**Laboratory 8: Class Invariants (Assert Statements)**

Video: [https://media.tntech.edu/media/Lab+8/1\_s4gvy3io](https://media.tntech.edu/media/Lab+8/1_s4gvy3io%20)

Using the BinarySearchTree implementation from the previous lab, you will investigate class invariants. **A class invariant** is an assertion that captures the properties and relationships, which remain stable throughout the life-time of instances of the class. You will **assert** Boolean methods, and if they return false, an exception is automatically thrown.

**Lab:**

* Class Invariants/Assert
  + Binary Search Tree Property
  + Number of Items Property

**Part I: Class Invariants and Assert**

Add public and protected methods to BinarySearchTree to check the class invariants of the binary search tree. **Use the assert statement in appropriate places in BinarySearchTree (insert & delete) to call these methods.** Use good error messages for assertion exceptions. In particular, indicate the item or searchkey that was being processed when the assertion exception occurred:

**Ex: assert validateBSTProperty() : "not a binary search tree after insertion of key: " + item.getKey();**

* boolean validateBSTProperty() // use an iterator and setInorder() to step through the tree, cast items pulled from the tree to KeyedItems and then to convert to type Comparable (via the getKey() method), and finally compare the items using compareTo(). Each subsequent item should be greater in value than the previous item to satisfy the BST property.
* boolean validateSize() //add a size instance variable to BST and update it appropriately (look at insert and delete methods) then compare this size to a method that manually computes the size of the BST using an iterator.
* boolean isBalanced() //recursive and requires a method to compute the height of a subtree, see below

Note: Some of these need convenience methods as well, where you will pass in the root node by default

To determine whether the BST is balanced, the left height and the right height cannot be different by more than one. Also, every subtree in the BST must be balanced for the entire BST to be balanced.

**Part II: Testing with BSTDriver**

Complete BSTDriver to thoroughly test all of the public methods in SearchTreeInterface. **You may use my provided BSTTest as a guide, but copying it completely will result in a point deduction.** That is, add, remove, and retrieve cds in an arbitrary fashion**. Also, don't simply add them all in and then remove them all, although doing this to get your testing started is fine**. Use the -ea option in the command line to enable assertions when your program runs.

**To compile & run (note: use ‘:’ instead of ‘;’ for OSX and Linux):**

javac -cp .;queue.jar \*.java

java -ea -cp .;queue.jar BSTDriver

**Labs will open for everyone by Monday morning (most likely earlier though) and due the following Sunday by 11:59 PM. Lab partners should be in the same lab section (it’s much easier to grade this way 😊). Only one submission per team is necessary, but please make sure to include both names at the top of your source code, as well as in the comments section when submitting the lab, so both people can get credit.**