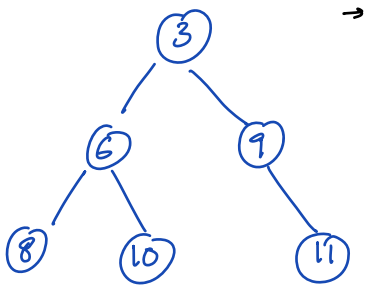
Recursion:

Ass: { Decide, what your function should do, assume it does }

Main logic: { Solving problems with subproblems }

Base Condition: { When code stops }

1(S) Given BT, search for k.



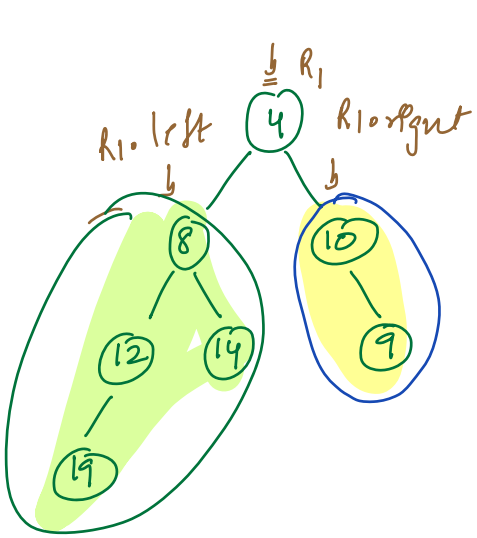
→ k = 9 ✓

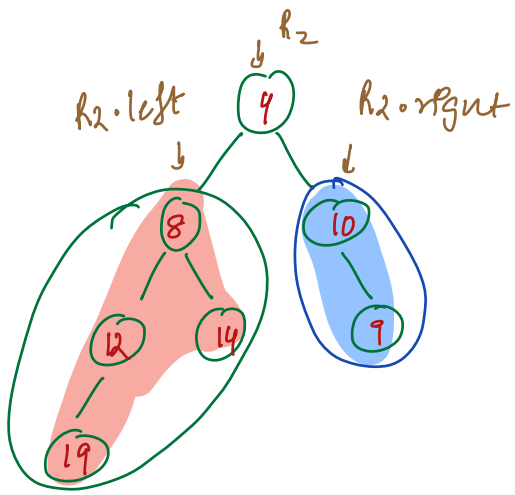
k = 20 ✗

Idea: Apply Inorder/ Postorder/ Preorder/ Search for k

TC: O(N)

2(S) Given 2 BT check if they are similar





Ass: Check if Both Trees are exactly same.

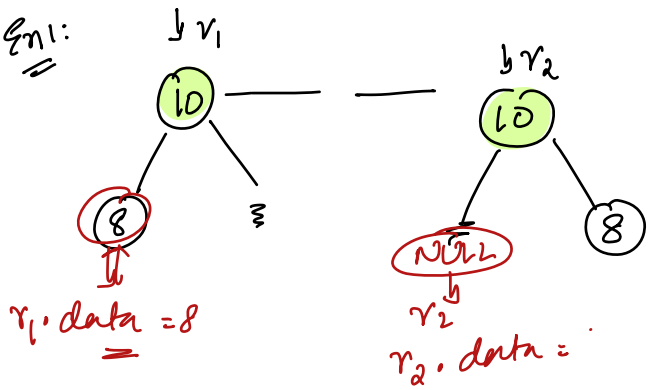
↳ logical AR

```
bool checkSim(Node r1, Node r2) {
    if (r1 == NULL && r2 == NULL) { return True }
    if (r1 == NULL || r2 == NULL) { return False }

    if (r1.data != r2.data) { return false }
        LST same
    if( checkSim(r1.left, r2.left) &&
                checkSim(r1.right, r2.right) {
        RST same
        return True
    }
    return False
}
```

