SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2017

Homework Set 2

Due: Start of Class, ☘ Friday, March 17, 2017 ☘

You may work alone or with a partner on this assignment. However, in order to make sure you learn the material and are well-prepared for the exams, you should work through the problems on your own before discussing them with your partner, should you choose to work with someone. In particular, the "you do these and I'll do these" approach will not prepare for the exams.

Please submit a hard copy: print your code (consider 2-up double-sided printing), a typeset printout of the empirical analysis writeup, and either a typeset (preferred) or handwritten (must be legible) set of responses for the written problems. Only one submission per group is needed.

There is a significant amount of work to be done here, and you are sure to have questions. It will be difficult if not impossible to complete the homework set if you wait until the last minute. A slow and steady approach will be much more effective.

Empirical Analysis of Sorting Algorithms

Your largest task for this homework set, worth a total of 50 points, is to conduct an empirical study of sorting algorithms. We have studied three sorting algorithms so far: bubble sort, selection sort, and insertion sort. We will soon see two more: mergesort and quicksort.

A Sorting Framework for Gathering Timings

Your first task, for 15 points, is to develop a program to perform an empirical study of the efficiency of these 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 (use the class you wrote for the previous assignment for array generation). Design your program to make it easy to implement additional sorting algorithms later.

- 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.
- You will need one or more methods to implement each sorting algorithm. You should write your own code, but you may base it on any descriptions of these algorithms you wish (pseudocode from our text might be a good choice). Beware of recursive implementations, which will be slower and more likely to run out of memory than their iterative counterparts.

- Write one big program rather than lots of little ones. This will help you avoid repeated code as you implement each of the sorting algorithms.
- Even though a single program will be able to run a variety of sorting algorithms on different sizes and initial ordering of input, a single execution of your program should only run one such combination (though possibly many trials of that same run to improve timing accuracy).
- 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, selection sort, insertion sort, mergesort, and quicksort, 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. (35 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. 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, *e.g.*, n^2, behavior.
- Include a summary of your raw timing results (in tabular form), your graphs of those results and your analysis of the results in your writeup.

Homework Set Problems

1. Determine closed forms for each of the following recurrence formulas using backward substitution. (24 points)

For each problem: (i) show at least the first 3 substitutions (for 4 points), (ii) show the pattern for the i^{th} substitution (2 points), (iii) state the value that i will have in the last substitution (1 point), (iv) give the closed form (4 points), and (v) give the representative Θ growth rate of the recurrence (1 point).

(a) $T(n) = T(n-1) + 5$ for $n > 1, T(1) = 3$

(b) $T(n) = T(n-1) + 5n$ for $n > 1, T(1) = 3$

(c) $T(n) = T(n/2) + 4$ for $n > 1, T(1) = 1$

(d) $T(n) = T(n/2) + 4n$ for $n > 1, T(1) = 1$

(e) $T(n) = 2T(n/2) + 3$ for $n > 1, T(1) = 1$

(f) $T(n) = 2T(n/2) + 3n$ for $n > 1, T(1) = 1$

(g) $T(n) = 2T(n/2) + an$ for $n > 1$ and any positive constant a, $T(1) = 1$

(h) $T(n) = 2T(n/2) + n^2$ for $n > 1, T(1) = 1$

Note: although your final closed form for (g) above will include the constant a, your Θ growth rate should only be in terms of n.

We will choose 2 of the 8 problems above to grade out of 12 points each. Any omitted problem will be chosen automatically (remember, we hate grading), so do them all!

---

2. (10 points) On exam 1 you were asked to write a $\Theta(n^2)$ brute-force algorithm that counted, in a given text, the number of substrings that start with an A and end with a B. For example, there are four such substrings `CABAAXBYA`.

Improve upon this by writing a linear time algorithm solving this problem, starting with the pseudocode below. (8 points)

```
ALGORITHM countABSubstrings( text[0...n] )
```

Briefly justify why your algorithm runs in linear time. (2 points)

---

3. (16 points) 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 Impulse stock over the last 15 days. The values indicate the amount by which the stock price changed from the start of the trading day to the end of the trading day. Positive values indicate the stock increased in price on that day, and negative values indicate the price fell.

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

We want to calculate the maximum amount of profit you could have earned assuming you buy exactly once (at the beginning of a day) and sell exactly once (at the end of a day, possibly the same day you bought) during the 15 day period. For example, if you bought on day 2 and sold on day 5, then you would have a net profit of 20 dollars. However, if you bought on day 2 and sold on day 11, then you would have a net profit of 23 dollars. (Note: the first day above is day 0.)

We could easily solve this problem in $\Theta(n^2)$ time using brute force by examining all pairs of buy and sell days, where n is the number days in the period being considered. Improve upon this by giving a divide and conquer algorithm solving this problem that runs asymptotically faster than $\Theta(n^2)$ Start with the pseudocode below. It returns the maximum profit you could have made if you bought and sold exactly once within the range of days indicated. The base case is already done for you. (10 points)

```
// Returns maximum profit possible based on past history
// in array A[sday...eday], for the range of days starting
// at sday and ending at eday.

ALGORITHM MaxProfit( A[sday...eday] )

   if ( sday = eday )
     // only one day in range, so must buy and sell on that day
     return A[left]
```

Once you have your algorithm, complete the recursive call tree below showing all the recursive calls your algorithm will make and the return value from each when it is called with input array `A = {6, -50, 2, 13, 5, -14, 1, 6}`

The root of your tree should look like what you see below, with `MP` (short for `MaxProfit`) and the blank filled in with the answer that the call returns:

```
MP( A[0...7] )
returns _________
```

All other nodes in your tree should be formatted similarly. (4 points)

Finally, give a recurrence formula for the worst case running time of your algorithm, and then give its representative Θ running time. (You may find the closed form of your recurrence in any way you like and you do not need to show your work in getting it.) (2 points)