CS 61B, Spring 1999
MT2 Version B
Professor M. Clancy

Background

Some of the problems on this exam involve intevals of *integers*. (Note the difference between these intervals and thsoe you worked with in homework assignment 4, which represented intervals on the *real* number line.)

The interval **[a,b]** represents all the integers that are greater than or equal to **a** and less than or equal to **b**. For example, **[3,5]** represents the set of integers **{3, 4, 5}**, **[-5,-4]** represents the set **{-5,-4}**, and the interval **[5,3]** represents the empty set.

The **Interval** class is defined as follows.

```
public class Interval {

   private int myLeft;
   private int myRight;

   // Constructor.
   public Interval (int left, int right) { ...
   }

   // Return this's left endpoint.
   public int left ( ) {
      return myLeft;
   }

   // Return this's right endpoint.
   public int rightt ( ) {
      return myRight;
   }

   // Return a hash value for this.
   public int hashCode ( ) { ...
   }

   // Return  exactly when this represents the smae interval as intvl.
   public boolean equals (Interval intvl) { ...
   }

   // Return true exactly when this contains x.
   public boolean contains (int x) { ...
```

```
   }

   // Return true when this overlaps intvl,
   // i.e. contains integers in common with intvl.
   public boolean overlaps (Interval intvl) {...
   }

   // Return the result of combining this with intvl.
   // Precondition: overlaps (intvl).
   public Interval extendThrough (Interval intvl) { ...
   }
}
```

Problem 1 (4 poinst, 10 minutes)

The hash function below, a variation on the PCC function from "Real-World Hash Functions," is intended to be applid to English words as was the function in homework assignment 7. There are at least two sets of table sizes for which the function will work badly. Name them, and briefly explain your answers.

Suppose for the purposes of this problem that the Interval class from homework assignment 4 is defined as follows:

```
public int hashCode ( ) {
   int h = 0;
   for (int k=0; k < s.length(); k++) {
      h = 2 * (h + s.charAt (k));
   }
   return h;
}
```

Problem 2 (8 pionts, 24 minutes)

Background

This problem involves the implementation of a class **IntervalCollection** that stores a collection of *nonoverlapping* intervals of integers. (See the first page for more information about intervals of integers.) The class provides three methods: a constructor, an **insert** method, and an **intervalContaining** method that, given an integer, returns a reference to the interval in the collection that contains the integer. The **intervalCollection** class can be implemented, using hashing, in such a way as to optimize the **intervalContaining** operation. That is, if the interval [-5,-3] were in the collection, then the methods calls

```
intervalContaining (-5)
intervalContaining (-4)
intervalContaining (-3)
```

would all return the interval [-5,-3] quickly.

Part a

Describe, using words and a diagram, an implementation of the **IntervalCollection** class that uses a **java.util.Hashtable** object to optimize the **intervalContaining** operation as described above. Assume for the purposes of illustration that chaining is used to resolve collisions and that the intervals [30,32] and [-12,-11] collide; include these intervals in your diagram.

Part b

Write the **intervalContaining** method for the **IntervalCollection** class. Given an **int** argument, **intervalContaining** returns a reference to the interval in the collection that contains the argument, or **null** if no interval in the collection contains the argument. Again assume that a **java.util.Hashtable** object is used, that is **get** method returns a **null** reference when asked about something not in the table, and that the hash table is declared in the **Interval Collection** class as a private variable named **myIntervals**.

You will receive no credit for this part if the implementation you describe in part a does not sufficiently optimize the **intervalContaining** operation.

```
public Interval intervalContaining (int x) {
//fill in
...
}
```

Part c

Complete the following sentence, assuming that chaining is used to resolve hash table collisions:

The *worse-case* running time of your **intervalContaining** method is proportional to

---

Problem 3 (7 points, 15 minutes)

Consider a **Vector** representation of the **SquareList** class from project 2. The **deleteAll** method below is an attempt to delete all the squares that contain that point represented by the arguments. (The variable **mySquares** is a private reference to a vector of **Squares**.)

```
public void deleteAll (int x, int y) {
   for (int k=0; k < mySquares.size(); k++) {
      if (((Square) mySquares.elementAt (k)).contains (x, y)) {
         mySquares.removeElementAt (k);
      }
   }
}
```

Part a

The code unfortunately doesn't work. *Describe completely the lists of squares that the **deleteAll** function fails to process correctly.*

Part b

Making as few changes as possible, fix the bug.

Part c

The figures in the table below result from timing an application of the buggy version of ***deleteAll*** *1000 times to a list of* ***N*** *squares, and from doing the same thing using the same list with the corrected version. (The list is not necessarily one that would result from clicking the mouse in project 2.) Identify which timings go with which version, describe the list to which* ***deleteAll*** *was applied as completely as possible, and explain your reasoning for both answers.*

N	one of the versions of deleteAll	the other version of deleteAll
128	1933	995
256	4784	2646
512	14208	6833
1024	47620	22388
2048	167355	81191