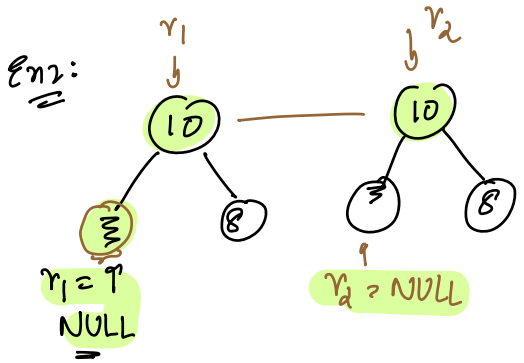
Base Condt

Edge Cases:

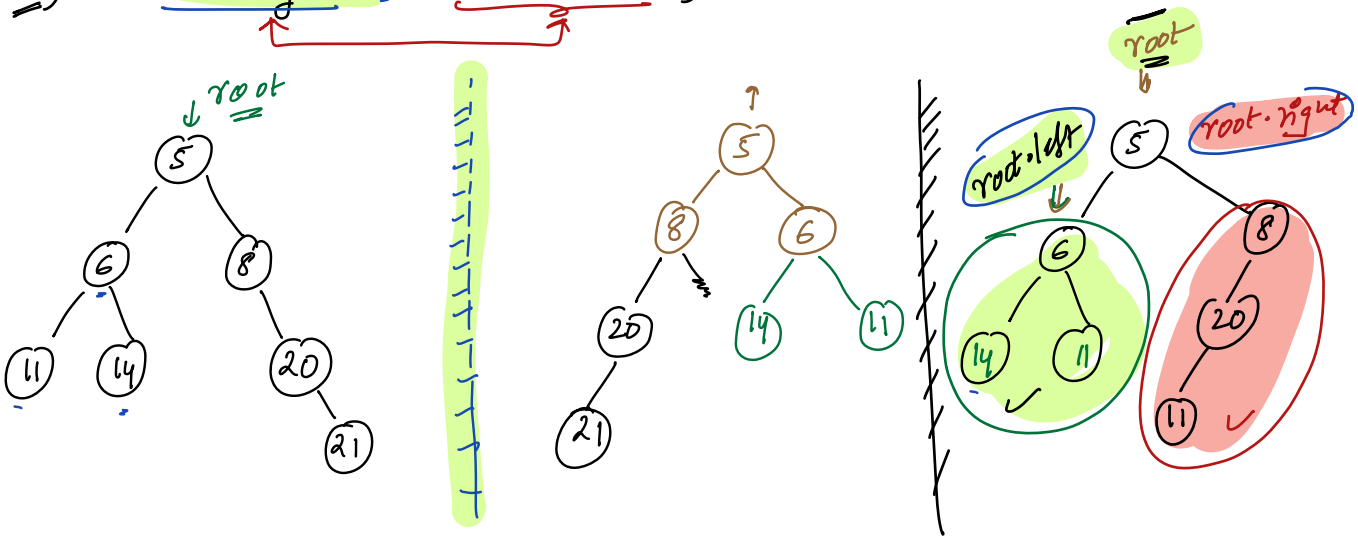
Eg1:



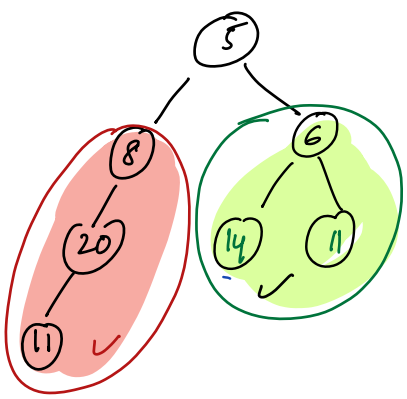
Eg2:



