**NAME : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ NETID : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CS2110 In-Class Quiz 3 (Fall 2010)**

**Problem 1: Ten T/F questions:**

|  |  |  |
| --- | --- | --- |
| **T** | **F** | **Suppose that Sally has a proof that some algorithm runs in time O(*n3* *log n*) but the Victor has a proof that the algorithm runs in time O(*2n*). One of their proofs must have an error.** |
| **T** | **F** | **“This statement is false.”** |
| **T** | **F** | **A program calls buildTree and treeAnalysis on *n* objects. buildTree performs *1.5n2* operations, while treeAnalysis performs *nlog(n)+sqrt(n)* operations. If these are the only parts of the program that we’re interested in, the overall complexity is O(*n2*).** |
| **T** | **F** | **If some parameter is declared to be of type ArrayList<? super Frog>, then the corresponding arguments must be ArrayLists with an element type T such that Frog is a subtype of T.** |
| **T** | **F** | **The “observer” pattern lets you attach a list of methods to an object. Each time the object is changed, the attached methods will be invoked one by one.**  |
| **T** | **F** | **No matter how you pick the pivot, there is some risk that the Quicksort algorithm will give performance O(*n2*) for some kinds of inputs.** |
| **T** | **F** | **We talk about abstract data types (ADTs) because sometimes we want to express more about a kind of object than just the methods for accessing it. For example, the performance might be an important property of an ADT, or even some aspect of the implementation. In contrast a Java interface only lets us specify the type signatures of methods, and a Java class lets us provide code and also specify fields, but we still can’t talk about big-O complexity or algorithms used.**  |
| **T** | **F** | **Suppose that Animal X = new Animal(“Cat”). Now we execute Animal Y = X. At the end of this sequence there are two (identical) Animal objects; X points to one, and Y points to the other.** |
| **T** | **F** | **Suppose we create X = new Animal(“Cat”) and then we execute X.setWeight(7.2). X will still point to the same object, although its weight field should be equal to the animal’s weight.** |
| **T** | **F** | **If multiple keys hash to the same value, a HashMap will still function correctly.** |

**Problem 2: For a BST, define the “imbalance” of a node to be the absolute difference of the number of left children from the number of right children (e.g. a node with identical numbers of left and right children has imbalance 0, but a node with 3 left children and 1 right child would have imbalance 2). Write a method “public static int imbalance(BST n)” that returns the imbalance of node n.**

**Problem 3: What was the worst-case complexity of your solution to problem 2?**