1. **[20 Points] The Millisoft Party Problem!** After your success at Snapple, you’ve decided to accept a job as senior algorithm designer at the well-known Millisoft Corporation. One day, the President of Millisoft, 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, Millisoft 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 Millisoft 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 his or her 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 nevermind 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 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. There are 1000 employees at Millisoft. Millisoft has also just purchased a Crayfish YMP that can process one trillion ($10^{12}$) subsets per second. How long will it take to find the optimal solution using this brute force approach on the Crayfish?

   (b) Describe in detail an efficient dynamic programming algorithm for this problem. What is the asymptotic running time of your algorithm?

   (c) How can you modify your algorithm to find the optimal solution in which Gill gets invited to his own party?

2. **[30 Points] Implementing the “Change” Dynamic Program.** In class we saw a dynamic programming 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 the algorithm. Your program should first ask the user to specify the number of coins in the coinage system. Then, 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 then report the minimum possible number of coins that can be used and the **actual list of coins used in this optimal solution.** Turn in a printout and an example of your program running with input coinages 1, 5, 30, 40, 60 and the amount $A = 70$.  


3. [15 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. None of them are correct! To show this, construct a separate counterexample (a set of intervals) for each algorithm below such that the algorithm does not find the optimal solution for your counterexample. Make sure to explain what the algorithm would do on your counterexample and what the optimal solution would be.

**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. (This is the hardest of the four parts of this problem!)