CS 321 - Algorithms - Fall 2018 | *Your names go here*

Assignment 8

Due BEFORE 8:00AM on Monday 11/5/2018

On time / 20% off / no credit

Total points: 50

You are REQUIRED to work in teams of two or three on this assignment. Make sure to include all names above and in the Java files, but submit only ONE zip file to D2L.

This assignment will help you develop your algorithm analysis skills with a focus on empirical testing, as opposed to the theoretical analyses we have focused on so far in this course. You will conduct an empirical study of three variants of quicksort. As always, you must write up your solutions to this assignment IN THIS FILE using LaTeX by filling in (or replacing) all of the boxes below. If your submitted `.tex` file does not compile, then you will receive 0 points.

You should NOT add any LaTeX packages to this `.tex` file.

In this assignment, you are required to implement a few methods. Make sure to develop and test your code using **JDK version 8**, since this is the version that I will be using for grading. **Make sure to use library methods whenever possible to shorten your code and make it more efficient.** You are allowed to fill in the body of the existing methods but may NOT modify the given source code files in any other ways, that is, by adding new methods, intance or class variables, import statements, classes, etc.

Submission procedure:

1. Complete this file, called `a8.tex`, with your full names and answers typed up below.
2. Compile this file to produce a file called `a8.pdf`. Make sure that this file compiles properly and that its contents and appearance meet the requirements described in this handout.
3. Create a directory called `a8` and copy exactly ten files into this directory, namely:
   - `a8.tex` (this file with all of your answers added)
   - `a8.pdf` (the compiled version of the file above)
   - `Sort.java` (the file containing your answer to part 1)
   - `Rand.java` (the file containing your answer to part 2)
   - `A8.java` (the file containing your answers to part 3 through 9)
   - `part3.xlsx`, `part4.xlsx`, `part5.xlsx`, `part6.xlsx`, `part7.xlsx` (the files containing your data and charts for parts 3 through 7)
4. Zip up this directory to yield a file called `a8.zip`
5. Submit this zip file to the D2L dropbox for A8 before the deadline above.
6. Submit a STAPLED, SINGLE-SIDED, hard copy of your `a8.pdf` file BEFORE the beginning of class on the due date above.

Problem statements

1. **(5 points) In this part, you will implement (or call) three variants of quicksort. You will do all of your work for this part in `Sort.java`. The comment blocks in this source file provide all necessary details. Make sure that your three top-level methods return the actual run time measured in SECONDS. You must use `System.currentTimeMillis()` to obtain the start and end times of the execution and then compute the difference.**

   Your answer for this part will be fully contained in your submitted `Sort.java` file.

2. **(5 points) In this part, you will implement two algorithms for generating a random permutation of a given array. The first algorithm is the Fisher-Yates algorithm. The second algorithm is a variant of Fisher-Yates that only partially shuffles the given array. All of your work for this part will be completed in the file `Rand.java`, which contains all necessary details in comment blocks.**

   Your answer for this part will be fully contained in your submitted `Rand.java` file.

3. **(10 points) In this part, you will study the asymptotic efficiency of all three sorting algorithms with random inputs. You will run each algorithm with increasing values of the array size and measure their running time. You will collect these data into an Excel workbook called `part3.xlsx` containing three sheets called `algo1`, `algo2`, and `algo3`, respectively.**

   Each sheet will contain three columns of numbers: one column for the array size in increasing order (which MUST be formatted in scientific notation with two decimal places), a second column for the corresponding running time in seconds (in general number format), and a third column for the "theoretical" running time.

   While the data in the first two columns come from your experiments, the data in the third column come from a formula that you must tYoe in and that represents the best model for the algorithm's running time. This formula is simply the function that you would use in the Θ bound to characterize the running time in this experiment multiplied by a constant. You must determine both the function and the constant in each case.

   The constant you choose MUST be stored explicitly in the top cell of the fourth column (note that this constant is not shown in the sample image below but must be included in all of your sheets). This cell must be referenced in the formula used in your third column. THIS CELL MUST BE FORMATTED AS A NUMBER IN SCIENTIFIC NOTATION WITH EXACTLY 3 DECIMAL PLACES. **Of course, all cells in the third column must contain exactly the same formula.** IF YOUR FORMULA USES THE LOG FUNCTION, YOU MUST USE LOGARITHM BASE 2.