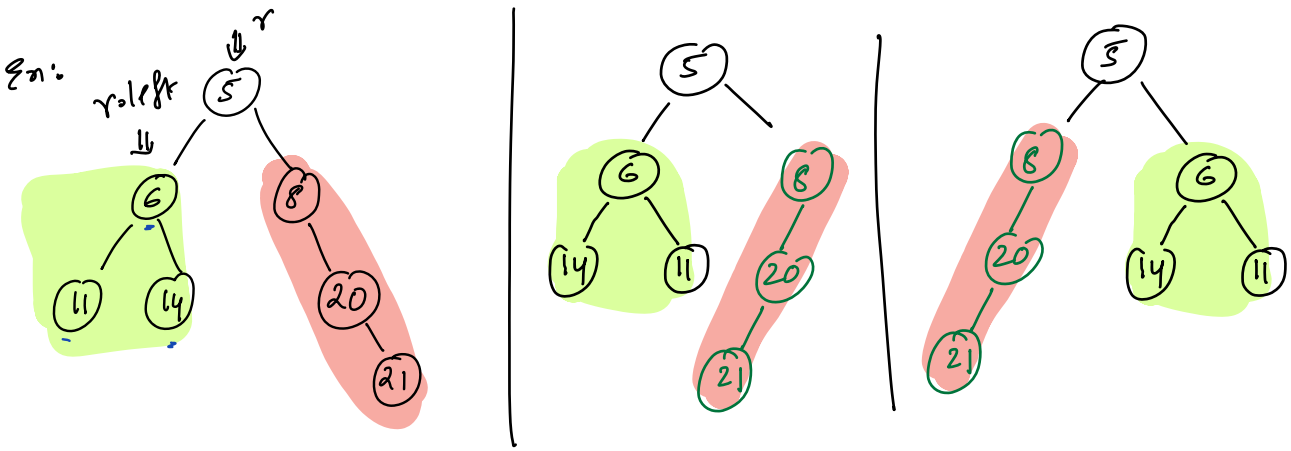
28) (Inverting BT - LinkedIn Meme)



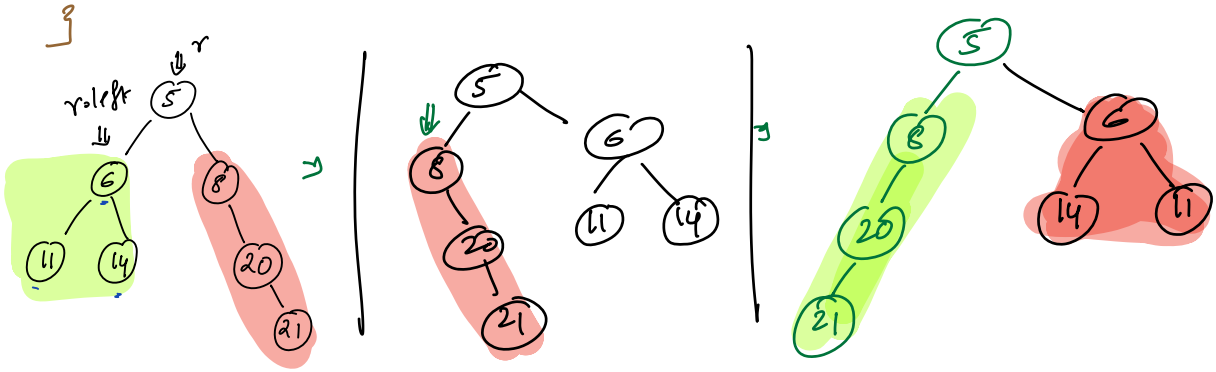
Ass: Given root Node Invert Binary Tree



```
Void    Invert ( Node root) {

    if ( root == NULL) { return }

    Invert ( root.left)

    Invert ( root.right)

  // swap root.left & root.right

  { Node temp = root.left
    root.left = root.right
    root.right = temp }  => changing a particular node

}
```




```
Void    Invert ( Node root) {

    if ( root == NULL) { return }

    Swap root.left & root.right

    Node temp = root.left

    root.left = root.right

    root.right = temp

    Invert (root.right)

    Invert (root.left)

}
```

