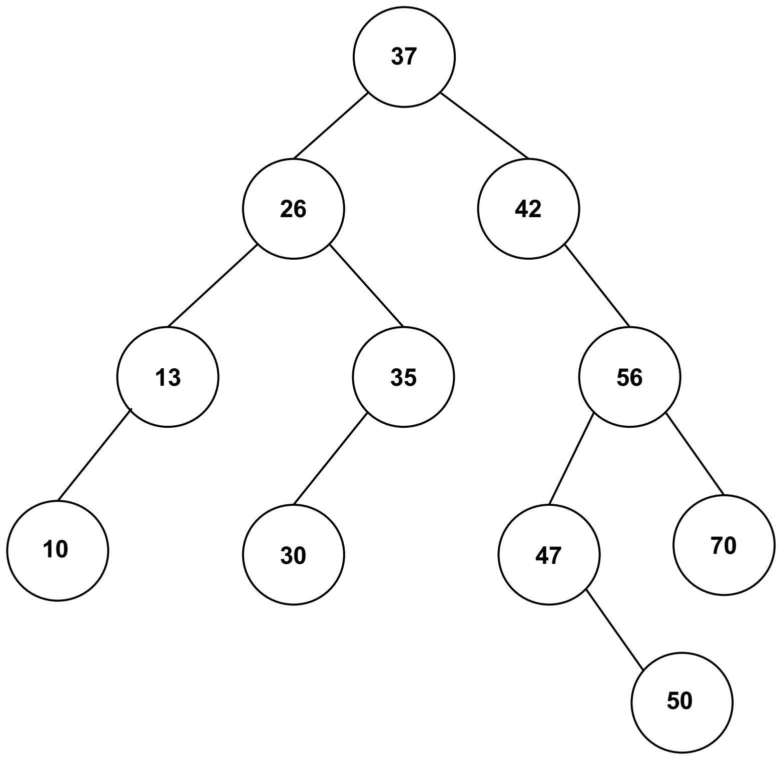
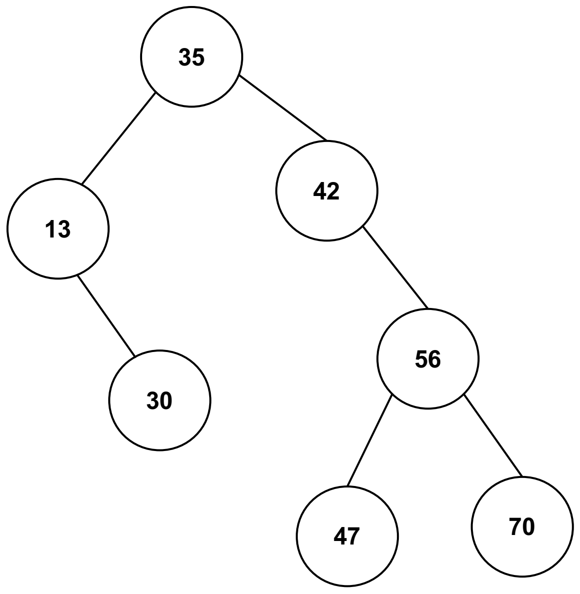
Problem Set 7, Part I

Problem 1: Binary search trees
1-1)