   You will choose the function and constant by graphing the actual running time and the model function while tweaking the value of the constant in the model until the two chart lines are as close as possible (ideally on top of each other) for the rightmost half of the chart, that is, for the top half of the input size range. You will have to tYoe up

the function and the final value that you choose for your constant (in scientific notation with no approximation) in the boxes below for each algorithm.

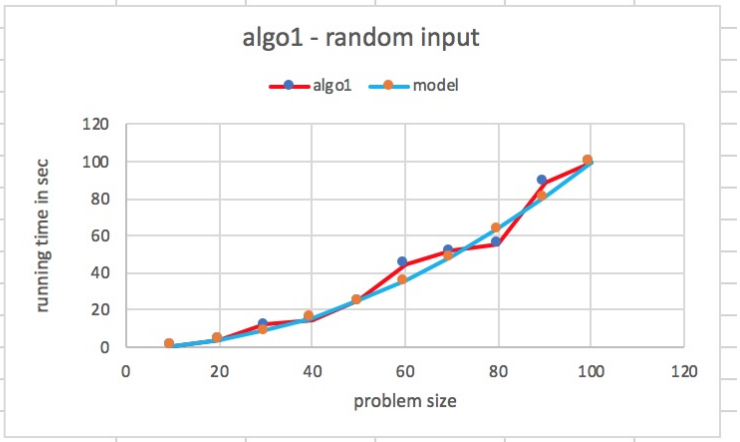
To the right of your four columns of numerical data (and constant) in each sheet, you will include an X Y (Scatter) chart with the array size on the X axis and the actual/modeled running time on the Y axis. Therefore, each chart will contain two lines of different colors: red for the actual running time and blue for the model. Make sure to also indicate each data point with a visible mark. For full credit, your axes must be properly labeled, and your chart must contain a title and a clear legend above the chart and below its title.

To produce your data, you will write your code in `A8.java`. For this part, you only need to complete the `part3` method. You may NOT modify any other code for this part. Running your code will send its output to the console. Then you will copy and paste the output into the `part3.xlsx` workbook's sheet for the appropriate algorithm and produce the chart as described above. The next page contains (made up) answers as an example of what is expected. Make sure that all three charts fit on the same page in your pdf submission.

Obviously, your three snapshots on the next page will be different from each other. The input sizes will be larger, etc. Make sure that they are legible without a magnifier and with appropriate use of colors and other options in the Excel charts. I should not have to open up your `.xlsx` file to understand your answers.

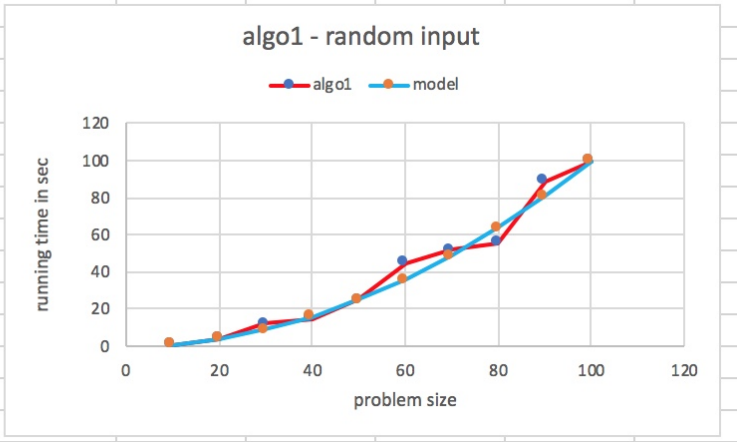
Algo 1: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



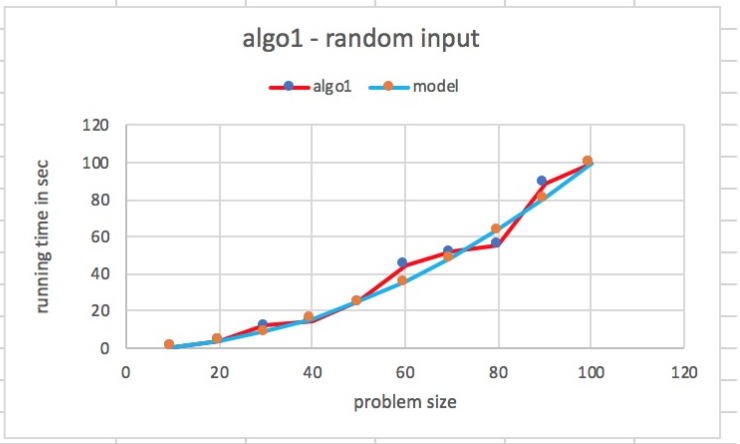
Algo 2: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



Algo 3: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



Your answer for this part will be split into:

(a) your submitted `A8.java` file,

(b) your submitted `part3.xlsx` file, and

(c) your three constant values and three figures on the previous page.