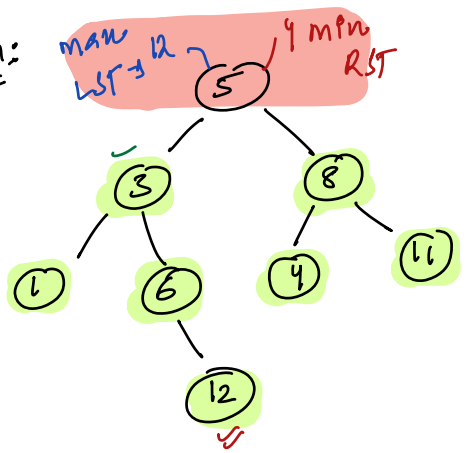


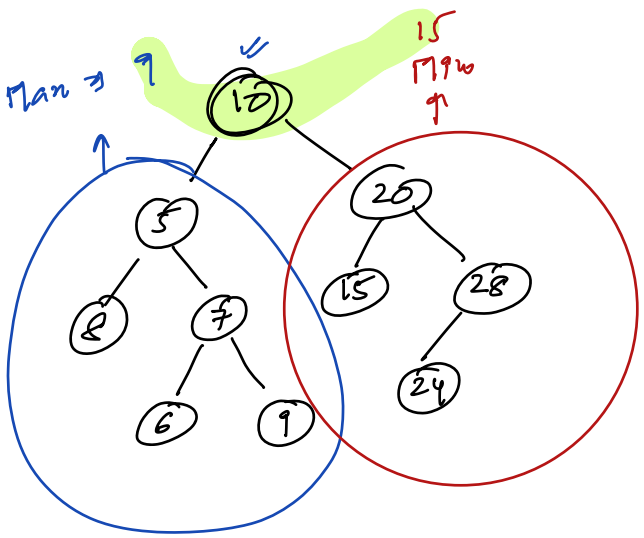
BST : Binary Search Tree

For all Nodes if (Max of LST) < (Node.data) < (Min of RST)

⇒ Given BT ⇒ BST

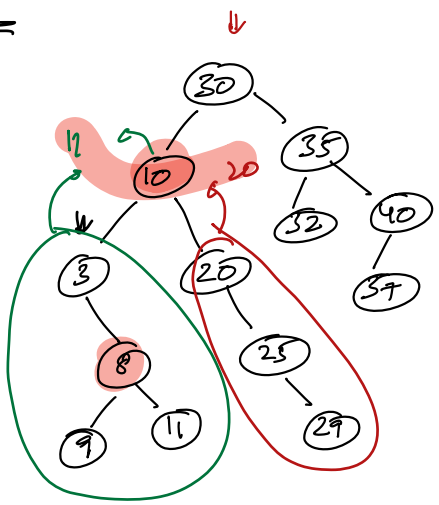
Ex:





Not BST

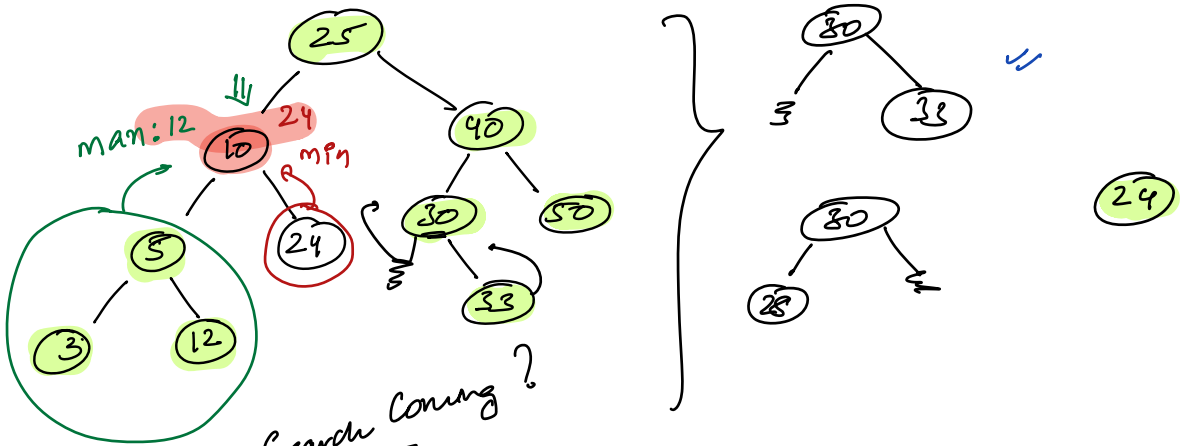
Ex:



