(1) Exercise 12.2-4 on Page 293 [10 points]

**12.2-4**
Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key $k$ in a binary search tree ends up in a leaf. Consider three sets: $A$, the keys to the left of the search path; $B$, the keys on the search path; and $C$, the keys to the right of the search path. Professor Bunyan claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counterexample to the professor’s claim.

**Solution:**

![Diagram of a binary search tree with sets A, B, and C labeled.]

Figure 1: Counter example to Professor Bunyan’s claim. The set $A$ is empty. The search path, where search is performed for the key 3 is marked. The search proceeds from root 8 to the node 4 and then to node 3. So $B = \{8, 4, 3\}$. Set $C$ is the only key to the right of the path, i.e., $C = \{6\}$.

The claim is wrong. A simple counter example is shown in figure 1. In the figure, the search is being done for leaf node 3, so the set $B = \{8, 4, 3\}$. There is nothing to the left of the path and so set $A = \{\phi\}$. Set $C$ is all elements to the right of the path, so set $C = \{6\}$. For any element $a \in A$, and $b \in B$ the claim is true, since $A$ is an empty set. But if set $b = 8$ and $c = 6$, the claim fails to hold.

(2) Exercise 12.2-8 on Page 294 [15 points]
12.2-8
Prove that no matter what node we start at in a height- \( h \) binary search tree, \( k \) successive calls to TREE-SUCCESSOR take \( O(k + h) \) time.

Suppose \( x \) is the starting node and \( y \) is the ending node. The distance between \( x \) and \( y \) is at most \( 2h \), and all the edges connecting the \( k \) nodes are visited twice, therefore it takes \( O(k + h) \) time.

(3) Exercise 12.3-1 on Page 299 [10 points]

12.3-1
Give a recursive version of the TREE-INSERT procedure.

\[
\text{TREE-INSERT}(z, k) \\
\text{If } z = \text{NIL} \text{ then} \\
\quad \text{key}[z] \leftarrow k \\
\quad \text{left}[z] \leftarrow \text{NIL} \\
\quad \text{right}[z] \leftarrow \text{NIL} \\
\text{else} \\
\quad \text{if } k < \text{key}[z] \text{ then} \\
\quad \quad \text{TREE-INSERT}(\text{left}[z], k) \\
\text{else} \\
\quad \quad \text{TREE-INSERT}(\text{right}[z], k)
\]

(4) Problem 12-1 on Page 303 [30 points]

12-1 Binary search trees with equal keys
Equal keys pose a problem for the implementation of binary search trees.

\( a. \) What is the asymptotic performance of TREE-INSERT when used to insert \( n \) items with identical keys into an initially empty binary search tree?

We propose to improve TREE-INSERT by testing before line 5 to determine whether \( z.\text{key} = x.\text{key} \) and by testing before line 11 to determine whether \( z.\text{key} = y.\text{key} \).
If equality holds, we implement one of the following strategies. For each strategy, find the asymptotic performance of inserting $n$ items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of $z$ and $x$. Substitute $y$ for $x$ to arrive at the strategies for line 11.)

b. Keep a boolean flag $x.b$ at node $x$, and set $x$ to either $x.left$ or $x.right$ based on the value of $x.b$, which alternates between FALSE and TRUE each time we visit $x$ while inserting a node with the same key as $x$.

c. Keep a list of nodes with equal keys at $x$, and insert $z$ into the list.

d. Randomly set $x$ to either $x.left$ or $x.right$. (Give the worst-case performance and informally derive the expected running time.)

d. Worst-case: every random choice is to the right (or all to the left) this will result in the same behavior as in the first part of this problem, $\Theta(n^2)$.

Expected running time: notice that when randomly choosing, we will pick left roughly half the time, so, the tree will be roughly balanced, so, we have that the depth is roughly $\lg(n)$, $\Theta(n \lg n)$.

(5) Exercise 13.3-2 on Page 322 [15 points]

13.3-2
Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree.

Solution:
The resulting red-black trees are shown in the figure:
Exercise 14.1-5 on Page 344 [20 points]

14.1-5

Given an element $x$ in an $n$-node order-statistic tree and a natural number $i$, how can we determine the $i$th successor of $x$ in the linear order of the tree in $O(\log n)$ time?

Solution:
The data structure should support the following two operations:

OS-RANK($T$, $x$), which returns the position of $x$ in the linear order determined by an in order tree walk of $T$ in $O(\log n)$ time,

OS-SELECT($x$, $i$), which returns a pointer to the node containing the $i$th smallest key in the subtree rooted at $x$ in $O(\log n)$ time.

The $i$th successor of $x$ is given by OS-SELECT($x$, OS-RANK($T$, $x$) + 1), which will also run in $O(\log n)$ time.