CSC 1350 Practice Final Exam

May 1, 2014

Section (1/2)

NAME: _____

- ⊙ This exam consists of two parts.
- ⊙ Use only Java™Standard Platform Edition 7.0 compliant syntax in your code.
- ⊙ Blue book is required. Fill in the information on the cover of your blue book and on the exam sheet.
- ⊙ Do exercise 1A on the exam sheet and all other exercises in your blue book.
- ⊙ Calculators are not allowed.
- ⊙ Use the back of the exam sheets if you need scratch paper.
- ⊙ Read the instructions preceding each section carefully before beginning the section.
- ⊙ Turn in the exam and your blue book before you leave.

DURATION: 120 Minutes

Table 1: Distribution of Points

PART	WORTH	SCORE
I	$x_1 = 40$	
II	$x_2 = 60$	
Total	$\sum_{i=1}^{2} x_i$	
Exam Score	100	/30

DO NOT TURN THIS PAGE UNTIL YOU ARE TOLD TO DO SO.

1 Problems

Instruction: This section is worth 40 points. Each exercise is worth 20 points. points.

A. Binary search is a divide-and-conquer algorithm. It works by repeatedly reducing the size of the array to be searched until a target is found or is determined not to be in the array. Consider the array *list* given below. Trace the action of binary search, including listing the values of *low*, *high*, and *mid*, indices of the array. (In each table, use as many columns as needed.) For found, write **Y** when X is found and **N** otherwise.

```
int list[ ] = {29, 57, 63, 72, 79, 83, 96, 104, 114, 136};
```

for each of the following elements X.

(i) X = 96

low					
mid					
high					
found					

Number of keys (array elements) visited:_____
List the keys in the order visited:_____
Return Value:_____

(ii) X = 64

low					
mid					
high					
found					

Number of keys (array elements) visited:_____
List the keys in the order visited:_____
Return Value:_____

B. What would the program below output?

```
public class Mystery
{
   public static void main (String[] args)
   {
      double x[][] = {{1,-2,3},
                      {-2,3,-1},
                      {3,-1,2}};
      int i,j;
      for (i=0; i<x.length; i++)
      {
         for(j=0; j<x[0].length; j++)
         {
            if (i == 0 && j == 0)
               x[i][j] = x[j][i];
            else if (i == 0)
               x[i][j] = x[i][j] + x[i][j-1];
            else if (j == 0)
               x[i][j] = x[i][j] + x[i-1][j];
            else
               x[i][j] = x[i][j] + x[i-1][j] + x[i][j-1] - x[i-1][j-1];
         }
      }

      for (i=0; i<x.length; i++)
      {
         for(j=0; j<x[0].length; j++)
         {
            System.out.print(x[i][j]+" ");
         }
         System.out.println();
      }
   }
}
```

2 Programming

Instruction: This section of the exam is worth 60 points. Define the indicated classes. Import all relevant packages for each class. Use descriptive names for identifiers but no documentation is required.

2.1 The CensusData Class

Complete the implementation of the *CensusData* class.

```
/* 1. Complete the class signature so that it implements the generic
      Comparable interface */
public class CensusData ____________________________________
{
   /**
    * a two letter code for a U.S. State.
    */
   private String state;
   /**
    * population in millions
    */
   private double popInMil;

   /**
    * creates an object of this class using the specified
    * parameters
    * @param sCode a two character code for the state
    * @param pInMil the population in millions
    * @throws IllegalArgumentException when
    * 1. the state code is more than two characters long
    *    or both characters are not letters of the alphabet
    * 2. the population is not a positive number
    */
    public CensusData(String sCode, int pInMil)
    {
       /* 2. complete this method
    }

    /**
     * give the two-character code
     * @return the two-character code for the state
     */
    public String getState()
    {
       /* 3. complete this method
    }

    /**
     * gives the population in millions
     * @return the population in millions
     */
    public double getPopInMil()
    {
       /* 4. complete this method
    }
```

```
    /**
     * compares the census data of this state and the
     * specified state using the state code as the
     * primary key and the population in millions
     * as the secondary key (state+popInMil order)
     * @param s a census data for a state
     * @return 1 when this state comes after the  specified
     * state in (state+popInMil order); -1 when this state
     * comes before the specified state; otherwise, 0
     */
    public int compareTo(CensusData s)
    {
       /* 5 complete this method
    }

    /**
     * gives a string representation of the census
     * data of a state in the form
     * [state-code-in-all-caps, pop to the thousandths of a million].
     * eg: [LA, 4.602M]
     * @return the census data for the state in the
     * format [XX, 9.999M] where XX represents the
     * state code in all caps and the 9.999
     * represents the state population to the nearest
     * thousandths of a million.
     */
    public String toString()
    {
       /* 6. complete this method
    }
}
```

The CensusDataComp

7. Define a class *CensusDataComp* that implements the generic *Comparator* interface that compares two CensusData objects using the population in millions as the primary key and the two-letter state code as the secondary key (popInMil + two-letter state code order). The *Comparator* interface is defined in the *java.util* package.

CensusDataSorter

Complete the client class below to test your implementation of both the *CensusData* and *CensusDataComp* classes.

```
import java.util.*;
import java.io.*;

public CensusDataSorter
{
   public static void main(String[] args)
   {
      try
      {
         /*  8. write code to define an empty array list whose contents
                will be CensusData objects; */

         /*  9. write code to prompt the user for the data file name;
                define a Scanner to read data from the file;
                define a while loop to read the data from the file
                and create CensusData objects and insert them into the
                array list  and then close the input file */

         /* 10. create an array whose contents is the same as the array
                list */

         /* 11. Sort the array list using the Comparable interface and
                print the sorted data on the screen.*/

         /* 12. Sort the array created in step 10 using a Comparator
                of the CensusDataComp class. */

         /* 13. Write code to prompt the user for an output file name;
                write the sorted data from the array to output file;
                and, close the output file. */
      }
      /* 14. write code (a catch-block) to catch any IOException,
             print a message about the exception,
             and exit the program */
   }
}
```

Sample Program Interaction

Consider the sample input file **census.data** below.

```
LA 4.602
OK 3.815
NY 8.337
TX 26.060
CA 36.040
RI 1.050
AK 0.731
AL 4.822
MA 6.646
```

A typical program run will be:

```
Enter the name of the input file -> census.data

   Census Data Sorted By State Abbreviation and Population
   [AK, 0.731M]
   [AL, 4.822M]
   [CA, 36.040M]
   [LA, 4.602M]
   [MA, 6.646M]
   [NY, 8.337M]
   [OK, 3.815M]
   [RI, 1.050M]
   [TX, 26.060M]

Enter the name of the output file-> sortedbypop.txt
```

The contents of **sortedbypop.txt** when the program is finished executing should be:

```
AK 0.731
RI 1.050
OK 3.815
LA 4.602
AL 4.822
MA 6.646
NY 8.337
TX 26.060
CA 36.040
```