For all Nodes (All LST) < (Node.data) < (All RST)

For all Nodes (Max of LST) < (Node.data) < (Min of RST)

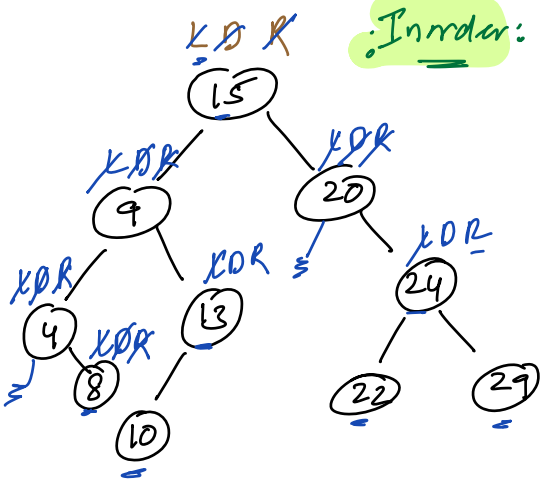
For all Nodes Max of (LST) < (Node.data) < (Min of RST)



Special BST property Search Coming ?

Increasing order

Inorder: 4 8 9 10 13 15 20 22 24 29

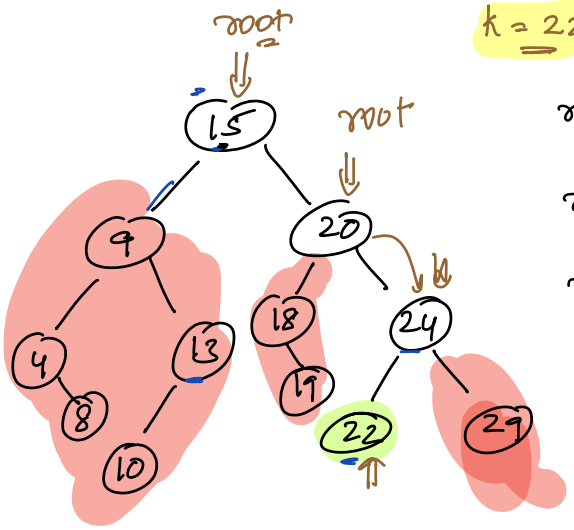


For a BST, Inorder : Sorted

: Given BT, check BST or not?

: BT → (Inorder) → Sorted or not

Search k in BST



k = 22

Search in

root.data < 22 : right, root = root.right

root.data < 22 : right, root = root.right

root.data > 22 : left, root = root.left

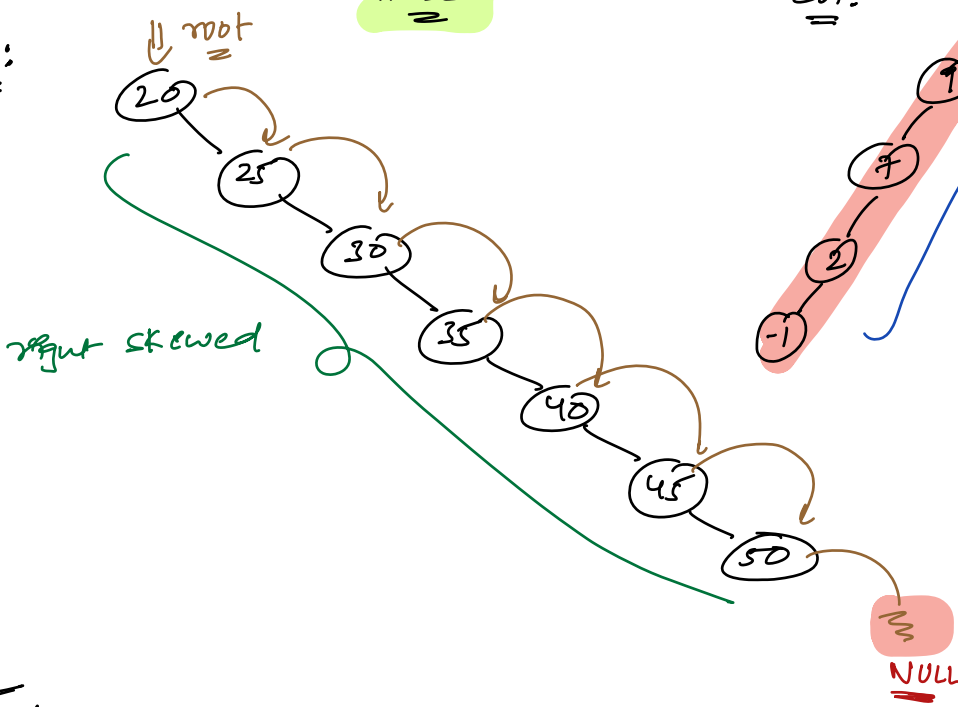
root.data == 22 : (return True)

```
bool searchC(Node root, k) {
    while(root != NULL) {
        if(root.data == k) {
            return True
        }
        else if(root.data > k) {
            root = root.left
        }
        else {
            root = root.right
        }
    }
    return false;
}
```