Finally, here are some questions for you to think about before moving on to the next part:

- **Do the three algorithms have the same order of growth on random inputs?**
- **Can you rank them in order of most to least efficient on random inputs?**
- **What is the unit for the constant values you chose? Do you understand their magnitude?**

4. **(5 points) In this part, you will study the asymptotic efficiency of all three sorting algorithms with already sorted inputs. The description of what you have to do is identical to the one for part 3 above. The only difference is that the input array is now fully sorted.**

   Your work for this part will be done in `A8.java`, in which you only have to complete the body of the method called `part4`. You may not modify any other code for this part.

   Your answer for this part will be split into:

   (a) your submitted `A8.java` file,

   (b) your submitted `part4.xlsx` file, and

   (c) your three constant values and three figures on the next page.

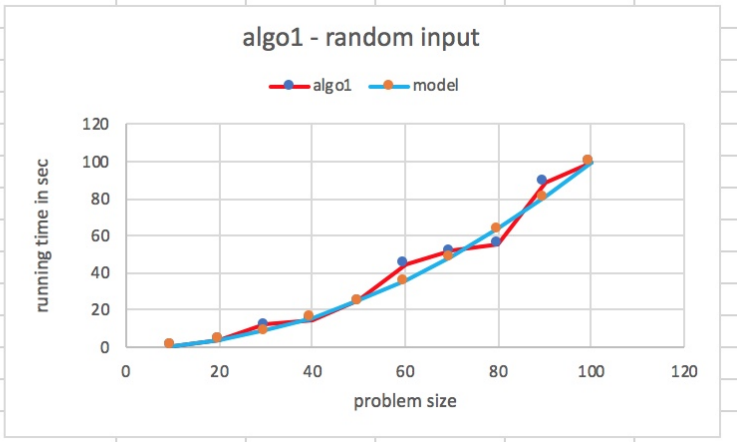
   Finally, here are some questions for you to think about before moving on to the next part:

   - **Do the three algorithms have the same order of growth on sorted inputs?**
   - **Can you rank them in order of most to least efficient on sorted inputs?**
   - **What is the unit for the constant values you chose? Do you understand their magnitude?**

   Make sure that all of your answers and charts for this part fit fully on the next page. Of course, all chart titles will have to be updated to"sorted input" instead of "random input".

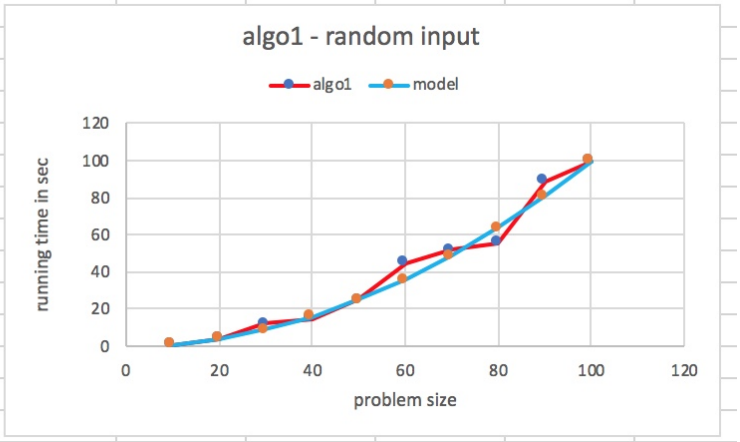
Algo 1: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



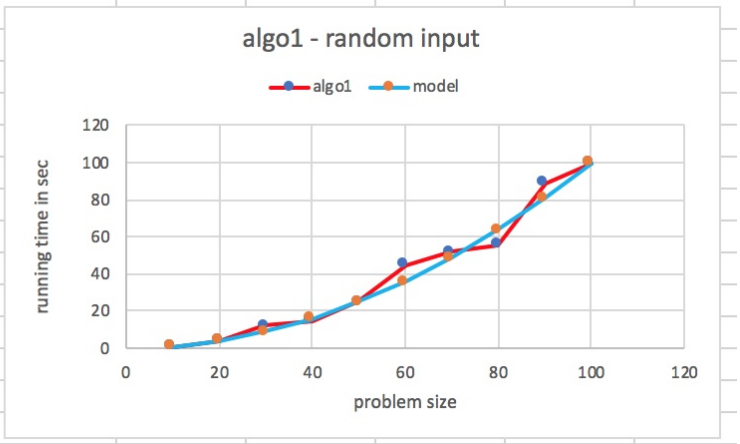
Algo 2: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



