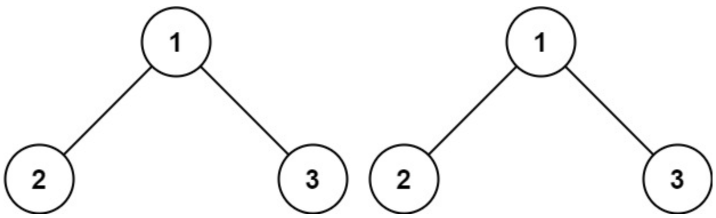
ASSIGNMENT ON TREES

1. **Given the roots of two binary trees `p` and `q`, write a function to check if they are the same or not.**

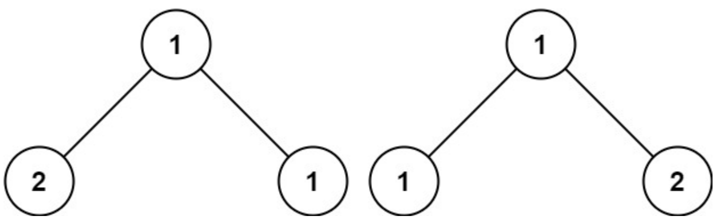
Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

EXAMPLE 1



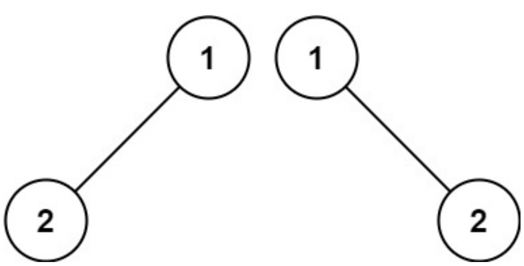
```
Input: p = [1,2,3], q = [1,2,3]Output: true
```

Example 2



```
Input: p = [1,2,1], q = [1,1,2]Output: false
```

Example 3

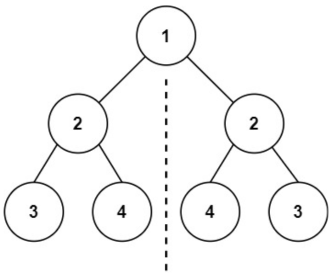


```
Input: p = [1,2], q = [1,null,2]Output: false
```

2. **Symmetric Tree**

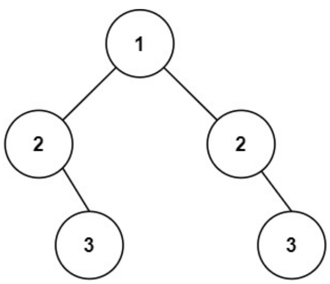
Given the `root` of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

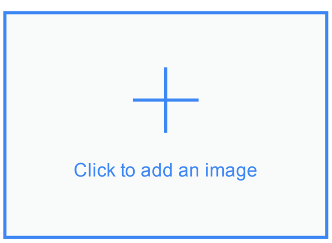
Example 1:



> **Input:** root = [1,2,2,3,4,4,3]**Output:** true

Example 2:

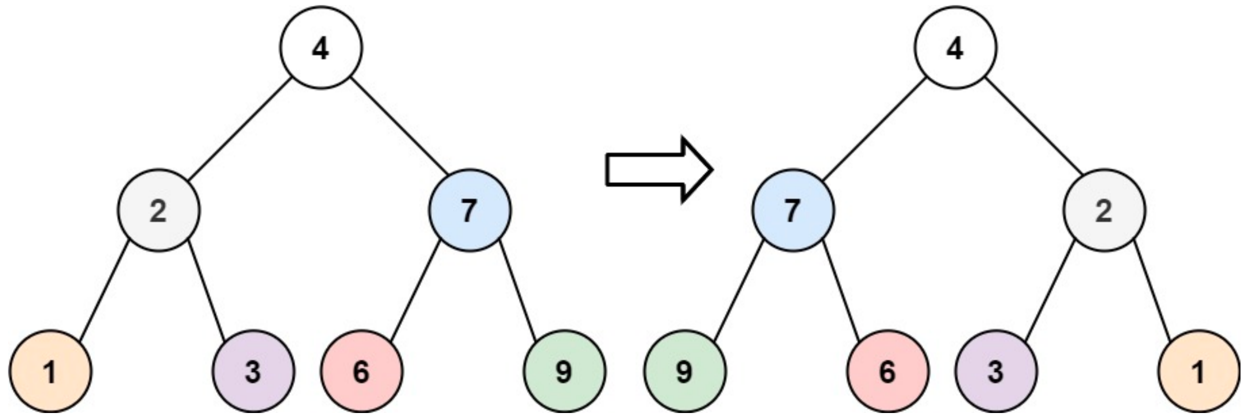




> **Input:** root = [1,2,2,null,3,null,3]**Output:** false

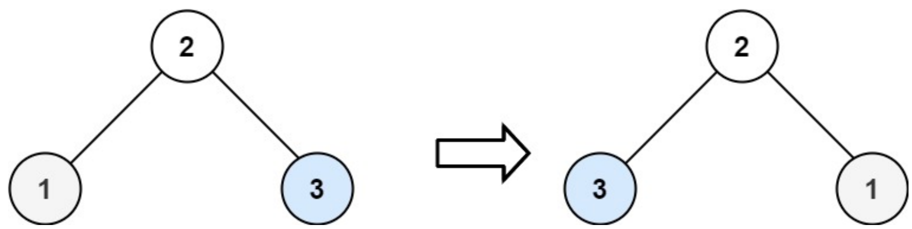
3. **Invert Binary Tree**

Given the `root` of a binary tree, invert the tree, and return *its root*.
Example 1:




```
Input: root = [4,2,7,1,3,6,9]Output: [4,7,2,9,6,3,1]
```

Example 2:



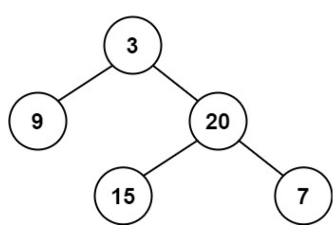

```
Input: root = [2,1,3]Output: [2,3,1]
```

4. **Balanced Binary Tree**

Given a binary tree, determine if it is height-balanced

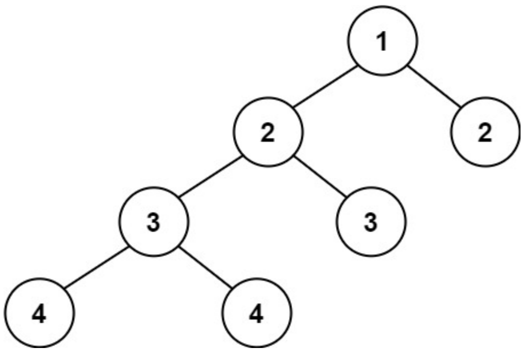
.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]Output: true
```

Example 2:



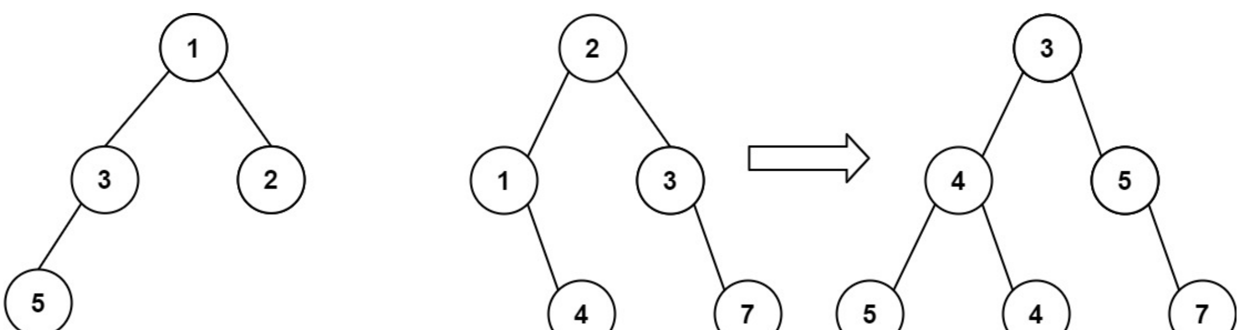

```
Input: root = [1,2,2,3,3,null,null,4,4]Output: false
```

Example 3:

```
Input: root = []Output: true
```

5. **Merge Two Binary Trees**

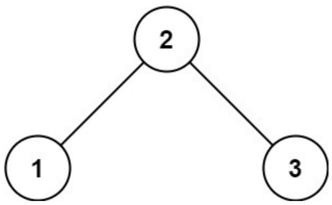
You are given two binary trees `root1` and `root2`.
Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.
Return *the merged tree*.
Note: The merging process must start from the root nodes of both trees.



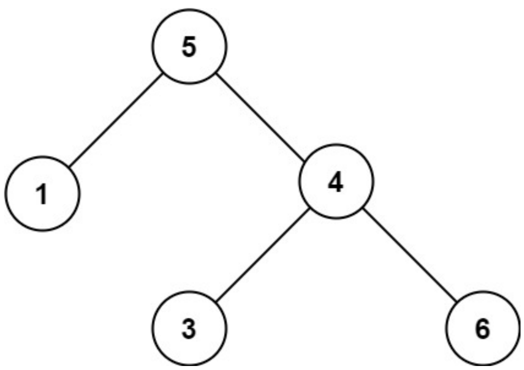

```
Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]Output: [3,4,5,5,4,null,7]
```

6. **Validate Binary Search Tree**

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*



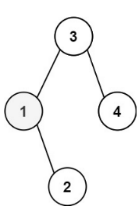

