Name: _____

This homework is may be used in two ways, which are *mutually exclusive*. First, those who submit this homework will have their lowest homework grade dropped. Second, this homework may be used as *a seed* for the term project by implementing what is described below and extending that work with a variety of additional features. This assignment focuses on gaining more experience with (1) designing and analyzing simple algorithms and (2) implementing a graphical user interface (GUI) using the *Swing* library, described in Appendix B of the textbook. Please refer to the previous assignments for important instructions on the allowable use of resources and the requirements for electronic submission. Ask for clarifications if needed.

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) Package your solutions to the programming questions as in previous assignments. Submit your work via `http://cs.umaine.edu/~chaw/u/`. *After* submitting your work, complete the following:
   File name: _____ Size, in bytes: _____
   MD5 checksum: _____
   Timestamp: _____

3. (28 pts.) Describe an efficient algorithm that takes as input a set S of n points in the x–y plane and produces as output a list containing all subsets of S that contain four or more collinear points. Try to design the most efficient algorithm you can. Avoid the obvious $O(n^4)$ algorithm and note that a simple $O(n^2 \log n)$ algorithm exists. Justify the correctness and claimed running time of your algorithm.

   The output list should be sorted in lexicographic order of the strings obtained by concatenating the points in each subset. The points themselves are ordered using the lexicographic order of their x and y coordinates. For example, $(1, 20) < (2, 4) < (2, 7) < (3, 1)$ and $\{(1, 20)\} < \{(1, 20), (2, 7)\} < \{(2, 1)\} < \{(2, 1), (1, 20)\} < \{(2, 1), (3, 1)\} < \{(3, 1)\}$.

[additional space for answering the earlier question]

4. (20 pts.) Implement the algorithm you describe for Question 3 by providing a class `MyLineFinder` that implements the `LineFinder` interface of Figure 1. Submit your work as in earlier assignments. Your submission should include a README file that describes how your implementation may be tested. You should include a suitable driver program, test inputs, and expected outputs.

```
import java.awt.geom.Point2D.Double;
import java.util.Set;
import java.util.List;

public interface LineFinder {
    /**
       @param p a set of points in the x-y plane.
       @return a list l of subsets of p. Each element of the list is a
       set of four or more collinear points belonging to p.  The
       subsets in the list l are sorted in lexicographic order of the
       strings obtained by concatenating the points in each subset;
       the points are ordered using the lexicographic order order of
       their x and y coordinates.
     */
    public List<Set<Point2D.Double>> collinearFourPlus(Set<Point2D.Double> p);
}
```

Figure 1: The LineFinder interface of Question 4.

5. (10 pts.) Perform experiments to study the running time of your implementation for Question 4. Summarize your results in tabular form in the README file of your submission. The file should also describe the exact steps that should be followed to reproduce these results using your submitted code. Your submission should also include a chart (in PDF format) that summarizes your results graphically. This chart should plot input sizes on the x axis and the experimentally observed running times on the y axis. Compare the the observed growth rate of running times to the worst case asymptotic growth rate claimed for your algorithm in Question 3 (e.g., by plotting a suitably fitted curve) and explain any differences. If the observed growth rate is larger than the claimed one, determine the cause of the discrepancy and fix it.

6. (30 pts) Provide a graphical user interface (GUI) to your implementation of Question 4 using only pure Java and Swing. Your GUI should provide at least the following functionality: On start-up, your program should display a square canvas, approximately 500 pixels a side, that depicts the x-y plane in the usual form, with the origin at the center. The lines representing the x and y axes should be labeled with values ranging from -100 to 100 in increments of 10. Clicking the left mouse button at any point on the canvas should result in a point with the corresponding x and y coordinates being added to the input set of points (the set S of Question 3). In this way, we may add several points to the input set. As each point is added to the input set, its location on the canvas should be marked with a dot (small filled circle). The interface should also include two buttons, labeled *find* and *clear*. Clicking on the find button should invoke your implementation of the `collinearFourPlus` method of Figure 1, using the current set of points displayed on the canvas as input. The subsets in the output of this method

should be highlighted graphically by connecting their constituent points using lines. (You may wish to use lines of different colors and thicknesses to improve visibility.) Clicking the clear button should return the interface, and underlying program, to the original, pristine state with a blank canvas.

You should submit your implementation as usual with the rest of your work, making sure to include suitable instructions in the README file.

7. (5 pts.) Add a checkbox, labeled *auto-find*, to the your interface for Question 6. When this checkbox is unchecked, the program should behave as before. When the checkbox is checked, the program should behave as though every click on the canvas (and resulting addition of a point to the input) is followed by a click on the find button. That is, the new output of the `collinearFourPlus` method should be displayed as soon as a new point is added to the input.

8. (5 pts.) Perform experiments to study the response time of the auto-find feature of Question 7. In particular, quantify how the response time varies as the number of points increases. Summarize the results by following the directions in Question 5.

9. (30 pts.) $\star$ Describe an efficient algorithm for and *online* version of the problem of Question 3, where the points in S are provided one at a time and the algorithm must produce any new qualifying subsets in response to each point's addition. (Review the distinction between online and offline algorithms in the textbook.) The output per added point should be sorted as before. Describe the asymptotic response time, where response time is the time from when the new point is added to the input to when the new output is produced. Justify the correctness and claimed response time of your algorithm. (Answer this question on separate sheets of paper attached to your hardcopy submission.)

10. (20 pts.) $\star$ Implement your algorithm for Question 9 and submit your implementation with the rest of the assignment as usual. Be sure to include the appropriate testing instructions in your README file.