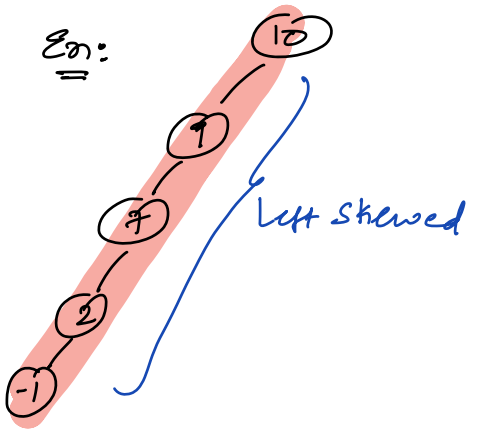
↳ Worst Case Scenario

TC: (O(N))

Eg:



Eg:



{ return False }

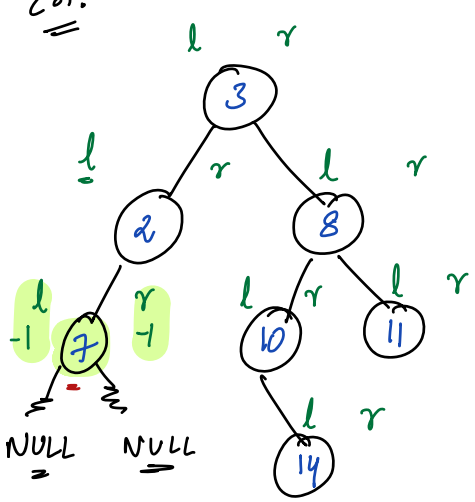
I/P

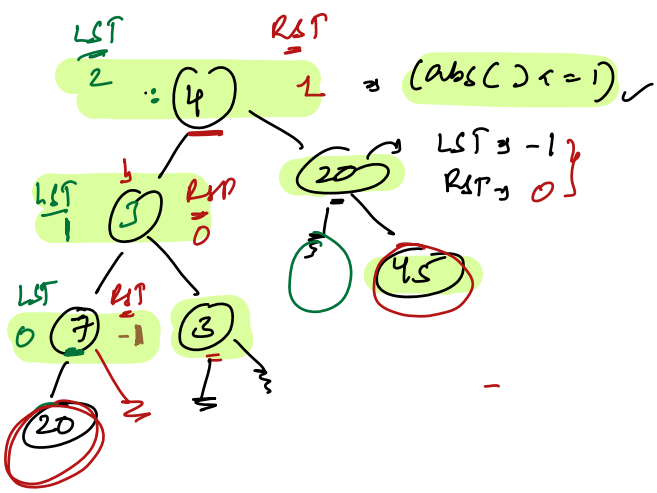
Balanced Binary Tree :

height = (Edges)

