(1) Exercise 12.2.3 on Page 293 [15 points]
Write the TREE-PREDECESSOR procedure.

Answer:

TREE-PREDECESSOR(x)
1  if x.left != NIL
2    return TREE-MAXIMUM(x.left)
3  y = x.p
4  while y != NIL and x == y.left
5    x = y
6    y = y.p
7  return y

(2) Exercise 12.2.4 on Page 293 [15 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.

Answer:

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.
(3) Exercise 12.3-1 on Page 299 [15 points]
Give a recursive version of the TREE-INSERT procedure.

Answer:
RECURSIVE-TREE-INSERT(z, k)
1     If z == NIL then
2         z.key = k
3     z.left = NIL
4     z.right = NIL
5     else
6         if k < z.key then
7             RECURSIVE-TREE-INSERT(z.left, k)
8         else
9             RECURSIVE-TREE-INSERT(z.right, 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.key = x.key and by testing before line 11 to determine whether z.key = y.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.)

Answer:
a. Each insertion will add the element to the right of the rightmost leaf because the inequality on line 11 will always evaluate to false. The height is n. This will result in the runtime being \( \sum_{i=1}^{n} i \in \Theta(n^2) \)

b. This strategy will result in each of the two children subtrees having a difference in size at most one. This means that the height will be \( \Theta(\lg n) \). So, the total runtime will be \( \sum_{i=1}^{n} \lg n \in \Theta(n \lg n) \)

c. This will only take linear time since the tree itself will be height 0, and a single insertion into a list can be done in constant time. The insertion time is \( \Theta(1) \) and n times insertion is \( \Theta(n) \).
**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 [25 points]
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.

**Answer:**
The resulting red-black trees are shown in the figure:

![Red-black tree diagram](image)