CSE 373 Section Handout #3

1. Consider inserting data with integer keys **34, 16, 45, 53, 6, 29, 37, 78, and 1** in the given order into a table of size 9 where the hashing function is **h(k) = k % 11**. Show how you would insert these values into the table using Linear Probing, Quadratic Probing, and Separate Chaining:

	Linear Probing		Quadratic Probing		Separate Chaining
0		0		0	
1		1		1	
2		2		2	
3		3		3	
4		4		4	
5		5		5	
6		6		6	
7		7		7	
8		8		8	
9		9		9	
10		10		10	

2. Consider the following table which inserts values using double-hashing with a primary hash function **h(k) = k % 10**, and a double hash function **g(k) = 7 - (k % 7)**:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

a. Insert the following values **21, 36, 26, 11, 6** into the hash table using the above hashing method.

b. Give a single integer that, when we attempt to insert it the table using the above hashing method after inserting the previous values from part a, results in an infinite loop.

c. Is there any way we can avoid double-hashing resulting in an infinite loop? Explain your answer.

3. What effect does the load factor have on the runtime of insert? For each of the following collision resolution schemes, give the worst case asymptotic runtime of insert for the given load factor:

	$\lambda = 0$	$0.5 < \lambda < 1$	$\lambda >= 1$
Linear Probing			
Quadratic probing			
Separate Chaining where chains are linked lists			
Separate Chaining where chains are AVLTrees			

4. 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?