**Homework #7**

**Hashing**

total points: 115

1. (5 points) Explain the importance of having a hash function that approximately satisfies the simple uniform hashing assumption.

2. Consider the following sequence of Map operations to be applied to a hash map with capacity 10 and the simple hash function $h\left(k\right)=k$.

add(57, "A")

add(107, "X")

add(27, "Q")

add(37, "B")

add(88, "M")

add(89, "R")

add(100, "Z")

add(66, "N")

a. (10 points) Show the contents of the hash map after this sequence of operations if it were using open addressing with linear probing. Assume that the hash table does not resize.

b. (5 points) What elements are in a primary cluster?

c. (10 points) Show the contents of the hash map after this sequence of operations if it were using closed addressing with chaining. Assume that the hash table does not resize.

d. (5 points) What is the load factor of the hash tables in parts (a) and (c)?

3. (15 points) As we saw in class, the hash map operations with open hashing run in $O(n/m)$ time. If we can assume Simple Uniform Hashing and ensure that $n\leq m$ then this reduces to $O(1)$ expected time per operation. The current implementation in CS232OpenHashMap does not guarantee that $n\leq m$ because the hash table is never resized. It thus does not guarantee $O(1)$ average time per operation. Add resizing to the put() operation such that the hash table is doubled in size whenever the load factor equals or exceeds the MAX\_LOAD\_FACTOR. The No3Tests class contains tests that you can use to check your implementation of this functionality. All but two of the tests pass using the provided code because they do not rely on the resize functionality. These tests are included, along with those that test the resizing, to ensure that the resize operation does not break the existing functionality.

4. (15 points) Complete the remove() method in the CS232OpenHashMap so that (i) it executes in $O(1)$ expected time, and (ii) removal from a bucket containing $m$ items executes in $O(m)$ worst-case time. (Hint: the CS232Iterator class has a remove() method.) Ideally, if the load factor became too small, the remove() method would decrease the capacity of the hash table to reduce wasted space. It is not necessary to implement this reduction in the capacity of the hash table for this problem. The No4Tests class contains tests that you can use to check your implementation of this functionality. All of the tests from No3Tests are also run to ensure that the remove() operation does not break the existing functionality.

5. Implement the following methods in the CS232ClosedHashMap using linear probing:

a. (20 points) put() and get(). You do not have to implement the resizing of the hash table in the put() method. The No5aTests class contains tests that you can use to check your implementation of this functionality.

b. (15 points) remove() with appropriate modifications to put() and get() to deal with deleted elements. The No5bTests class contains tests that you can use to check your implementation of this functionality.

6. (15 points) Consider each of the following possible hash functions where $m$ is the capacity of the underlying hash table:

 a. $h(k) = k/m$

 b. $h(k) = 1$

c. $h(k) = (k + rnd.nextInt(m))$

rnd.nextInt($m$) gives a uniformly distributed random integer in the range $[0, 1,…, m)$.

For each of these hash functions, discuss why it would not be a good hash function to use.