CS2110: Hashing Practice Questions

Instructors: Birrell, Gries, and Sampson *Made by: Your 2110 TAs*

1 Hash Functions

1. Suppose a.equals(b). What can you conclude about a.hashCode() and b.hashCode()? (Assume hashCode() is properly implemented)
   a.hashCode() == b.hashCode()

2. Suppose a.hashCode() == b.hashCode(). What can you conclude about a.equals(b)? (Assume equals() is properly implemented)
   Nothing (true or false)

3. The best hash functions have the most amount of clustering. (T/F) **False**

4. Below is part of class HashMe. Circle the best hash function for it from the list below. Also, underline any valid hash functions (they could be terrible, but as long as they work). Assume that `timeOfDayInSeconds()` returns an int. Justify your rationale

   (a) `return 0;` **valid but terrible**

   (b) `return id;` **valid and best**

   (c) `return x;` **invalid. Can return different number for equal objects**

   (d) `return timeOfDayInSeconds();` **invalid. Can return different number for equal objects**

```
public class HashMe() {
    private final int id; // Won't change after construction, used in equals().
    private int x; // Might change after construction, not used in equals().

    @Override public int hashCode() {
        // What goes here?
    }

    // ... other methods ...
}
```

2 Time Complexity

You can look up answers to these questions in the Java API specs.

1. What is the time complexity of `HashSet.add()`? **Constant**
2. What is the time complexity of `HashSet.contains()`? **Constant**
3. What is the time complexity of `HashMap.put()`? **Constant**
4. What is the time complexity of `HashMap.get()`? **Constant**

3 Collision Resolution

Assume that the following array backs a HashSet. The top line represents the index and the bottom represents the value in the set. Each of the questions in this section are **independent of each other**. Assume **for this section** that you don't have to worry about resizing.

0	1	2	3	4	5
MA			NY		VA

1. What is the load factor of this array?
   1/2.

2. Assuming that we are using chaining and that CA hashes to index 3, which bucket does CA end up in, and what is the size of the list in that bucket?
   Bucket 3. Length of 2.

3. Assuming that we are using linear probing and that CA hashes to index 3, which bucket does CA end up in, and how many state objects are in that bucket?
   Bucket 4. 1 state object.

4. Assuming that we are using linear probing, CA hashes to index 3 and CA has already been inserted. How many buckets would linear probing need to probe if we were to insert AK, which also hashes to index 3? (The initial bucket check counts as a probe)
   5. Probes index 3, 4, 5, 0, then 1.

5. Assuming that we are using quadratic probing, CA hashes to index 3 and CA has already been inserted. How many buckets would quadratic probing need to probe if we were to insert AK, which also hashes to index 3?
   3. Probes index 3, 4, 1. (3, 3 + 1, 3 + 2^2**)**

4 Rehashing Practice

The following is the initial configuration of an array backing a HashSet. On the right is a mapping from states to their hash values. Follow the directions below and draw the backing array at the end of each step. Assume that the largest load factor allowed in the HashSet is $\frac{1}{2}$ and that it uses linear probing. If you have to increase the length of the array, double its length.

0	1	2	3	4	5
MA					VA

State	Hash
MA	6
NY	4
VA	5
MO	14
AZ	2
PA	2

1. Add NY to the HashSet

0	1	2	3	4	5
MA				NY	VA

2. Add MO to the HashSet

0	1	2	3	4	5	6	7	8	9	10	11
		MO		NY	VA	MA					

3. Add AZ to the HashSet

0	1	2	3	4	5	6	7	8	9	10	11
		MO	AZ	NY	VA	MA					

4. Add AZ to the HashSet
   Same as above. Already in Set.

5. Add PA to the HashSet

0	1	2	3	4	5	6	7	8	9	10	11
		MO	AZ	NY	VA	MA	PA				

5 For Fun!

1. Hashing is used in many different asspects of computing. Do a Google search for md5sum. Read up a little. Understand that that's actually what you see in CMS after you upload a document.

2. Now search for salted password hashing. Read about the role that hash functions play in storing sensitive user data like passwords.