Algo 3: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



5. **(5 points) In this part, you will study the asymptotic efficiency of all three sorting algorithms with inputs that are sorted in reverse order. The description of what you have to do is identical to the one for part 3 above. The only difference is that the input array is now fully sorted in reverse order.**

   Your work for this part will be done in `A8.java`, in which you only have to complete the body of the method called `part5`. You may not modify any other code for this part.

   Your answer for this part will be split into:

   (a) **your submitted `A8.java` file,**

   (b) **your submitted `part5.xlsx` file, and**

   (c) **your three constant values and three figures on the next page.**

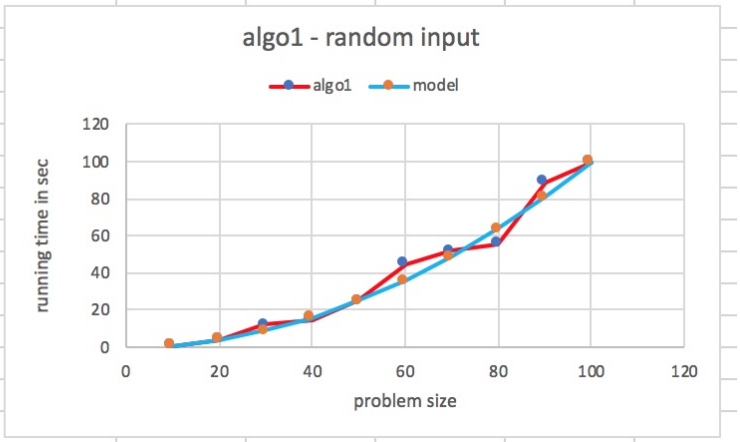
   Finally, here are some questions for you to think about before moving on to the next part:

   - **Do the three algorithms have the same order of growth on reverse-sorted inputs?**
   - **Can you rank them in order of most to least efficient on reverse-sorted inputs?**
   - **What is the unit for the constant values you chose? Do you understand their magnitude?**

   Make sure that all of your answers and charts for this part fit fully on the next page. Of course, all chart titles will have to be updated to "reverse-sorted input".

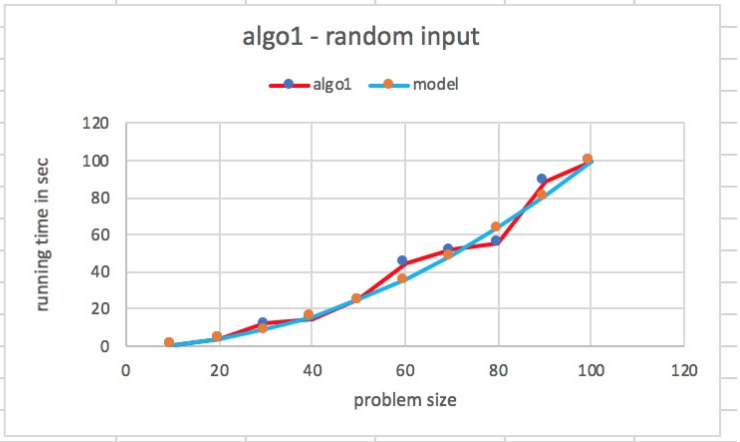
Algo 1: $T(N) = \Theta($ Your function $)$ **Your constant =** Your value

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



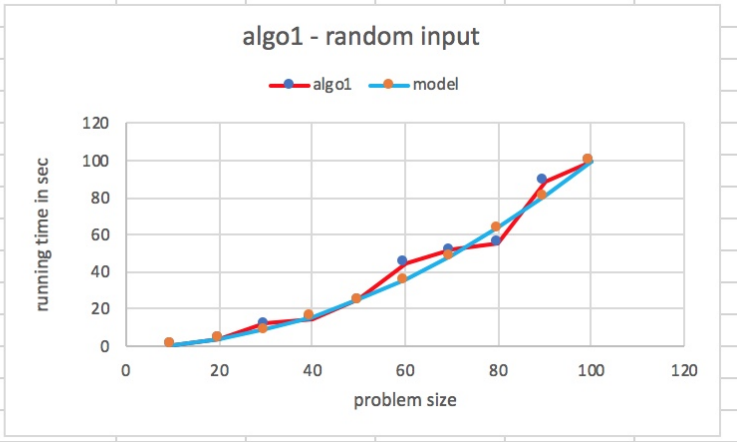
Algo 2: $T(N) = \Theta($ Your function $)$ **Your constant =** Your value

