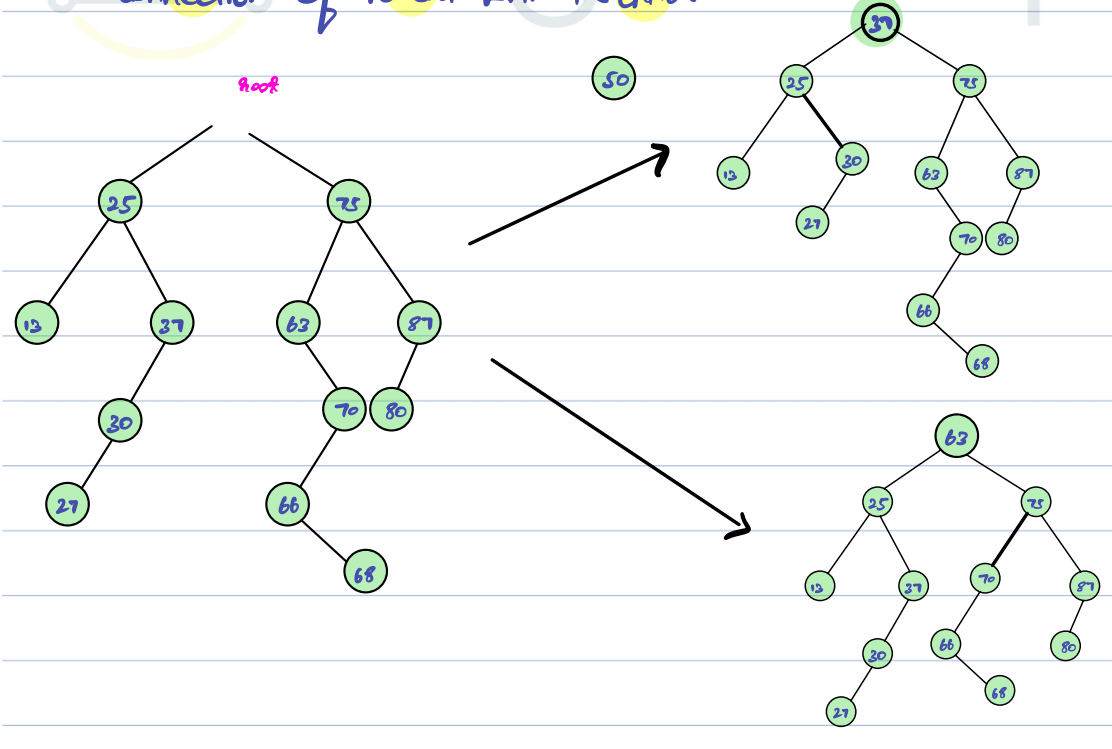



→ (i) removal of leaf node: remove the node.

→ (ii) Node with left child:
↳ Remove and make the connection of Parent with the child.

→ (iii) Node with right child:
↳ Remove and make the connection of Parent with the child.

(iv) Node with both the child.

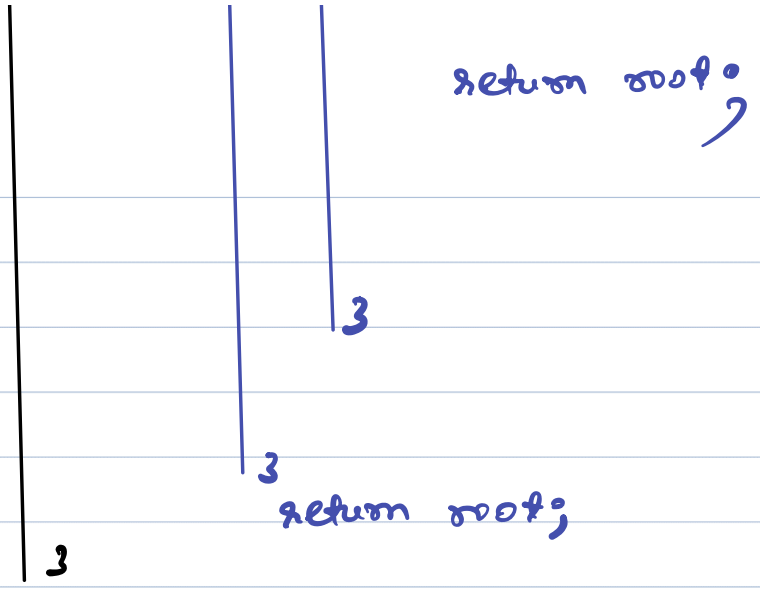


//Pseudo code

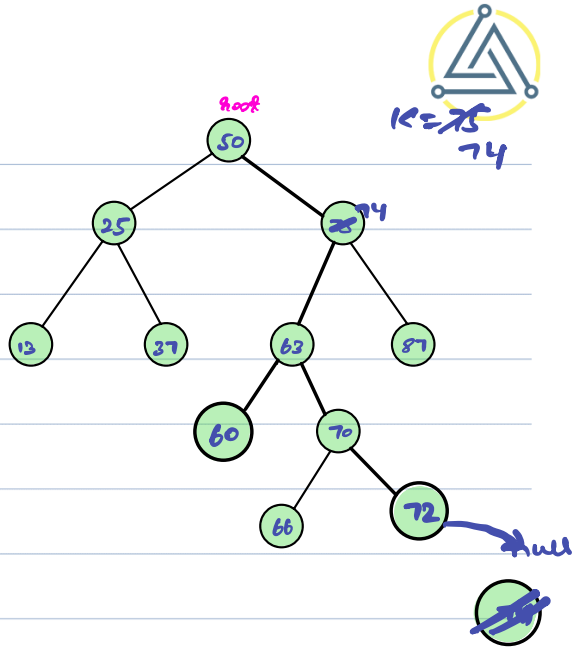
```
Node DeleteinBST (Node root, int k){
    if (root==null) { return null; }

    if ( root.val > k ) {
    root.left = DeleteinBST (root.left, k);
    }
    else if (root.val <k){
    root.right= DeleteinBST (root.right, k);
    }
    else{
        if (root.left== null && root.right==null){
            return null;
        }
        else if (root.left !=null && root.right==null){
            return root.left;
        }
        else if (root.left ==null && root.right!=null){
            return root.right;
        }
        else{
            int max = func (root.left);
            root.val = max;

        root.left= DeleteinBST (root.left ,max);
```

