2 Portfolio Problem

Goals

The goal of this assignment is to show you the importance of *stress tests* that measure the program performance and efficiency and can uncover potentially fatal flaws.

Hints

Some or all of the following interfaces and classes are likely to prove useful. In the *java.lang* package: *Comparable*, *Iterator*, *Collections*.

Stress Tests

The code for this assignment consists of three packages: *datasets*, *algorithms*, and *timertests*. This organization allows us to understand better the structure of the code, to see which classes are related, and to hide from the user the implementation of some of the classes.

The *datasets* package contains the classes that represents city data - as a recursively built list, and as *ArrayList*as well as three implementations of a *Comparator*: by he name of the city (and it state/zip), by zip code alone, and by latitude..

The *algorithms* package is given as a library *.jar* file. The provided documentation tells you what classes it contains and what public methods and fields are available for you to use. It contains several different implementations of sorting algorithms. Each of them consumes *City* data generated by a *Traversal* and produces an *Traversal* that generates the sorted data. The *init* method first copies the data into an internal data structure and the *sort* method then performs the sorting algorithm.

Your task is to play a detective: use the *timertests* package to set up and run the different algorithms, measuring the time needed to perform the tasks for different sizes of data.

Record your results in a table that specifies the algorithm tested, the size of data that was sorted, the *Comparator* used to determine the ordering (by zip code, by state, by latitude), and the time needed to perform the sorting. Highlight the anomalies in your test results and try to explain the reason for them.

Write the results down as a professional report.

Optional extra credit

For an extra credit, you may add your implementations of some of the following algorithms:

1. merge sort with immutable *AList*s
2. *in place* mutable merge sort using two *ArrayList*s
3. heapsort with *ArrayList* implementation
4. binary tree sort

Detailed description of the design of TimerTests

The files you need to read and edit to perform the timer tests are in the package *timertests*. You also need to know how the abstract class *ASortAlgo* is defined.

The class *DataSet* allows us to save each set of data to be sorted and supply the data to the sorting algorithm through a *Traversal*. That way all algorithms that sort a dataset of size 2000 organized randomly will sort the same data, making our comparison more accurate.

The constructor asks us to specify the expected size of the dataset and the expected organization. It then provides a method *add* for adding the data items in a sequential manner. The creator of the dataset is responsible for generating the data in either sequential or random manner. The creator of the data set is also responsible for adding as many data items as has been specified in the constructor.

The class *TestData* is constructed with the initial dataset of all 29470 cities and uses that to generate an array of 11 *Dataset*s that will be used in the tests (5 random at sizes 1000, 2000, 4000, 8000, and 16000 and 6 sequential ones at sizes 1000, 2000, 4000, 8000, 16000, and 29470);

The class *Result* represents one timing measurement for our *TimerTests*. It records which algorithm was used to perform the sorting, which *Comparator* was used to determine the sorting order, what was the size and the organization of the dataset that was sorted, and what was the measured runtime.

You should use all of these variables in your comparisons of various results. To run all tests would require that you run 5 algorithms with 3 comparators for each on each of the 11 datasets (5 random at sizes 1000, 2000, 4000, 8000, and 16000 and 6 sequential ones at sizes 1000, 2000, 4000, 8000, 16000, and 29470) for a total of 5 x 3 x 11 = 165 possible results. Some of the algorithms cannot handle such large datasets - it is your job to find out which ones.

Finally, the *TimerTests* class actually runs the timing tests. It runs each algorithm on the selected data sets. Before looking at how this work is organized, we need to know what methods and fields are provided by the *ASortAlgo* abstract class that all algorithm classes extend.

The abstract class *ASortAlgo* contains two fields, the *Comparator* used for determining the sorting order and the name of this sorting algorithm to report at the end of the test. It has two abstract methods, *initData* that consumes a *Traversal* and is used to initalize the data that the algorithm will use, and the method sort that takes no additonal arguments (besides this) and produces a*Traversal* for the sorted data. Our timing will only measure the time needed for this step.

The *TimerTests* class first initialzes an *ArrayList* of *Comparators* we will use. The method *runAllTests* organizes which test will be run by first creating the *ArrayList* of *ASortAlgo* objects, each representing an algorithm with a specific *Comparator*. (It can build all 15 possible combinations, but we can omit some algorithms by commenting out the appropriate lines in the *makeAlgos* method.) It also builds a list of indices for the test data *ArrayList* allowing us to choose which datasets should be sorted. For each combination of algorithm/comparator + dataset it then invokes the *runATest* method that performs one measurement and produces the *Result* object. All *Results* are collected in an *ArrayList* and printed in the *Console* after completion of all tests.

The *runATest* method first invokes for the algorithm/comparator being tested the *init* method providing as argumeant the *Traversal* for the dataset to be sorted. It then starts the timer, invokes the *sort* method and checks the elapsed time when the sort is completed. To make sure the sorting has been performed correctly, it verifies that the resulting data has been sorted before reporting the result. If the resulting data is not sorted, it throws the *UnsortedException*.

Files in the package *timertests*:
DataSet.java
Result.java
TestData.java
TimerTests.java
UnsortedException.java