CMPSC 112
Introduction to Computer Science II
Fall 2016

Laboratory Assignment Six: Doubling Experiments for Inferring Time Complexities

Introduction

The current module of the course has focused on the importance and purpose of both empirical and analytical evaluations of algorithm performance. For this laboratory assignment, we will learn how to use a tool, called ExpOse, that leverages the results from successive doubling experiments to infer the "actual-worst-case" time complexity of an algorithm. Using examples provided with ExpOse, you will experimentally determine the actual-worst-case time complexity of five sorting algorithms and two algorithms for determining if there are no duplicate elements in an array. Ultimately, you work aims to experimentally confirm some of the analytical evaluations of an algorithm provided in your textbook and in online sources. Throughout this assignment, you will take additional steps towards seeing the connection between the experimental and analytical evaluation of algorithms. Finally, you will continue to practice the use of software engineering tools.

Review Your Textbook

To do well on this laboratory assignment, you should review the content about sorting an array in Section 3.1.2 (optionally, students may investigate a more advanced analysis of these algorithms by reading Section 9.4.1 of the textbook). Next, you should carefully review the content in Section 4.1, paying particularly close attention to the results in Table 4.1 and Figure 4.1. To learn more about the uniqueness detection algorithms studied in this assignment, please read Section 4.3.3, specifically noting Code Fragments 4.7 and 4.8. Additionally, you should also examine the slides that we have discussed during our recent class sessions. If you have questions about this reading assignment or the material that was presented in class, then please see the course instructor. If done appropriately, you may also post your question to the `#laboratory` channel of our Slack team.

Downloading the ExpOse Tool from GitHub

You should complete this assignment with a partner; please ensure that you are working with someone who is different than your previous partners. Both members of the partnership are required to complete all of the steps for this laboratory assignment. To start this assignment, you should go to the `https://github.com/kinneerc/ExpOse` web site and click the large green button to the right of the screen to see the SSH-based name for the repository. Now, use this repository name as the command-line argument to the "`git clone`" command that you run in your terminal window.

At this point, you will have downloaded the entire ExpOse system from the GitHub servers and saved it on your own computer. Each person is the partnership should ensure that they have their own download of ExpOse as there is no need for your partnership to store it in joint repository. Next, you should use GVim to study the source code in the `build.xml` file that comes with the ExpOse tool. As in the past assignments, when creating and using a Java program you can type "`ant compile`" in your terminal window and it will compile the Java class and save the bytecode in the correct subdirectories inside of the `bin/` directory. Please see the course instructor if you

Ratio $f(2n)/f(n)$	Worst-Case Conclusion
1	constant or logarithmic
2	linear or linearithmic
4	quadratic
8	cubic

Table 1: Actual-worst-case time complexity conclusions drawn from the doubling ratio $f(2n)/f(n)$.

cannot get this to work. Next, you should load the source code of the `SortingExperiment` into GVim so that you can study it carefully. What sorting algorithms does ExpOse experimentally study? Finally, please examine the source code of the `UniqueExperiment`. What are the uniqueness detection algorithms that it studies? Make sure that you look at the source code of each of these algorithms, studying their structure as you try to discern their worst-case time complexity. Which of the algorithms are most efficient? Which of the algorithm seem to be less efficient? Why?

Understanding the Approach Taken by ExpOse

In the last assignment, you studied statements in your textbook about a pattern that you could follow when making observations about an algorithm's time overhead. For instance, when describing the results from running the `StringExperiment`, page 153 notes that "[A]s the value of n is doubled, the running time of `repeat1` typically increases more than fourfold." What does this suggest about the likely worst-case time complexity of the `repeat1` method? Additionally, page 172 includes the following statement when describing the performance of `repeat2`: "the running times in that table ... demonstrate a trend of approximately doubling each time the problem size doubles." Again, what would this observation suggest about the likely worst-case time complexity of `repeat2`?

A useful understanding of an algorithm's efficiency, the worst-case time complexity gives an upper bound on how an increase in the size of the input, denoted n, increases the execution time of the algorithm, $f(n)$. We have learned in class that this relationship is often expressed in the "big-Oh" notation, where $f(n)$ is $O(g(n))$ means that the time increases by no more than on order of $g(n)$. Since the worst-case complexity of an algorithm is evident when n is large, one approach for determining the big-Oh complexity of an algorithm is to conduct a doubling experiment with increasingly bigger input sizes. By measuring the time needed to run the algorithm on an input of size n and the time needed to run with input of size $2n$, the algorithm's order of growth can be empirically determined. The goal of a doubling experiment is to draw a conclusion about the efficiency of the algorithm from the ratio $f(2n)/f(n)$ that represents the factor of change in runtime from input sizes n to $2n$. For instance, a ratio of 2 would indicate that doubling the input size resulted in the runtime's doubling, thus leading to the conclusion that the algorithm under study is $O(n)$ or $O(n \log n)$. Table 1 shows some common time complexities and their corresponding ratios.

