**LAB ASSIGNMENT 5**

**Due: Tuesday, May 8th by the beginning of class**

Submit each .java file electronically through Canvas by the due date.

Submit a printout of your source code to the instructor by the due date.

**1)** **Revise the BSTInterface and the BinarySearchTree class to include the following methods**: **(72 points @ 12 points each)**

1. A public method that returns a count of the number of nodes in the tree that contain a value less than or equal to the argument value. The header is: **public int countLess(T maxValue)**.You can either use a private helper method that is recursive or write it using iteration.
2. A public method that returns a count of the number of nodes in the tree that contain a value greater than or equal to the argument value. The header is: public **int countGreater(T minValue)**.You can either use a private helper method that is recursive or write it using iteration.
3. Add a public method *height* that returns the height of the tree. The header is: **public int height().** Use a private helper method **that is recursive**.
4. Implement the **Balance** and **InsertTree** methods from the pseudocode given in section 7.8 on pages 469 thru 471 in the text. This is where you are balancing your BST Tree.
5. Implement and add a **Breadth-First Traversal** (Level Order Tree Traversal) from the pseudocode given in chapter 7 on pages 426 - 427.
6. Implement and add a **fRatio** method. The method determines the fullness ratio of your binary search tree. It is the ratio between the trees minimum height (based on the number of nodes in the tree and the trees actual height). For example, if a trees minimum height is 3 and its actual height is 3 then its fullness ratio is 1.00 and it is relatively balanced. However, if a trees minimum height is 3 and its actual height is 9 then its fullness ratio is 0.33 and the tree is unbalanced.

**2) Create a BST test driver program (use a menu) to: (28 points)**

1. **Create and add entries to the tree** using the BST classes in your textbook.
2. **Thoroughly test the added functionality by**:
   1. Displaying the count of all the values less than or equal to an argument value.
   2. Displaying the count of all the values greater than or equal to an argument value,
   3. Displaying the height of your binary search tree.
   4. Printing the binary search tree using the breadth-first order traversal.
   5. Calling the fRatio method and balancing the tree if the ratio is 0.50 or below. Printing the tree again in breadth-first order traversal after you balance.
3. You will need to come up with relevant test cases to test that all your added functionality works properly. This entails adding entries to form a balanced tree and testing the added methods. This also entails adding and/or removing entries to form a skewed tree with a fRatio at 0.50 or below that requires the tree to be balanced.