Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Fall 2004
Homework 6b
Due on Tuesday, 10/12/04 (at the beginning of class)

1. [**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.

In this problem you will explore various aspects of greed in the context of the Registrar problem.

(a) To show that an algorithm is incorrect—e.g. in the case of greed, does not always find an optimal solution—one only needs to demonstrate a specific example in which the algorithm might select a solution that is suboptimal. Explain in one or two sentences why this is sufficient to demonstrate an algorithm's incorrectness.

(b) Show that Professor Lai's algorithms are incorrect by giving a counterexample for each. You should explain your counterexample in pictorial form. Also, indicate in this picture what the greedy algorithm could choose and contrast this a better solution.

(c) Consider a registrar algorithm that first sorts on *start* time and then uses the following greedy choice: choose the course with the *latest* start time (remove conflicts and repeat). Argue by way of contradiction that the greedy choice is safe. Be careful and precise, so we can provide good feedback when grading. For the same reason, don't take a reasonable "shortcut" here, e.g. simply observing that, via symmetry, since this algorithm is isomorphic to the one we proved made a safe choice in class, its choice is also safe.

2. [**40 Points**] **Optimizing Party Fun!**

After your success at Snapple, you've decided to accept a job as senior algorithm designer at the well-known My-I'm-Soft Corporation. One day, the President of My-I'm-Soft, Gill Bates, comes to you with the following problem. "I'm throwing a company party," Gill says excitedly, "And I need your help! As you know, My-I'm-Soft has a hierarchical structure. You can think of it as a tree. The president, that's me, is at the root of the tree. Oh boy, I love being at the root!" You take a sip of your luke-warm diet coke (which My-I'm-Soft provides for free—what a perk!) and listen patiently as Gill continues. "Below the root are supervisors, below them are managers, below them are team leaders, etc., etc., until you get down to the leaves—the summer interns. Anyhow, to make the party fun, I thought it best that we don't invite an employee along with their immediate boss (their parent in the tree). Also, I've personally assigned every employee a real number (actually it's a double precision floating point, but never-mind that!) called their *coefficient of fun*. My objective is to invite employees so as to maximize the total sum of the coefficients of fun of all invited guests, while not inviting an employee with his or her immediate boss."

(a) The first algorithm that Gill thought up (some people question Gill's technical competence...that's one reason you were hired) was to simply enumerate all possible subsets of his employees, throw out those subsets that include an employee and his or her boss, find the score for each remaining subset, and finally choose the best one. My-I'm-Soft has 1000 employees and also has just purchased a Crayfish YMP that can process one trillion (10^{12}) subsets per second. How long will it take (in years) to find the optimal solution using this brute force approach on the Crayfish?

(b) Provide pseudocode for a recursive solution to this problem and explain in a few sentences why it is correct. This recursive solution should take an input (a node in the tree) and return the optimal fun available for a tree rooted at this node. Node v can use the following notation: $i \in \text{Child}(v)$, where i is a member of the set that contains all of v's children; $i \in \text{Grandchild}(v)$ is analogously defined.

(c) Write out the recurrence relation for this pseudocode and then use this to demonstrate why, in the worst case, run-time can be bounded from below exponentially (if you choose a specific tree shape for this worst case, be sure to specify what it is). Here, $i \in \text{Desc}(v, h)$ will be useful. Desc is the set of all descendents of node v that lie h "levels" below it. For example: $\text{Child}(v)$ is $\text{Desc}(v, 1)$, $\text{Grandchild}(v)$ is $\text{Desc}(v, 2)$, etc.

(d) Provide pseudocode that implements this recursive algorithm as a *memoized* dynamic program and explain in a few sentences what memoization is doing and how you are handling it. You can assume that each employee is identified by a unique integer in the range 1 through n (there are n employees overall; Gill is, naturally, numero uno!).

(e) What call would Gill have to make to find out just how much fun his company party can be? Use the accounting amortization method to derive what this call's overall running time would be.

(f) Will Gill necessarily be invited to his own party? Explain why or why not. If the answer is no, is there a simply way you could modify your algorithm to ensure that he was invited (and if so, what is it)?

(g) Suppose Gill later changes his hiring strategy, only offering jobs to applicants that are as similar to one another as possible (resistance is futile!). Thus, after some time, it comes to be the case that all of his employees' coefficients are identical! Because Gill is ever-paranoid to build the fastest, leanest, most elegant software, he now calls you back on board to develop a greedy algorithm (in the hopes that you might be able to streamline this special case):

   i. Identify what greedy choice you would use.

   ii. By way of contradiction, prove that your greedy choice is safe.

   iii. Always eager to save time, you recognize that you can now simply use the "Greedy Template" we discussed in class! Towards this end:

      A. Identify what X, A, $A*$ are.

      B. How does X need to be modified each time another greedy choice is added?

      C. Will this algorithm necessarily be asymptotically faster than a DP-based approach? Briefly justify your answer. (Should Gill be fired?)

   iv. Provide a (pictorial) counter-example where the greedy choice would fail when all coefficients were not necessarily the same value.