PROBLEM 4:  (Fluffer (12 points))

In class we discussed height-balanced trees. An alternative to height-balanced is node-balanced — for example we want the number of nodes in every left subtree to be close to the number of nodes in the corresponding right subtree. When this is the case we call the tree fluffy. More precisely we want the ratio of the number of nodes in each left subtree to the number in the right subtree to be at least 45% (or vice versa). The tree on the left below is fluffy at the root with a ratio of 50%. It might be completely fluffy but we can’t see the smaller subtrees. For example, the subtree labeled with 50 nodes on the left might have the 50 nodes divided into a left subtree of 40 nodes and a right subtree of 10 nodes as shown on the right. Then the ratio is $\frac{1}{4}$ or 25% so the tree isn’t fluffy.

![Tree Diagram](image)

Part A (6 points)

The method `isFluffy` below correctly determines if a tree is fluffy as described above. The method adds one to the count of the number of nodes in each subtree to avoid division by zero problems. What is the running time of `isFluffy` for an N-node tree? Justify your answer, use big-O. Provide two answers: one in the average case when trees are roughly balanced, and one when trees are not balanced.

```java
public int nodeCount(TreeNode root){
    if (root == null) return 0;
    return 1 + nodeCount(root.left) + nodeCount(root.right);
}

public boolean isFluffy(TreeNode root){
    if (root == null) return true;
    int leftCount = 1 + nodeCount(root.left);
    int rightCount = 1 + nodeCount(root.right);

    double ratio = leftCount * 1.0/rightCount;
    if (ratio > 1.0) ratio = 1/ratio;
    if (ratio > 0.45 && isFluffy(root.left) && isFluffy(root.right)){
        return true;
    }
    return false;
}
```
Part B (6 points)
A student turns in the code below for isFluffy but calls the method isBushy. The method works correctly, returning the same values as isFluffy. What is the running time of this method? Use big-O and justify your answer. Provide two answers: one in the average case when trees are roughly balanced, and one when trees are not balanced.

```java
class TreeNode {
    int data;
    TreeNode left, right;
}

public boolean isBushy(TreeNode root){
    double ratio = 1.0*(1+nodeCount(root.left))/(1 + nodeCount(root.right));
    if (nodeCount(root.left) >= nodeCount(root.right)){
        ratio = 1.0*(1+nodeCount(root.right))/(1+nodeCount(root.left));
    }
    if (ratio > 0.45 && isBushy(root.left) && isBushy(root.right)){
        return true;
    }
    return false;
}
```