Problem 1—Short answer (15 points)

1a) As written, the program leaves the array in the following state:

<table>
<thead>
<tr>
<th></th>
<th>50</th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
</table>

If you had wanted mystery to “rotate” the array elements, you would need to run the loop in the opposite order to ensure that no elements are overwritten, like this:

```java
private void mystery(int[] array) {
    int tmp = array[array.length - 1];
    for (int i = array.length - 1; i > 0; i--)
        array[i] = array[i - 1];
    array[0] = tmp;
}
```

1b) The code for `insertValue` has the following issues:

```java
private int[] insertValue(int value, int[] array) {
    int[] result = new int[array.length + 1];
    for (int i = 0; i < result.length; i++)
        result[i] = array[i];

    int pos = 0;
    for (int i = 0; i < array.length; i++)
        if (value > array[i]) {
            pos = i;
            break;
        }

    for (int i = result.length; i >= pos; i--)
        result[i] = result[i - 1];

    result[pos] = value;
    return result;
}
```

- This is the wrong default value; it fails if value is larger than the existing ones.
- Off by one; past end of array.
- Also off by one; want >.
- Incorrect sign; should be <.
- Off by one; past end of array.
- Also off by one; want >.
Problem 2—Using the `acm.graphics` library (15 points)

```java
/**
 * Creates a `<code>GCompound</code>` object that represents a pie chart composed of the data in the array. The reference point of the `<code>GCompound</code>` is the center of the circle.
 *
 * @param r The radius of the pie chart
 * @param data An array specifying the values to be plotted
 * @return A `<code>GCompound</code>` containing the pie chart
 */
private GCompound createPieChart(double r, double[] data) {
    GCompound gc = new GCompound();
    double total = sumArray(data);
    double start = 0;
    for (int i = 0; i < data.length; i++) {
        double sweep = 360.0 * data[i] / total;
        GArc arc = new GArc(-r, -r, 2 * r, 2 * r, start, sweep);
        arc.setFilled(true);
        arc.setFillColor(WEDGE_COLORS[i % WEDGE_COLORS.length]);
        gc.add(arc);
        start += sweep;
    }
    return gc;
}

/**
 * Returns the sum of the array.
 *
 * @param array An array of `<code>double</code>` values
 * @return The sum of those values
 */
private double sumArray(double[] array) {
    double total = 0;
    for (int i = 0; i < array.length; i++) {
        total += array[i];
    }
    return total;
}
```
Problem 3—Strings (15 points)

```java
/**
 * Checks to see whether a word ladder is legal.
 */
public class CheckWordLadder extends ConsoleProgram {

    public void run() {
        println("Program to check a word ladder.");
        println("Enter a sequence of words ending with a blank line.");
        String previous = null;
        String current = null;
        while (true) {
            while (true) {
                current = readLine();
                if (current.equals("")) break;
                if (isLegalLadderPair(previous, current)) break;
                println("That word is not legal.  Try again.");
            }
            if (current.equals("")) break;
            previous = current;
        }
    }

    /**
     * Checks to see if it is legal to link the two words in a 
     * word ladder.
     */
    private boolean isLegalLadderPair(String previous, String current) {
        if (!lexicon.isEnglishWord(current)) return false;
        if (previous == null) return true;
        if (previous.length() != current.length()) return false;
        return countCharacterDifferences(previous, current) == 1;
    }

    /**
     * Counts the number of character positions in s1 and s2 that contain 
     * different characters.
     */
    private int countCharacterDifferences(String s1, String s2) {
        int count = 0;
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) != s2.charAt(i)) count++;
        }
        return count;
    }

    /* Private instance variables */
    private Lexicon lexicon = new Lexicon("english.dat");
}
```
Problem 4—Arrays (10 points)

There are many ways to solve this problem. One of the most straightforward is to keep a tabulating array that counts how many times each element occurs, as follows:

```java
/**
 * Returns true if the dice array contains two pairs. This method
 * assumes that there are exactly five dice and that a die will
 * always show a number between 1 and 6.
 */
private boolean checkTwoPairCategory(int[] dice) {
    int[] counts = new int[6 + 1];
    for (int i = 0; i < 5; i++) {
        counts[dice[i]]++;
    }
    int pairCount = 0;
    for (int i = 1; i <= 6; i++) {
        if (counts[i] == 2) pairCount++;
    }
    return pairCount == 2;
}
```
Problem 5—Building graphical user interfaces (15 points)

