

LABORATORY EXPERIMENTS IN AN ALGORITHMS COURSE: TECHNICAL WRITING AND THE SCIENTIFIC METHOD

Jeff Matocha¹

Abstract *¾ Technical writing and use of the scientific method are two skills necessary for Computer Science students. This paper presents a set of laboratory experiments for students to perform experimental verification in the context of an Algorithm Design and Analysis course. These exercises focus on experimental design, the scientific method, data presentation, and technical writing. Although students have been exposed to the scientific method, many need practice in designing and conducting feasible experiments. In addition to example lab exercises, we attempt to point out some common problems encountered in a recent course using this type of exercises.*

Index Terms *¾ Algorithm Design and Analysis, Laboratory Experiments, Scientific Method, Technical Writing.*

INTRODUCTION

Most liberal arts colleges require science courses of all majors. This requirement ensures that students have exposure to the scientific method. Xavier University's Core Curriculum seeks to provide students with "acquaintance with the scientific method" [9].

In Computer Science, we expect students to have more than a cursory knowledge of the scientific method; in fact, the recently approved draft of the Computing Curricula 2001 says the following, "To develop a firm understanding of the scientific method, students must have direct hands-on experience with hypothesis formulation, experimental design, hypothesis testing, and data analysis. While a curriculum may provide this experience in various ways, it is vital that students must 'do science' - not just 'read about science.'" [2] ABET's Computing Accreditation Commission (CAC) requires "science courses or courses that enhance the student's ability to apply the scientific method," as a part of their accreditation criteria [1]. Many times, Computer Science curricula do not explicitly include coverage of the scientific method, but rather leave coverage of the method to required natural science laboratory courses. The Computing Curricula 2001 requires the introduction of "empirical measurements of performance" in Data Structures (CS103I) [2]. Although this paper describes a way to provide working experience with the scientific method in an Algorithm Design and Analysis course (CS210) [2], most of the discussion would also apply to a Data Structures course.

Liberal arts colleges also require English composition courses to give students "the ability to write ... effectively" [9]. Writing skills are also required as a part of a Computer Science curriculum by both the Computing Curricula and CAC criteria. "Communication skills should not be seen as separate but should instead be fully incorporated into the computer science curriculum and its requirements." [2] "The written communication skills of the student must be developed and applied in the program." [1] Requiring students to develop the laboratory reports presented in this paper satisfies requirements for technical writing experience.

In addition to the scientific and communicative goals of laboratory exercises, the labs reinforce algorithmic concepts. "Nothing burns in the difference between a quadratic algorithm and an $n \log n$ algorithm quite like watching the second hand on the wall clock crawl while a bubble sort runs." [6]

This paper presents several sample labs as well as an outline for lab report preparation in the section entitled Description of Sample Laboratory Experiments. The section entitled Goals discusses the overall goals of laboratory exercises. In the section entitled Observations, we discuss some observations made when this type of lab was required in a recent course offering. We end the paper with our Conclusions.

DESCRIPTION OF SAMPLE LABORATORY EXPERIMENTS

This section describes a set of laboratory exercises. We present these as a starting point to give a general idea of the expectations. Also, we list more labs than one might assign during a single semester. This is simply a list of labs conceived since beginning to use this approach.

As a first laboratory, due to the potentially large set of new concepts, we suggest giving students some working code and asking them to draw conclusions about it. One such example appears in [5], in which students are given code and executables and must determine which code matches which executable. Four algorithms are presented to solve maximum segment sum for theoretical analysis; additionally, executables for each of the algorithms are provided and students must determine, experimentally, which binary matches which algorithm [5]. This lab would exercise many of the goals described in the section entitled

¹ Jeff Matocha, Xavier University of Louisiana, Computer Sciences and Computer Engineering, 1 Drexel Drive, Box 50A, New Orleans, LA 70125-1098
jlmatoch@xula.edu

Goals, but would reduce the lab's complexity by not requiring students to develop code or to learn how to perform code timing. Running code gives students the ability to focus on analysis and writing.

A follow up lab could be constructed that would include the requirement of code timing. Object and header files containing functions to be timed could be provided. Students would have to develop a small amount of code and understand the issues with timing described in the subsection entitled Code Timing.

Additional labs include:

- **Determine the relationship of the height of a randomly generated binary search tree (BST) to the number of nodes.** Textbooks often include implementations of `BSTinsert` and `BSTdelete` in a data structures section. To gather data, a student must add a `BSTheight` function (good recursion practice).
- **Show the relationship between the running times of an iterative and recursive version of a particular function.** Students should understand the overhead inherent in function calls. On the other hand, students should appreciate the tradeoff that recursive functions are often more elegant.
- **Try to enhance an asymptotically slow algorithm to perform better than an asymptotically faster algorithm.** Students could be instructed to write an algorithm (e.g., Bubblesort) in assembly language and another (e.g., Mergesort) in Java and compare the times of the two. This lab reinforces the fact that fast computers can not compensate for slow algorithms. Another possibility is to use an optimizing compiler (e.g., gcc's `-O3` option) to generate faster code for the asymptotically slower algorithm.
- **For what values of n does Bubblesort have a shorter running time than Mergesort?** This lab helps students to understand asymptotic bounds. The results illuminate the clause in the asymptotic definitions about n_0 . Also, students should appreciate that Mergesort is not *always* the best choice for sorting.
- **Compare the running times of Mergesort, Quicksort, and Bubblesort on arrays varying in their "sortedness".** Building on the popularity of sorting, this laboratory allows students to see the algorithms' best and worst cases. The "enhanced" Bubblesort, that runs in linear time for sorted arrays, should be used. Students can be allowed to devise their own method for varying the "sortedness". A followup lab would vary the size of the array, as well as the array's sortedness, to create a 3-dimensional data plot.
- **Determine the compression savings using Huffman codes on ASCII and binary data.** This lab largely tests students programming skills. An implementation of the greedy Optimal Merge Patterns algorithm is provided and all other code must be developed. This

laboratory may be students' first exposure to bitwise processing.

Of course, laboratory exercises could examine the experimental memory usage of an algorithm. Space complexity is typically not done for two reasons: most algorithms texts treat storage complexity lightly and determining the memory usage of a process is more complex than determining its running time.

Over the progression of lab experiments, we expect that students will be required to take more responsibility for the design of the experiment. With this responsibility, justification for their experimental methodology must be more extensive than in earlier laboratories.

The general format for a lab report resembles the format of science fair projects, following the scientific method.

1. **Background Research:** This section describes the problem and what is known from the text, class, and other sources.
2. **Hypothesis Statement and Justification:** The student's prediction of the outcome, *determined before data is collected*, is presented and justified.
3. **Experimental Methodology:** Students must describe the manner in which the data was created. This section should include annotated code fragments and the entire code that was used in the experiment. Justification for data structures and other methods should be provided.
4. **Data Presentation and Analysis:** Data that supports or refutes the hypothesis is presented using appropriate means (e.g., tables, 2-d or 3-d graphs) in this section. Any data considered outlying should be explained.
5. **Conclusions:** This section consists of a discussion of how the data supports or refutes the hypothesis. If the data refutes the hypothesis, the data should be explained and a new hypothesis formed. Reflections on the lab as a whole should be included in this section and potential problems and enhancements to the experiment presented.

GOALS

In a scientific laboratory approach to Algorithm Design and Analysis course, we suggest a gradual introduction since each lab encompasses many potentially new concepts. In addition, many of these skills are not trivial (e.g., technical writing). In this section, we present some of the potential activities that are exercised using the laboratory approach and discuss the benefits. These include exposure to the scientific method and technical writing as described earlier, as well as experience with problem solving and code timing.

Scientific Method

The scientific method is one major focus of these exercises. Although the scientific method is typically taught in middle school and high school, students have been guided through the process. They must move beyond the memorization of

the method's steps to using the steps to actually *perform* science. These labs require the use of sound scientific and statistical methods (e.g., executing multiple trials, the need to explain aberrant results). The outline of a lab report in the Description of Sample Laboratory Experiments section can be used as a basis for discussion of the scientific method.

In performing a lab experiment, a student forms a hypothesis and must determine how to arrive at data to either support or oppose that hypothesis. This planning requires a backward reasoning approach to the experiment. First, the student must envision the final data needed to support the hypothesis. Then the student must create an experiment that will provide data of the sort expected in the first step.

Of course, when the data has been generated, it may differ from one's expectations. This stage of the experiment is difficult. The disparity, which can be classified by one of the following, must be either corrected or explained.