T.C: O(H)

S.C: O(H)

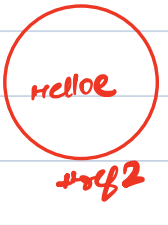


```
Node DeleteinBST (Node root, int k){
    if (root == null) { return null; }

    if ( root.val > k ) {
        root.left = DeleteinBST (root.left, k);
    }
    else if (root.val < k){
        root.right = DeleteinBST (root.right, k);
    }
    else{
        if (root.left == null && root.right == null){
            return null;
        }
        else if (root.left != null && root.right == null){
            return root.left;
        }
        else if (root.left == null && root.right != null){
            return root.right;
        }
        else{
            int max = func (root.left);
            root.val = max;
            root.left = DeleteinBST (root.left, max);

            return root;
        }
    }
    return root;
}
```




## StringBuilder

String s1 = "Hello" → 

String s2 = "Hello" → 

s1 = s1 + "e" → 

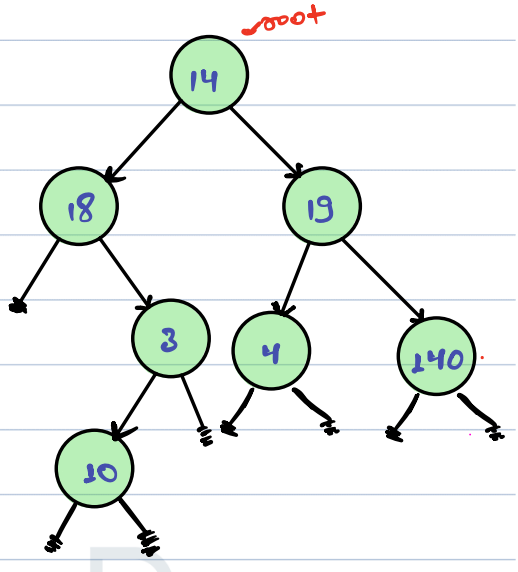
→ StringBuilder sb = new StringBuilder(s1);

↳ sb = sb.append("e");

→ sb.toString();

Q) Serialize and Deserialize a Binary tree

↳ Can you convert a tree in linear form.

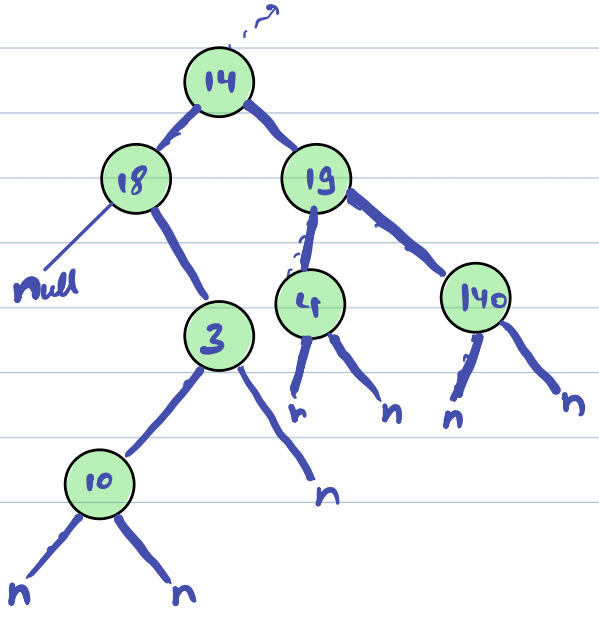



→ Sb = " "

Serialize: Do Preorder, add val with space and instead of null add # with space.

S: 14 _ 18 _ # _ 3 _ 10 _ # _ # _ # _ 19 _ 4 _ # _ # _ 140 _ # _ # _

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ch[]: | 14 | 18 | # | 3 | 10 | # | # | # | 19 | 4 | # | # | 140 | # | # |



// Pseudo Code

```
String serialize (Node root){
    StringBuilder sb = new StringBuilder();
    Helper1(root, sb);
    return sb.toString();
}

void Helper1 (Node root, StringBuilder sb){
    if (root == null){
        sb.append("# ");
        return;
    }
    sb.append(root.val + " ");

    Helper1(root.left, sb);
    Helper1(root.right, sb);
}

int i;
Node Deserialize (String data){
    String[] ch = data.split(" ");
    i=0;
    Node root = Helper2(ch);
    return root;
}
```

T.C: O(N)

S.C: O(H)

```
Node Helper2 (String[] ch){
    if(ch[i].equals("#")){
        i++;
        return null;
    }

    Node root = new Node (Integer.parseInt(ch[i]));
    i++;

    root.left = Helper2 ( ch );
    root.right= Helper2 ( ch );

    return root;
}
```

# Binary Tree Preorder Traversal

## Java Code:

```
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> ans = new ArrayList<>();
        TreeNode curr = root;
        while(curr != null){
            if(curr.left == null){
                ans.add(curr.val);
                curr = curr.right;
            } else {
                TreeNode currp1 = curr.left;
                while(currp1.right != null && currp1.right != curr){
                    currp1 = currp1.right;
                }
                if(currp1.right == null){
                    currp1.right = curr;
                    ans.add(curr.val);
                    curr = curr.left;
                } else {
                    currp1.right = null;
                    curr = curr.right;
                }
            }
        }
        return ans;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <stack>

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
class Solution {
public:
    std::vector<int> preorderTraversal(TreeNode* root) {
        std::vector<int> ans;
        TreeNode* curr = root;
        while (curr != nullptr) {
            if (curr->left == nullptr) {
                ans.push_back(curr->val);
                curr = curr->right;
            } else {
                TreeNode* currp1 = curr->left;
                while (currp1->right != nullptr && currp1->right != curr) {
                    currp1 = currp1->right;
                }
                if (currp1->right == nullptr) {
                    currp1->right = curr;
                    ans.push_back(curr->val);
                    curr = curr->left;
                } else {
                    currp1->right = nullptr;
                    curr = curr->right;
                }
            }
        }
        return ans;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def preorderTraversal(self, root):
        ans = []
        curr = root
        while curr:
            if not curr.left:
                ans.append(curr.val)
                curr = curr.right
```

```
            else:
                currp1 = curr.left
                while currp1.right and currp1.right != curr:
                    currp1 = currp1.right
                if not currp1.right:
                    currp1.right = curr
                    ans.append(curr.val)
                    curr = curr.left
                else:
                    currp1.right = None
                    curr = curr.right
        return ans
```

