Eric Roberts
CS 106B

Handout #38
February 18, 2015

Section #6—Trees

For problems 1, 2, and 3, assume that **`BSTNode`** is defined as follows:

```
struct BSTNode {
   string key;
   BSTNode *left, *right;
};
```

1. Tracing binary tree insertion (Chapter 16, review question 9, page 724)

In the first example of binary search trees, the text uses the names of the dwarves from Walt Disney's 1937 classic animated film *Snow White and the Seven Dwarves.* Dwarves, of course, occur in other stories. In J. R. R. Tolkien's *The Hobbit*, for example, 13 dwarves arrive at the house of Bilbo Baggins in the following order:

Dwalin, Balin, Kili, Fili, Dori, Nori, Ori, Oin, Gloin, Bifur, Bofur, Bombur, Thorin

Diagram the binary search tree you get from inserting these names into an empty tree in this order. Once you have finished, answer the following questions about your diagram:

1a. What is the *height* (defined on page 691 as the number of nodes in the longest path from the root to a leaf) of the resulting tree?

1b. Which nodes are leaves?

1c. Which nodes, if any, are out of balance (in the sense that the subtree rooted at that node fails to meet the definition of balanced trees on page 706)?

1d. Which key comparisons are required to find the string **`"Gloin"`** in the tree?

2. Calculating the height of a binary tree (Chapter 16, exercise 6, page 730)

Write a function

```
int height(BSTNode *tree);
```

that takes a binary search tree and returns its height.

3. Checking whether a tree is balanced (Chapter 16, exercise 7, page 731)

Write a function

```
bool isBalanced(BSTNode *tree);
```

that determines whether a given tree is balanced according to the definition in the section on "Balanced trees." To solve this problem, all you really need to do is translate the definition of a balanced tree more or less directly into code. If you do so, however, the resulting implementation will be inefficient because it has to call **`height`** at every node in the tree. The real challenge in this problem is to implement the **`isBalanced`** function so that it determines the result without looking at any node more than once.

Problem 4. Heap structures

In this problem, suppose that you start with a heap implementation of a partially ordered tree that is initially empty.

4a. Show the contents of the heap array after you enter each of the following values (i.e., draw one diagram for each of these values): 63, 24, 55, 93, 17, 65, 42, 13.

4b. Draw the diagram of the corresponding partially ordered tree at the end of the sequence of values from 4a.

4c Show the contents of the heap array after removing the lowest value in the heap, continuing the process until the heap is empty.