```
Input: root = [2,1,3]Output: true
```



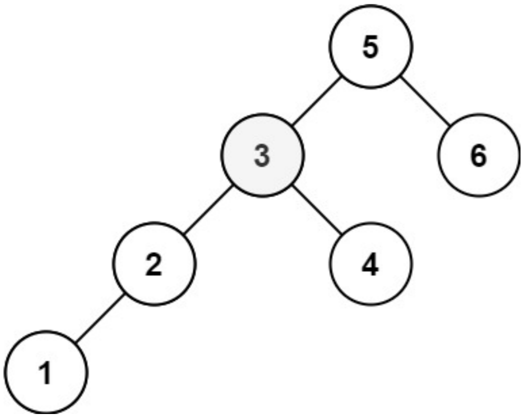

```
Input: root = [5,1,4,null,null,3,6]Output: false Explanation: The root node's value i
```

7. **Kth Smallest Element in a BST**

Given the `root` of a binary search tree, and an integer `k`, return *the* k^{th} *smallest value (1-indexed) of all the values of the nodes in the tree.*



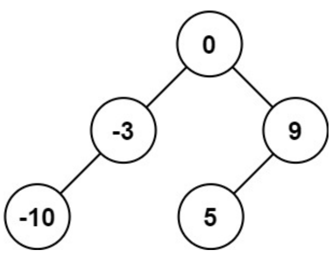

```
Input: root = [3,1,4,null,2], k = 1Output: 1
```



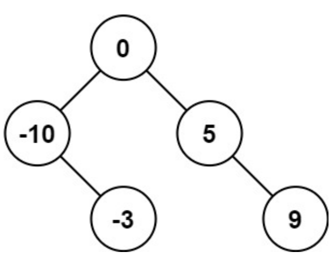

```
Input: root = [5,3,6,2,4,null,null,1], k = 3Output: 3
```

8. **Convert Sorted Array to Balanced Binary Search Tree**

Given an integer array `nums` where the elements are sorted in ascending order, convert *it to a height-balanced binary search tree*.




```
Input: nums = [-10,-3,0,5,9]Output: [0,-3,9,-10,null,5]Explanation: [0,-10,5,null,-3,
```

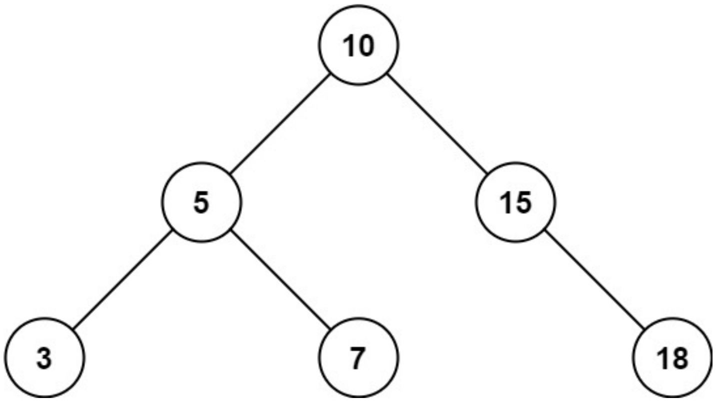


Q8 Range Sum of BST

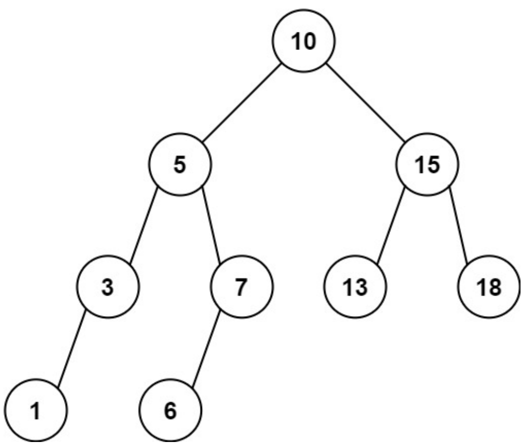
Google
Facebook
Amazon

Given the `root` node of a binary search tree and two integers `low` and `high`, return ***the sum of values of all nodes with a value in the inclusive range*** **`[low, high]`**.

```
Input: root = [10,5,15,3,7,null,18], low = 7, high = 15Output: 32Explanation: Nodes 
```




```
Input: root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10Output: 23Explanation:
```

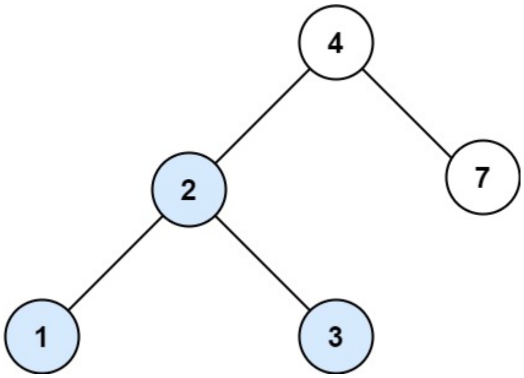


Q9

Search in a Binary Search Tree

You are given the `root` of a binary search tree (BST) and an integer `val`.

Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node. If such a node does not exist, return `null`.




```
Input: root = [4,2,7,1,3], val = 2Output: [2,1,3]
```