Project 5 – Theory & Practice

Computer Science 2413 – Data Structures

Fall 2014

This project is individual work. Each student must complete this assignment independently.

User Request:

"Create a system to read, merge, purge, search, write, and sort bibliographic data efficiently. Then test that system to evaluate its efficiency."

Objectives:

1.	Ensure that your program meets all sorting, merging, and purging requirements for Project 4.	*5 points*
2.	Test the time to sort the initial bibliographic entries in a vector by name.	*25 points*
3.	Test the time to merge bibliographic entries in a vector sorted by name.	*25 points*
4.	Test the time to remove bibliographic entries from a vector sorted by name.	*25 points*
►	Develop and use an appropriate design.	*10 points*
►	Use proper documentation and formatting.	*10 points*

Description:

For this project, you will create PST (for PublicScholar Tester) to test the runtime performance of the algorithms you incorporated into PublicScholar4.0 and compare them to the theoretical runtime performance discussed in your analysis in Project 4. This includes the time for sorting the initial vector of entries, the time for merging new bibliographic entries into the database, and the time for purging bibliographic entries from the database. PST will use your sorting, merging, and purging code from PublicScholar4.0 but add new code to record the actual runtimes observed for the sorting, merging, and purging operations plus code to rerun the tests several times to get reliable data.

Operational Issues:

PST is not meant for end users beyond you, the software developer. Instead, it is intended to help you understand if your code is performing according to theory. As such, it does not need to have a user interface, per se. Instead, it should be possible for the user to simply run PST, wait for it to complete, and have it output its data in a human-readable format.

Implementation Issues:

Because the intention is to compare the performance of your sorting, merging, and purging operations to their theoretical Big O running times, you need to ensure that PST tests these algorithms for different values of *n*, that is, different numbers of bibliographic entries. The exact values of *n* you test is up to you to decide but you should test at least ten different values of *n*. (For example, you might use *n* = 1, 2, 4, … , 512, or you might use *n* = 100, 200, 300, …, 1,000.) You might implement this by having ten (or more) different files that PST reads from, each containing *n* bibliographic entries for your various values of *n*, or you might have a single file that PST reads from but uses only *n* entries from that file for your various values of *n*. **However you choose to handle your data files, you must ensure that you submit your files with your code in the D2L dropbox so that the TA can run PST on your files.**

Besides testing your algorithms for different values of *n*, PST should test each algorithm multiple times and provide both individual and summary data for each *n*. This is because, depending on how you are measuring time, the runtime you measure may be affected by many factors, such as other programs running on the same computer concurrently. For this reason, you should run each test with each value of *n* at least 10 times and report the 10 (or more) individual values measured plus some summary value(s) such as the median and/or the mean and standard deviation. In addition, you should interleave the runs for each value of *n* with one another (preferably in a random order), rather than doing all the runs for one value of *n* first, then moving to the second value of *n*, etc.

As PST tests your algorithms, it should test the individual parts of each algorithm separately. For example, if your merge operation performs an O(*n*) operation, followed by an O(*n* log *n*) operation, followed by another O(*n*) operation, PST should report on those three operations individually, rather than reporting the runtime for all three combined, so that you can determine whether the supposed O(*n*) operation really performed in O(*n*) time, etc.

All of the data that PST reports should be captured to a file and submitted along with your project in D2L. In addition, you should use the data reported by PST to create graphs plotting the measured runtimes of the operations and comparing them to the theoretical Big O runtimes you predicted for those operations in Project 4.

You should also turn in a report (typed up and saved in PDF) that includes the data from PST, the graphs you created, and an explanation of whether your algorithms performed as expected.

Due Date:

You must submit an electronic copy of your source code through the dropbox in D2L by **Friday, December 5th by 2:45pm**.

Notes:

In this project, the only libraries you will use are iostream, fstream, cstring, and whatever libraries you decide to use for measuring runtime.

Be sure to use good object-oriented design in this project. That includes appropriate use of encapsulation, inheritance, overloading, overriding, accessibility modifiers, etc.

Be sure to use good code documentation. This includes header comments for all classes and methods, explanatory comments for each section of code, meaningful variable and method names, consistent indentation, etc.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.