COMP 3160 – Fall 2014
Homework 2: Binary Search Trees

Number of People: Individual. Feel free to ask me for help, or visit the Computer Science Learning Center (www.cs.memphis.edu/cslc).

Due: Thurs., Sept. 18 by 1:00 pm

Submission: Zip all of your Java source files (you can zip the entire project folder if using an IDE) into a single file and upload it to the proper folder in the eCourseware dropbox at https://elearn.memphis.edu. The non-coding portions can be submitted within your zip file, or as a hard copy at the beginning of class.

Coding Style: Use consistent indentation. Use standard Java naming conventions for **variableAndMethodNames**, **ClassNames**, **CONSTANT_NAMES**. Include a reasonable amount of comments.

Grader: TA, Kyle Cherry (kcherry2@memphis.edu). Questions about grading? Please contact him first!

1. **(4 pts)** Draw the binary search tree that results from adding the following elements in the specified order. Assume the tree is initially empty.

   8, 5, 6, 2, 7, 10, 9, 12

2. **(5 pts)** In your BST from #1, indicate the order in which the nodes are visited for a(n):

   a. *(1 pt)* In-order traversal
   b. *(2 pts)* Post-order traversal
   c. *(2 pts)* Pre-order traversal

3. **(4 pts)** Draw what your BST from #1 looks like after each of the following deletions (assume they are performed sequentially):

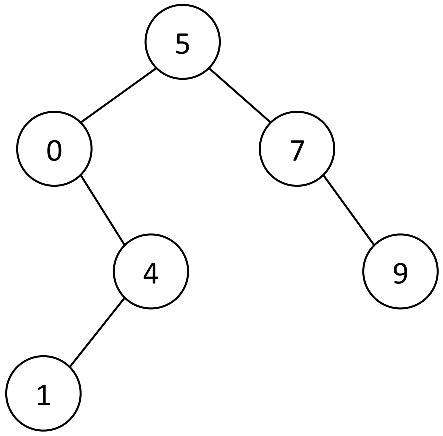
   a. *(1 pt)* Delete the 12
   b. *(1 pt)* Delete the 6
   c. *(2 pts)* Delete the 8

4. **(12 pts)** For these coding problems, use the **BinarySearchTree** class that we wrote in lecture as a starting point. Non-honors students: pick any 2 *(6 pts each)*. Honors students: do all 3 *(4 pts each)*.

   a. Write a non-recursive **addIterative(E newItem)** method. In other words, do not call this method within its own definition! (Hint: create a temp reference to the root node and move it down the tree as you search for the correct add position.)

   b. Write a non-recursive **findIterative(E someItem)** method. This should behave the same way as the **find** method we wrote in class – it should return the item from the tree if found, or **null** if not found.

   c. In class we discussed pre-order, in-order, and post-order traversals. Another common type of tree traversal is a *level-order traversal*. This involves visiting the nodes of the tree left-to-right, top-to-bottom (just like the way you normally read a page of text).



For example, in the BST above a level-order traversal would visit the nodes in this order: **5 0 7 4 9 1**

Write a **levelOrderTraverse** method. The method should print the nodes in the order that they are visited. (Hint: one non-recursive algorithm for this is to use a queue to keep track of the nodes that you want to visit. Start by enqueuing the root node, and see what needs to be done from there.)