- **Implementation errors:** An erroneous program must be modified since data from an incorrect implementation is potentially meaningless.
- **Timing problems:** An inconsistent load on timesharing systems or insufficient running time in relation to the clock granularity can cause imprecise data. Timing of a larger number of iterations should correct most granularity problems. The subsection entitled Code Timing discusses timing functions, clock granularity, and system load.
- **Input data problems:** Of course, if the input data provided is incorrect, useful data must be generated to test the desired behavior of the algorithm.
- **Incorrect hypothesis:** The only *acceptable* disparate data occurs when it is possible to explain that the original hypothesis is incorrect. In this case, the hypothesis should not be modified, but a modified hypothesis based on the new understanding should be presented in the conclusions of the lab report.

In addition to predicting the final data's pattern, a student must be able to interpret that data. When an algorithm is analyzed asymptotically, details, such as constants and low order terms, are ignored; the timing data from an actual run contains all such details. A student must correlate experimental data with theoretical predictions [5]. Some of the variation between the two sets of data can be easily explained by dividing the actual running time by the theoretical running time. For instance, consider the following example from [7]. If we expect Quicksort to require $T(n) \leq c n \log n$ time for a particular set of data, the observed times divided by $n \log n$ should result in a constant (within a reasonable range) [7]

When performing analysis for labs, one often has more data than necessary. Although it is important to report all the data to ensure that nothing is hidden [7], a visual presentation of the data that supports or refutes the hypothesis is important. Determining which data to present and the presentation method to use is an important decision.

Are multiple 2-d data plots acceptable or would a single 3-d plot be more useful? Is the data continuous or discrete, requiring a scatter plot or a line graph?

If Microsoft Excel is used to present data, students must understand the difference between Excel's Line graph and Excel's XY graph; the Line graph uses the X-axis values as labels, whereas the XY graph uses the X-axis values as actual values. The gnuplot tool is a better choice for scientific data, with the ability to create 3-dimensional plots and more.

Technical Writing

Writing is a skill that is difficult but fundamental. Ways to include writing throughout a Computer Science curriculum have been presented by many authors [3][8]. Laboratory reports allow instructors a natural opportunity to include technical writing skills in an Algorithm Design and Analysis. The major sections of the report should focus on justifying the hypothesis, the experimental methods, and the data. The following items, taken from a list in [8] and intended as ideas for including writing in CS1, could act as a set of guidelines for lab reports.

- "Describe what you observe."
- "Discuss advantages of one approach over another."
- "Justify your answer."
- "Justify your selection of data structures or algorithm." [8]

A percentage of each lab grade should reflect a student's technical writing skills. "If CS teachers are serious about the importance of communication skills, why don't teachers take points off for writing errors in student work?" [8] Of course, technical writing is different from the writing that is taught in English Departments. We expect that the writing grades in early lab reports may focus on simple English skills (e.g., grammar and punctuation) and additional comments made about technical writing; as the semester progresses, simple English skills will be assumed, and errors greatly penalized, with the focus of grading shifting to technical writing skills.

Code Development and Problem Solving

Of course, in some laboratory experiments, we expect students will develop code to test their hypotheses. Although programming is not the focus, programming is a necessary skill; in fact, some laboratory experiments may not require a large amount of programming. For instance, we suggest a gentle introduction to labs using something like the approach in [5] as described earlier. This exercise focuses on correlating theoretical results with experimental results.

Other labs require students to test algorithms included in their text (e.g., comparing sorting algorithms), but some labs should require students to use an algorithm in a novel way or to solve a problem in a particular domain. With the wealth of code currently available on the Internet, we feel it

necessary for instructors to modify labs, creating unique problems to which solutions are not freely available. We must be sure that students perform the experiment rather than simply seeking results on the Internet.

Code Timing

Code timing, although simple in concept, causes many problems for students. Several issues, which naturally bring a discussion of topics from other courses, must be considered.

Using C in a Unix environment, one may use the `times` function like a stopwatch. This function “returns the number of clock ticks that have elapsed since an arbitrary time in the past.” [4] The number of clock ticks per second is usually available either by a call to the `sysconf` function or defined as a constant in `limits.h`. The `times` function may be called before and after the code in question. The results provide an elapsed time of running the code.

The granularity of the system clock is typically very coarse, therefore, code segments that run quickly register no elapsed time. The average time of many repetitions of the code is necessary to obtain reasonable results. Students must then determine *how many* repetitions are needed to result in reasonable precision. A discussion of clock issues can be related to topics in Computer Architecture (CS220) [2].

In addition to problems due to the coarse system clock, time sharing systems, like Unix, cause additional variance. A system’s load can greatly affect timing results, therefore multiple repetitions are necessary, even with a clock of acceptably small granularity, when using timesharing systems. Executions at certain times may even provide useless results; most important is that the load level be fairly consistent over the whole of the runs to result in meaningful data. Discussion of this problem can be easily related to Operating Systems concepts (CS225) [2].