# Binary Tree Inorder Traversal

## Java Code:

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
         List<Integer> ans = new ArrayList<>();
        TreeNode curr = root;
        while(curr != null){
            if(curr.left == null){
                ans.add(curr.val);
                curr = curr.right;
            } else {
                TreeNode currp1 = curr.left;
                while(currp1.right != null && currp1.right != curr){
                    currp1 = currp1.right;
                }
                if(currp1.right == null){
                    currp1.right = curr;
                    curr = curr.left;
                } else {
                    currp1.right = null;
                    ans.add(curr.val);
                    curr = curr.right;
                }
            }
        }
        return ans;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <stack>

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    std::vector<int> inorderTraversal(TreeNode* root) {
        std::vector<int> ans;
        TreeNode* curr = root;
        while (curr != nullptr) {
            if (curr->left == nullptr) {
                ans.push_back(curr->val);
                curr = curr->right;
            } else {
                TreeNode* currp1 = curr->left;
                while (currp1->right != nullptr && currp1->right != curr) {
                    currp1 = currp1->right;
                }
                if (currp1->right == nullptr) {
                    currp1->right = curr;
                    curr = curr->left;
                } else {
                    currp1->right = nullptr;
                    ans.push_back(curr->val);
                    curr = curr->right;
                }
            }
        }
        return ans;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def inorderTraversal(self, root):
        ans = []
        curr = root
        while curr:
            if not curr.left:
                ans.append(curr.val)
                curr = curr.right
            else:
                currp1 = curr.left
                while currp1.right and currp1.right != curr:
                    currp1 = currp1.right
                if not currp1.right:
                    currp1.right = curr
                    curr = curr.left
                else:
                    currp1.right = None
                    ans.append(curr.val)
                    curr = curr.right
        return ans
```

# Binary Tree Postorder Traversal

## Java Code:

```
class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
         List<Integer> ans = new ArrayList<>();
        TreeNode curr = root;
        while(curr != null){
            if(curr.right == null){
                ans.add(curr.val);
                curr = curr.left;
            } else {
```

```
            TreeNode currp1 = curr.right;
            while(currp1.left != null && currp1.left != curr){
                currp1 = currp1.left;
            }
            if(currp1.left == null){
                currp1.left = curr;
                ans.add(curr.val);
                curr = curr.right;
            } else {
                currp1.left = null;
                curr = curr.left;
            }
        }
    }
    Collections.reverse(ans);
    return ans;
  }
}
```

## C++ Code:

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>

// Definition for a binary tree node.
struct TreeNode {
  int val;
  TreeNode* left;
  TreeNode* right;
  TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
  std::vector<int> postorderTraversal(TreeNode* root) {
    std::vector<int> ans;
    TreeNode* curr = root;
    while (curr != nullptr) {
      if (curr->right == nullptr) {
        ans.push_back(curr->val);
        curr = curr->left;
      } else {
        TreeNode* currp1 = curr->right;
        while (currp1->left != nullptr && currp1->left != curr) {
          currp1 = currp1->left;
```

```
            }
            if (currp1->left == nullptr) {
                currp1->left = curr;
                ans.push_back(curr->val);
                curr = curr->right;
            } else {
                currp1->left = nullptr;
                curr = curr->left;
            }
        }
    }
    std::reverse(ans.begin(), ans.end());
    return ans;
  }
};
```

## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Solution:
    def postorderTraversal(self, root):
        ans = []
        curr = root
        while curr:
            if not curr.right:
                ans.append(curr.val)
                curr = curr.left
            else:
                currp1 = curr.right
                while currp1.left and currp1.left != curr:
                    currp1 = currp1.left
                if not currp1.left:
                    currp1.left = curr
                    ans.append(curr.val)
                    curr = curr.right
                else:
                    currp1.left = None
                    curr = curr.left
        ans.reverse()
        return ans
```

# Height of Binary Tree

## Java Code:

```
class Solution {
    //Function to find the height of a binary tree.
    int height(Node node)
    {
        if(node == null) return 0;
        int lht = height(node.left);
        int rht = height(node.right);

        return Math.max(lht,rht)+1;
    }
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int x) : data(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    // Function to find the height of a binary tree.
    int height(Node* node) {
        if (node == nullptr) return 0;
        int lht = height(node->left);
        int rht = height(node->right);

        return std::max(lht, rht) + 1;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class Solution:
    # Function to find the height of a binary tree.
    def height(self, node):
        if node is None:
            return 0
        lht = self.height(node.left)
        rht = self.height(node.right)

        return max(lht, rht) + 1
```

# Diameter of a Binary Tree

## Java Code:

```
class Solution {
    // Function to return the diameter of a Binary Tree.
    int ans = 0;
    int diameter(Node root) {

        height(root);
        return ans+1;
    }
     int height(Node node)
    {
        if(node == null) return -1;
        int lht = height(node.left);
        int rht = height(node.right);
        ans = Math.max(ans , lht+rht+2);
        return Math.max(lht,rht)+1;
    }
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int x) : data(x), left(nullptr), right(nullptr) {}
};

class Solution {
private:
    int ans = 0;

public:
    // Function to return the diameter of a Binary Tree.
    int diameter(Node* root) {
        height(root);
        return ans + 1;
    }

    int height(Node* node) {
        if (node == nullptr) return -1;
        int lht = height(node->left);
        int rht = height(node->right);
        ans = std::max(ans, lht + rht + 2);
        return std::max(lht, rht) + 1;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class Solution:
    def __init__(self):
        self.ans = 0

    # Function to return the diameter of a Binary Tree.
```

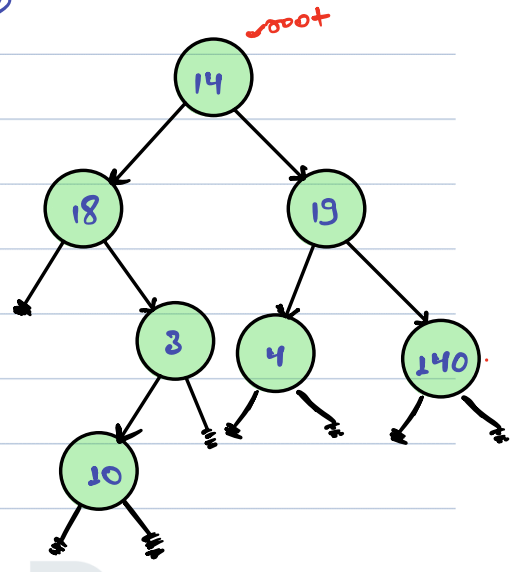
```
def diameter(self, root):
    self.height(root)
    return self.ans + 1

def height(self, node):
    if node is None:
        return -1
    lht = self.height(node.left)
    rht = self.height(node.right)
    self.ans = max(self.ans, lht + rht + 2)
    return max(lht, rht) + 1
```

Today's agenda

- left view
- Right view
- Top view
- bottom view
- vertical order
- Construct BT from inorder & preorder

Q) left view

↳ Print Nodes visible from left side.




//Idea:

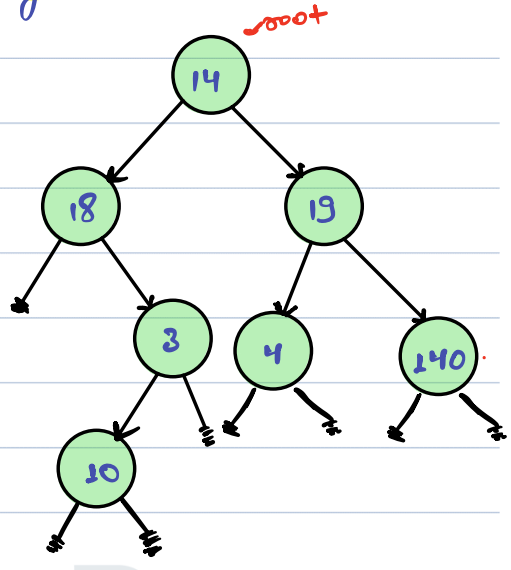
↳ Do level order and Print the first node of each level.

```
void levelorder(Node root){
    Queue<Node> q;
    q.add(root);
    while(q.size()>0){
        int n: q.size();

        for(int i=1; i<=n; i++){
            Node temp = q.remove();
            if(i==1){
                System.out.print(temp.val);
            }
            if(temp.left!=null){
                q.add(temp.left);
            }
            if(temp.right!=null){
                q.add(temp.right);
            }
        }
        System.out.println();
    }
}
```

14 18 3 10

Q) Right view

↳ Print Nodes visible from right side.




Ideas

↳ Print left Node of each level.

```
void levelorder (Node root){
    Queue < Node > q;
    q.add (root);
    while ( q.size () > 0 ){
        int n = q.size ();

        for(int i=1; i<=n; i++) {
            Node temp = q.remove();
            if (i==n){
                System.out.print (temp.val);
            }
            if ( temp.left != null){
                q.add (temp.left);
            }
            if ( temp.right != null) {
                q.add (temp.right);
            }
        }
        System.out.println();
    }
}
```

14 19 140 10

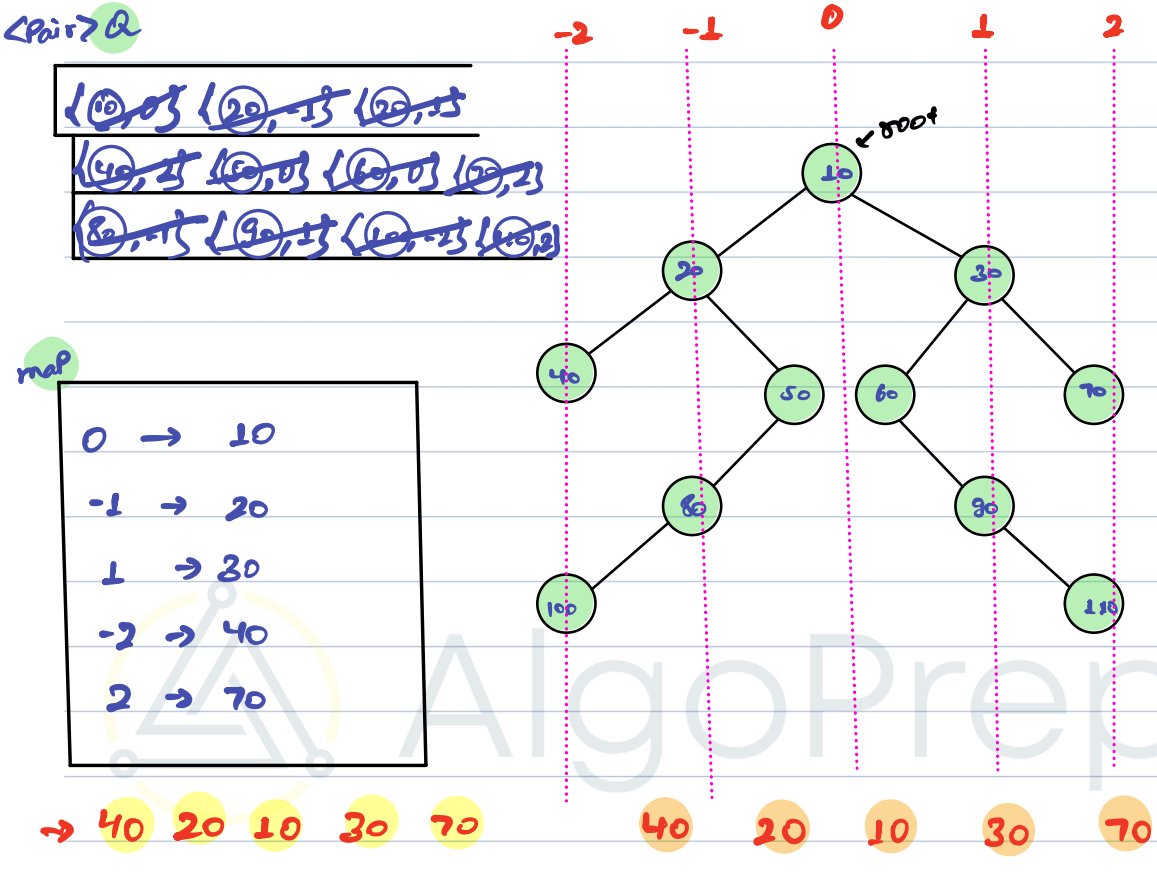
T.C: O(N)

S.C: O(N)

