Data Structures II - HW3

Instructions:

1. Download the hw3-files.zip file from the COL submission folder for HW3. Similar to how you did it for hw2, unzip the file into the src folder in the workspace you created for this class. If you are unsure where that is, inside Eclipse, right click the src folder and select properties (at the bottom).

2. Start up Eclipse. You will probably need to refresh the project before you see the hw3 folder. (Right click on the src folder in the Project Explorer and select refresh.)

3. Open the hw3 folder inside Eclipse. You will find 2 files. The only file you will be modifying is IntTree.java. Implement each of the functions labeled with the comment "// TODO". You must use recursion by calling a helper function that takes an additional argument of type Node. Make sure to read all the comments in the source file for extra instructions and hints. In particular:
    - You are not allowed to use any kind of loop in your solutions.
    - You may not modify the Node class in any way
    - You may not modify the function headers of any of the functions already present in the file.
    - You may not add any fields to the IntTree class.
    - You may not change or remove the line that reads "package hw3;"

4. You should write your own small tests, especially if you are having difficulty understanding my tests. To help you in writing your tests, I added two methods in the IntTree class: **`fromString`** and **`setElementsUsingArray`** that are described in the Testing section of this writeup.

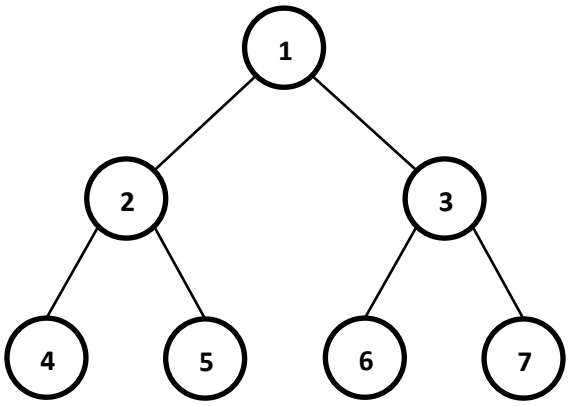
Submission and Grading:

Your submission consists of a single file (IntTree.java) and in the comment box, and a screenshot of what happens when your run HW3Test. Your grade will be the sum of the points of all tests that pass but 10 points will be deducted if you fail to include the required screenshot.

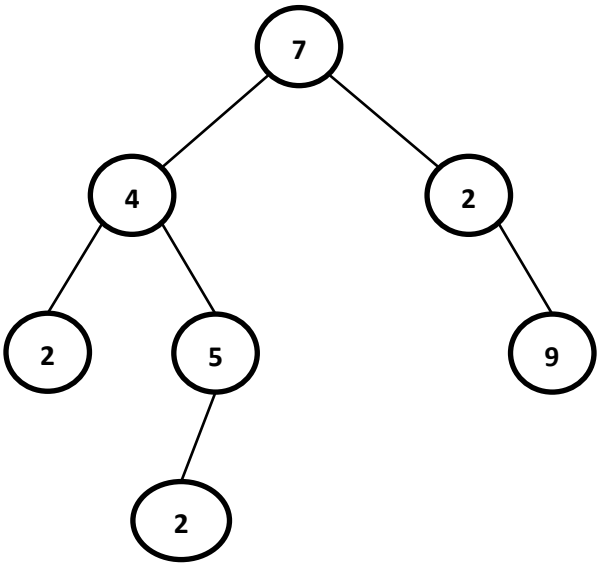
Testing

To help you with testing, I've created a function called **setElementsUsingArray**. Call it on an IntTree object with an array of ints and it will replace the contents of the tree so the contents of the tree match the contents of the array as described below.

First, imagine numbering the positions in a full binary tree starting with 1 at the root and proceeding in level order. A full binary tree is a binary tree in which all nodes have 2 children, except for the leaves (which have 0 children). Below is a full binary tree of height 2 numbered in this way.



Now imagine the tree you want to build. Suppose it looks like:



You need to create an array that has the values you want in the indices corresponding to the tree positions in the first picture picture. For example, in the second picture, the 9 is in position 7. So in the array, slot 7 should have a 9. We don't have a number in position 6, so in the array, slot 6 should have a 0. (This means we cannot have 0s in our tree when using this method.) To create the tree in the second picture, we could use:

```
int[] a = {0, 7, 4, 2, 2, 5, 0, 9, 0, 0, 2};
IntTree t = new IntTree();
t. setElementsUsingArray(a);
```

You will not use the setElementsUsingArray method in your solution. It is only there to help you make trees for testing purposes. You can create a small test in main using this method.

There is another way to create a tree for testing. You can see it at the end of the main method in IntTree.java. You use a string to list out the values and positions of the various keys. They are inserted in the order they appear in the String, so you need to make sure parents are inserted before children. To create the same tree as before using this method, you could use:

```
IntTree s2 = IntTree.fromString("7: 4:l 2:r 2:ll 5:lr 9:rr 2:lrl");
```

This method has the advantage of not having to list nodes that aren't present, but it has the drawback that you must list the path to each node. Again, this method is provided solely to help you write tests. You will not be using this method in the solution.