/*
 * File: EtchASketch.java
 * ----------------------
 * This program solves the Etch-a-Sketch problem from the practice final.
 */

import acm.graphics.*;
import acm.program.*;
import acm.util.*;
import java.awt.event.*;
import javax.swing.*;

public class EtchASketch extends GraphicsProgram {

    /** Cross size */
    private static final double CROSS_SIZE = 10;

    /** Step size */
    private static final double STEP_SIZE = 20;

    /** Initialize the application */
    public void init() {
        add(new JButton("North"), SOUTH);
        add(new JButton("South"), SOUTH);
        add(new JButton("East"), SOUTH);
        add(new JButton("West"), SOUTH);
        x = getWidth() / 2;
        y = getHeight() / 2;
        double delta = CROSS_SIZE / 2;
        cross = new GCompound();
        cross.add(new GLine(-delta, -delta, delta, delta));
        cross.add(new GLine(-delta, delta, delta, -delta));
        add(cross, x, y);
        addActionListeners();
    }

    /** Called when an action event is detected */
    public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();
        if (cmd.equals("North")) {
            moveCross(0, -STEP_SIZE);
        } else if (cmd.equals("South")) {
            moveCross(0, STEP_SIZE);
        } else if (cmd.equals("East")) {
            moveCross(STEP_SIZE, 0);
        } else if (cmd.equals("West")) {
            moveCross(-STEP_SIZE, 0);
        } else if (cmd.equals("West")) {
            moveCross(-STEP_SIZE, 0);
        }
    }
}
Problem 5—Building GUIs (continued)

```java
/**
* Moves the cross and adds a red line to the canvas connecting its old and new positions.
*/
private void moveCross(double dx, double dy) {
    GLine line = new GLine(x, y, x + dx, y + dy);
    line.setColor(Color.RED);
    add(line);
    x += dx;
    y += dy;
    cross.move(dx, dy);
}

/* Private instance variables */
private GCompound cross;
private double x, y;
```
Problem 6—Using Java collections (15 points)

/*
 * File: CachingHailstone.java
 * ---------------------------
 * This program prints the number of steps required to compute
 * the hailstone sequence for each value from 1 to 100, but does
 * so in a way so that each value is computed only once.
 */

import acm.program.*;
import java.util.*;

public class CachingHailstone extends ConsoleProgram {

public void run() {
    HashMap<Integer,Integer> cache = new HashMap<Integer,Integer>();
    for (int i = 1; i <= 100; i++) {
        int nSteps = countSteps(i, cache);
        println("Hailstone(\" + i + \") requires \" + nSteps + \" steps\")
    }
}

/**
 * Returns the number of steps in the hailstone sequence beginning
 * at \<code\>n</code\>. The \<code\>cache</code\> parameter keeps track
 * of all previously calculated chain lengths to speed up the
 * computation.
 *
 * @param n The starting value
 * @param cache A map of previously calculated step counts
 * @return The number of steps in the hailstone sequence
 */
private int countSteps(int n, HashMap<Integer,Integer> cache) {
    ArrayList<Integer> numbersSoFar = new ArrayList<Integer>();
    while (n != 1 && !cache.containsKey(n)) {
        numbersSoFar.add(n);
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
    }
    int nSteps = (n == 1) ? 0 : cache.get(n);
    for (int i = numbersSoFar.size() - 1; i >= 0; i--) {
        nSteps++;
        cache.put(numbersSoFar.get(i), nSteps);
    }
    return nSteps;
}
}
Problem 7—Essay: Extensions to the assignments (10 points)
This extension will require the following changes to the Adventure program:

- In `AdvObject`, add a new field to record the current location; implement methods `setLocation` and `getLocation` to set and retrieve the location of the object. Use 0 as a location to indicate that it is being carried.

- In `Adventure`, update the take, drop, and initial placement logic to call `setLocation` on the object.

- Also in `Adventure`, expand the code that interprets commands so that it recognizes `ZAP` along with the other built-in commands such as `LOOK`, `TAKE`, and `DROP`. The exact form of the code will vary depending on how you implemented the command logic, but presumably the code will look something like this:

```java
if (verb.equalsIgnoreCase("ZAP")) {
    game.executeZapCommand(obj);
}
```

- Also in `Adventure`, implement the `executeZapCommand` command so that it has the desired functionality. The code will look something like this:

```java
public void executeZapCommand(AdvObject obj) {
    int location = obj.getLocation();
    ArrayList<AdvObject> source;
    if (location == 0) {
        source = inventory;
    } else {
        source = rooms[location].getObjectList();
    }
    source.remove(obj);
    location = obj.getInitialLocation();
    rooms[location].getObjectList().add(obj);
    obj.setLocation(location);
}
```

- You should update the list of commands displayed by the `Help` command to include the new `ZAP` command.