1. Several of the questions in this assignment rely on code in the Sample Code project on the SVN server. Update your local copy of the Sample Code project. There is nothing to turn in for this question.

2. Show the binary search tree (BST) that would be constructed if the following elements were added in order: {12, 16, 17, 18, 8, 4, 6, 13, 2}

3. Consider the BST shown to the right:

Show the BST after each of the following operations are performed. You should assume that the BST is reset to the above configuration before each operation.

   a. remove(15)
   b. remove(11)
   c. remove(5)
   d. remove(10)

4. The contains(E element) method and the descendSubTree method in the LinkedBinarySearchTree class contain very similar functionality. Rewrite the contains(E element) method using the descendSubTree method. JUnit tests have been provided for the contains method and these tests should pass both before and after you rewrite it.

5. Write the remove method in the LinkedBinarySearchTree class. Note that JUnit tests have been provided for the remove method.

6. Give asymptotic bounds (Ω and O, or Θ, whichever is appropriate) for successful calls to the add and remove methods in the LinkedBinarySearchTree class. By successful, I mean ignore any situations in which exceptions would be thrown. Also, for add, give separate bounds for an empty and non-empty tree. Briefly justify your bounds.

7. Adding and removing nodes from an AVL tree can result in 4 different types of unbalanced trees. The class slides illustrated each of these (LL, RR, LR, RL). It also showed in detail how to rebalance three of the cases through either a single rotation (for LL and RR) or through a double rotation (left and then right for LR). The RL case also requires a double rotation, first a right rotation and then a left rotation. Draw a picture showing the rotations required to rebalance the RL case. Your picture should be similar to the one given for the LR case in the course slides.

8. Consider the AVL tree shown to the right:

Show the above AVL tree after each of the following operations are performed. You should assume that the AVL tree is reset to the above configuration before each operation.

   a. add(1)
   b. add(8)
   c. add(31)
   d. add(40)

9. Implement the `rotateLeft` method in the `LinkedAVLTree` class. Note that JUnit tests have been provided that test this method.