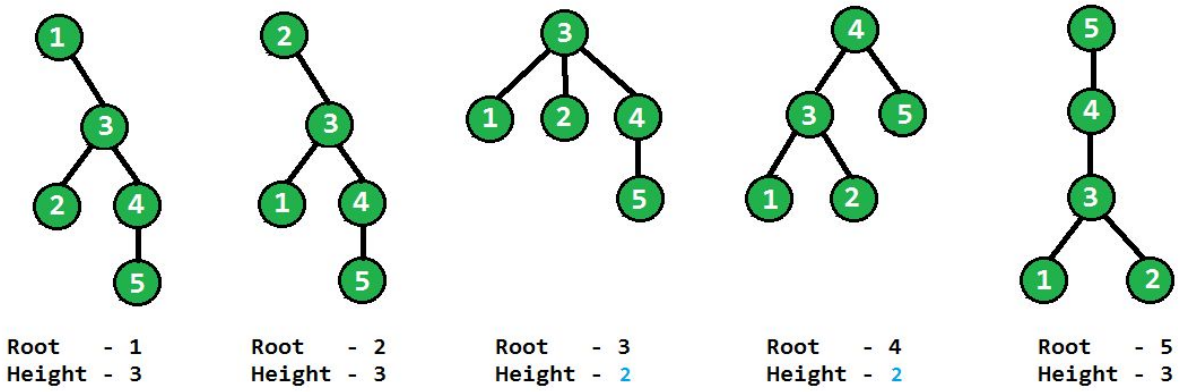
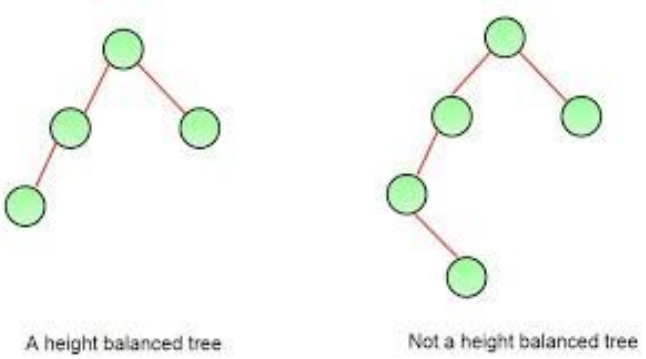
Find number of Balanced Binary Tree Violations

Balanced Binary tree : At every node the difference of height of left subtree and height of right subtree should not exceed more than 1.

Height : Length of the longest path from root to the leaf.



Violation : Any node which violates the above property in the binary tree.



The property is violated at root and root's left on RHS image

image credits : geeksforgeeks.

Approach 1 : O(N ^ 2)

Since this is a tree problem we will try and solve this recursively. Let's see how we should go about it.

1. One thing is for sure, when we are at any node we have to check whether there is a violation at that node and If there is a violation this would contribute 1 to our final result. The pseudo code for that would look something like this

```
left = get_height(root.left)
right = get_height(root.right)
```

```
If absolute difference of left and right is greater than 1
        make sure you add 1 to final answer
```

2. Implementing get_height function recursively is straight forward. The pseudo code for get_height function is

```
get_height(root)
        if root is null
                return -1
        return 1 + max(get_height(root.left), get_height(root.right))
```

3. Continuing from point 1, now we know whether the current node violates the property or not, next we need to examine all the nodes in the left subtree and all the nodes in the right subtree.

```
left = get_height(root.left)
right = get_height(root.right)
violation = 0;
If absolute difference of left and right is greater than 1
        violation++;
return violation + find_violation(root.left) + find_violation(root.right)
```

4. The base case for find_violation is simple i.e there are no violations if the tree is empty.

Approach 2

1. Can we do better than O(n ^ 2), so notice that the order we traverse the nodes while calculating height and number_of_violations is same
2. So we can eliminate calling the number_of_violation function and calculate the violations also in height function only so that we can avoid calculating height over and over again.
3. If we do this, we get O(n) as the final time complexity.