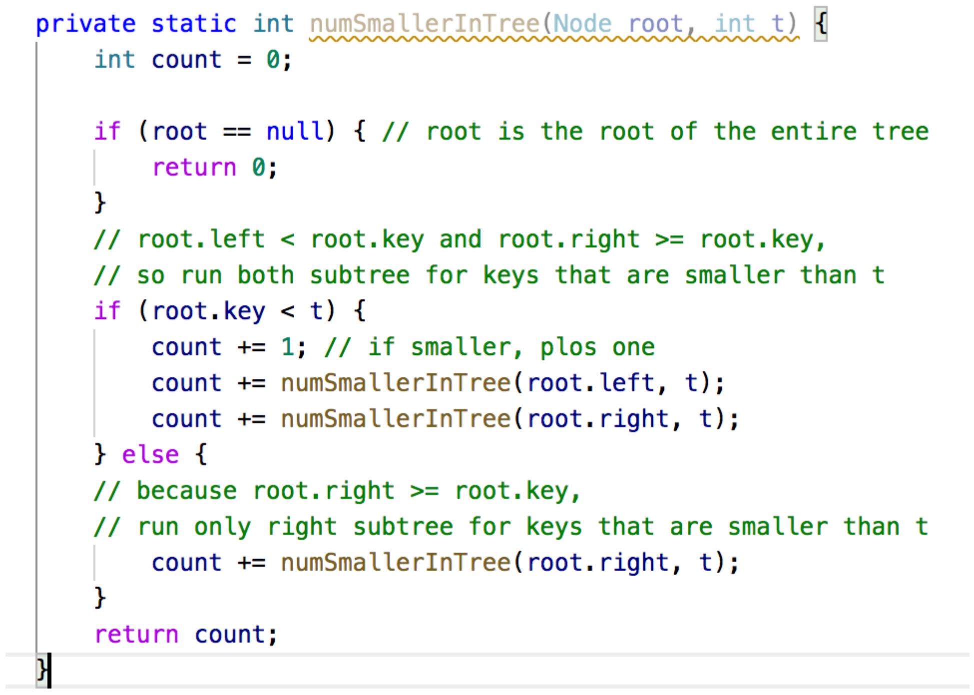


1-2)



Problem 2: Counting keys below a threshold
2-1)
In the best case is O(1), when the root = null, which means root is the root of the entire tree, and in the worst case is O(n), when both the left subtree and right sub tree are not null and all the keys in the entire tree are less than t.

2-2)

