COMP161 - Project 2 - Complexity and the Real World

Spring 2016

For your final project you'll carry out empirical studies of key searching and sorting algorithms in order to better appreciate the interplay between complexity and real world performance. The ultimate goal is the collection of execution time data and the presentation of that data in the form of three mini-posters.

The Data Set

One of the primary goals of this project is to generate data sets that allow you to visualize and explore the running times of classic searching and sorting algorithms. We'll look at the following set of procedures:

- binary search
- *binary_search* from the C++ *algorithm* library
- linear search
- *find* from the C++ *algorithm* library
- mergesort
- quicksort
- *sort* the C++ *algorithm* library
- selectionSort
- insertionSort

This list includes procedures we'll develop and analyze in class and lecture notes as well as C++ standard library implementations. This lets us look at the efficiency of DIY implementations versus well engineered libraries. Additionally, the complete set of algorithms represents an important series of complexity classes[1] and should allow us gain some appreciation for the order of magnitude differences these classes represent.

[1] $O(\log n), O(n), O(n \log n), O(n^2)$

Execution Times

Your first data set is the execution time for the theoretical worst case and five randomly selected procedure inputs for each of the 9 algorithms listed above and for each of the following input sizes: 10, 50, 100, 500, 1000, 5000, 10000, 25000, 50000, 75000, 100000, 150000, 200000, and 250000. When explicit generation of a worst-case generating input is not feasible or possible, then another random input should be used in its place. To be clear that's 14 sizes, 9 algorithms, and 6 executions per size-algorithm combination for a total of 756 timed executions[2].

Gathering data on only 6 cases is, for most of the sizes we're looking at, too little data. We're being hyper conservative about scaling programs up to larger numbers of executions as the input vectors get bigger. Several of the procedures that you'll be dealing with can and will take on the order of minutes to execute. So, a procedure that takes 5 minutes repeated 20 times will take over an hour to run. Given that we're all working off the same server, we'd like to avoid overloading the server with a class sized set of compute jobs each needing hours of compute time. This is also the reason we're not looking at writing one uber-program that gathers all of our data in a single execution

Posters

Posters are common means of presenting information in the sciences. A lot of work can go into making high quality posters. The posters you'll be creating are not meant as a complete poster in the traditional sense. They will simply present the data without the usual introductory material and without drawing conclusions from the data. In short, we're simply presenting the results and the methodology by which they were gathered so that readers can draw their own conclusions.

To tell the complete story of our data we need to combine both images and text. Posters are a simple and effective way to do this. You'll need to develop three 11x17 posters:

1. A poster that presents a table of worst case execution times for all algorithms and all sizes
2. A poster that presents point plots of all collected execution times for the four search algorithms
3. A poster that presents point plots of all collected execution times for the five sorting algorithms

Each poster should stand on its own. Towards this end, you must include the following items in each poster.

- A Title
- Your name and affiliation[3]

[2] 84 per algorithm

[3] Monmouth College, Department of Mathematics and Computer Science.

- Some text that briefly describes the data, the visualizations, and the overall design of the visualizations so that the reader knows how to read and interpret the graphics[4].
- The visualizations, each with a caption that includes a figure number, title, and brief description.

The Visualizations

Our posters require the following visualizations, which we'll generate with Mathematica.

- A single table in which each row contains the maximum times for each procedure on a given size. The first row of data should be the times for the smallest size and the last row should be the times for the largest size. This table lets us easily compare the worst case execution times for the nine different procedures.[5].
- Nine point plots, one for each of the nine algorithms. These show all the times gathered for each procedure as a point in 2D space where the x-coordinate is vector size and the y-coordinate is execution time.

The Profiling Programs

The code you're writing for this project is used to gather execution times for our set of nine procedures. The complete program has four components:

1. A library of the 7 algorithm implementations discussed in the lecture notes, i.e. the 7 non-C++ standard library procedures listed in the beginning. Much of this work will be done as a class and in lecture notes.
2. A library for generating random *vector<int>* data, algorithm specific worst-case data, and reading some vector data to files . Much of this was carried out in*labp2*.
3. A series of 9 programs, one per procedure, used to gather performance data.
4. A Mathematica notebook used to process data and generate visualizations.

Algorithm Library

The first library should contain six procedures[6]– one for each of following algorithms on a *vector<int>* variable.

[4] Note we're describing what should be read from them, just how to read them. We leave conclusions to the reader

[5] A similarly structured table for average times would also be really interesting, but we'll focus our energies on worst case execution for now

[6] and any helpers needed to implement those procedures

- binary search
- linear search
- selectionSort
- insertionSort
- mergesort
- quicksort

These will be discussed in class and in lecture notes. Some code will be provided for you. You just need to be certain you have access to these procedures, that they work as specified, that you know how to use them, and that you understand their complexity analysis.

Random Vectors and Files Library

The second library is used to generate data for testing and writing test results to files for processing with Mathematica. It should be comprised of the following procedures:

1. *rand_ints*
   A procedure for computing a random permutation as a vector of integers.
2. *sorted_ints*
   A procedure for computing a vector of integers in greatest to least order.
3. *write_ints*
   An output procedure which takes a vector of integers and an output file stream and then writes the contents of the vector to the file as comma separated values.
4. *write_times*
   An output procedure which takes a vector of durations and an output file stream and then writes the durations, as seconds in double format, as comma separated values to the stream.

These three procedures can be used to generate and report on all the test data we need.

Profile Data Gathering Programs

Your need to write 9 programs, one per procedure. This lets you incrementally work through the algorithms and gather data as you go. Your goal is to get data and visualize that data. Work one procedure at a time, get the program written, gather and visualize the data, then move on to another procedure. Do not wait to get and work with timing data until you have a complete set. There is going to be a lot of code shared between these programs, so you'll likely save some time by copying and modifying as needed.

The goal of a profiling program is to produce a csv[7] file with 15 lines of values on it. The first line is the sizes to profile the procedure. The remaining 14 lines are the execution times for each size. You should profile the procedure in smallest to largest size order. This means the second line of your file contains the times for the smallest size profiled and the last line of your file contains the times for the largest size profiled.

Mathematica Notebook

You should us a single Mathematica notebook to process all of your data.

Logistics and TLDR

- *Lab 4/20* - Labp2 starts work on some library procedures
- *Lab 4/27* - Open lab time to work.
- ***Wednesday 5/4*** Program code, data files, Mathematica Notebook, and digital copies of posters submitted via handin as *proj2*.

[7] comma separated values