Cornell net id ____________________ **Name** ________________________________

CS 100J Prelim 3 14 November 2006

0 ___________	out of	02
1 ___________	out of	20
2 ___________	out of	22
3 ___________	out of	15
4 ___________	out of	20
5 ___________	out of	21
Total ________	out of	100

This 90-minute exam has 6 questions (numbered 0..5) worth a total of 100 points. Spend a few minutes looking at all questions before beginning so that you can see what is expected. Budget your time wisely. Use the back of the pages, if you need more space. We have a stapler at the front of the room, so you can tear the pages apart. You need not write loop invariants unless they are explicitly required in the question.

Question 0 (2 points). Write your netid and your name, legibly, at the top of each page (Hint: do it now).

Question 1 (20 points). Array `b` is in ascending order. Each value may occur many times, and, since `b` is sorted, the equal values are together. Here is an example: {3, 3, 5, 5, 5, 5, 7, 9, 9, 9}. In this example, the length of the longest segment of equal values is 4, since 5 occurs 4 times and the other values occur fewer times.

Write a single loop (either a while-loop or a for-loop) that stores in x the length of the longest segment of equal values in array b. The post-condition is given below, as is the invariant. **No credit will be given for a loop that does not use this invariant at all**, so remember the four loopy questions and use them in developing the loop. We *will* attempt to give as much partial credit as possible.

You may assume that `b` has at least one element, although it is not necessary.

In the repetend, you have to decide when to increase x by 1. In thinking about this, ask yourself when extending the processed segment by 1 (to the left) gives a segment of x+1 equal values.

		0	k	b.length
// invariant:	b	?	x = length of longest segment of equal values in this part of b	

	0	b.length
// Postcondition: b	x = length of longest segment of equal values in this part of b	

Cornell net id ____________________ **Name** ________________________________

Question 2 (22 points). It will soon be the time to buy textbooks for next semester. Gries is considering using the following classes `Textbook` and `Deal` to search for the best deals. Write the bodies of function `findBest` and procedure `sortSellers`, given on the next page. **These methods go in class Textbook**. Read the spec of `sortSellers` carefully! It gives additional requirements on your solution.

Gries's definition for a better deal is:

(1) the deal with lower price, and
(2) if two deals have the same price, the one with the better condition, and
(3) if several deals have the same price and condition, any one will do.

These methods in `Vector<Deal>` may be useful (not all of them are), for `v` a `Vector<Deal>`:

Return	Method	Purpose
`Object`	`v.get(int k)`	= `v[k]`
`int`	`v.capacity()`	= the number of elements currently allocated for v's list
`int`	`v.size()`	= the number of elements in `v`'s list
`<Deal>`	`v.set(j, d);`	Set v[j] to d and return the element previously in v[j].
`int`	`v.indexOf(Object ob)`	= `i`, where `v[i]` is the first occurrence of `ob in v`

```
import java.util.*;

/** An instance is a text with a title and
   a list of sellers of the text. */
public class Textbook {
   // the title of the textbook.
   private String title;

   // The list of sellers.
   private Vector<Deal> deal;

   /** Constructor: an instance of a textbook
        with title t and no deals. */
   public Textbook(String t) {
      title= t;
      deal= new Vector();
   }

   /** = the title of the textbook. */
   public String getTitle() {
      return title;
   }
}
```

```
/** An instance is a deal for a textbook. It
   contains a seller name, a price, and a
   condition of the text. The condition is in 1..10,
   (10 is the best condition and 1 the worst) */
public class Deal {
   private String name;    // name of seller

   private double price; // price for the book

   private int condition; //book condition

   /** Constructor: a Deal with name n, price p,
      and condition c.
      Precondition: n != null, p >=0, c in 1..10. */
   public Deal(String n, double p, int c) {
      name= n;
      price= p;
      condition= c;
   }

   /** = the name of the seller. */
   public String getName() {
      return name;
   }

   /** = the price of the textbook.*/
   public double getPrice() {
      return price;
   }

   /** = the condition of the textbook. */
   public int getCondition() {
      return condition;
   }
}
```

Cornell net id ____________________ Name ________________________________

/** = the index of the best deal in Vector segment `deal[h..k].`
Precondition: `d[h..k]` is not empty */
private int findBest(**int** h, **int** k) {

}

/** Sort the deals for the text by price and condition, i.e. permute Vector `deal` so that the seller with the lowest price and best condition appears in `deal[0],` the next lowest in `deal[1],` etc.

