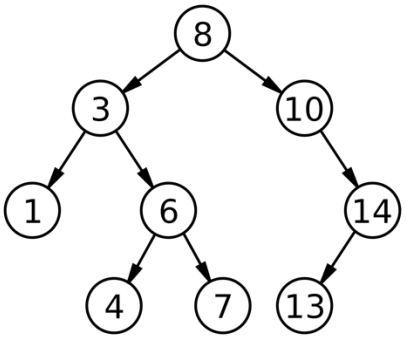
Project: Examining Trees One by One

In this project you will work in a group of 2 or 3* to implement a BST and an AVL tree.

**If you choose to work alone, you must submit your solution to #1-4 earlier, by March 25th (the rest of the assignment can be submitted on the final due date. If you choose to work in a group of more than 3, you must email me by March 15, your group will be assigned more work to account for more members. If you work in a group of 2-3, the above doesn't apply.*

1. Read the entire assignment before beginning. Ask questions on canvas, email, and in class if you are stuck!
2. In all of the classes and methods you will implement, the value type stored in the binary node will be an int. Assume unique values so no need to handle duplicates. All work must be done in Java. You cannot use any Java standard libraries, unless the problem states you're allowed. You can use the book's code as a resource (http://users.cis.fiu.edu/~weiss/dsaajava3/code/). The slides use this code as well.
3. Write a `BST class: insert, remove, contains` as we discussed in class slides: TreesLecture.pdf.
4. Write an `AVL class: insert and contains` as we discussed in the class slide: AVLTreesLecture.pdf. For this method, you may need to write helper methods like `balance, getHeight`. But we will only be grading for a correct `insert and contains`.
5. In your BST and AVL class, write a method which will print to console the elements of the tree level by level. For example,



Your print out would look like the following:

Level 1: 8

Level 2: 3, 10

Level 3: 1, 6, 14

Level 4: 4, 7, 13

6. What is the runtime and space complexity of the method in the previous problem?
7. In your BST and AVL tree class, write the method `findMax` as we discussed in class slides, TreesLecture.pdf.
8. What is the runtime and space complexity of the `findMax` method?
9. In this question you will empirically examine runtimes of your AVL and BST.

    a. Use a random number generator (from a Java standard library) to insert nodes into your AVL tree and BST tree until you can get a height of 3, 4, and 5. For each height and tree (3, 4, and 5) use your method in number 5 to print the tree you created.
    b. Utilize Java system timers or other libraries which have timing capability to time how long in milliseconds contains takes to complete on the 3, 4, and 5 height AVL and BST trees. Call the method which runs this test: "runtimeTester." You can choose any parameter or return value as long as your code shows timing is being determined for the different trees.
    c. Using any document editor, create a table to record your findings for the above for the different height AVL and BST trees.

d. Write a short conclusion describing your findings (30 words max).

Turn in instructions:

Create a zip file with the following documents: All your code used, a ReadMe.txt on specific instructions if needed to run your code, a Answers.doc for your solutions to any written responses needed.