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



Algo 3: $T(N) = \Theta($ Your function $)$ **Your constant =** Your value

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



6. **(5 points) In this part, you will study the asymptotic efficiency of all three sorting algorithms with input arrays whose elements are all equal. The description of what you have to do is identical to the one for part 3 above. The only difference is that the input array is now filled with the value 1.**

   Your work for this part will be done in `A8.java`, in which you only have to complete the body of the method called `part6`. You may not modify any other code for this part.

   Your answer for this part will be split into:

   (a) **your submitted `A8.java` file,**

   (b) **your submitted `part6.xlsx` file, and**

   (c) **your three constant values and three figures on the next page.**

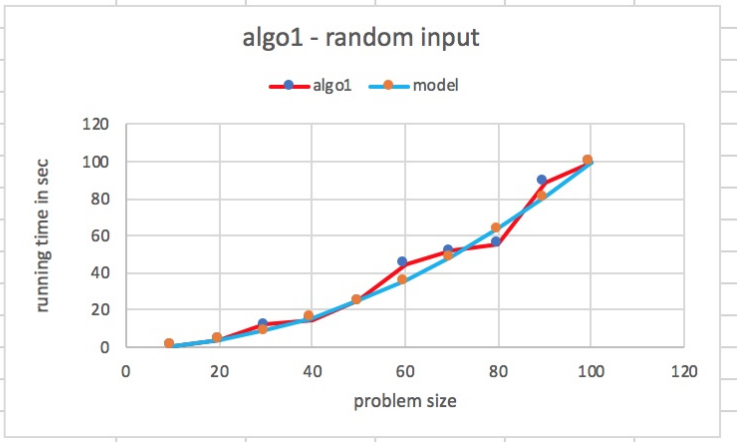
   Finally, here are some questions for you to think about before moving on to the next part:

   - **Do the three algorithms have the same order of growth on equal-value inputs?**
   - **Can you rank them in order of most to least efficient on equal-value inputs?**
   - **What is the unit for the constant values you chose? Do you understand their magnitude?**

   Make sure that all of your answers and charts for this part fit fully on the next page. Of course, all chart titles will have to be updated to "equal-value input".

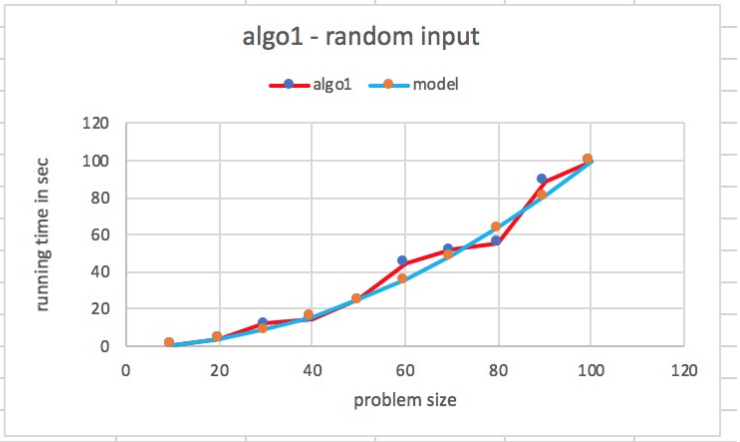
Algo 1: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



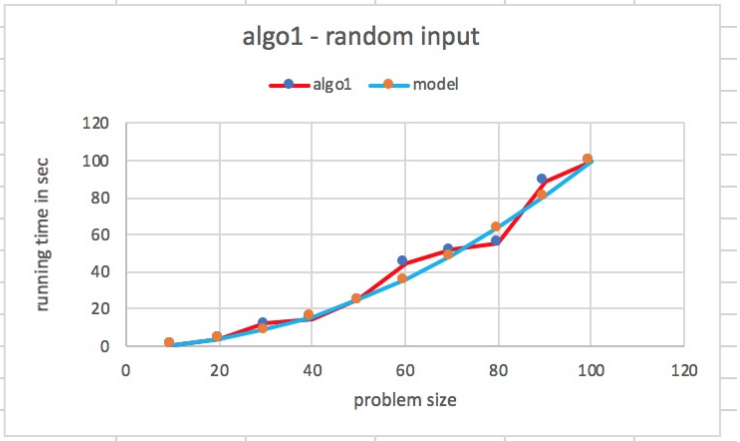
Algo 2: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



ion Algo 3: $T(N) = \Theta($ **Your function** $)$ **Your constant =** **Your value**