```
class Pair {
    Node n;
    int vl;
}
```

Q) Top view

↳ Print Nodes visible from Top side.

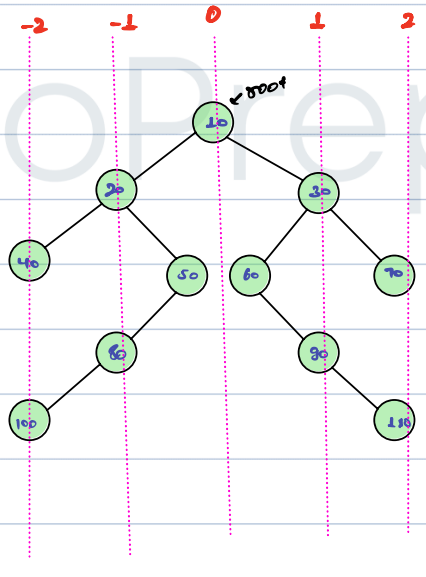


```
//Pseudo code

Public class Pair{
    Node n;
    int vtlevel;
    Pair (Node n1, int n){
        n=n1;
        vtlevel=n;
    }
}

void topview (Node root){
    Queue < Pair > q;
    HashMap < Integer, Integer > map;

    q.add ( new Pair (root, 0));

    while (q.size()>0) {
        Pair rem= q.remove();
        Node remn = rem.n;
        int remVL = rem.vtlevel;

        if (map.containsKey (remVL) == false){
            map.put(remVL, remn.val);
        }
        if (remn.left != null){
            q.add (new Pair (remn.left, remVL-1));
        }
```

```
        }
    }
    if (remn.right != null) {
        q.add(new Pair(remn.right, remVL+1));
    }
}
```

Iterate on map and Point

```
}
q.add(new Pair(root, 0));
while (q.size()>0) {
    Pair rem = q.remove();
    Node remn = rem.n;
    int remVL = rem.vslevel;
    if (map.containsKey(remVL) == false) {
        map.put(remVL, remn.val);
    }
    if (remn.left != null) {
        q.add(new Pair(remn.left, remVL-1));
    }
    if (remn.right != null) {
        q.add(new Pair(remn.right, remVL+1));
    }
}
```




{10,0} {20,-1} {30,1}

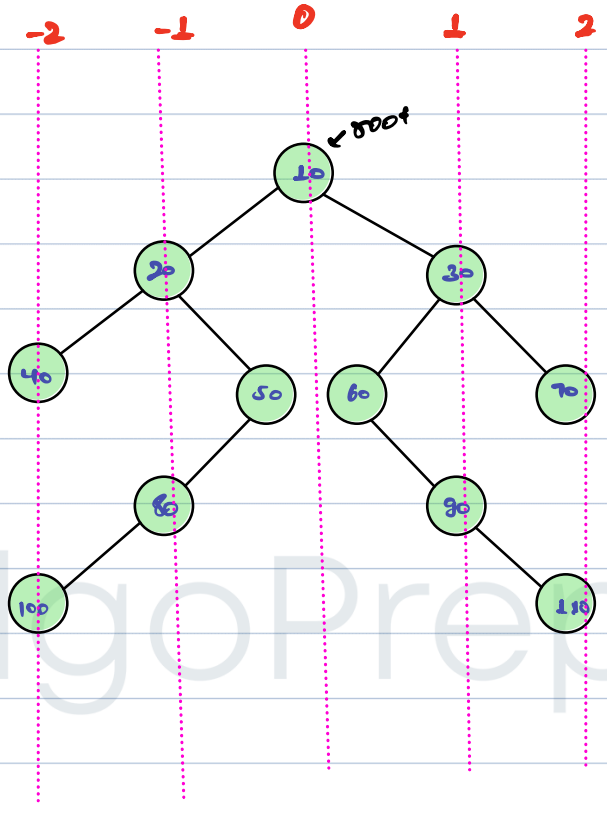
rem: {10,0}

remn: 10 remVL: 0

0 → 10

Q) Bottom view

↳ Print Nodes visible from bottom side.




//Ideal

↳ Print the last node of each vertical level.

↓

Keep updating the HashMap value.

//Pseudo code

T.C: O(N)

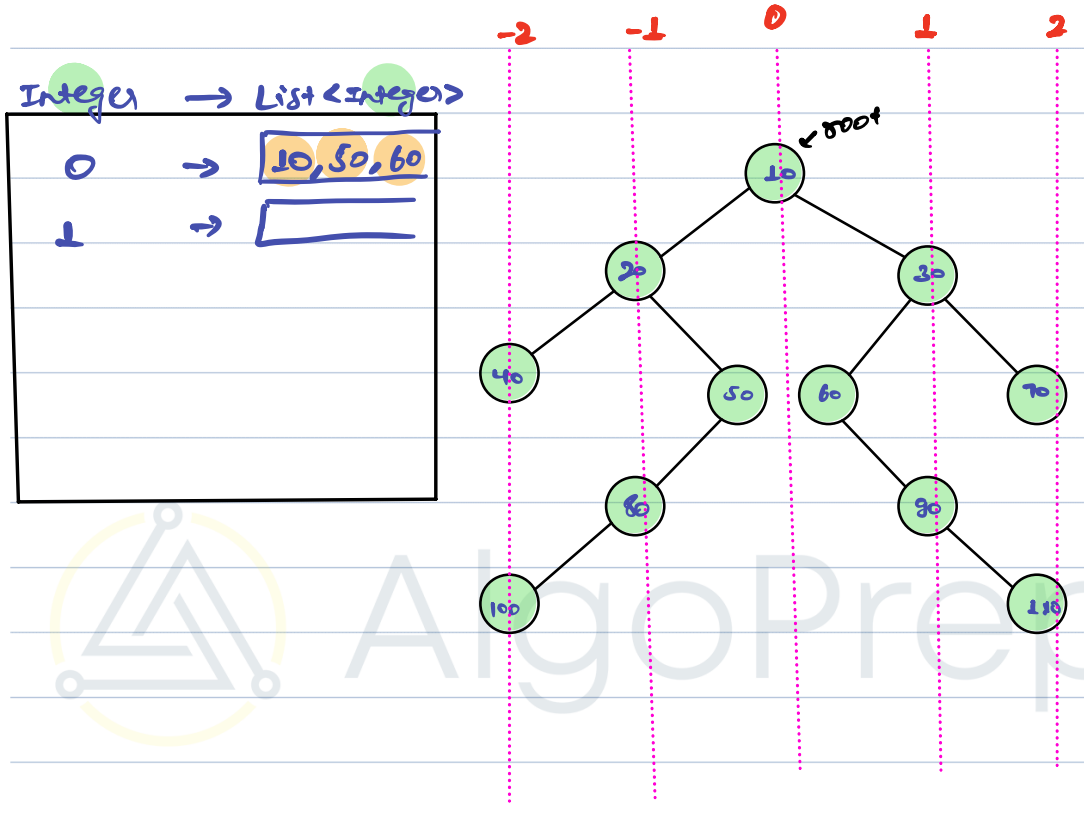
S.C: O(N)

