April 18, 2016

Computer Science 182
Programming Assignments #5

Due: May 2, 2016

You are to complete the classes below so that all of the methods do what they are supposed to do in the way they are supposed to be done. As always, leave all variable and method names exactly as they appear and keep in mind that the specification, required algorithms, etc. may change according to lab discussions. Leave both toString methods as I have them. You can, of course, write your own output/toString methods for debugging purposes. Also, notice that there is no setNum setter. This is an exercise in manipulating pointers, not moving data. For the same reason, creating unnecessary intNodes's is not allowed.

The quality of your algorithms will be an important part of your grade and they should be recursive whenever it makes sense. They should be simple and easy to understand. Comments in your recursive methods should almost never contain phrases like “keep going” or “until we get to” or “while”, etc.

You are responsible for making sure your program adheres to all the most recent requirements. I will need a printout of your program listing, a printout of the output produced by my *final* test program, and an email with an attached file named prog5.java with a subject of “Noël Coward – prog 5” if your name is Noël Coward.

The first test program will appear soon. **My test programs do not provide a good way to debug your program.** It only helps you decide whether or not your program is working correctly. If after running my test program you find errors then use your own test program to home in on where the error is. For example, if delete is not working first see if it correctly deletes leaves. Then try the other easy cases. Then try some nodes close to a leaf. Then try the root. You get the idea. Be systematic with your own test program. You will be much more productive that way.

```
class IntNode {

    private int num;
    private IntNode left, right;

    public IntNode(int num) {

    public IntNode(int num, IntNode lf, IntNode rt) {

    public int getNum() {

    public void setLeft(IntNode pt) {

    public void setRight(IntNode pt) {

    public IntNode getLeft() {

    public IntNode getRight() {
}

class IntBST {

    // member variable pointing to the root of the BST
    private IntNode root;

    // default constructor
    public IntBST() {

    // copy constructor
    public IntBST(IntBST t) {

    // for output purposes -- override Object version
    public String toString() {
        return toString(root, 1);
    }
```

```
private static String toString(IntNode l, int depth) {
    String s = "";
    if (l == null)
        ; // nothing to output
    else {
        if (l.getLeft() != null) // don't output empty subtrees
            s = '(' + toString(l.getLeft(), depth + 1) + ')';
        s = s + l.getNum() + "-" + depth;
        if (l.getRight() != null) // don't output empty subtrees
            s = s + '(' + toString(l.getRight(), depth + 1) + ')';
    }
    return s;
}

// Is num in this tree
public boolean search(int num) {

// Insert num into this tree
public void insert(int num) {

// Returns the height of the tree
public int height() {

// The depth of the leaf closest to the root
public int closeLeaf() {

// Still think about what this should be. The name will change.
public int doSomething() {

// Replace with the “inorder successor”, i.e., the smallest in the right subtree
public void delete(int num) {

// Rotate the node containing num to the left -- do nothing if not possible
// or num is not in the tree
public void rotateLeft(int num) {

// Rotate the node containing num to the right -- do nothing if not possible
// or num is not in the tree
public void rotateRight(int num) {

// remove the largest leaf
public void removeBigLeaf() {

public static String myName() {
```