Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Spring 2004
Homework 5b
Due on 2/26/04 (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 and answer a few questions:

   (a) Before coding, finish the skeleton of the recursive pseudo-code I gave you in class for OPT_CH.

   (b) Implement the algorithm. Your program should begin by asking 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.* Turn in a copy of your code and take care to write it with some modicum of style (up to 1/4 of your grade may depend upon your code being reasonable—i.e. readable, well organized).

   (c) 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 $A = 70$.

   (d) 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. **[30 Points] Ski Optimization!**

   Your job at Snapple is pleasant but in the winter you've decided to become a ski bum. You've hooked up with the Mount Baldy Ski Resort (they actually happen to have some snow this year!). They'll let you ski all winter for free (and drink as much free Snapple as you want!) in exchange for helping their ski rental shop with an algorithm to assign skis to skiers.

   Ideally, each skier should obtain a pair of skis whose height matches his or her own height exactly. Unfortunately, this is generally not possible. We define the *disparity* between a skier and his or her skis to be the absolute value of the difference between the height of the skier and the pair of skis. Our objective is to find an assignment of skis to skiers that minimizes the sum of the disparities. We'll solve some interesting subproblems along the way to finding an efficient algorithm for the ski assignment problem.

(a) First, let's assume that there are n skiers and n skis. The previous computer scientist at the ski resort, Professor R. U. Kitting, proposed the following algorithm for this case. Consider all possible assignments of the n skis to the n skiers. For each one, compute the sum of the disparities. Finally, select the assignment that minimizes the sum of the disparities. Explain why Professor Kitting was fired. How much time would this algorithm take on a computer that performs 1 billion operations per second if there were 50 skiers and 50 pairs of skis?

(b) The next computer scientist at the resort, Dr. G. I. M. Fast, spent most of her time on the slopes. She did however make an interesting discovery one day on the chair lift. She observed that if we have a short person and a tall person, it would never be better to give the shorter person a taller pair of skis than were given to the tall person. Show that this is always true. (It would be a good idea to split your argument into cases and consider each separately.)

(c) After Dr. Fast's brief stint at the ski resort, you were hired. Your first job is to find a greedy algorithm that minimizes the sum of the disparities assuming that there are n people and n pairs of skis. (*Hint:* First sort the people by increasing height and sort the skis by increasing height.) Argue carefully that your algorithm does indeed find an optimal solution (you'll want to utilize part b here). In other words, show that no better solution is possible. What is the time complexity of your algorithm?

(d) Now, your task is to design an efficient dynamic programming algorithm for the more general case that there are m skiers and n pairs of skis and $m \leq n$. Again, start by sorting the skiers and skis by increasing height. Let h_i denote the height of the i^{th} skier in sorted order and let s_j denote the height of the j^{th} pair of skis in sorted order. Let $A[i,j]$ be the optimal cost (sum of absolute differences of heights) for matching the first i skiers with skis from the set $\{1, 2, \ldots, j\}$. The solution we seek is then simply $A[m,n]$. Fill in the blanks below to define A recursively:

$$A[i,j] = \begin{cases} \underline{\hspace{4cm}} & \text{if } i = 0 \\ \underline{\hspace{4cm}} & \text{if } i = j \\ \underline{\hspace{4cm}} & \text{if } i < j \end{cases}$$

Now give pseudocode (with explanatory comments) or a concise and complete English description of a dynamic programming algorithm that solves the optimal assignment problem of skis to skiers using a table for $A[i,j]$.

(e) What is the running time of your program? Explain.

(f) Describe how your algorithm can be modified to allow you to find an actual optimal assignment (rather than just the cost) of skis to skiers. How does this affect the running time of your algorithm?

(g) Illustrate your algorithm by explicitly filling out the $A[i,j]$ table for the following sample data:

- Ski heights: $1, 2, 5, 7, 13, 21$.

- Skier heights: $3, 4, 7, 11, 18$.

3. **[25 Points] Load Balancing on the Snapple Computer!**

   Snapple Computers is an emerging manufacturer of parallel computers. You've been hired to design algorithms for the operating system of Snapple's new personal computer, the Snapple Mango Melon Madness (referred to hereafter as the "M^3"). The M^3 has **two** identical UltraSuperSparkingMangoPower processors. When run in batch mode, the operating system receives an array S of the *integer* running times of n tasks. That is $S[1], \ldots, S[n]$ are integer running times for n tasks. The objective of the operating system is to find an optimal **load balance** for these tasks on the two processors. That is, we wish to find a partition of the tasks in S into two sets (such that the tasks in one set will be executed on one of the processors and the set of tasks in the other set will be executed on the second processor in parallel) that minimizes the total elapsed time required to complete all the tasks.

   For example, if S contains five tasks with running times 2, 1, 3, 5, 7 then we can assign the tasks with running times 2 and 7 to one processor and the tasks with running times 1, 3, and 5 to the other, resulting in an elapsed execution time of 9. On the other hand, if the five running times had instead been 2, 1, 3, 5, 8 the best we could do is have a partition of 2, 3, and 5 on one processor and 1 and 8 on the other, with an elapsed running time of 10.

   We will solve this problem in a sequence of steps. Be patient.

   (a) Your boss, Dr. I. Stee, has proposed a brute-force algorithm that involves simply enumerating all possible partitions of the task set S and finding the best one. If the M^3 can evaluate one million partitions per second and the set S contains just 50 tasks, how long does it take the M^3 to find the optimal load balance? Explain how you found your answer.

   (b) Now consider a variant of this problem called the *Partition Problem*. In this problem we are given the set S of n running times and we are also given a positive integer C. Our objective is return the boolean **true** if there exists a subset of S which adds up to exactly C and otherwise return the boolean **false**. To this end, give pseudo-code for a recursive function `partition`(i, C) which returns the boolean **true** if there is a subset of $S[1], \ldots, S[i]$ which adds up to exactly C and returns **false** otherwise.

   (c) Give a lower bound argument to explain why `partition`(n, C) can take exponential time.

   (d) Next, describe a dynamic programming algorithm that takes as input the set S and a number C and determines if there exists a subset of S that which adds up to exactly C. Describe your algorithm carefully and explain why it works.

   (e) What is the asymptotic running time of your algorithm as a function of n and C?

   (f) Now let's reconsider Snapple's load balancing problem. Show how your dynamic programming algorithm can be used to determine an optimal partition of S into two subsets such that the total elapsed running time of the resulting partition is minimized. Your algorithm must determine the actual partition. What is the running time of this algorithm?