<u>EECS 233 Homework 4</u>

General requirements:

- Due at 11:00 PM on the posted due date.
- Include your name and network ID as a comment at the top of all of your programs.
- Create a typed document (.docx or .pdf) for the report with your name and network ID at the top.
- Upload your document and all .java files as a .zip file to Canvas. Do <u>not</u> use other formats such as .rar.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus. File sharing is prohibited.
- ***NEW: To assist with grading, the following are required (note that <u>no</u> template file is provided):***
  1. ***All programs must be named "ProgramX.java", where X is the problem number.***
  2. ***All methods must be named as explained in the specific instructions.***

1. This problem is based on two sample programs: (a) Fractal2.java was provided in class and uses the exact same algorithm discussed in the textbook (use as a reference). (b) Fractal3.java is provided with this document and demonstrates some useful techniques for this problem. Create a program "Problem1.java" that creates an animation showing the fractal sequence at each recursion level. Stop at a predefined maximum recursion level of your choice instead of the amount of change in vertical pixels. Include the following features:
   a. Revise the randomFactorial() method to receive an additional argument that is the recursion level of any previous call to randomFactorial(). For example, pass the value 0 the first time that randomFactorial() is called. The next level of calls to randomFactorial() should pass the value 1, then 2, etc.
   b. Change the recursion base case (stopping condition) to be when the predefined maximum recursion level has been reached. For example, if the
   c. To perform the animation, include a loop in the constructor that repeatedly calls repaint() after setting the maximum recursion level to a higher value. Begin with a maximum recursion level of 1 and increment up to at least 10. It is recommended that you include a delay of some time, such as with a dummy loop, to slow the animation down.
   d. What is the big-O time complexity of this algorithm if *N* is the recursion level? Briefly explain your answer. Hint: How many new points are added?

   Include at least one screenshot of the end of your animation in your report. Note that a video is provided on Canvas that shows an example animation. The example in the video has two features that are <u>not</u> required for this assignment, but were added for effect: (1) It displays the maximum recursion level using drawString(), as shown in Grapher.java for HW #1. (2) New dots are displayed in red using Color.RED.

2. Write two programs that both track the user's path on a rectangular grid. Allow the user to repeatedly enter a direction (left, right, or straight) in which they will take a step. When the user chooses to quit, the program should print a complete list of the necessary directions for each step to return to the original starting point. Include a message indicating when the directions are done. Note that the user should turn 180 degrees before following a reverse path. Below is an example of the possible output (user input in **bold**):

```
Enter a direction for a step (s=straight, l=left, r=right, q=quit): l
Enter a direction for a step (s=straight, l=left, r=right, q=quit): r
Enter a direction for a step (s=straight, l=left, r=right, q=quit): s
Enter a direction for a step (s=straight, l=left, r=right, q=quit): l
Enter a direction for a step (s=straight, l=left, r=right, q=quit): q
Turn 180 degrees
Take a step and turn right
Take a step and remain straight
Take a step and turn left
Take a step and turn right
Done!
```

Write the following two programs, each of which should produce the same output using different techniques:

Problem2A.java: Use one loop to push each direction onto a Stack. Use another loop to pop the directions and determine the reverse path. Do not use recursion for this program. Include an example of your program input/output in a separate report.

Problem2B.java: Use a single, recursive function that handles all user input and printing (except for indicating when the reverse path is done). It should be a static method that receives nothing, returns nothing, and is called once from the main method to begin. The stopping case is when the user chooses to quit. Note that there should be no loops anywhere in your program! Include an example of your program input/output in a separate report.

Include the following in your report:

a. Provide sample output from both programs.
b. What is the big-O time complexity of this algorithm if *N* is the number of steps? Briefly explain your answer.

3. The program PowerDemo.java from the textbook is provided and demonstrates two approaches to computer values raised to a power (exponentiation). Note that it requires the file EasyReader.java that is also included. Answer or do the following:

   a. What is the big-O time complexity of the "power" method in the sample code if *N* is the exponent? Briefly explain your answer.
   b. What is the big-O time complexity of the "pow" method in the sample code if *N* is the exponent? Briefly explain your answer.
   c. Write a program "Problem3.java" that uses the recursive algorithm and solution from Self-test exercise 13 (chapter 8) that was given in class (including the solution!). **Provide sample output in your report.**
   d. What is the big-O time complexity of the recursive method in (c) if *N* is the exponent? Briefly explain your answer. Hint: How many multiplication operations occur?

Tip (why run the fractal algorithm multiple times?): It may seem strange to run the fractal algorithm using different recursion levels. It might be more intuitive to just modify paintComponent() to pause as it adds the dots. However, the screen will not be updated until paintComponent() has totally finished. That would mean no animation. Forcing paintComponent() to be called for different recursion levels is one way to get around this limitation.

Comment (weird random number technique): Though not required, you may be curious about the weird use of the Random class in Fractal3.java. In particular, the bit shifting with the seed value is very unusual because it uses the midX value in the seed! This was done because of the animation problem that is mentioned in the tip above. The random sequence will a different quantity of values for every recursion level (1, 2, 4, etc.). To see the animation appear correctly, the values need to be repeatable for every recursion level. The weird technique used here accomplishes that. If it weren't for the animation and the issue with paintComponent(), we wouldn't make random values in this weird way.

Tip (fractal recursion level): In Problem #1, the term "recursion level" refers to how many times the recursive function has called itself. Fractal3.java does not keep track of this. The goal is to control how many points are added to the fractal line in order to see the animation progress. The first step (a) is to add an argument that the method uses to tell the next method call what level it was at when the call was made. Below is an example of several such calls with values for this new argument:

randomFractal(..., 0) // First call from paintComponent(), level is 0 because it hasn't called itself
↓
randomFractal(..., 1) // 2nd call from paintComponent(), level is 1 because it has been called once
↓
randomFractal(..., 2) // First call from paintComponent(), level is 2 because it has been called twice
↓
continues until the stopping case

A relatively easy change to randomFractal() is to make the stopping case (if statement) check whether the recursion level has reached some maximum level. We can have an instance variable in the class for this maximum level. You could name it "STOP", as in Fractal3.java, or you could call it something else like "maxLevel". In the fractal program, stopping at level 1 would add 1 point to the line. Stopping at level 2 would add 3 points (1 point in the middle and 2 points on the sides). We can set our new instance variable to the desired max level (1, 2, etc.) and let randomFractal() always check to see if the current recursion level is equal to it.