Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Fall 2003
Homework 4a
Due on 9/26/03 (Friday at 2pm)

1. **[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?

2. **[5 Points] Load Balancing on the Snapple Computer Revisited.**

   Argue why the partition problem you propose above *minimizes* the time needed to run all jobs.