EECS 268: Spring 2009
Laboratory 8: Time Complexity of Sorting Algorithms

Due: 11:59 pm the day before your lab meets the week of April 20th

Lecture Topics: Algorithmic Efficiency and Sorting
Lab Topics: Timing Measurement of Program Behavior, Random Number Generation
Outside Topics: Data Presentation

1. Introduction

Through this lab, you will examine a number of sorting algorithms, specifically: (1) selection sort, (2) insertion sort, (3) bubble sort, (4) mergesort, and (5) quicksort.

You will chose four of the five algorithms to evaluate. Note, however, that your "Sort Algorithm Tester" program should be able to run all five algorithms.

You will empirically evaluate the time complexity of these algorithms in terms of best, worst, and average case datasets. This will be accomplished by measuring and comparing their actual performance on a variety of specifically constructed data sets. You will also look specifically at the quicksort algorithm and evaluate how the choice of pivot affects performance.

Note: The source code for these algorithms is provided on the lab website. However, only the C++ source (*.cpp) for the algorithms is provided in the separate source files. Because your testing code will need to use the provided sorting algorithms, you will be required to create a header file that provides the function prototype for each of the sort algorithms. Your testing code will then, of course, be required to include this new header file in order to use the sort functions.

Also note that you should create a makefile that builds each sorting algorithm separately as a object file (g++ -c). When producing the final executable, the makefile will then need to include the object file of each sort algorithm during the final linking stage.

2. Building the "Sort Algorithm Tester"

For this project, you will first create a utility program that generates a specified number of floating point values in a predefined order. These values will then be sorted according to the sort algorithm specified on the command line. The array values will be generated in one of 3 ways:

- *Ascending* – The first element in the array contains the smallest value and the values incrementally increase, with the largest value being in the final position of the array.
- *Descending* – The largest value is the first element of the array and the values decrease until you reach the last element.
- *Random* – Each element in the array contains a randomly generated value as described below.

Your application will receive from the command line (using argv[]) the following input parameters:

- The size of the data set to be tested,
- The initial order of data to be included within the data set,
- The algorithm you wish use to sort the data set.

To execute your program, you will pass in a number of parameters in the terminal to specify how the program will execute. The valid options for each parameter:

```
$ ./main <data_size> <random|ascending|descending> <selection|insertion|bubble|merge|quick>
```

Meaning your program could be run as follows:

```
$ ./main 1000 random insertion
$ ./main 10000 ascending merge
etc.
```

2.1 Data Generation

Given the data set size and the data order requirement, you are to dynamically allocate an array of the appropriate size and populate it with the appropriate data. For the "random" arrangement of the test data set, you must generate floating point numbers in the range $0.0 \leq x \leq 100.0$. Details of the random number generation process will be given in lab.

2.2 Algorithm Execution

Given the desired algorithm and your array of test data, you should then execute the specified sort algorithm on the test data. Your program must be able to execute all of the five sorting algorithms listed above. For execution of the quicksort algorithm, you can use the default pivoting scheme for this section of the project. In Part 3, the second component of this project will be detailed in which two other pivoting methods are implemented and compared.

2.3 Timing

The main purpose of this lab is to experimentally measure the time complexities of the sort operation as performed by each algorithm. Obviously, the measured behavior is expected to match the theoretical complexities for each algorithm. However, you may find that at first this is not true. Learning to measure program performance and correctly interpret the data is not too hard, but it is often a bit harder than it first appears. To that end, you will add the appropriate code to your program so that you can measure the total amount of time taken to sort the array by the specified algorithm. Be certain to *exclude* from your measurement of sort timing the amount of time taken to generate the array of test data to be sorted. More information on timing measurement—such as averaging across multiple runs—will be provided in lab.

2.4 Evaluation

You will collect timing information on each of the sorting methods using various sizes and different initial array orderings. For each data set ordering: ascending, descending, and random, you will record the execution time of four sorting algorithms, using each of the following data set sizes: 100, 1,000, 10,000, 50,000, and 100,000.

For example, for the insertion sort algorithm, you will test the execution time of sorting an ascending array of size 100, then of size 1,000, 10,000, 50,000, and finally 100,000. You will then test the same sizes for an array sorted in descending order, and finally do the same for a randomly ordered data set.

