Objective: More fun with recursion. Check the provided starter code to make sure you have the correct prototypes. Feel free to modify prototypes if something else is more useful.

In the comments to each function, provide a big-Oh expression for the complexity of the functions you write, assuming trees are roughly balanced (depth = log(n) for n nodes). Use recursion where appropriate, but if something isn't logically recursive, don't use recursion.

1. Write functions that return the minimum and maximum nodes in a binary search tree rooted at root. Assume trees store unique elements (no duplicates).

2. Write a function to find the inorder successor of a node. You should use the parent link. This function can be written in fewer than 10 lines of code – that’s a guideline, not a requirement.

3. Write the function levelCount that returns the total number of nodes on the specified level. For this problem, the root is at level zero, the root’s children are at level one, and for any node, the node’s level is one more than its parent’s level.

4. Write the function count that returns the total number of nodes in the tree.

5. Write the function findKthInOrder. Given an integer k and a binary search tree with unique values, findKthInOrder returns a pointer to the node that contains the kth element if the elements are in sorted order – the node with the smallest value is returned if k = 1, the node with the second smallest is returned if k = 2, and so on.

For example, in the tree t shown below (t points to the root), findKthInOrder(t,4) returns a pointer to the node with value 9, findKthInOrder(t,8) returns a pointer to the node with value 18, and findKthInOrder(t,12) returns NULL since there are only 9 nodes in the tree.

Consider the following strategy. Using the function count, if the left subtree has fewer than k nodes, simply adjust the k you are looking for and perform findKthInOrder of the right subtree.
6. By definition, the *diameter* of a tree (sometimes called the width) is the number of nodes on the longest path between two leaves in the tree. The diagram below shows two trees each with diameter nine, the leaves that form the ends of a longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).

It can be shown that the diameter of a tree $T$ is the largest of the following quantities:

- The diameter of $T$'s left subtree
- The diameter of $T$'s right subtree
- The longest path between leaves that goes through the root of $T$ (this can be computed from the heights of the subtrees of $T$)

Here's code that's almost a direct translation of the three properties above (assuming the existence of a standard $O(1)$ max function that returns the larger of two values).

```c
// returns diameter of tree rooted at t
int diameter(TreeNode * t)
{
    if (t == 0) return 0;

    int leftD = diameter(t->left);
    int rightD = diameter(t->right);
    int rootD = height(t->left) + height(t->right) + 1;

    return max(leftD, max(rootD, rightD));
}
```
return max(rootD, max(leftD, rightD));
}

However, the function as written does not run in $O(n)$ time. Write a version of
diameter that runs in $O(n)$ time. Hint, use a function as described below.

```c
int diameterHeight(TreeNode * t, int & height)
{
    // pre:  t is a binary tree
    // post: return (via reference param) height = height of t
    //       return as value of function: diameter of t
    
    }
```

7. Two binary trees $s$ and $t$ are isomorphic if they have the same shape; the values stored in the nodes do not affect whether two trees are isomorphic. In the diagram below, the tree in the middle is not isomorphic to the other trees, but the tree on the right is isomorphic to the tree on the left.

8. Two trees $s$ and $t$ are quasi-isomorphic if $s$ can be transformed into $t$ by swapping left and right children of some of the nodes of $s$. The values in the nodes are not important in determining quasi-isomorphism, only the shape is important. The trees below are quasi-isomorphic because if the children of the nodes $A$, $B$, and $G$ in the tree on the left are swapped, the tree on the right is obtained.

Write a function `isQuasiIsomorphic` that returns true if two trees are quasi-isomorphic.