CSI: Temple

Daniel Yee
CIS 2168

Purpose

We have learned that sorting is one of the fundamental algorithmic problems in computer science and is still a current research field. Empirical and theoretical measurements claim that as many as 25% of all CPU cycles are spent sorting. There are numerous algorithms that can be used with their own performance signatures – some are performant on small collections, while others are prohibitively non-performant.

In this assignment, you will implement several of the sorting algorithms discussed in class to further help you understand how they work. The second part of the assignment will allow you to analyze and compare the run-time for each algorithm.

NOTE: You are required to test another student's algorithms and present your findings in a meaningful way.

Background

Way back in 2000, the TV show *CSI: Crime Scene Investigation* premiered on CBS. The show featured a team of crime scene investigators in Las Vegas as they use evidence to solve the murder. While much of the show involved "Hollywood magic" and distorted the true nature of crime scene investigation, the show was popular and lasted for 15 seasons.

The Scenario

Welcome to *CSI: Computer Science InvestigationI*. You've come highly recommended so I have no doubt that your skills will make this training assignment easy.

The Crime Scene

Your first task is to set up the crime scene for your fellow computer science investigators to examine during their training assignment. You will need to implement Insertion Sort, Quicksort, and one other sorting algorithm from the list below:

- Shellsort
- Mergesort
- Heapsort
- Timsort

For the benefit of your future performance with CSI, I strongly suggest that you first try to implement the algorithms without any help. Then, if you find that you need a refresher on the algorithms, please review this week's briefing (a.k.a. lecture) or your CSI manual (a.k.a. textbook). Remember that if you do use code from the book, class, or outside sources, you must cite the source.

Once you have implemented the algorithms and tested them, modify each sorting algorithm to keep track of the number of comparisons performed, the number of exchanges performed. These should be returned in a Map<String, int> from the methods for easier evidence collection.

Finally, rename your algorithms to *Sort1*, *Sort2*, and *Sort3*.

Evidence Collection

You will need to find a fellow computer science investigator to examine your mystery sort methods. But before you do, you will need to be able to examine the methods yourself. So, before finding a fellow computer science investigator, write your test class and test method(s). You should have, at minimum, a *runTests()* method.

As the investigator, you will need to determine which algorithm that each mystery sort is implementing. You should examine each mystery sort method for its runtime characteristics. To test, give your testing class to your partner who will place it in their project and run it. Your partner will take the output of your test and give it back to you.

Test each algorithm on collections of varying sizes. Good tests have many data points of varying sizes and elements. The results of your tests should be outputted to a file (.csv if you plan on using Excel in the next step).

Courtroom Presentation

A CSI will sooner or later be called into the courtroom to testify on their findings. Your objective here is to present your data so that the jury can understand the results of your investigation (which method implemented which algorithm and why). You have a good amount of freedom in how to do this, but if you are collecting a lot of data on big sample sizes, I strongly suggest outputting your results to a file.

Some options include, but are not limited to:

- A well-formatted table
- Graphing your data (using Excel or some other program)

In your findings, be sure to indicate which of your fellow computer science investigators' methods you examined.

Summary of Tasks

1. Implement the required sorting algorithms
2. Modify the sorting algorithms to track and return comparisons and exchanges

3. Obfuscate the names of the algorithms
4. Implement a testing method(s) and store the results for later processing
5. Give your test code to a classmate to gather the evidence
6. Present your findings in a meaningful way

Hints

- Java API System.nanoTime()
- Export to a .csv file for easier importing into another program for data processing
- If you are having a hard time finding a classmate to for evidence collection, please see me or the TA immediately. Do not wait until the last minute!

Challenge

5 points

Implement Timsort (as presented in the book) as the third algorithm