Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Fall 2003
Homework 4b
Due on 9/30/03 (Tuesday in class)

1. **[25 Points] Implementing the "Change" Dynamic Program.**

   In class we discussed an algorithm which takes as input a coinage system and an amount of money to be made out of coins in this coinage system. The algorithm returns the smallest number of coins required to make up this amount. Your job is to implement this algorithm in two ways: recursively and as a dynamic program.

   Your program should ask the user to specify which algorithm they want to run. Next, it should ask the user to specify the number of coins in the coinage system. Next, the user should be prompted to enter the value of each coinage from smallest to largest. (The program should report an error if the user fails to do this in order or if the first coin does not have value 1.) Finally, the program asks the user for the amount A that needs to be changed into coins.

   The program should report the minimum possible number of coins that can be used, the *actual list of coins used in this optimal solution*, and the overall runtime needed to calculate this solution. To calculate runtime on turing, use the `time` command. You'd want to report user time (which appears to be accurate to 100-ths of seconds). Report execution time in second units. You can use any programming language you like but please run the code on turing so that it makes more sense to compare various students reported run-times.

   Turn in an example of your program running both recursively and with DP. Use input coins $1, 5, 30, 40, 60$ and a change amount of $A = 70$. You might want to experiment with other values of A to get other run-time values as well. Also turn in an example of your code; up to 1/4 of your grade may depend upon your code being reasonable (i.e. readable, well organized).

   Finally, answer the following question: Why should your program error if the user enters a series of coins that does not contain a value of 1?

2. **[20 Points] LCS is Optimal!**

   In class we talked about the problem of finding the longest common subsequence (lcs) Z_k among two strings X_m and Y_n where:

   $$z_l = x_{i_l} = y_{j_l}$$

   for all $l \in \{1, 2, \cdots, k\}$ and $i_l < i_{l+1}$ and $j_l < j_{l+1}$.

   Carefully reconstruct the argument of why the recursive `LCS` program discussed in class produces a correct lcs. (Please send me email or come to office hours if you need help with this.) In proving this, feel free draw pictures to make your argument clear.

3. **[25 Points] Greed Must be used with Extreme Caution!** In class we talked about the problem of finding the maximum number of non-conflicting courses from a given set of courses. We argued that if the courses are sorted in order of completion time, then the following greedy algorithm is guaranteed to find the maximum number of non-conflicting courses: Choose the course with earliest completion time. Cancel out all courses that conflict with that course. Now repeat the process for the remaining courses.

   Professor I. Lai of the Pasadena Institute of Technology has proposed the following four alternative greedy algorithms for this problem.

   Algorithm 1: Sort the courses by ending time as before. Now, run our original greedy algorithm in the opposite direction. That is, choose the course that ends **latest**. Then cancel out all courses that conflict with that course. Now repeat the process for the remaining courses.

   Algorithm 2: Sort the courses by increasing starting time (rather than ending time). Now, choose the course that starts first. Then cancel out all courses that conflict with that course. Now repeat the process for the remaining courses.

   Algorithm 3: Forget about sorting the courses. Choose a course of shortest duration (that is the course that has the least length). Then cancel out all courses that conflict with that course. Now repeat the process for the remaining courses.

   Algorithm 4: Don't sort the courses. Choose a course that conflicts with the fewest other courses, breaking ties arbitrarily. Then cancel out the courses that conflict with that course. Now repeat the process for the remaining courses.

   Show that each of these greedy algorithms does not always find an optimal solution. Showing this for **Algorithm 4 is optional and is worth 10 Bonus Points**.

   To show that an algorithm does not always find an optimal solution, you need only demonstrate a specific example and show that the given algorithm could select a particular solution which is not the best one. Also answer in one or two sentences why to find a single-counter example is sufficient for proving the algorithm's incorrectness.

   Now formally argue (similarly to what we did in class) why the following algorithm is correct: Choose the course with latest start time. Cancel out all courses that conflict with that course. Now repeat the process for the remaining courses.