CSE 373 Section Handout #4

BinarySearchTrees and AVLTrees

1. This problem will give you some practice with the basic operations on AVL Trees. Show the result of inserting 43, 8, 5, 10, 4, 7, 32, 2, 1, 3 in that order into an initially empty AVL tree. Show your work.

2. Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

3. Given a binary search tree, describe how you could convert it into an AVL tree with worst-case time O(n lg(n)). What is the best case runtime of your algorithm?

4. HeapVL Trees
    a. Is there an AVL Tree that isn't a heap?

    b. Is there a heap that isn't an AVL tree?

    c. Is there a binary search tree that is neither?

    d. Is there a binary search tree that is both?

5. Write pseudocode to determine if an AVLTree is balanced. You may assume that the nodes of the tree keep track of the height of their subtree. What would change if the height wasn't stored?

   You may assume the nodes of the tree look like:

```
class AVLNode {
     int key;
     V value;
     int height;
     AVLNode left;
     AVLNode right;
}
```

Hashing

For these next few problems, assume we are dealing with the following **Business** class. Our goal is to develop a hash function for this object, so we can use it as a key in a hash table. Remember, objects that are considered equal must hash to the same value.

```
public class Business {
	private String name;
	private String city;
	private int numEmployees;

	public int hashCode() {
		// your hash functions here
	}
}
```

1. Design a hash function that has a 100% collision rate (all keys map to the same "bucket")

2. Come up with a hash function that should have a 50% collision rate.

3. Try to come up with a hash function that is better than those you provided in (1) and (2).

4. If we were unable to come up with a good hash function from scratch on my own, how could we use existing hash functions to help? Provide an example.

5. Insert the keys **8, 3, 2 , 20, 0, 5,** into the table below, using the following hash function. Use **Linear Probing** to resolve any collisions.

   h(k) = (k * 2 + 1) % tableSize

0	
1	
2	
3	
4	
5	
6	
7	

6. What is the load factor of the above table?