Assignment 2 – Due Monday, October 3 at Noon

Note: This assignment has an unusual deadline, which is not on a class day. Your first exam is on Tuesday, October 4, and I don't want you to have to choose between a last minute push to complete this assignment and studying for the exam. I would still encourage you to try to get this finished by the preceding Friday, at the latest.

Objectives: There are three objectives for this assignment:

- Master details of red-black tree insertion
- Get further practice writing recursive functions on trees
- Learn about basic experimental analysis and profiling

Background: Insertion into a red-black tree can either be done with bottom-up tree balancing, as described in Section 19.5.1, or with top-down tree adjustments, as described in Section 19.5.2. The code in the book uses the top-down method, which is nice because all tree adjustments are done in a single pass down the tree when searching for the insertion point. In contrast, the bottom-up balancing requires two passes: one down the tree to find the insertion point, and then a pass back up the tree for adjusting the tree. However, the bottom-up approach makes some applications of red-black trees much easier, so it's worth considering this approach further. We can simplify the bottom-up method (and avoid recursion) by including a "parent" pointer in each node, allowing us to easily move up the tree as well as down.

Here's the technique in a nutshell: Do a standard unbalanced binary search tree insertion first, and color the new node red. After just the insertion of the new red node into a non-empty red-black tree, it's clear that all red-black tree properties are still satisfied, with the possible exception of property 3 (page 715). In other words, if the parent of the new node is also red, then we have just created a red child of a red node, which is not allowed (call this a "red-red violation"). If we have a red-red violation, we can correct it at this position with a rotation (single or double), as described in Section 19.5.1, which may create a single red-red violation higher in the tree. If there's a new violation, we move up the tree and repeat this process, continuing up the tree until there are either no more violations or we reach the root. If the algorithm goes all the way up the tree to the root, there is a possibility that the root is changed to a red node — in that case, it should simply be recolored black so that red-black tree property 2 is satisfied. More details on this are in the textbook, and we will discuss it thoroughly in class.

What To Do: Start with the code in Bitbucket, as in previous assignments: fork the "Assign2" repository, rename it to include your username, grant read access to the class administrators, and then use NetBeans on your computer to clone it so you can work with it. This NetBeans project contains the full unbalanced binary search tree implementation (from Section 19.1), the full top-down red-black tree implementation (from Section 19.5), and an incomplete class `AltRedBlackTree` that is the start of a

bottom-up rebalancing implementation. I have slightly modified this code so that the `RedBlackTree` and `AltRedBlackTree` are subclasses of the `BinarySearchTree` class, so that operations that don't depend on the rebalancing operations on red-black trees can be done by generic code that will work for all binary search trees (operations such as the finding keys, counting numbers of nodes, etc.). In object-oriented design terminology this is a good way to organize these classes, since a `RedBlackTree` IS-A `BinarySearchTree` with some additional functionality for maintaining the red-black balance properties.

Your first task is to study the book's red-black tree implementation, along with the explanation in the book, and make sure you completely understand it. Then you should do the following:

- Finish the implementation of the `AltRedBlackTree` class for the bottom-up rebalancing technique. Note that the node definition in this class (`AltRBNode`) includes a parent pointer, as described above. The basic insertion routine is provided, and you need to write the `adjustTree()` method that does the bottom-up rebalancing. Make sure you test your implementation thoroughly, which will require writing some internal testing procedures (some basic tests are provided in the `main` method in the `AltRedBlackTree` class, although this probably isn't enough to test everything – you can run this `main` function by selecting the file in the projects pane of NetBeans, right clicking and selecting "Run File").

- The main project class, `Assign2`, includes code to print out statistics about the tree, but you will need to implement a few new methods in the `BinarySearchTree` class: `getSize()`, `getHeight()`, and `getInternalPathLength()`. [*Hint: See Figures 18.19 and 18.21 in the book.*]

- Once you are confident that your tree implementation and statistics methods are working correctly, run the main program in the `Assign2` class, which will builds trees and prints out statistics about the trees that these implementations build. This will run six tests: each of the three implementations is run once with randomly ordered insertions, and once with insertions in sorted order.

- NetBeans has a built-in profiler that will measure how much time each of the testing methods takes to build its tree, and you should use this to get a feel for the real-world timing of these implementations. Specific instructions on how to run the profiler and interpret the results are on the class web site. When the run completes, make sure you take a snapshot of the results, and then do a "Save Snapshot to Project" to store the results (do this before you commit your files and push to upstream!).

Hint: Note that some of the internal methods from the provided `RedBlackTree` class can be used either directly or with small changes (especially the various rotation methods). You can cut and paste these into the `AltRedBlackTree` class as a starting point. The biggest change you'll need to make to the rotation functions is to make sure parent pointers are modified correctly.

Submission Instructions: Using NetBeans, commit all changes to your project and do a "push to upstream" to put the most up-to-date files on the Bitbucket server. Remember: Do *not* create a pull request — I will clone your repository (if it exists and you granted me access) at noon on the due date, and will assume that is your submission. If you intend to keep working on your project and submit late, please let me know by email, and I will ignore your repository until the late submission deadline.