CSC 355. Discrete Structures and Basic Algorithms
Project 4. Sorting Algorithms

General Guidelines.

There are ten different provided text files *array1.txt, array2.txt, array3.txt, array4.txt, array5.txt* files that would be the inputs for part 1 and part 2 of the project, and another five different *testGrid1.txt, testGrid2.txt, testGrid3.txt, testGrid4.txt, testGrid5.txt* that will be the input for part 3. Make sure that your code works with the provided test cases. Your solution must be coded in Java.

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! It is also considered academic dishonesty to share your code with another student. Anyone who is found to have violated this policy will be subject to consequences according to the syllabus and university policy.

Note on grading and provided tests: The provided tests are to help you as you implement the project and (in the case of auto-graded sections) to give you an idea of the number of points you are likely to receive. Please note that the points indicated when you run these tests locally are not your final grade. Your solution will be graded by the TAs after you submit. Please also note that these test cases are not likely to be exhaustive and find every possible error. Part of programming is learning to test and debug your own code, so if something goes wrong, we can help guide you in the debugging process, but we will not debug your code for you.

Project Overview.

This project has three parts. The first part requires you to implement a version of Shellsort. The second part requires you to implement a second sorting method. The third part requires you to sort a grid.

Part 1. Shellsort

In a file called `Shellsort.java`, implement a Shellsort method. The method must take an `Array` parameter and sort it. You need to implement your own SortingTest.java class to test the `Shellsort.java` implementation. You need to verify that works with the files that the instructor provided `(array1.txt, array2.txt, array3.txt, array4.txt and array5.txt)`. You may keep track of the number of passes that your algorithm performed to sort the 5 arrays and display your sorted array for each file.

Note: Although you may use the sequences given in class, you are encouraged to experiment with your own list of integers to verify your performance.

Required Method

Method Signature	**Description**
`public static void sort(Array a)`	`sorts the Array using Shellsort`

Part 2. ArraySort

In a file called `ArraySort.java`, implement another sorting method (apart from Shellsort). The method must take an `Array` parameter and sort it. Use the same array.txt files to test your sorting algorithm and count the number of passes that took to sort the entire arrays and compare it with part 1 (you can use the same `SortingTest.java` file that you created).

Required Method

Method Signature	**Description**
`public static void sort(Array a)`	`sorts the Array`

Part 3. SortGrid

In a file called `SortGrid.java`, implement a sorting method for a grid. The method must take a `Grid` parameter and sort it. You can use the same SortingTest.java class to test your algorithm, just make sure that works with the files that the instructor provided (`testGrid1.txt, testGrid2.txt, testGrid3.txt, testGrid4.txt, testGrid5.txt`).

A grid is considered sorted when each row is sorted and the last element in each row is less than or equal to the first element in the next row.

I highly recommend that you work with this as a grid and not treat it as a one-dimensional array. That is, take advantage of the fact that it is a grid with rows and columns.

Example:
<u>Original Grid</u>

2	9	1	0
1	6	3	5
0	2	7	2
9	8	7	6

Sorted Grid

0	0	1	1
2	2	2	3
5	6	6	7
7	8	9	9

Required Method

Method Signature	**Description**
`public static void sort(Grid g)`	`sorts the Grid`

Other notes about grading:

- Good Coding Style includes using good indentation, using meaningful variable names, using informative comments, breaking out reusable functions instead of repeating them, etc.
- If you implement something correctly but in an inefficient way, you may not receive full credit.

Submission

- To submit your code, please upload your java files that I will need to test your project to D2L.