You must: (1) Write a selection sort algorithm, with ONE loop.
(2) Write the invariant of the loop before you write the loop —as a picture, as a formula, or in English.
(3) Write, in the repetend, a call on method findBest, written above. */

public void sortSellers() {

}

Cornell net id ____________________ **Name ________________________________**

Question 3 (15 points)

(a) Write a single statement that declares and initializes a two-dimensional **`int`** array `b` to look like the following table.

1	3	6	10
2	5	9	13
4	8	12	15
7	11	14	16

(b) Consider the program segment in the box on the right. Draw all variables and objects created by execution of this program segment.

```
int[][] c= new int[3][];
c[1]= new int[] {4, 5, 6};
c[2]= new int[] {1, 2};
```

(c) Consider part (b) above. Assuming that the values c[1] and c[2] may be changed to other arrays, give an expression for the length of the third row of array c.

Cornell net id ____________________ **Name** ________________________________

Question 4. (20 points) Consider class GUI on the right, which, for the purposes of this question, has no comments.

(a) Consider evaluation of:

```
new GUI(2)
```

This results in an object being created and a constructor being called. Draw the object as it initially appears —you need not write in all the methods of class JFrame— and draw the frame for the constructor call GUI(2) before execution of the method body.

```
import javax.swing.*;
import java.awt.*;

public class GUI extends JFrame {
   Box box;

   public GUI(int n){
      super("quest 2");
      Container cp= getContentPane();

      cp.add(addButtons(n, 1),
                              BorderLayout.EAST);
      cp.add(new JLabel("center"),
                           BorderLayout.CENTER);
      cp.add(addButtons(n, 2),
                              BorderLayout.WEST);
      pack();
      show();
   }

   public Box addButtons(int r, int c){
      box= new Box(BoxLayout.X_AXIS);
      for (int i= 0; i != c; i= i+1){
         Box boxc= new Box(BoxLayout.Y_AXIS);
         boxc.add(new JButton("col " + i));

         for (int j= 0; j != r; j= j+1){
            boxc.add(new JButton(j + "." + i));
         }

         box.add(boxc);
      }

      return box;
   }
}
```

(b) Draw the `JFrame` that results from evaluating the following expression. Important is not the shape and shading of components but the placement of components and the labels on the buttons.

```
new GUI(2);
```

(c) In evaluating the new-expression of part (a), how many buttons are placed in the `JFrame`?

Cornell net id ____________________ **Name** ________________________________

Question 5 (21 points). Consider the classes provided below and answer the following questions.

(a) In class `Positive`, write the body of the constructor.

(b) In class `Rational`, write the bodies of the constructor and procedure `setPositive`. In doing these, keep in mind that **the rational number must always be maintained with the denominator > 0 and in lowest possible terms** —e.g. the rational number 15/45 is maintained as 1/3 and 5 / –3 as –5/3.

(c) Explain why class `Rational` overrides procedure `setPositive`.

```
/** An instance wraps a positive
    integer */
public class Positive{
    // the positive integer
    private int k;

    /** Constructor: an instance
        with value k.
        Precondition: k > 0 */
    public Positive (int k) {



    }

    /** = this instance's value */
    public int getPositive() {
        return k;
    }

    /** Set the value of this instance
        to n.
        Precondition: n > 0 */
    public void setPositive(int n){
        k= n;
    }
}
```

```
/** An instance is a rational number */
public class Rational extends Positive {
   /** The rational number is num / k, where k is the value
       wrapped by the super class.
       Restrictions on fields:
       k is always > 0 and
       num / k is always in lowest possible terms.
       E.g. instead of 10/5 or –5/10, these numbers
       are stored as 2/1 and –1/2.

   private int num;
    /** Constructor: an instance with rational number
        num / denom.
        Precondition: denom != 0 */
   public Rational(int num, int denom) {



   }

/** Set the value of the denominator to n.
       Precondition: n > 0 */
   public void setPositive(int n){



    }

   /** Reduce this rational number to the lowest
       possible terms, e.g. 8/24 becomes 1/3 */
   public void reduce(){
     // YOU DO NOT HAVE TO WRITE THIS BODY
   }
}
```