0. Review

a.) In the worst case, how long does it take to insert into a Linked List?

b.) In the worst case, how long does it take to add an element into an array?
(Assuming the array has space)

c.) In the worst case, how long does it take to retrieve an element from a Linked List?

d.) In the worst case, how long does it take to retrieve an element from an Array?

e.) How long does it take to resize a Linked List by one element?

f.) How long does it take to resize an Array by one element? (No space left in array)

Given the above, when would we use an array over a linked list? Could you give an example?

1. HashMaps

Let's build a Map out of an array. Recall (a subset) of the map interface:

interface Map<K, V> {

```
  public int size();
  public void put(K key, V value);
  public boolean containsKey(K key);
  public V get(K key);
}
```

Consider this put method, which assume no collisions (and no negative hashCodes):

```
public void put(K key, V value) {
   arr[key.hashCode() % arr.length] = value;
}
```

a.) What is the point of mod (%)?

b.) Now what are collisions, and how do we address them? (What changes do we have to do to the above code?) What should our array declaration look like? Assume you have a nested class Entry, where each Entry models one key-value pair.

c.) How would we write size? Describe in a few sentences.

d.) How do we write get and contains key? Assume the solution for collisions you did above.

2. HashCodes

Which of the hashCodes are invalid? Assume we are trying to hash the following class:

```
import java.util.Random;
class Point {
  private int x;
  private int y;
  private static count = 0;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
    count += 1;
  }
}

// a.)
public void hashCode() {
  System.out.print(this.x + this.y);
}

// b.)
public int hashCode() {
  Random randomGenerator = new Random();
  return randomGenerator.nextInt(Int);
}

// c.)
public int hashCode() {
  return this.x + this.y;
}

// d.)
public int hashCode() {
  return count;
}

// e.)
public int hashCode() {
  return 4;
}
```

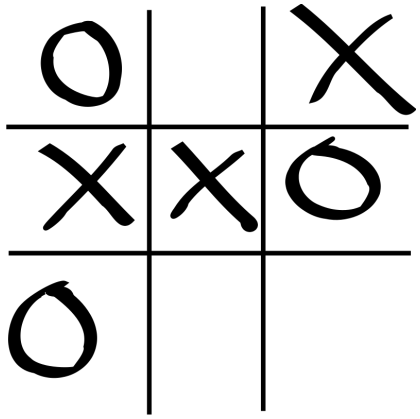
The following is a hashcode for a "binary" string which only contains characters that are ‘0’ and ‘1’. Is this a good hash code and why? You may have seen something similar from lecture.

```
public class BinString {
      public int hashCode() {
            int total = 0;
            String s = this.str;
            for (int i = 0; i < s.length; i++) {
                  char c = s.charAt(i);
                  total = 2 * total + (c - '0');
            }
            return total;
      }
}
```

A ternary string consists of only of ‘0’,’1’,’2’ instead of just ‘0’ and ‘1’. Implement a hash code for TriString that encodes the TriString as a int.

```
public class TriString {
      String str;

      public TriString(String triStr) {
            str = triStr;
      }

      public int hashCode() {
            int total = 0;
            String s = this.str;
            for (int i = 0; i < s.length; i++) {



            }
            return total;
      }
}
```

The class TicTacToe is a game state of the game tic tac toe, which has a 3x3 grid where every square is either empty or contains an ‘X’ or an ‘O’. Empty squares are represented as ‘0’, ‘O’s are represented as ‘1’, and ‘X’s are represented as ‘2’.



```
public class TicTacToe {
      char[][] board;

      public TicTacToe() {
            board = new char[3][3];
      }

      public TriString toTriString() {
            String triString = “”;
            for (int i = 0; i < board.length; i++) {
                  for (int j = 0; j < board[0].length; j++) {
                        triString += board[i][j];
                  }
            }
            return new TriString(triString);
      }

      public int hashCode() {




      }
}
```

Extra Question!

Design a hashcode for strings. Assume the string is only made up of lower case characters and spaces.

```
public class String {
      public int hashCode() {









      }
}
```