Introduction to Algorithms	**Problem Set 8**
CS 482 Spring 2000	**Due May 3, 2000**

(1) Recall the problem of *Monotone Satisfiability with Few True Variables* from Problem Set 5. Let's define *Monotone 3-SAT with Few True Variables* as the following special case. You're given a monotone instace of 3-SAT with n variables and m clauses: each clause has three terms, and each term is a non-negated variable. You're also given a number k. You want to decide: is there a satisfying assignment for the instance in which at most k variables are set to 1?

Give an algorithm that solves this problem with a running time of the form $O(f(k) \cdot p(m))$, where $p(\cdot)$ is a polynomial function, and $f(\cdot)$ is an arbitrary function that depends only on k, not on n or m. (You may assume that $m \geq n$, though this is not crucial.)

(2) The designers of the new Mars probe *U.S.S. Remember-to-Measure-Everything-in-Metric* are trying to pack as much cargo onto the craft as weight limits will allow.

Here's the problem they face. There are n items of cargo that could potentially be placed on the probe; the weight of the i^{th} item is w_i Newtons. The probe has a limit L on the maximum number of Newtons of cargo it can carry. The goal is to choose a subset of the items whose total weight is as large as possible, subject to the constraint that the weight limit not be exceeded.

Give a polynomial-time algorithm that approximates this maximization problem to within a factor of 2. Specifically, your algorithm should have the following property: if there is a subset of items of total weight $L^* \leq L$ (i.e. a subset that fits on the probe, and has weight L^*), your algorithm should return a subset of total weight at least $\frac{1}{2}L^*$.

(3) Consider the load balancing problem from Chapter 9.1. Some friends of yours are running a collection of Web servers, and they've designed a local search heuristic for this problem, different from the algorithms in Chapter 9.1.

Recall that we have m machines $M_1, \ldots, M_m$, and we must assign each job to a machine. The load of the i^{th} job is denoted t_i. The *makespan* of an assignment is the *maximum load* on any machine:

$$\max_{\text{machines } M_i} \sum_{\text{jobs } j \text{ assigned to } M_i} t_j.$$

Your friends' local search heuristic works as follows. They start with an arbitrary assignment of jobs to machines, and they then repeatedly try to apply the following type of "swap move":

> Let $A(i)$ and $A(j)$ be the jobs assigned to machines M_i and M_j respectively. To perform a swap move on M_i and M_j, choose subsets of jobs $B(i) \subseteq A(j)$ and $B(j) \subseteq A(j)$, and "swap" these jobs between the two machines. That is, update $A(i)$ to be $A(i) \cup B(j) - B(i)$ and update $A(j)$ to be $A(j) \cup B(i) - B(j)$. (One is allowed to have $B(i) = A(i)$, or to have $B(i)$ be the empty set; and analogously for $B(j)$.)

Consider a swap move applied to machines to machines M_i and M_j. Suppose the loads on M_i and M_j before the swap are T_i and T_j respectively, and the loads after the swap are T_i' and T_j'. We say that the swap move is *improving* if $\max(T_i', T_j') < \max(T_i, T_j)$; in other words, the larger of the two loads involved has strictly decreased. We say that an assignment of jobs to machines is *stable* if there does not exist an improving swap move, beginning with the current assignment.

Thus, the local search heuristic simply keeps executing improving swap moves until a stable assignment is reached; at this point, the resulting stable assignment is returned as the solution.

Example: Suppose there are two machines: in the current assignment, the machine M_1 has jobs of sizes $1, 3, 5, 8$, and machine M_2 has jobs of sizes $2, 4$. Then one possible improving swap move would be to define $B(1)$ to consist of the job of size 8, and define $B(2)$ to consist of the job of size 2. After these two sets are "swapped," the resulting assignment has jobs of size $1, 2, 3, 5$ on M_1, and jobs of size $4, 8$ on M_2. This assignment is stable. (It also has an optimal makespan of 12.)

(a) As specified, there is no explicit guarantee that this local search heuristic will always terminate — i.e., what if it keeps cycling forever through assignments that are not stable?

Prove that in fact the local search heuristic terminates in a finite number of steps, with a stable assignment, on any instance.

(b) Show that any stable assignment has a makespan that is within a factor of 2 of the minimum possible makespan. *(It may be helpful to consider the bounds* $\frac{1}{m}\sum_j t_j$ *and* $\max_j t_j$ *that we considered in Chapter 9.1.)*