CS 152h: Data Structures Homework #4
BST TREES

4.4 Show that in a binary tree of N nodes, there are $N + 1$ `NULL` links representing children.

Ans: There are N nodes. Each node has two links, so there are $2N$ links. Each node but the root has one incoming link from its parent, which accounts for $N - 1$ links. The rest are `NULL`.

4.5 Show that the maximum number of nodes in a binary tree of height h is $2^{h+1} - 1$. *Hint*: Prove by induction.

Ans: Proof is by induction. The theorem is trivially true for $h = 0$. Assume true for $h = 1, 2, \cdots, k$. A tree of height $k + 1$ can have two subtrees of height at most k. These can have at most $2^{k+1} - 1$ nodes each by the induction hypothesis. These $2^{k+2} - 2$ nodes plus the root prove the theorem for height $k + 1$ and hence for all heights.

4.32 Design a recursive linear-time algorithm that tests whether a binary tree satisfies the search tree order property at every node.

Ans: Have the recursive routine return a triple that consists of a boolean (whether the tree is a BST) and the minimum and maximum items. Then a tree is a BST if it is empty or both subtrees are (recursively) BSTs, and the value in the node lies between the maximum of the left subtree and the minimum of the right subtrees. Coding details are omitted.

4.37 Write a function that takes as input a binary search tree, T, and two keys k_1 and k_2, which are ordered so that $k_1 \leq k_2$, and prints all elements X in the tree such that $k_1 \leq key(X) \leq k_2$. Do not assume any information abut the type of keys except that they can be ordered (consistently). Your program should run in $(O(K + \log N)$ average time, where K is the number of keys printed. Bound the running time of your algorithm.

Ans: This is known as one-dimensional range searching. The time is $O(K)$ to perform the inorder traversal, if a significant number of nodes are found, and also proportional to the depth of the tree, if we get to some leaves (for instance, if no nodes are found). Since the average depth is $O(\log N)$, this gives an $O(K + \log N)$ average bound. The following code is written in Java:

```
static void printRange( Comparable lower, Comparable upper, BinaryNode t )
{
  if( t != null )
  {
    if( lower.compareTo( t.element ) <= 0 )
       printRange( lower, upper, t.left );
    if( lower.compareTo( t.element ) <= 0 && t.element.compareTo( upper ) <= 0 )
       System.out.println( t.element );
    if( t.element.compareTo( upper ) <= 0 )
       printRange( lower, upper, t.right );
  }
}
```