**Hashing Questions**

1. A hash function computes an integer hash code from an object. The goal of a good hash function is to:
   1. determine the size of the hash table.
   2. provide space for the object that is to be inserted.
   3. provide a method of dealing with collisions.
   4. provide a key for efficiently sorting the objects in the hash table.
   5. provide an integer value for the object so that the objects are uniformly distributed in the hash table.
2. If there are no collisions in the hash table, then locating a hash table element takes
   1. O(1)
   2. O( log n)
   3. O(n)
   4. O(n log n)
   5. O(n2)
3. Every object has an equals method which is inherited from the \_\_\_Object\_\_\_\_ class.

4. There are two situations in which foo2.equals(foo3) will return **true**. What are they?

When foo2 and foo3 point to the same object

When the equals method is overridden in the Foo class to be based on the values rather than the object.

1. Suppose we have a hashtable of size 7 that stores integers using the hash function f(n) = n % 7. The hashtable handles collisions using chaining. For the given sequence of integers, show what the hashtable will look like after the elements have been added.

19, 8, 59, 40, 71

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 71 -> 8 |  | 59 |  | 40 ->19 |  |

1. Suppose we have a hashtable of size 7 that stores integers using the hash function f(n) = n % 7. The hashtable handles collisions using linear probing. For the given sequence of integers, show what the hashtable will look like after the elements have been added.

19, 8, 59, 40, 71

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 8 | 71 | 59 |  | 19 | 40 |

7. A student is writing a Position class that does **not** implement Comparable. Position objects store x and y coordinates as their private data.

public class Position

{

private int x;

private int y;

. . .

The student adds 100 random positions to a set, like so:

Set<Position> allPositions = **new** HashSet<Position>();

for( int x=0; x<100; x++)

{

allPositions.add(**new** Position((int)(Math.random() \* 20-10,

(int)(Math.random() \* 20-10);

}

1. Then the student asked me, “There are duplicate (x,y) positions in the set. Why???”

What is your explanation? What should the student do now?

Even though the Position objects had the same value, the hashcodes were unique because Java’s hashcode method generates unique hashcodes for each object. To fix this, the student needs to override the hashcode and equals method for the position class.

1. Write the code so that positions are not duplicated in the set:

**Hashcode = (x + y + “”).hashcode();**

**If(x == arg.getX() && y == arg.getY())**

**Return true;**

1. Would the problem be solved if you used a TreeSet?