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



7. **(5 points) In this part, you will NOT study the asymptotic efficiency of the algorithms. Instead, you will study how their running time varies with the level of "sortedness" of the input array. More precisely, you will run each algorithm on a fixed-size array, namely 2^{20} elements. Starting from a fully sorted array (in increasing order), you will increase the number of elements that are swapped using the `Rand.shuffleArray` method with values of n (the third argument) ranging from 0 ro 5,000, in increments of 100, for a total number of 51 runs for each algorithm.**

   The output of your program will be the values of n and the corresponding running times in seconds. Once you have this data for all three algorithms, you will build an Excel workbook called `part7.xlsx` containing a single sheet. This sheet will contain four columns of numbers, namely, from left to right: the value of n, followed by the running times for algorithms 1, 2, and 3, in this order. To the right of these columns, you will create an X Y (Scatter) Chart with the value of n on the X axis and the running times on the Y axis.

   This chart should follow the directions for the previous parts, with the following differences:

   - **It will contain three lines: a blue line for algorithm 1, an orange line for algorithm 2, and a gray line for algorithm 3. There is no model lines for this experiment.**
   - **It will not include any markers for the data points (just a line, as described above).**
   - **The tick marks on the X axis must range from 0 to 5,000 using the (default) linear scale.**
   - **The tick marks on the Y axis must range from 0.001 to 1,000 using a logarithmic scale in base 10.**
   - **The tick labels for the X axis must appear at the bottom of the chart, whereas the tick labels for the Y axis must appear at the left of the chart.**
   - **The title of the chart must be "FROM SORTED TO RANDOM".**

   Your work for this part will be done in `A8.java`, in which you only have to complete the body of the method called `part7`. You may not modify any other code for this part.

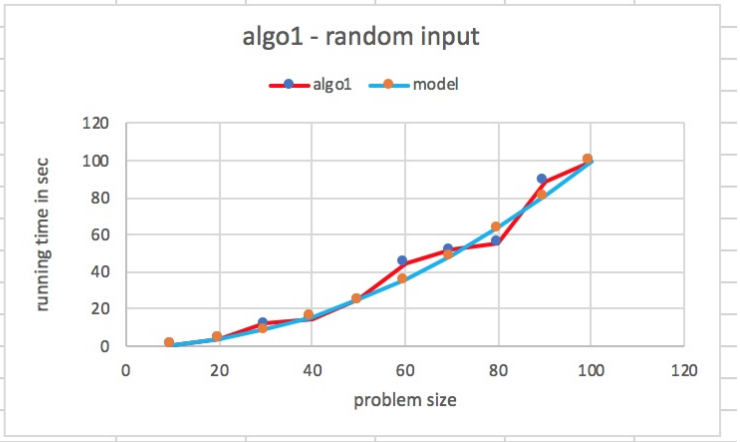
   Your answer for this part will be split into:

   (a) your submitted `A8.java` file,

   (b) your submitted `part7.xlsx` file, and

   (c) the figure on the next page.

10	1	1
20	4	4
30	12	9
40	15	16
50	25	25
60	45	36
70	52	49
80	56	64
90	89	81
100	100	100



8. **(5 points) The implementation of the `Arrays.sort` method in Java 8 is rather complex. It uses quicksort at its core as well as other sorting algorithms in special cases. Look up the code for all of these methods (there is one implementation for each primitive type) and list below the name of all of the sorting algorithms that they use:**

   Algorithm #1 = algorithm name

   Algorithm #2 = algorithm name

   Algorithm #3 = algorithm name

   Algorithm #4 = algorithm name

   Algorithm #5 = algorithm name

   Algorithm #6 = algorithm name

   Note: You may not be able to fill in all of the boxes above. Delete the boxes that you end up not needing, if any.

9. **(5 points) After studying the implementation of the `Arrays.sort` method for `int[]`, figure out which kinds of inputs will force the code to use (at least once) a one-pivot quicksort.**

   Once you have figured out one working pattern, complete the method called `part9` in `A8.java` that will generate the shortest possible array of `int`s that meets the requirement above.

   Your answer for this part will be fully contained in your submitted `A8.java` file.