CSE 373 Section Handout #5

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

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

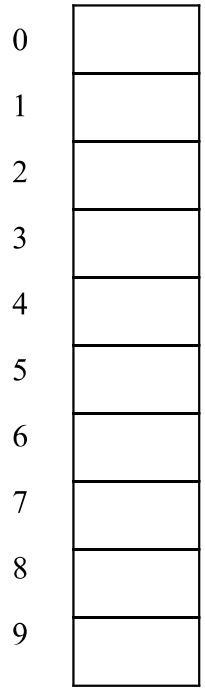
Quadratic Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Separate Chaining

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
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)**:



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. Write **pseudocode** for a **`rehash`** method for your TextAssociator HashTable from HW3. This method, inside the `TextAssociator.java` class, should be called when the load factor gets too large ($\lambda >= 1$), and should rehash and reinsert every element inside your TextAssociator's table field into a new, larger table. This pseudocode is similar to any other rehash pseudocode for a HashTable using separate chaining for collision resolution.

4. 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			

5. Consider the following class (haha) that doesn't adequately protect its data. Provide client code that could cause a **`NullPointerException`** (when you attempt to call a method on a null object).

```
public class Student {
	private String name;

	public Student(String name) { setName(name); }

	public void setName(name) { this.name = name; }

	public boolean equals(Student other) {
		return this.name.equals(other.name);
	}
}

public class Classroom {
	private List<Student> students;

	// make a new classroom with the given students
	public Classroom((List<Student> classList) {
		if (classList == null) {
			throw new IllegalArgumentException("classlist is null");
		}

		// make sure we don't just copy the reference (students = classList)
		students = new ArrayList<Student>();
		for (Student s : classList) {
			if (s == null || s.name == null) {
				throw new IllegalArgumentException("student is null");
			}
			students.add(s);
		}
	}

	public List<Student> getStudents() { return students; }

	// assume a isn't null
	public boolean hasStudent(Student a) {
		for (Student b : students) {
			if (b.equals(a)) {
				return true;
			}
		}
		return false;
	}
}
```