CS:5350 Midterm Exam (Retake), Spring 2016

Your answers will be graded primarily for correctness, but as in the homeworks, clarity, precision, and conciseness will also be important. Problem 1 is worth 60 points (each part is worth 20 points), and the remaining three problems are worth 70 points each. Note that this total is 270, which means that this exam contains 20 extra credit points.

1. This problem is on recurrence relations.

   (a) You are given a *divide-and-conquer* algorithm, Algorithm $\mathcal{A}$. The running time of Algorithm $\mathcal{A}$ is $T(n) = \Theta(n^2)$. You are also told that, in the DIVIDE step, Algorithm $\mathcal{A}$ does some work that leads to three subproblems (which it then solves recursively). Write down a possible recurrence for the running time of Algorithm $\mathcal{A}$. Justify your answer by solving your recurrence and showing that the solution is indeed $T(n) = \Theta(n^2)$.

   (b) Your friend has designed a search algorithm, and she tells you that the algorithm's running time is given by the recurrence $T(n) = T(\lfloor\sqrt{n}\rfloor)+1$, for $n > 1$, and $T(1) = 1$. For a certain problem, you are trying to decide between using your friend's algorithm and *binary search*. If asymptotic running time is your only criterion in making this decision, which algorithm would you choose? Justify your answer.

   (c) Your friend clearly loves to design algorithms! She has designed another algorithm whose running time is given by the recurrence: $T(n) = T(n/10) + T(n/20) + 1$, for $n > 1$, and $T(1) = 1$. For a certain problem, you are trying to decide between using your friend's algorithm and a *linear-time* algorithm. If asymptotic running time is your only criterion in making this decision, which algorithm would you choose? Justify your answer.

2. You are given an array $A[1..n]$ that may contain both positive and negative integers. Design a *divide-and-conquer* algorithm running in $\Theta(n \log n)$ time for finding the largest sum of a contiguous subarray of numbers within A. (Among all contiguous subarrays of A, one [or more] of them has the largest possible sum; you want to return this largest sum.) As an example, if the given array is $[-2, -5, \mathbf{6}, \mathbf{-2}, \mathbf{-3}, \mathbf{1}, \mathbf{5}, -6]$, then the maximum subarray sum is 7 (see highlighted elements). Your answer should contain two parts: (a) A clear description of the algorithm, either in plain English or in pseudocode; and (b) An analysis showing that the algorithm has running time $\Theta(n \log n)$.

3. You have 1 dollar and you want to invest it for n months. At the beginning of each month, you must choose one option from the following three:

   (I) Purchase a savings certificate from the local bank. In this case, all of your money will be tied up for one month. If you buy the certificate at the beginning of month t, there will be an upfront fee of $C_S(t)$ dollars, and after one month, it will return $S(t)$ for every dollar invested. That is, if you have k dollars at the beginning of month t and you choose the savings certificate option, then you will have invested k dollars to purchase a savings certificate at the beginning of month t and you will have $(k - C_S(t)) \cdot S(t)$ dollars at the end of month t (i.e., at the beginning of month $t+1$).

   (II) Purchase a state treasury bond. In this case, your money will be tied up for six months. If you buy the bond at the beginning of month t, there will be an upfront fee of $C_B(t)$ dollars, and after six months, it will return $B(t)$ for every dollar invested. That is, if you have k dollars at the beginning of month t and you choose the treasury bond option, then you will have invested k dollars to purchase a treasury bond at the beginning of month t and you will have $(k - C_B(t)) \cdot B(t)$ dollars at the beginning of month $t + 6$.

(III) Store the money in a sock under your mattress for a month. If you choose this option and you start month t with k dollars, then you'll have k dollars at the beginning of month $t+1$ as well.

Suppose you know the values of the functions $S(t)$, $B(t)$, $C_S(t)$, and $C_B(t)$ for each of the next n months, i.e., for $t = 1, 2, \ldots, n$. You should assume that each of these functions is stored in an array of length n. This question will lead you through a dynamic programming solution to this problem, running in $O(n)$ time, for computing the *maximum* amount of money you can have at the end of the n months.

(a) For $j = 1, 2, \ldots, n$, let MaxInvest(j) denote the problem in which you have 1 dollar at the beginning of month j and you want to make investment decisions at the beginning of months $j, j+1, \ldots, n$ so as to maximize the amount of money you will have after month n. Let $OPT(j)$ denote the amount of money you would have at the end of month n if you optimally solve MaxInvest(j). Write down a recurrence relation for $OPT(j)$. Make sure you include base cases.

(b) Turn your recurrence into clear pseudocode. Clearly identify the data structure you are using to compute the values of $OPT(j)$ and make sure to fill this data structure in the correct order.

4. You are a painter who has just been awarded a very large contract – you have been hired to paint all the houses on a particular street! There are n houses on the street and you have k colors of paint, and you must paint each house with a single color. Each house is a different size and shape, and so the costs of painting each house varies. These costs are given by an $n \times k$ array called $Cost[1..n, 1..k]$, where $Cost[i, j]$ is the cost of painting house i with color j. It is worth emphasizing that there is no relationship between the different costs in this array.

The residents of the street like all the colors, so each doesn't care about which color you decide to use to paint their particular house, EXCEPT for one thing – each doesn't want their house to be the same color as a neighbor's house. So, for instance, in the case of $n = 5$ houses and $k = 3$ colors (say, Red, Green, and Blue), then G-R-G-B-G and B-R-G-B-R would be acceptable color schemes, but G-R-B-G-G would not be acceptable.

Since you are a businessperson, you want to maximize your profit by using the *cheapest* possible color scheme (i.e., an assignment of colors to houses) satisfying the constraint described in the previous paragraph. Note that the cost of a particular color scheme is just the sum of the costs of painting each individual house with its assigned color.

(a) As a first step toward a dynamic programming solution to this problem, describe the subproblems your algorithm will need to solve. What is the number of these subproblems?

(b) Write down the recurrence relation(s) that express the minimum cost of painting the n houses using the k colors. Make sure you specify all base cases.

(c) Describe an algorithm, using pseudocode, which takes as input an array $Cost[1..n, 1..k]$ and returns the cost of a cheapest acceptable assignment of colors to houses.

(d) Analyze the asymptotic running time of your algorithm in terms of n and k.