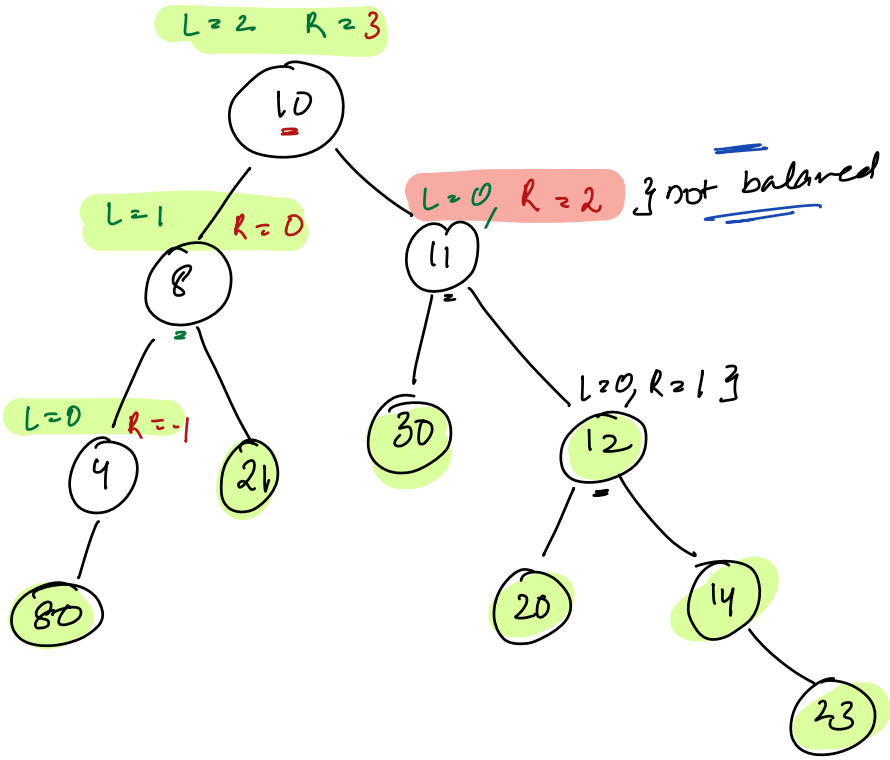
For all Nodes | abs (height(LST) - height(RST)) <= 1

Eg:





Eg:



// Given BT check Balance Balanced or Not?

```
boolean IsBBT = True

int height (Node root) {

    if (root == NULL) { return -1 }

    hl = height (root.left)        → height of LST

    hr = height (root.right)       → height of RST

    if (abs (hl - hr) > 1) { IsBBT = False }

    return (max (hl, hr) + 1)

}
```

```
solution : solve(Node *root){
    IsBST = True
    height(root)
}
```

Note: Reset global variable before each function

In advanced Data

obs: If Tree is Balanced, height of tree = $(\log N)$

N = Number of Nodes

Todays Announcement :

→ March - 9th →

Content :

Set 1
- Arrays → 3
- Bits M → 2
- Maths → 4
- Recursion → 1½

Set 2
- Sorting → 3½
- Searching + 2 Pointer = {5}
- Hashing + Strings → {4}

HS1

Set 3
- Linked List → {4}
- Stacks / Que / Deq → {4}

Set 4
- Trees / BT / BST / Tries → {7}
- Heaps & greedy → {3}
- Segment Trees → {2}

Set 5
- Back tracking → {2/3}
- Dynamic Program → {7}
- graphs → {5}

→ Space / Tips of Fun → (I will keep a remainder)

1) To Revise a particular from your notes : [Revise]

2) Revise + : (To do from intermediate)
   Revise 2 : (To do from advanced)

3) Revise ↦ topic / technique / Pseudo Code /

→ 56 →
- Ass : 4 : Can you do it in same day / Next day : prc 1
- HW : 3 : next day after Session

→ Total → (56 × 7) = 400

Weeks: M — W — F Sat Sat

Ass e / HW

Must: Al eat try to complete Assignments of a topic

→ 2hr: 30

→ (Don't skip a Session)