Project 3 provides experience with binary search tree (BST) operations and experimental analysis of algorithms. You will design, implement, and verify a program that experimentally measures the structure of a BST<E> after insertions and deletions. The experiment has the following 3 experimental conditions: 3 operations (create, delete, reinsert) on a random BST. The create and reinsert operations use BST<E> insert(BSTNode<E>) method. The experiment has 6 dependent variables (max and average tree levels for the three experimental condition). You will use 6 instances of your Sample class to hold and display the results of the experiment's trials (Sample createMax, createAverage, deleteMax, deleteAverage, reinsertMax, and reinsertAverage).

	Operations		
	create (2^n - 1 nodes)	**delete (2^{n-1} - 1 nodes)**	**re-insert (2^{n-1} - 1 nodes)**
dependent variables	average and max level	average and max level	average and max level

You will design and develop classes for a generic binary search tree (BST<E>) and use your Sample class (from P1) for statistics. You will use a class named "BSTElement<T>" that contains Integer "key" values and implements Comparable. Your BST implementation must use reference variables to reference the left and right subtrees. You cannot implement the BST in an array or an ArrayList. **Your BST insert and delete methods should be recursive.** You must design and develop your own classes for BSTP3 (the application or program), BSTree<E>, BSTNode<E>, BSTElement<T>, and Sample. You cannot use Java's TreeMap (for the BST).

The three experiment operations will be done with the help of an ArrayList<Integer> key. Your application class (say BSTP3) will have, create, and use key to run the simulation experiment. The array holds the Integers you are going to use to instantiate the BSTElement<T> objects that you insert and delete into your BST.

In the experiment there will be 1,000 trials. Each trial will run three experimental conditions in the order: create, delete, re-insert. In each trial, first a random BST is created with 2^n - 1 nodes (the Integer "keys" in ArrayList<Integer> key). Second, nodes with key values equal to the first 2^{n-1} - 1 key values are deleted from the tree (half the tree). Third, the same 2^{n-1} -1 nodes that were deleted are re-inserted into the tree. For each of the 3 experimental operations the level of all the nodes in the resulting tree will be added to three temporary (local) Samples (createTrial, deleteTrial, and reinsertTrial). Your Sample class will need double getMax() and double getAverage() methods. Those methods will be called after computing the stats on your 3 temporary Samples. The max and Average values will be added to the respective 6 Sample instance variables of BSTP3. Again, the 6 instance Samples are: createMax, createAverage, deleteMax, deleteAverage, reinsertMax, and reinsertAverage. After the 1,000 trials the statistics for the six Sample instances will be computed and displayed.

Assume your BSTP3 application's constructor has an int argument "power". This is the power of 2 used to determine the number of Integers to put into ArrayList<Integer> key. For power = = 3, n == 7 or $2^{power} - 1$. For each simulation trial the ArrayList<Integer> key should be randomized to a

different order. Create and initialize the ArrayList<Integer> key once, but shuffle it 1,000 times to randomize key for each trial. Look at example ShuffleAssertion.java posted with this assignment.

```
int n = (int) Math.pow(2, power) - 1;
for (int i = 0; i < n; i++) key.add(i);        // initialize key
Collections.shuffle(key);                      // randomize key
```

Your BSTree must have a recursive boolean verify() and a void nodeLevel(Sample s) method. Method verify() uses Java's assert to test if the tree is a valid or not a valid tree. You must run your program with the java " – ea " runtime option. Method nodeLevel(Sample s) visits every node in the tree and adds that node's height to Sample s. The root of the tree is at level 1. A tree with level = = 1, has $2^{level} - 1$, or 1 node (the root).

A possible 3 phase implementation plan for your consideration. This "3 phase plan" is presented to support the idea "solve a smaller problem first" approach. Its use is optional. You can develop and use a different design / implement / test plan.

Phase 1. Design, draw UML, and validate classes: BSTP3, BSTree<E>, BSTNode<E>, BSTElement<T>, and Sample. Validation can be done by reviewing the UML. Are the necessary instance constants, instance variables, and methods specified for the classes? Next, generate the pseudo-code for the experiment. Can you walk-through the application's pseudo-code by assuming the BSTree<E>, BSTNode<E>, BSTElement<T>, and Sample methods are correct and "call them" in the correct order? Last, generate the pseudo-code for the other classes and validate with a walk-through.

Phase 2. Using phase 1, implement and verify your classes for small numbers of nodes. Test by using "power" values of 3 or 4. Can you create, delete, and re-insert into your BST? After each update to the tree (e.g. insert) is verify() true? After all the operations (e.g., all inserts in create) do the three Sample instances provide the correct min, max, and Average for the tree's node levels? Phase 2 is a single simulation trial using the local (temporary) Samples (createTrial, deleteTrial, and reinsertTrial).

Phase 3. In this phase you can conduct your experiment -- run the program to generate your experimental results with a power value of 10. You will have 1,000 trials each with a different random order of key. You will need use the 6 instances of Sample for your statistics. When you have your results you should write a brief report (with table of data) presenting and discussing your dependent variables. You must compare your experimental results with the expected results of a balanced BST with the same number of nodes.

Manage your time well, this project requires design, implementation, testing, and analysis on your part. The lab is a good time to work on the design and implementation of the project. This is the time for one-on-one question/answering with me, or your group. Remember to learn what everyone in the group knows, this material is also on the exams.

… "Plan well, test sufficiently, and prosper."