```
Public class Pair{
    Node n;
    int vlevel;
    Pair(Node n1, int m){
        n = n1;
        vlevel = m;
    }
}

void topView(Node root){
    Queue<Pair> q;
    HashMap<Integer, Integer> map;
```

```
q.add(new Pair(root, 0));

while (q.size()>0) {
    Pair rem = q.remove();
    Node remn = rem.n;
    int remVL = rem.vlevel;

    if (~~condition~~) {
        map.put(remVL, remn.val);
    }

    if (remn.left != null) {
        q.add(new Pair(remn.left, remVL-1));
    }

    if (remn.right != null) {
        q.add(new Pair(remn.right, remVL+1));
    }
}

//iterate on map and Point

}
```

Q) vertical level

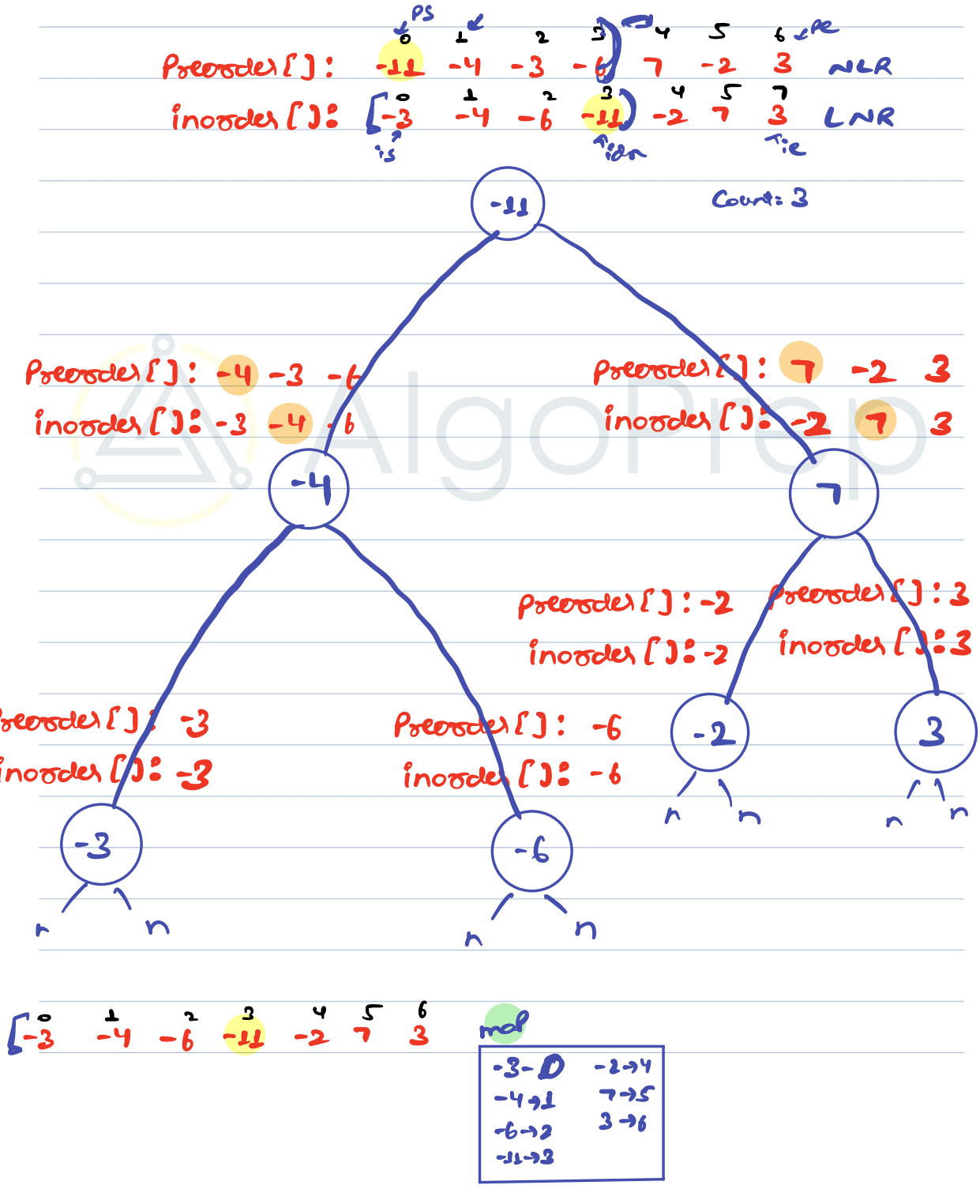
↳ Print Nodes vertical level wise.




Break till 9:36 PM

## Q) Construction of tree

↳ Given preorder and inorder of BT of distinct values, generate the trees.




[0 -3] [1 -4] [2 -6] [3 -11] [4 -2] [5 7] [6 3]

mp

| | |
|---|---|
| -3 → 0 | -2 → 4 |
| -4 → 1 | 7 → 5 |
| -6 → 2 | 3 → 6 |
| -11 → 3 | |

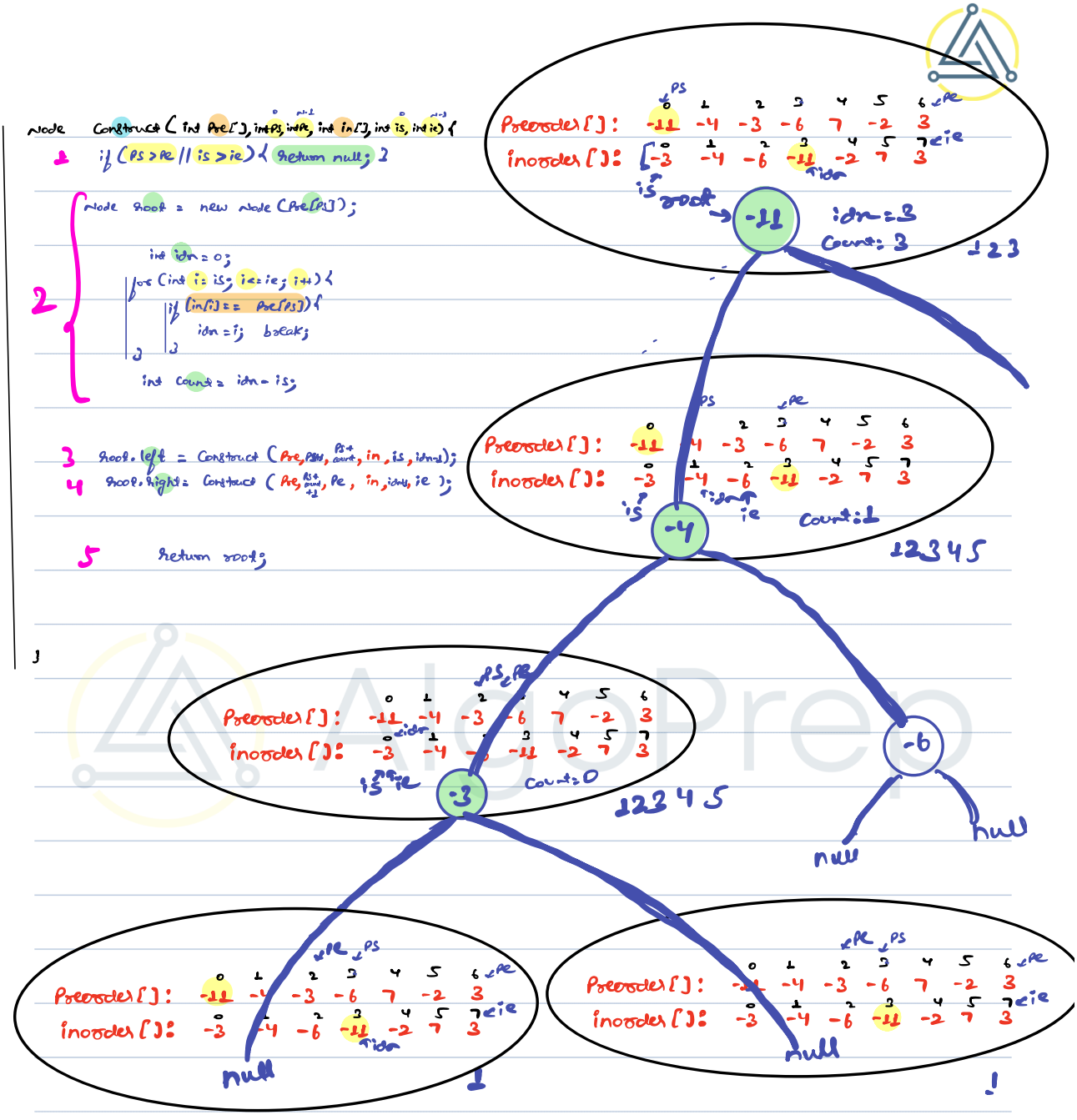
//Pseudo code

```
Node Construct ( int Pre[], int PS, int PE, int in[], int IS, int IE) {
