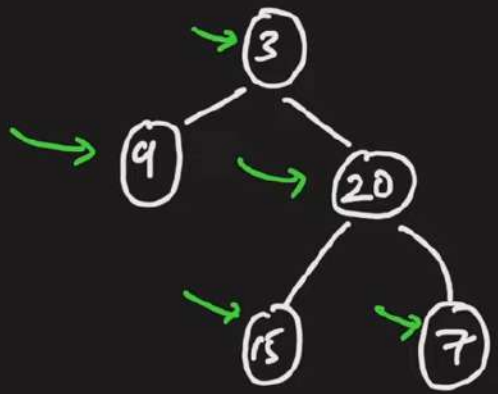
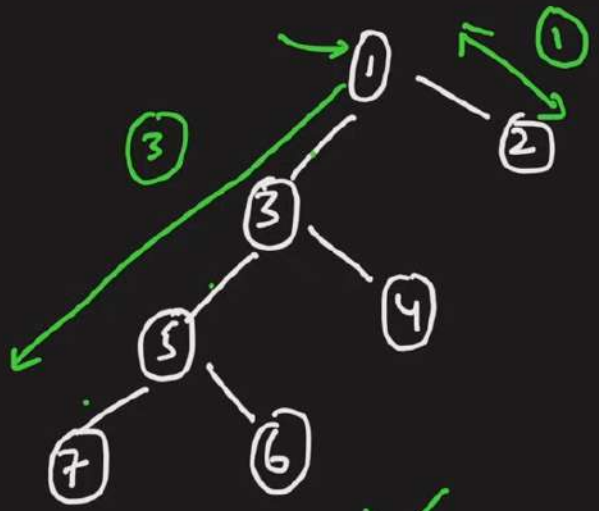
Check for Balanced Binary Tree