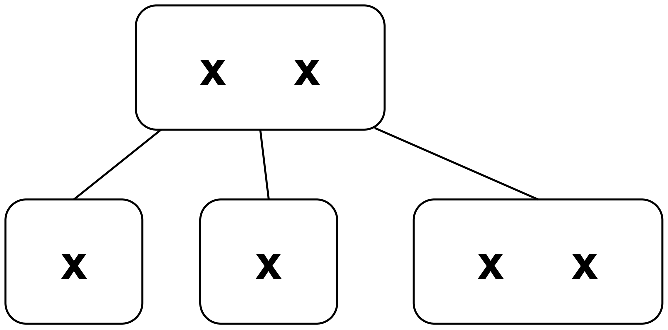


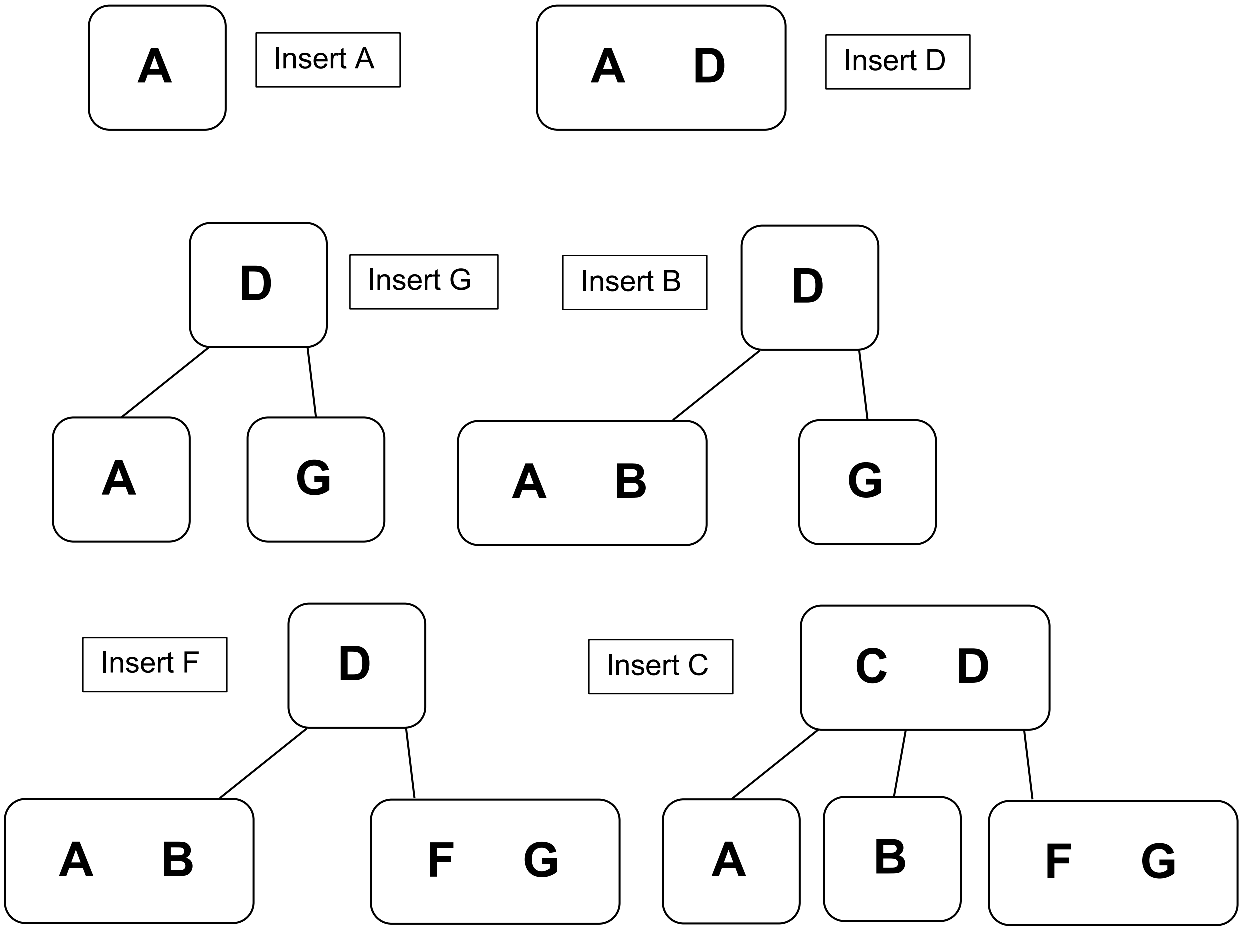
```
private static int numSmallerInTree(Node root, int t) {
    int count = 0;

    if (root == null) { // root is the root of the entire tree
        return 0;
    }
    // root.left < root.key and root.right >= root.key,
    // so run both subtree for keys that are smaller than t
    if (root.key < t) {
        count += 1; // if smaller, plos one
        count += numSmallerInTree(root.left, t);
        count += numSmallerInTree(root.right, t);
    } else {
    // because root.right >= root.key,
    // run only right subtree for keys that are smaller than t
        count += numSmallerInTree(root.right, t);
    }
    return count;
}
```

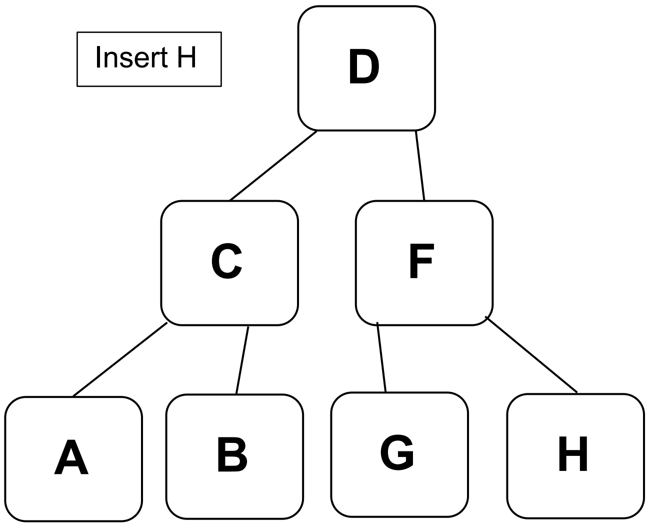
2-3)
In the best case, same as the original algorithm, is O(1) when the root = null, which means root is the root of the entire tree, and in the worst case, also same as the original algorithm, is O(n), when both the left subtree and right sub tree are not null and all the keys in the entire tree are less than t. However, each time the key is smaller than t, we don’t need to consider the left subtree because all nodes in k’s right subtree are >= k, so the revised algorithm is still more efficient than the original one.