As you saw in the previous laboratory assignment, it is time consuming and challenging to implement your own framework for conducting a doubling experiment. Therefore, in this assignment, you will learn how to use ExpOse, a tool that can automatically conduct a doubling experiment for you. Once your algorithm is added to ExpOse (like the sorting and uniqueness detection algorithms that have already been integrated), you can use the tool to automatically conduct an experiment and reach a conclusion about the likely worst-case time complexity of an algorithm (since this inference is based on data observations from actual runs of an algorithm, the suggested complexity is called "actual-worst-case" in the remainder of this assignment sheet). It is nice to use ExpOse when you want to confirm that a likely worst-case time complexity is correct or you want to develop an intuition about an algorithm's worst-case behavior. Overall, using a tool like ExpOse enables you to make a clear link between the analytical and experimental evaluation of an algorithm's efficiency. In the remainder of this assignment, we will use ExpOse to confirm the conclusions that your textbook's authors have reached concerning the worst-case time complexity of two different types of algorithms (e.g., sorting algorithms and uniqueness detection algorithms).

Determining "Actual-Worst-Case" Time Complexity

Before you and your partner start to use ExpOse to infer the actual-worst-case time complexity of an algorithm, you should study the `SortingExperiment` and `UniqueExperiment` classes and determine each of the algorithms that they are currently configured to study. Next, you should use your textbook and online resources to determine the well-known worst-case time complexity of each algorithm. For instance, the bubble-sort algorithm implemented in the `public static void bubbleSort(int data[], int n)` of the `BubbleSort` class is know to be in the $O(n^2)$ worst-case time complexity. What is the already-known time complexity of all of the other sorting and uniqueness detection algorithms? How do you know that these time complexities are correct?

Now you are ready to start running ExpOse to infer the time complexities by typing:

```
java edu.allegheny.expose.examples.sort.SortingExperiment bubble --verbose
```

What is the output of this program? Does the output confirm the well-known worst-case time complexity of bubble-sort? How does ExpOse implement the concept of a doubling experiment to infer the actual-worst-case time complexity? Does the ExpOse tool determine the time complexity in an efficient manner? What are the benefits and drawbacks of the approach taken by ExpOse? Please see the instructor if you are not able to use this tool to determine that bubble-sort is $O(n^2)$.

Next, you and your partner should repeatedly run ExpOse for all possible configurations of the `SortingExperiment` and `UniqueExperiment`, carefully checking to see if the tool confirms the well-known time complexities for the different algorithms. For instance, this means that you must repeatedly run `SortingExperiment` so that you infer the actual-worst-case time complexity for the five integrated sorting algorithms and then run `UniqueExperiment` for the two chosen uniqueness detectors. Please run both of these programs multiple times, recording their output in a separate file. You and your partner should also complete a summary table that records the name of the algorithm, the well-known worst-case time complexity, and the complexity suggested by `ExpOse`. In situations in which the tool's output differs from the time complexity that you found in a textbook or an online source, you should note why this is the case. Otherwise, you should use the source code of the algorithm as you articulate why the inferred time complexity is, in fact, correct.

You and your partner should create a Git repository on Bitbucket according to the naming scheme `cs112F2016-lab06-<user name one>-<user name two>`. You can now use this repository as you collaboratively complete your report on the worst-case time complexity of the five sorting algorithms and the two uniqueness detectors. Please note that you do not have to incorporate ExpOse into your Bitbucket repository. Instead, each team is responsible for using the web-based interface to GitHub to submit at least one "pull request" that contains an improvement to the

ExpOse tool. Students who have not previously used GitHub to create a pull request can review the tutorial that is available at `https://yangsu.github.io/pull-request-tutorial/`. Intuitively, each partnership is going to add some new feature to ExpOse and then ask the maintainers of this project to incorporate their improvement into the main system that is available for download from GitHub. You and your partner may consider adding comments to the source code, reformatting the source code so that it is easier to read, or adding a new example like the `SortingExperiment`. While each team must add at least one pull request (which may or may not be merged into the open-source project at the discretion of the project's maintainers), you and your partner can earn extra credit by attempting more complex improvements or by initiating multiple pull requests. Note that completing this task is an important step towards building an online portfolio that will be reviewed by your prospective employer or graduate school.

Carefully Review the Honor Code

The Academic Honor Program that governs the entire academic program at Allegheny College is described in the Allegheny Academic Bulletin. The Honor Program applies to all work that is submitted for academic credit or to meet non-credit requirements for graduation at Allegheny. This includes all work assigned for this class (e.g., examinations, laboratory assignments, and the final project). All students who have enrolled in the College will work under the Honor Program.

It is understood that an important part of the learning process in any course, and particularly one in computer science, derives from thoughtful discussions with teachers and fellow students. Such dialogue is encouraged. However, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone else's work. While it is acceptable for partners in this class to discuss their programs, data sets, and reports with their classmates, deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code.

Summary of the Required Deliverables

This assignment invites you to submit a signed and printed version of the following deliverables:

1. A sample output from running all doubling experiments in all of their relevant configurations.
2. A report that explains the well-known and inferred time complexities for the seven algorithms.
3. A "pull request" that you submit to the public GitHub repository for the ExpOse tool.
4. A reflective commentary on the challenges that you faced when creating the pull request.
5. A reflective commentary on the challenges that you faced when running the ExpOse tool.

Before you turn in this assignment, you must ensure that the course instructor has read access to your Bitbucket repository that is named according to the convention `cs112F2016-lab06-<user name one>-<user name two>`. Please note that each team in the class is responsible for submitting one version of this assignment. Students who have questions about any aspect of this laboratory assignment, including how they should complete it under the structure of the Honor Code, are encourage to schedule a meeting during the course instructor's office hours. Students are also invited to post questions or comments about this laboratory assignment to the `#laboratory` channel in our Slack team; either a teaching assistant or the instructor will answer these questions.