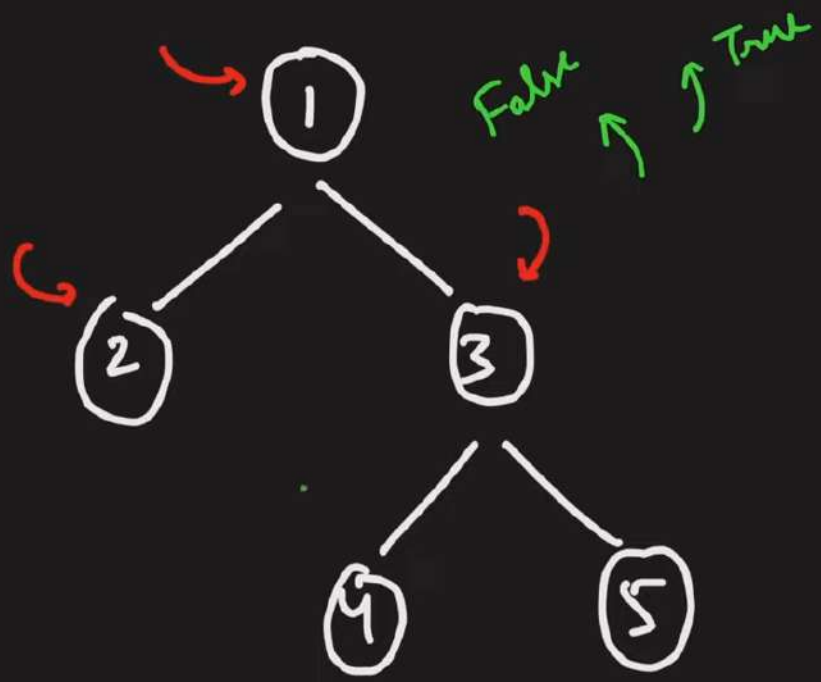




3 - 1 = 2

1

Balanced BT → for every node, height(left) − height(right) <= 1



False

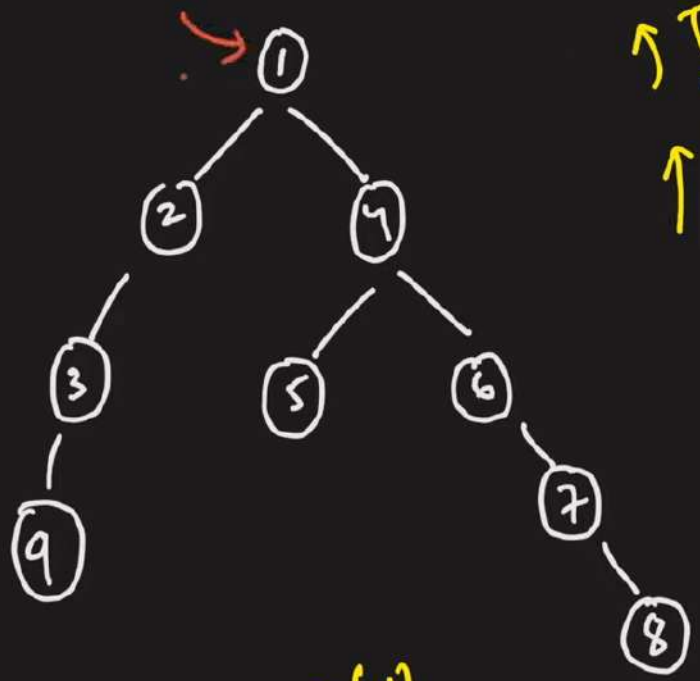
True

```
Bool check(Node)
    If node == null
        Return true

    Lh = findHLeft(node->left)
    Rh = findHRight(node->right)

    If(abs(rh - lh) > 1) return false;

    Bool left = check(node->left);
    Bool right = check(node->right);

    If(!left || !right) return false;

    Return true;
```

O(N).

O(N) x O(N) = O(N²)

O(N)



↑ True

↑ false

O(N)

(Height of a tree)

→ O(N)

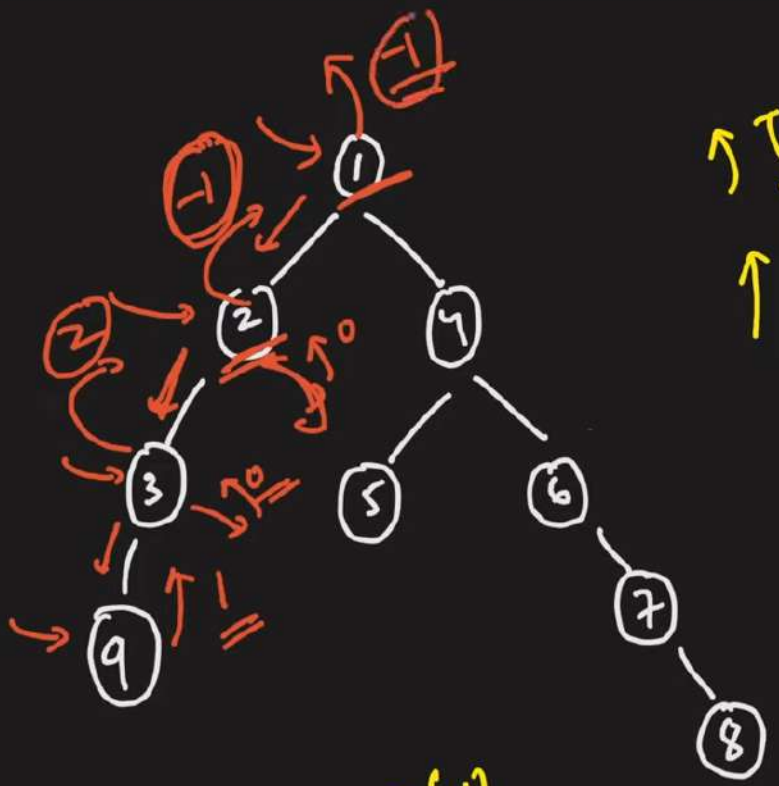
```
int check (node)
{
  if (node == null)
      return 0
```

↑ height

↑ -1 X

```
  lh = check (node → left)
  rh = check (node → right)
  if (lh == -1 || rh == -1) return -1;
  if (abs(lh - rh) > 1) return -1;
  return max(lh, rh) + 1;
}
```

X



↑ True

↑ false

O(N)

(Height of a tree)

↳ O(N)

lh = 2

rh = 0

```
int check (node)
{
  if (node == null)
    return 0

  lh = check (node -> left)
  rh = check (node -> right)
  if (lh == -1 || rh == -1) return -1;
  if (abs (lh - rh) > 1) return -1;
  return max (lh, rh) + 1;
}
```

↑ height

-1 X

X

Java

Autocomplete

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isBalanced(TreeNode root) {
        return dfsHeight (root) != -1;
    }
    int dfsHeight (TreeNode root) {
        if (root == null) return 0;

        int leftHeight = dfsHeight (root.left);
        if (leftHeight == -1) return -1;
        int rightHeight = dfsHeight (root.right);
        if (rightHeight == -1) return -1;

        if (Math.abs(leftHeight - rightHeight) > 1)  return -1;
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```