OBSERVATIONS

This section presents problems that students encountered in performing laboratory experiments. These problems range from flawed logic to lack of understanding of the topics described in the section entitled Goals.

The timing issues described in the Code Timing subsection cause many problems. The fact that the same code on the same data can vary in its running time (due to granularity issues or system load) or that a section of code can run in less than one time unit causes major conceptual difficulties. After explaining that multiple iterations of a code section should be run and then an average time per run be calculated, many students submitted the next lab report with a loop surrounding the code, but instead of timing all the iterations as a whole (starting and stopping the “stopwatch” once), they accumulated the sum of the times to run the piece of code (starting and stopping the “stopwatch” inside the loop and averaging the results). A clear

explanation, possibly including a handout or an example, may be necessary to eliminate timing problems.

A lesser problem is the tendency of students to time the entire program rather than the algorithmically interesting section. Although the setup code typically does little to the asymptotic running time of a program, it adds difficulty in the analysis. For instance, if we are to time a sorting algorithm, it is not necessary to time the generation of the random array values. This seems simple, but sometimes requires more planning and upfront work. In our sorting example, multiple runs are necessary to combat the coarse clock and differing load levels, so many arrays should be filled *first*. Following the setup, the timing is started and the sorting function is repeatedly called with each of the preloaded arrays.

If a technical writing course is not a part of the required curriculum, it may be necessary to prepare a handout of the characteristics and common mistakes of technical writing. Students have little writing experience at the college level and the writing expected for lab reports is of a different style than that done in English courses. As stated in the Technical Writing subsection, a gradual approach to writing is recommended. Making a sizable percentage of the lab report grade contingent on writing ensures that students take writing seriously.

In some cases, students seem to have difficulty backward planning from the goal to the experiment. Some students tend to begin the program without understanding of the final goal, implementing the parts that are well defined (e.g., Bubblesort or a binary search tree). This tendency potentially stems from semesters of traditional programming classes. We, as instructors, must guide students to *first* form a hypothesis to predict how the data will look. Students should be encouraged to sketch graphs that they expect to produce as part of the Data Presentation and Analysis section before starting their experiment.

Another problem that may be a result of programming classes is students tendency to underestimate the amount of time necessary for data collection, presentation, and writeup. Since students have had many classes where a working program is the only deliverable, they must realize that the code is not the most important part of these exercises. In our use of this type of labs, reports are often warm from the printer when submitted.

Students must realize that the work they do to experimentally verify a hypothesis gives no *proof* of its correctness. Rather, the data obtained merely *supports* the hypothesis. Students often use these words interchangeably, but the words “support” and “verify” should be favored over the word “proves”.

CONCLUSIONS

Technical writing and use of the scientific method are two skills we should require of Computer Science majors. The laboratory exercises presented in this paper exercise these skills. We have presented a set of sample laboratory experiments, the set of goals for these labs, and some observations of potential problem areas. The author maintains a website which includes a completed lab report and a section where instructors can share labs they have developed at <http://webusers.xula.edu/jlmatoch/alglabs/>.

REFERENCES

- [1] ABET/Computing Accreditation Commission, *Criteria for Accrediting Computing Programs*, November 23, 2001.
- [2] ACM/IEEE Curriculum Committee on Computer Science, *Computing Curricula 2001*, December 15, 2001.
- [3] Jackowitz, P, Plishka, R, and Sidbury, J, "Teaching Writing and Research Skills in the Computer Science Curriculum", *SIG-CSE Bulletin*, 22, 1, February 1990, pp. 212-215.
- [4] Linux Manual Page, `man 2 times`.
- [5] McCloskey, R and Beidler, J, "An Analysis of Algorithms Laboratory Utilizing the Maximum Subset Segment Sum Problem", *SIG-CSE Bulletin*, 27, 4, December 1995, pp. 21-26.
- [6] McCracken, D, "Three 'Lab Assignments' for an Algorithms Course", *SIG-CSE Bulletin*, 21, 2, June 1989, pp. 61-64.
- [7] Morris, J, "Experimental Design", website at http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/Labs/exp_design.html
- [8] Walker, H, "Writing Within the Computer Science Curriculum", *SIG-CSE Bulletin*, 30, 2, June 1998, pp. 24-25.
- [9] Xavier University of Louisiana Catalog 2000-2002.