Problem 3: Balanced search trees

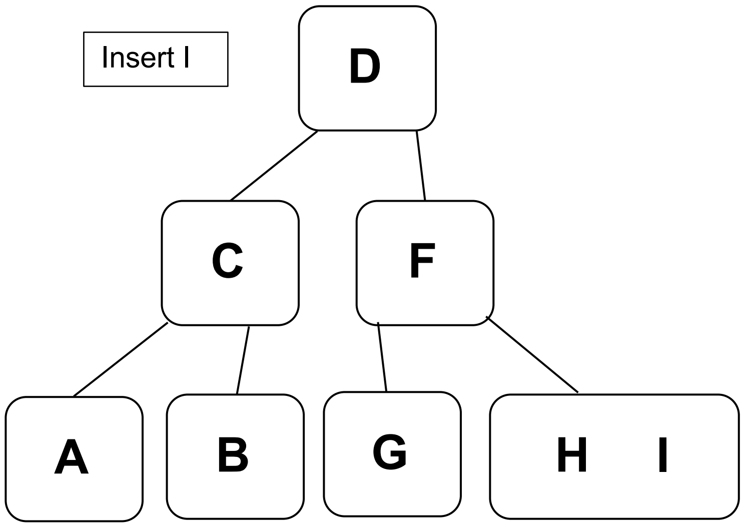




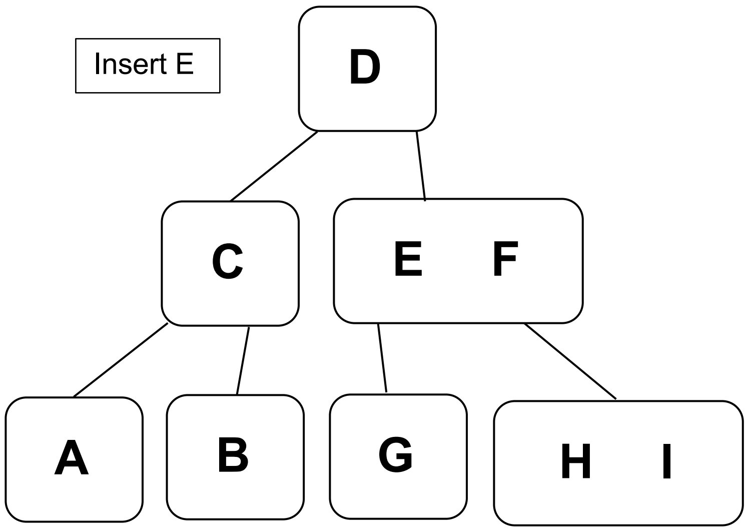
Insert H



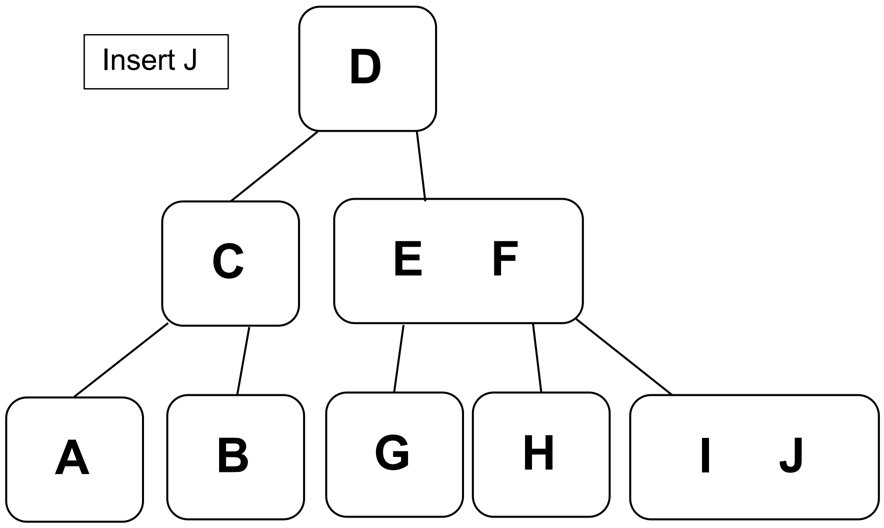
Insert I



Insert E



Insert J


D
C
E F
A
B
G
H
I J