(Create an Eclipse Project and starter code)

1. Create an eclipse project, called BSTQuery or some other name of your choice.
2. Right click the project name, and click on Properties at the bottom of the pop-up menu. Then click on java build path, add algs4.jar as an external jar and junit as a library.
3. Create a new classed called BasicBST.java.
4. In Eclipse Projector Explorer, find algs4.jar under Referenced Libraries, click on the jar file icon to expand it. Then click on the edu.princeton.cs.algs folder (package) and expand it.
5. Find BST.class and click on it. You should see the source code in the Eclipse editor.
6. Select all code in BST.java, copy and paste it to BasicBST.java.

(Edit BasicBST.java)

7. Use ctrl-F (or the edit menu, click find/replace) to replace all BST with BasicBST, and save BasicBST.
8. Comment out or delete the package statement at the top, “package edu.princeton.cs.algs4”.
9. You will get some error messages, since BasicBST uses classes in algs4.jar but does not import them yet. If you do have errors, you can open the Eclipse Window menu, click “Show View”, and choose “Problems”. The problems view will show the compilation errors. You can use the information there to figure out what algs4 classes to import. Then add import statements for these classes, and your BasicBST.java should pass compilation.

Please note that you should NOT use import edu.princeton.cs.algs4.* to import all the classes in algs4.jar. This approach may feel convenient to you but is considered bad practice by the industry standard. It can lead to unintended, undesirable consequences. For example, if a student program imports both edu.princeton.cs.algs4.* and java.util.*, and both libraries have a Queue class: same name, but different APIs. Which Queue class will be used?

10. In your BasicBST.java, replace the private keyword for the root instance variable, the Node class and its instance variables (key, val, left, right, size) to protected. This will allow these variables to be accessible in the sub(child) classes of BasicBST.

(Develop your own BSTExercise.java)

11. create a new java class, BSTExercise, and let it extend BasicBST. In other words, BSTExercise is a child class of BasicBST. Use the following API for your code. This exercise is not going to be graded or tested by us. So you can add more methods if you want. Please work on at least the following methods, and you need to submit your code by next Sunday.

```java
public class BSTExercise<Key extends Comparable<Key>, Value>
    extends BasicBST<Key, Value> {

    /**
     * returns the depth of a specified key.
     *
     * If the tree is empty or the key is not in the tree, return -1.
     *
     */
    public int depth(Key key)
```
/**
 * Returns all keys of the leaf nodes in the tree, in the sorted order.
 * @return an iterable of all leaf keys in this tree in the
 * sorted order. Return null if the tree has no leaf.
 */
public Iterable<Key> leaves()

/**
 * print this tree in the indented form. More information below.
 */
public void indentedPrint()

/**
 * builds an (intuitively) balanced BST based on a sorted Key array.
 * For simplicity and clarity, use null (or some other value) as
 * the value for all keys. (Inspired by exercise 3.2.25)
 * @param a  Key[], sorted data used to build a balanced BST.
 *          precondition: a does not have any duplicates
 */
public void balancedBuild(Key[] a)

We also encourage you to write test code for your class. You can have some testing code in the main method of BSTExercise. But it'd be better to have a separate Junit test class for systematic testing.

Finally, the indented print, the main idea is to use indentation to reflect the tree structure. For the following tree, the indented print is shown in the textbox on the right.