Computer Science 431
Algorithms
The College of Saint Rose
Spring 2015

Problem Set 3: Theoretical and Empirical Analysis

Due: 11:59 PM, Friday, February 20, 2015

You may work alone or in groups of size 2 or 3 on this assignment. Only one submission per group is needed.

Programming Task: Generating Example Arrays

Below, you will be asked to perform empirical analysis on the sorting algorithms we will be studying. As part of that task, you will be required to generate input arrays for the sorting algorithms. In order to test the best, worst, and average cases of some of these algorithms, you will need to generate input arrays with various characteristics. For simplicity, we will sort arrays of `int`.

- an array filled with n random values within a given range
- an array filled with n values sorted in ascending order
- an array filled with n values sorted in descending order
- an array filled with n values "nearly" sorted in ascending order

You may use any programming language. If you use Java, implement it within a class `IntArrayGenerator` that includes `static` methods to generate and return arrays of `int` with each of the above characteristics, and a `main` method that tests these methods for various values of n and ranges. Be sure you can achieve similar functionality if you choose a different language. It is important that you generate these efficiently – for example, you should not generate sorted input by generating random input then sorting it. Generate it in sorted order right from the start. (10 points)

Programming Task: Timing Sorting Algorithms

We have seen two sorting algorithms so far: bubble sort and selection sort. Develop a program that will help us to perform an empirical study of the efficiency of these, and later, other sorting algorithms. Your program should operate on arrays of `int` values. It should have options to set the array size, the number of trials (to improve timing accuracy), the ability to generate initial data that is sorted, nearly sorted, completely random, and reverse sorted. Yes, it should use the class you wrote for the previous part of this assignment to do all of this. Design your program to make it easy to implement additional sorting algorithms later. You will then use this program to answer one of the written problems below. (12 points)

- Use command line parameters rather than prompts, as this makes it much easier when running many (likely hundreds or thousands) of trials to generate timing results. In Java, `args[]` has what you need! If you don't know how to run with command-line parameters inside your IDE, run your Java program at the command line. That's what you'll want to do when generating timing results anyway.
- Have one big program rather than lots of little ones. This will help you avoid repeated code as you implement each of the sorting algorithms we will be studying within the same framework.
- Be careful that you don't reuse an array of values for multiple runs, since all but the first could end up having already-sorted data as input.
- A simple tabular format of output will help you manage the creation of tables and/or graphs that you'll need later. Something like

  ```
  10000 bubble random .034693
  ```

  might indicate for an input size of 10,000, using a bubble sort on random input took .034693 seconds.

First Empirical Analysis Study

Use your sorting algorithm program to generate timing data for bubble sort and selection sort on an appropriate range of data sizes and distributions. Compare your timing results with your expectations based on our (theoretical) efficiency analysis of these algorithms. (16 points)

- Please include as many of the details of your test environment as you can: the type of processor or processors in the computer including clock speed, cache sizes, memory sizes, the operating system and version running on the computer, and the Java version (or whatever other language) you are using.
- Include a brief description of the methodology for the tests. For this first one, you would describe the set of runs you are going to perform and state the expected results based on the theory (*e.g.*, n^2 running times).
- If you find discrepencies between your theoretical expectations and the timings you gather, explain to the best of your ability.
- The runs should vary the input array size for each combination of sorting algorithm and input data type. To get meaningful results, you want a pretty wide range of sizes. You might start with an array of size 1000 (or better yet 1024) and double the size until you have an input size of 1,000,000 (or better yet 1,048,576). Take the average or best times (and justify your choice) for some number of runs, probably a few dozen to a few hundred. Then, plot your results. Make sure your graphs have a meaningful title, legend, and axis labels. See if the numbers fit the expected, in this case n^2, behavior.

- Include your graphs and your analysis of them in your writeup. The raw numbers are useful, but should be submitted separately as part of a big table or spreadsheet, or possibly as an appendix. There are likely to be too many numbers for anyone to want to look at them all.

Written Problems

1. Levitin Exercise 2.4.9, p. 78 (4 points)
2. Levitin Exercise 2.4.10, p. 78. Set up and solve a recurrence relation to find your answer. (4 points)
3. Levitin Exercise 3.2.4, p. 107 (3 points)
4. Levitin Exercise 3.2.6, p. 107 (3 points)
5. Determining the best time to buy and sell a stock is a problem of interest to many people for obvious reasons. One way of predicting future behavior is by studying past behavior. For example, consider the following rise and fall of a stock over a period of 15 days. Positive values indicate the amount by which the stock rose on that day and negative values indicate the amount by which the stock fell.

   -6 -1 +5 + 7 +5 +3 -5 -7 -1 +10 +1 +5 -20 +10 +1

   We want to determine what the best days were to buy and sell this stock. You are allowed to buy only once and sell only once during the 15 day period.

   For example, if you bought on day 3 and sold on day 6, then you would have a net profit of $20. However, if you bought on day 3 and sold on day 12, then you would have a net profit of $23.

   Give pseudocode for a brute-force algorithm that computes the best day to buy and the best day to sell given the stock's history over the past `n` days. Assume that the rise and fall values are stored in an array `A[1...n]`. Give the Θ efficiency class of your algorithm and briefly explain how you arrived at this bound (7 points)
6. Given an array of positive numbers `A[1...n]` and a goal value `G`, are there two numbers in the array that sum to exactly `G`? Give pseudocode for an exhaustive search algorithm solving this problem. Give its Θ efficiency class and briefly explain how you arrived at it. (6 points)

Submitting

Before 11:59 PM, Friday, February 20, 2015, submit your problem set for grading. To complete the submission, upload an archive (a `.7z` or `.zip`) file containing all required files using Submission Box at `http://sb.teresco.org` under assignment "PS3".

Grading

This assignment is worth 65 points, which are distributed as follows:

Feature	Value	Score
Example array generation methods	8	
Example array generation tests	2	
Sorting algorithms code	12	
Empirical analysis writeup	16	
Exercise 2.4.9	4	
Exercise 2.4.10	4	
Exercise 3.2.4	3	
Exercise 3.2.6	3	
Brute-force stock algorithm	7	
Sum values search algorithm	6	
Total	65	