SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2018

Problem Set 5
Due: 4:00 PM, Friday, March 23, 2018

You may work alone or in a group of 2 or 3 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 is sure to leave you unprepared for the exams.

Groups must be formed and "registered" as a group by an email to *jteresco@siena.edu* by 4:00 PM, Monday, March 18, 2018. This applies to those who choose to work alone as well!

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 assignment if you wait until the last minute. A slow and steady approach will be much more effective.

Submitting

Please submit a hard copy (typeset preferred, handwritten OK but must be legible) for all written questions. Only one submission per group is needed.

Written Problems

In the fraudulent voters problem, the input consists of two arrays: `V[0...n-1]` contains the names of n voters that participated in a recent election, and `IE[0...m-1]` contains the names of m persons ineligible to vote (because of something such as a felony record, deceased, etc...) In this problem, the task is to determine how many of the voters in `V` were actually ineligible to vote. For example, for the two arrays below, 2 should be returned as the answer, because voters "T. Smith" and "J. Doe" were ineligible to vote. You may assume there are no duplicate names in the arrays.

```
V[0...7] = {"A. Turing", "T. Smith", "B. McKeon", "G. Gordon",
   "J. Doe", "M. Taylor"}
IE[0...4] = {"Z. McBeth", "J. Doe", "T. Jones", "T. Smith", "N. Wirth"}
```

Question 1:
You could easily write a brute force algorithm solving this problem in $\Theta(nm)$ time. But the number of voters in both arrays is very large, and so an asymptotically more efficient algorithm is required. Write such an algorithm. (10 points)

ALGORITHM NUMFRAUDULENTVOTERS(V, IE)
//Input: $V[0..n]$, names of actual voters
//Input: $IE[1..m]$ names of ineligible voters
//Output: the numbers of names that appear in both V and IE

Question 2:
Give the Θ efficiency class of your algorithm and briefly explain how you arrived at this bound. (3 points)

Spring Break is right around the corner, but it sure has been snowing a lot recently. So let's consider an algorithm that can be used to do a little analysis of that recent snowfall. Suppose you have a large array of numbers representing the snowfall totals in the most recent storm (at least as of the date when this assignment came out) across several weather reporting stations, *e.g.*, the storm totals shown below for 11 locations around the region, as reported by the Albany National Weather Service field office on March 15.

$$A = \{15.8, 9.8, 5.5, 7.5, 16.8, 12.0, 10.8, 7.8, 44.0, 23.0, 18.4\}$$

Your task is to find the pair of snowfall totals that are closest together in value. For the example above, this would be the 7.5" (which fell in Dolgeville) and the 7.8" (which fell in Middleburgh), a difference of only 0.3".

Question 3:
Develop pseudocode for an algorithm to solve this problem the operates in better than $\Theta(n^2)$ time. (10 points)

Question 4:
Give the Θ efficiency class of your algorithm and briefly explain how you arrived at this bound. (3 points)

Read Section 8.1 of Levitin to learn all about the robot coin collection problem. There, you will find a bottom-up dynamic programming solution to this problem.

Question 5:
Complete the pseudocode below that uses a **recursive exhaustive search algorithm** (**not** dynamic programming) to solve the robot coin collection. (5 points)

ALGORITHM ROBOTCOINCOLLECTION(r, c, C)
//Input: r, c, starting row and column position
//Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
//Output: the max number of coins collected when the robot reaches (n, m)

Question 6:
Rewrite your algorithm from above so that it uses **top-down dynamic programming** to get a recursive algorithm that is more efficient. (10 points)

ALGORITHM ROBOTCOINCOLLECTION($r, c, C, sols$)
//Input: r, c, starting row and column position
//Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
//Input: $sols[1..n, 1..m]$ initialized to all -1
//Output: the max number of coins collected when the robot reaches (n, m)

Question 7:
Give the Θ efficiency class of your top-down dynamic programming algorithm and briefly explain how you arrived at this bound. (3 points)

Read Section 8.3 of Levitin to learn all about optimal binary search trees.

Question 8:
Answer Levitin Exercise 8.3.1, p. 303. (3 points)

Question 9:
Answer Levitin Exercise 8.3.5, p. 303. Justify your answer. (3 points)