CSE 3318-003 Lab Assignment 4

Due April 16

Goals:

1. Understanding of binary search trees.
2. Understanding of recursive binary tree processing.
3. Understanding of subtree *root ranks* in binary search trees for supporting ranking queries.

Requirements:

1. Modify the provided C code for maintaining a binary search 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. Besides inserting the key, subtree root ranks must be updated. (Error message if *x* is a duplicate.)

   `2` *x* - Find the rank of *x*, i.e. the number of keys in the tree that are smaller or equal to *x* (error message if *x* is not in the tree).

   `3` *k* - Find the key with rank *k* (error message if *k* is not between 1 and *n*, inclusive).

   `4` – Print the tree. For each node, except the sentinel, provide the key and the root rank

   `5` - Perform an audit on the subtree rank at each node. Either give a final indication that the tree is "clean" or `exit(0)`.

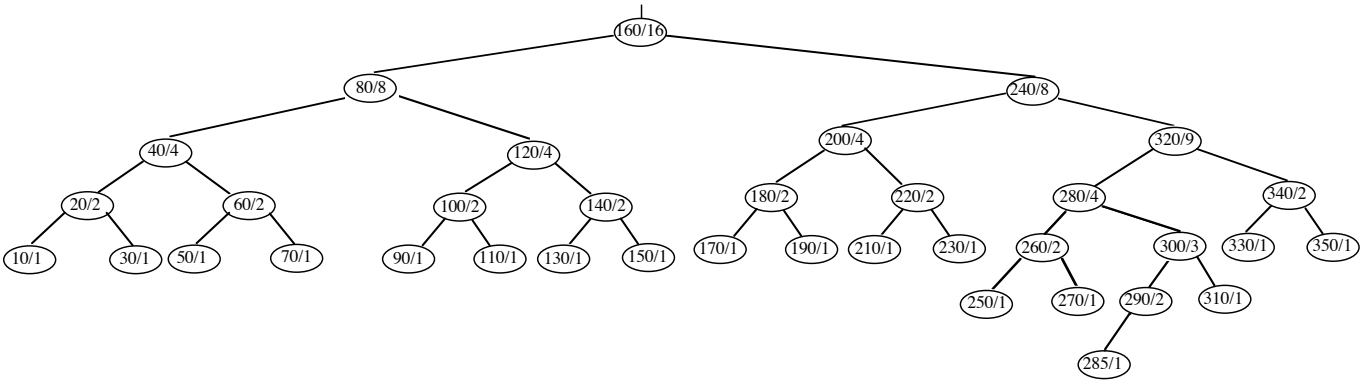
   Each command must be echoed to standard output. Commands 1, 2, and 3 must be processed in time proportional to the height of the tree.

2. Submit your program source files as a zipped file on Canvas by 3:45 p.m. on Wednesday, April 16. One of the comment lines should include the compilation command used on OMEGA.

Getting Started:

1. The code changes for maintaining subtree root ranks during command `1` are minor.
2. The code for commands `2` and `3` may be coded as variations on recursive search of a binary search tree, but recursion is not required.
3. Command `5` should traverse the tree, compute the rank of the root of each subtree, and verify the stored ranks. Optionally, you could also check the inorder key property.
4. You are not required to maintain any satellite data related to the integer keys.
5. You may use the driver in the lab directory.

6. An example of a binary search tree with subtree root ranks:



7. The same tree as 6., but using subtree sizes: