Name: ______________________________

Please submit this homework by following the homework submission instructions on the class Web site. Reminder: You are welcome, and encouraged, to use any resources (e.g., Web sites) to help you with your work. However, *all such help must be clearly noted* in your submissions. Further, no matter what you use, *you must be able to explain* how and why it works. Refer to the class policy for details, and ask for clarifications if you are unsure if something is allowed. Questions marked with ⋆ are optional and may be answered for extra credit.

1. (1 pt.) Write your name in the space provided above.
2. (1 pt.) Package and submit your solutions to the programming questions as in previous assignments. After uploading your jar file to the FTP server, complete the following:
   File name: ____________________ Size, in bytes: __________
3. Read about the textbook's description of the mergesort algorithm and its implementation[1] and answer the following:
   (a) (5 pts.) Will the merge method of Figure 8.9 produce correctly merged output if the `<= 0` on line 19 is replaced with `< 0`? If so, explain why. If not, provide a counterexample.

[1]Mark Allen Weiss, *Data Structures and Problem Solving Using Java*, 3rd edition (Addison-Wesley, 2006), Section 8.5, pp. 313–317.

(b) (5 pts.) Explain, as precisely as possible, the meaning of the expression on line 5 of Figure 8.8:

```
<AnyType extends Comparable<? super AnyType>>
```

In particular, explain what happens if we replace the above with:

i. `<AnyType>`

ii. `<AnyType extends Comparable<AnyType>>`

4. Read the textbook's description of external sorting[2] and answer the following questions:

   (a) (7 pts.) Let $D(x, d, n)$ denote the sequence n integers obtained by reading the first dn digits of x (following the decimal point) as n d-digit integers. For example, $D(1/3, 5, 4)$ is the rather boring sequence $(33333, 33333, 33333, 33333)$ but $D(\pi, 2, 10)$ is more interesting: (14, 15, 92, 65, 35, 89, 79, 32, 38, 46).

   Illustrate the key configurations of the simple mergesort algorithm (Section 21.6.3) with $D(\pi, 2, 33)$ as input, assuming that we have four tapes and that internal memory can hold four records at a time (i.e., $M = 4$). Use a notation similar to that used in the textbook.[3] Here are the first 100 digits of π for reference:

```
3.1415926535 8979323846 2643383279 5028841971 6939937510
  5820974944 5923078164 0628620899 8628034825 3421170679
```

[2] *Idem*, Section 21.6, pp. 764–771.

[3] *Idem*, Figures 21.29–21.33.

(b) (7 pts.) Repeat Question 4a, but use a *four-way merge with eight tapes*.

(c) (7 pts.) Repeat Question 4a, but use a *four-way polyphase merge with five tapes.*

(d) (7 pts.) Repeat Question 4c, but use *replacement selection* to generate the initial runs.

5. (20 pts.)[4] Describe an algorithm that takes as input a set S of points in a plane and produces as output a list containing those subsets of S that contain four or more collinear points, and whose worst-case running time is $O(n^2 \log n)$, where n is the number of points in S. [Hint: $O(n^2 \log n)$ seems to suggest sorting $O(n^2)$ items.]

[4] *Idem*, p. 341.

6. (10 pts.) $\star$ Describe an *online* algorithm for the problem of Question 5. (Recall the distinction between online and offline algorithms from the second assignment.) 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.

7. (30 pts.) Implement the algorithm you describe for Question 5 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.Point;
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 descending order of
       cardinality...
     */
    public List<Set<Point>> collinearFourPlus(Set<Point> p);
}
```

Figure 1: The LineFinder interface of Question 7.

8. (10 pts.) Perform experiments to study the running time of your implementation for Question 7.

   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 $O(n^2 \log n)$ worst case asymptotic growth rate claimed for your algorithm (e.g., by plotting a fitted curve $y = ax^2 \log x + b$) and explain any differences. If the observed growth rate is larger than the claimed one, determine the cause of the discrepancy and fix it.

9. (20 pts) $\star$ Provide a graphical user interface (GUI) to your implementation of Question 7. Be sure to use only pure Java (i.e., no platform-specific extensions); the Swing package is a good option and is described in the textbook.[5]

   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 5). In this way, we may add several points to the input set. As each

[5] *Idem*, Appendix B, p. 867.

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.

10. (2 pts.) ⋆ Add a checkbox, labeled *auto-find*, to the your interface for Question 9. 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.

11. (3 pts.) ⋆ Perform experiments to study the response time of the auto-find feature of Question 10. In particular, quantify how the response time varies as the number of points increases. Summarize the results by following the directions in Question 8.

12. (5 pts.) ⋆ Implement your algorithm for Question 6 and submit your implementation with the rest of the assignment as usual. Be sure to include the appropriate testing instructions in your README file.