You may work in pairs for this assignment. If you choose to work with a partner, make sure only one of you submits a solution, and you paste a copy of the Partners Template that contains the names and PIDs of both students at the beginning of each of the files you submit.

You will submit your solution to this assignment to the Curator System (as `HW01`). Your submission will consist of two files, described below. Place the two files in a flat zip file (not a jar file, not a tar file, not a gzip'd tar file, or any other type), and submit that. See the instructions on the Resources page for how to create an acceptable zip file. Files that do not satisfy the requirements will not be graded.

Part I BST Algorithms

For questions 1 and 2, refer to the BST generic interface provided for Project 2. You may assume you have a complete, working implementation of all the methods shown in that interface, and you may make use of any of them (or their implied private helper methods) that you like. However, you may not assume any modifications to the interfaces of those methods.

Download the posted zip file and unpack it. See the Testing section at the end of this specification for further instructions.

Strive for the most efficient solution you can devise. For example, if it's possible to report the correct answer without performing a full traversal of the tree, then that's what your solution should do. You may not use any Java collections, including ones you implement yourself, to store any additional information (like copying the data elements from the BST into an array).

You may, and should, write private (or package-access) helper functions. Just be sure that your solutions do not depend on any code other than that in your version of `BSTUtils.java`, Java library classes, and the BST generic class.

Also: in addition to showing a Java representation of your solution to each of questions 1 and 2, you must write a short paragraph that provides a clear, complete explanation of the logic used in your algorithm. Our evaluation of your explanation will count 40% of the score for the question.

1. **[30 points]** Design an algorithm to determine whether a given binary tree has the BST property. (For this question we will assume the implementation of the BST does allow the insertion of duplicate values.) Express your solution using Java code as if it were to be implemented as a public member function (with private helper) of a class belonging to the same package as the BST generic specified in Project 2; your answer (which will include both the public and private functions) must conform to the public interface shown below:

```
// Pre:      tree is a structurally-valid binary tree object;
//             i.e., the nodes are correctly linked
// Post:     the tree is unchanged
// Returns:  true iff the tree satisfies the BST property
//
// Logic:    DESCRIBE THE LOGIC OF YOUR ALGORITHM HERE
//
public static boolean hasBSTProperty( BST<Integer> tree ) {

   // up to you. . .
}
```

Since the function is in the same package as the BST generic, it may access the internal members of the BST object `tree`. You may use member functions of the BST generic, but none of them are likely to be useful.

2. [30 points] Design an algorithm to find, within a given BST, the least upper bound (LUB) of a given value, according to the `compareTo()` method for the stored data type. Express your solution using Java code as if it were to be implemented as a public member function (with private helper) of a class belonging to the same package as the BST generic specified in Project 2; your answer (which will include both the public and private functions) must conform to the public interface shown below:

```
// Pre:      tree is a valid BST<Integer> object
// Post:     the BST is unchanged
// Returns:  the smallest value in the tree that is larger than floor;
//           null if there is no such value
//
// Logic:    DESCRIBE THE LOGIC OF YOUR ALGORITHM HERE
//
public static Integer LUB( Integer floor, BST<Integer> tree ) {

   // up to you. . .
}
```

You may use any of the public functions defined in Project 2 for the BST generic. And, since the function is in the same package as the BST generic, it may access the internal members of the BST object `tree`.

Note: in grading each question in Part I, we will allocate 12 points for the quality and correctness of the description of your logic, 12 points for the correctness of your solution in testing, and 6 points for the efficiency of your solution.

Part II BST Theorems

Prepare your answers to the following questions, either in a plain text file or a typed MS Word document. Solutions submitted in other formats will be discarded. Partial credit will only be given if you show relevant work.

3. [20 points] Use Mathematical Induction to prove the following theorem:

Suppose that T is a nonempty, full binary tree with N nodes and L leaves. Then $N = 2L - 1$.

You may not use any clause of the Full Binary Tree Theorem in your proof, even if you prove that clause in your answer. Hint: use induction on the number of leaf nodes in the tree.

4. [20 points] Suppose that T is a BST, containing two values X and Y, such that Y is in the left child of X and the left child of X is a leaf.

Prove that the tree cannot also contain another value, Z, such that Y < Z < X.

Hint: if there were such a value Z, then Z was either inserted before both X and Y, after both X and Y, or after X but before Y. QTP: why must X have been inserted before Y?

Testing

The posted zip file includes testing code for your solutions to questions 1 and 2 in Part I. You will find the following files:

`testDriver.java`	test manager
`Summer2017/HW1/DS/BSTUtils.java`	shell for implementing solutions
`Summer2017/HW1/DS/UtilsTests.class`	64-bit Java test harness
`Summer2017/HW1/DS/BST.class`	64-bit Java implementation of the BST as in J2
`Summer2017/HW1/DS/BST$BinaryNode.class`	

Edit `BSTUtils.java`, and complete the implementations of the two Java functions described in questions 1 and 2. Note that these are not member functions, but rather functions belonging to a class in the same package as the BST implementation. Therefore, these functions can access any members of the BST class that were declared with package protection.

Compile the code by executing the command "`javac testDriver.java`" in the directory where that file resides. That will also compile your BSTUtils.java file in the directory `./Summer2017/HW1/DS/`.

Note: I'm describing this using Linux directory notation, but the same commands work in the Windows shell as long as the JDK is properly installed.

The command "`java testDriver`" will run the testing code and create a log file for the testing details for each of the specified functions, and a summary file showing how many tests were passed for each function. Once you've run the testing code once, running it with the switch `-repeat` will reuse the same test data as on the previous run; that's useful in debugging. Do not assume that your solutions are correct merely because they pass a single run of the testing code (or even several).