Balanced Binary Tree (LeetCode 110 - Easy)

```java
/**
 * Top Down.
 * @param root
 * @return if root is a balanced binary tree.
 * @timecomplexity - O(nlogn)
 * time complexity for depth is O(n) (DFS)
 * F(n) = 2*depth(n/2) + 2F(n/2) = n + 2F(n/2) = O(nlogn)
 */
public static boolean isBalanced_1(TreeNode root) {
	if (root == null) {
		return true;
	}
	return Math.abs(depth(root.left) - depth(root.right)) <= 1 &&
		isBalanced_1(root.left) && isBalanced_1(root.right);
}

private static int depth(TreeNode root) {
	if (root == null) {
		return 0;
	}
	if (root.left == null && root.right == null) {
		return 1;
	}
	return 1 + Math.max(depth(root.left), depth(root.right));
}

/*
 * O(n) solution
 * This solution only calculates height once for each node.
 * Bottum Up.
 */
public static boolean isBalanced_2(TreeNode root) {
	return dfsHeight(root) != -1;
}

private static int dfsHeight(TreeNode root) {
	if(root == null) {
		return 0;
	}
	int leftHeight = dfsHeight(root.left);
	int rightHeight = dfsHeight(root.right);
	if (leftHeight == -1 || rightHeight == -1 ||
			Math.abs(leftHeight - rightHeight) > 1) {
		return -1;
	}
	return 1 + Math.max(leftHeight, rightHeight);
}
```