CMSC 451 Project 1

The first project involves benchmarking the behavior of Java implementations of two of the following sorting algorithms: Quick Sort, Bubble Sort, Insertion Sort, Heap Sort, Radix Sort, Bucket Sort, Selection Sort, Merge Sort, Shell Sort, Bucket Sort. You must post your selection of the two algorithms that you chose in the "Ask the Professor" conference. Every student must have a unique combination of algorithms..

You must write the code to perform the benchmarking of the two algorithms that you selected. You do not have to write the sorting algorithms yourself, you may take them from some source, but you must reference your source.

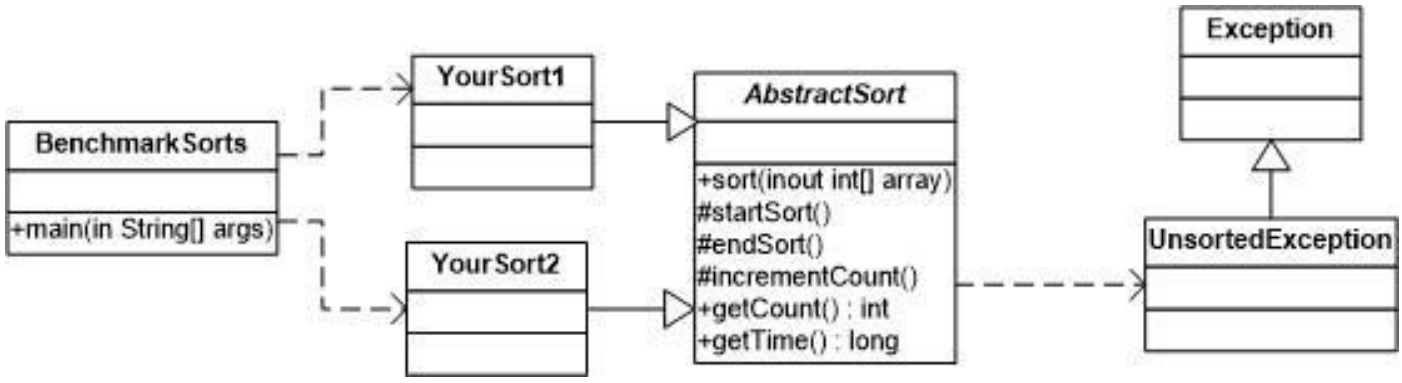
You must identify some critical operation to count for each algorithm that reflects the overall performance and modify each of the two sorting algorithms so that they count that operation. In addition to counting critical operations you must measure the actual run time in nanoseconds.

In addition, you should examine the result of each call to verify that the data has been properly sorted to verify the correctness of the algorithm. If the array is not sorted, an exception should be thrown.

It should also randomly generate data to pass to the sorting methods. It should produce 40 data sets for each value of *n*, the size of the data set and average the result of those 40 runs. The exact same data must be used for both sorting algorithms. It should also create 12 different sizes of data sets. Choose sizes that will clearly demonstrate the trend as *n* becomes large. Be sure that the data set sizes are evenly spaced so this data can be used to generate graphs in project 2

This project should consist of two separate programs (BenchmarkSorts.java and Report.java). The first of those programs should perform the benchmarking described above and generate two data files, one for each of the two sorting algorithms.

The benchmarking program must be written to conform to the following design:



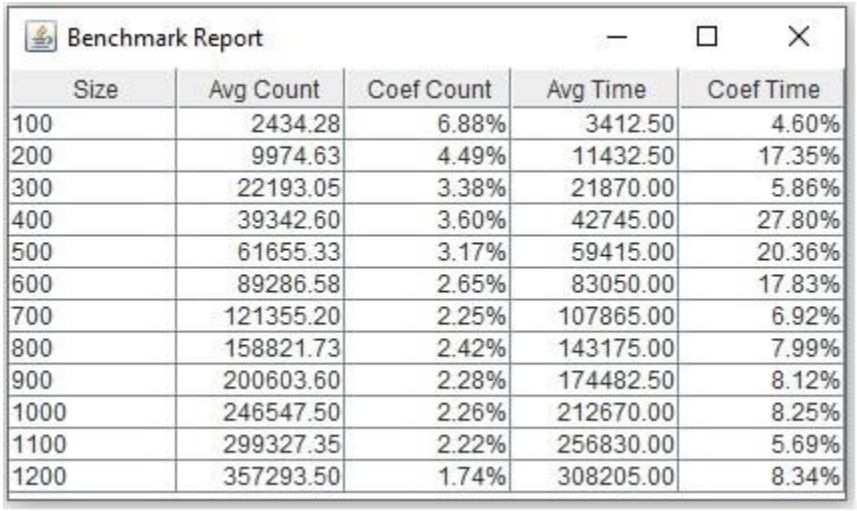
The purpose of the methods in the abstract class `AbstractSort` are as follows:

- The method `sort` is an abstract method that must be implemented in both of your classes that contain the sorting methods that you have selected
- The `startSort` method should be called before the sort begins and it should initialize the counter and record the starting time of the sort
- The `endSort` method should be called after the sort ends and it should compute the elapsed time of the sort
- The `incrementCount` method should be called whenever the critical operation that you selected is executed and it should increment the critical operation counter
- The `getCount` method should return the final value of the counter
- The `getTime` method should return the elapsed time

The output files should contain 12 lines that correspond to the 12 data set sizes. The first value on each line should be the data set size followed by 40 pairs of values. Each pair represents the critical element count and the time in nanoseconds for each of the 40 runs of that data set size. The values on each line should be delimited by spaces.

The second program (Report.java) should produce the report. It should allow the user to select the input file using `JFileChooser`. The report should contain one line for each data set size and five columns and should be displayed using a `JTable`. The first column should contain the data set size the second the average of the critical counts for the 40 runs and the third the coefficient of variance of those 40 values expressed as a percentage. The fourth and fifth column should contain similar data for the times. The coefficient of variance of the critical operation counts and time measurement for the 40 runs of each data set size provide a way to gauge the data sensitivity of the algorithm.

Shown below is an example of how the report should look:



Size	Avg Count	Coef Count	Avg Time	Coef Time
100	2434.28	6.88%	3412.50	4.60%
200	9974.63	4.49%	11432.50	17.35%
300	22193.05	3.38%	21870.00	5.86%
400	39342.60	3.60%	42745.00	27.80%
500	61655.33	3.17%	59415.00	20.36%
600	89286.58	2.65%	83050.00	17.83%
700	121355.20	2.25%	107865.00	6.92%
800	158821.73	2.42%	143175.00	7.99%
900	200603.60	2.28%	174482.50	8.12%
1000	246547.50	2.26%	212670.00	8.25%
1100	299327.35	2.22%	256830.00	5.69%
1200	357293.50	1.74%	308205.00	8.34%

On the due date for project 1, you are to submit a `.zip` file that includes the source code for both programs. All the classes should be in the default package.

You must research the issue of JVM warm-up necessary for properly benchmarking Java programs and ensure that your code performs the necessary warm-up so the time measurements are accurate.