    if (PS > PE || IS > IE) { return null; }

    Node root = new Node (Pre[PS]);

    int idn = 0;
    for (int i = IS; i <= IE; i++) {
        if (in[i] == Pre[PS]) {
            idn = i; break;
        }
    }

    int count = idn - IS;

    root.left = Construct (Pre, PS+1, PS+count, in, IS, idn-1);
    root.right = Construct (Pre, PS+count+1, PE, in, idn+1, IE);

    return root;
}
```

T.C: O(N)+O(N) = O(N)

S.C: O(N)

int idn = map.get(Pre[PS]);


Node Construct ( int Pre[ ], int ps, int pe, int In[ ], int is, int ie) {
1 if (ps > pe || is > ie) { return null; }
2
Node root = new Node (Pre[ps]);
int idn = 0;
for (int i = is; i <= ie; i++) {
if (In[i] == Pre[ps]) {
idn = i; break;
}
}
int count = idn - is;
3 root.left = Construct (Pre, ps+1, ps+count, In, is, idn-1);
4 root.right = Construct (Pre, ps+count+1, pe, In, idn+1, ie);
5 return root;
}
Preorder[ ]: -11 -4 -3 -6 7 -2 3
Inorder[ ]: -3 -4 -6 -11 -2 7 3
root
idn=3
Count: 3
123
-4
Count:1
12345
-3
Count=0
12345
-6
null
null
null
null
1
!

# Delete Node in a BST

## Java Code:

```
class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if(root == null) return null;

        if(root.val>key){
            root.left = deleteNode(root.left , key);
        }else if(root.val < key){
            root.right = deleteNode(root.right , key);
        }else{
            if(root.left == null && root.right ==null){
                return null;
            }else if(root.left != null && root.right == null){
                return root.left;
            }else if(root.left == null && root.right != null){
                return root.right  ;
            }else{

                int max = func(root.left);
                root.val = max;
                root.left = deleteNode(root.left , max);
                return root;
            }
        }
        return root;
    }
    public int func(TreeNode root){
        while(root.right != null){
            root = root.right;
        }
        return root.val;
    }
}
```

## C++ Code:

```
#include <iostream>

// Definition for a binary tree node.
struct TreeNode {
```

```
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    TreeNode* deleteNode(TreeNode* root, int key) {
        if (root == nullptr) return nullptr;

        if (root->val > key) {
            root->left = deleteNode(root->left, key);
        } else if (root->val < key) {
            root->right = deleteNode(root->right, key);
        } else {
            if (root->left == nullptr && root->right == nullptr) {
                return nullptr;
            } else if (root->left != nullptr && root->right == nullptr) {
                return root->left;
            } else if (root->left == nullptr && root->right != nullptr) {
                return root->right;
            } else {
                int maxVal = findMax(root->left);
                root->val = maxVal;
                root->left = deleteNode(root->left, maxVal);
                return root;
            }
        }
        return root;
    }

    int findMax(TreeNode* root) {
        while (root->right != nullptr) {
            root = root->right;
        }
        return root->val;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val):
        self.val = val
```

```
        self.left = None
        self.right = None

class Solution:
    def deleteNode(self, root, key):
        if root is None:
            return None

        if root.val > key:
            root.left = self.deleteNode(root.left, key)
        elif root.val < key:
            root.right = self.deleteNode(root.right, key)
        else:
            if root.left is None and root.right is None:
                return None
            elif root.left is not None and root.right is None:
                return root.left
            elif root.left is None and root.right is not None:
                return root.right
            else:
                max_val = self.findMax(root.left)
                root.val = max_val
                root.left = self.deleteNode(root.left, max_val)
                return root

        return root

    def findMax(self, root):
        while root.right is not None:
            root = root.right
        return root.val
```

# Serialize and Deserialize Binary Tree

## Java Code:

```
public class Codec {
    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuilder sb = new StringBuilder();
        Helper1(root , sb);
        return sb.toString();
    }
```

```
    public void Helper1(TreeNode root , StringBuilder sb){
        if(root == null){
            sb.append("# ");
            return;
        }
        sb.append(root.val+" ");
        Helper1(root.left , sb);
        Helper1(root.right , sb);
    }
    // Decodes your encoded data to tree.
    int i;
    public TreeNode deserialize(String data) {
        String[] ch = data.split(" ");
        i = 0;
        return Helper2(ch);
    }
    public TreeNode Helper2(String [] ch){
        if(ch[i].equals("#")){
            i++;
            return null;
        }
        TreeNode root = new TreeNode(Integer.parseInt(ch[i]));
        i++;
        root.left = Helper2(ch);
        root.right = Helper2(ch);
        return root;
    }
}
```

## C++ Code:

```
#include <iostream>
#include <sstream>
#include <vector>

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Codec {
public:
    // Encodes a tree to a single string.
```

```
    std::string serialize(TreeNode* root) {
        std::ostringstream ss;
        Helper1(root, ss);
        return ss.str();
    }

    void Helper1(TreeNode* root, std::ostringstream& ss) {
        if (root == nullptr) {
            ss << "# ";
            return;
        }
        ss << root->val << " ";
        Helper1(root->left, ss);
        Helper1(root->right, ss);
    }

    // Decodes your encoded data to tree.
    int i;

    TreeNode* deserialize(std::string data) {
        std::istringstream ss(data);
        std::vector<std::string> tokens;
        std::string token;

        while (ss >> token) {
            tokens.push_back(token);
        }

        i = 0;
        return Helper2(tokens);
    }

    TreeNode* Helper2(const std::vector<std::string>& tokens) {
        if (tokens[i] == "#") {
            i++;
            return nullptr;
        }
        TreeNode* root = new TreeNode(std::stoi(tokens[i]));
        i++;
        root->left = Helper2(tokens);
        root->right = Helper2(tokens);
        return root;
    }
};
```

## Python Code:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

class Codec:
    def serialize(self, root):
        result = []
        self.Helper1(root, result)
        return ' '.join(result)

    def Helper1(self, root, result):
        if root is None:
            result.append("#")
            return
        result.append(str(root.val))
        self.Helper1(root.left, result)
        self.Helper1(root.right, result)

    def deserialize(self, data):
        tokens = data.split()
        self.i = 0
        return self.Helper2(tokens)

    def Helper2(self, tokens):
        if tokens[self.i] == "#":
            self.i += 1
            return None
        root = TreeNode(int(tokens[self.i]))
        self.i += 1
        root.left = self.Helper2(tokens)
        root.right = self.Helper2(tokens)
        return root
```

# Binary Tree Cameras_HW

## Solution Vid:

https://youtu.be/5iLSl8fNIHQ

## Java Code:

```
int camera;
public int minCameraCover(TreeNode root) {
    camera = 0;
    int rootreturn = traversal(root);

    if(rootreturn == 0){
        camera++;
    }

    return camera;

}

public int traversal(TreeNode node){
    if(node == null){
        return 2;
    }

    int left = traversal(node.left);
    int right = traversal(node.right);

    if(left == 0 || right == 0){
        camera++;
        return 1;
    }

    if(left == 1 || right == 1){
        return 2;
    }
```

```
        return 0;
    }
}
```