CSE 2320 Lab Assignment 3

Due April 15, 2014

Goals:

1. Understanding of red-black trees.
2. Understanding of recursive binary tree processing.
3. Understanding of subtree sizes in binary search trees for supporting ranking queries.

Requirements:

1. Modify the provided C code for maintaining a red-black tree to process a sequence of commands (standard input) from the following list:

   `0` - Exit the program

   `1` *x* - Insert positive integer key *x*, unless *x* is already present in the tree. (Error message if *x* is a duplicate.) Besides inserting the key, *odd subtree sizes* must be updated. The odd subtree size of a node is the number of odd keys in the subtree rooted by the node.

   `2` *x* - Find the *odd rank* of *x*, i.e. the number of odd keys in the tree that are smaller than *x* (error message if *x* is not in the tree). The numbers to the left of each node in the diagram below are the odd ranks.

   `3` *k* - Find the smallest and largest keys with odd rank *k* (error message if there is no key with odd rank *k*).

   `4` - Perform an audit on the odd subtree size at each node to give a final indication that the tree is "clean" or "corrupt".

   Each command must be echoed to standard output. Commands 1, 2, and 3 must be processed in $\Theta(\log n)$ time.

2. Email your program source files (e.g. `topdownRB.h`, `topdownRB.c`, `topdownRB.test.c`) to `adnan.khan@mavs.uta.edu` by 3:15 pm on April 15.

Getting Started:

1. Commands `2 and 3` ***should not*** be based on directly applying tree traversal.

2. Command `4` should traverse the tree, compute the odd size of each subtree, and verify the stored odd sizes. You may leave the code alone for checking 1) the inorder key property, 2) that there is no downward path with consecutive red nodes, and 3) that the black-heights are consistent.