In total, each of the four algorithms will have 15 time values associated with them, 5 data set sizes for each of 3 data set types. Thus, the total number of experiments run for this project will be 60. Writing a simple Bash Shell script to run your experiments may be helpful. This information will be communicated in the form of a report as described in Part 4.

3. Comparison of Pivot Point Selection in Quicksort

The second evaluation part of this project will be to implement two additional pivoting methods to be used in the quicksort algorithm. These two schemes are as follows:

- *Median of Three* – discussed in class.
- *Random* – a random number generator will be used to select an index into the subarray you are examining. You must ensure that this index is within the range of the current subarray.

To indicate which pivot selection to use, you will allow a fourth input parameter to be passed into the application via the command line. This parameter will be optional and only utilized when quicksort is the algorithm used for sorting. The valid options for this new input parameter will be:

- *default* – indicates to use the default pivot method.
- *median* – indicates to use the median of three pivot method.
- *random* – indicates to use the random pivot method.

So you could still call the program with only three input parameters:

```
$ ./main 1000 random insertion
```

But, if quicksort is the algorithm used, then the fourth input parameter can be used:

```
$ ./main 1000 random quick median
```

You can may also allow quicksort to be run using the "default" pivot method if no fourth parameter is used. The following example would call the quicksort algorithm using the "default" pivot method:

```
$ ./main 1000 random quick
```

3.1 Evaluation

For each of the three pivot methods, you will gather timing information for the three data types (random, ascending, descending) and for the five data sizes (100, 1000, 10000, 50000, 100000). This information will be used in the report as described below.

4. Report

The timing information collected in this project will be combined together in the form of a report. This report will detail your findings and include explanations for why the algorithms performed as they did, as well as how these algorithms compared to each other. The report will be broken down into two sections, corresponding to the two experiments of this project.

Section 1: A Comparison of Sorting Algorithms.

In this section you will write an analysis of what you found during the timing experiment. Discuss which algorithms performed better with smaller data sets versus larger data sets. Also, write a summary about how each of the algorithms performed given the initial order of the data set, including which data ordering demonstrated the best or worse case of a given algorithm where appropriate. In your discussion, consider specific properties of each algorithm and why you feel that they may have contributed to the observed results.

Your analysis should include a number of graphs comparing the performance of the various algorithms. You should create relevant graphs that illustrate the trends you discuss and which thus allow you to communicate effectively about the performance of your algorithms.

For example, you might want a graph for each sorting algorithm that compares the time required for each of the three different types of data arrays for all sizes of arrays. In this graph you would have a line for each of the three types of data sets, with the x-axis indicating the size of the dataset and the y-axis relating the time it took for each test to run.

You could also generate graphs that allow for a more direct comparison between the various sorting algorithms. For example, you could create graphs that displayed the timing differences between the four different algorithms for a particular data set type. In this case you would have four lines with the x-axis again being size and the y-axis being time. You would need 3 separate graphs corresponding to the three data set types examined in this lab.

Include these graphs within the report as you discuss the sorting algorithm behavior illustrated by the graphs. Also provide the spreadsheet or other source you used to create the graphs in your submission for this project.

Section 2: A Comparison of Pivot Selection in QuickSort

Here you will provide an analysis similar to that of Section 1 considering effects of the quicksort pivot selection. Discuss the performance of each pivot selection with respect to the initial order of the data set and the data set size. Is there one pivot selection method that you would choose? If so, which? If not, why not?

You will also include graphs to display the results of these tests, as you did in Section 1. Include in your spreadsheet for these graphs as well.

5.1 What to turn in

Include in your submission for this project:

- The source code used for your timing tests including the original algorithm code.
- The report with the included graphs and discussion.
- The spreadsheets used to create graphs, including all the raw data collected for your experiments.

7. Grading Criteria

20 Sort Algorithm Tester

- Dynamically allocating memory and generating data.
- Timing the sorting algorithm.

10 **Pivot selection algorithms**
- Implementing median of three and random pivots.

30 **Report – Section 1: A Comparison of Sorting Algorithms.**
- Analysis and comparison of sorting algorithms
- Graphs that support your conclusions

30 **Report – Section 2: A Comparison of Pivot Selection in QuickSort**
- Analysis and comparison of pivot selections
- Graphs that support your conclusions

10 **Programming Style & Output Formatting**
- Use standard paragraphing conventions and design practices.

100 **Total**