Introduction to Algorithms	**Problem Set 9**
CS 482 Spring 2004	**Due April 30, 2004**

Please hand in each problem on a separate sheet with your name and NetID on each.

Reading: Sections 8.1-8.2, 9.1-9.2, 9.6, 9.8.

(1) *(This is Problem 5 at the end of Chapter 9.)* Suppose you're acting as a consultant for the Port Authority of a small Pacific Rim nation. They're currently doing a multi-billion dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Here's a basic sort of problem they face. A ship arrives, with n containers of weight $w_1, w_2, \ldots, w_n$. Standing on the dock is a set of trucks, each of which can hold K units of weight. (You can assume that K and each w_i is an integer.) You can stack multiple containers in each truck, subject to the weight restriction of K; the goal is to minimize the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don't have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers $1, 2, 3, \ldots$ into it until you get to a container that would overflow the weight limit. Now declare this truck "loaded" and send it off; then continue the process with a fresh truck.

(a) Give an example of a set of weights, and a value of K, where this algorithm does not use the minimum possible number of trucks.

(b) Show that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of K.

(2) *(This is Problem 9 at the end of Chapter 9.)* Consider an optimization version of the *Hitting Set* problem defined as follows. We are given a set $A = \{a_1, \ldots, a_n\}$ and a collection $B_1, B_2, \ldots, B_m$ of subsets of A. Also, each element $a_i \in A$ has a *weight* $w_i \geq 0$. The problem is to find a hitting set $H \subseteq A$ such that the total weight of the elements in H, $\sum_{a_i \in H} w_i$, is as small as possible. (Recall from Problem Set 7 that H is a hitting set if $H \cap B_i$ is not empty for each i). Let $b = \max_i |B_i|$ denote the maximum size of any of the sets $B_1, B_2, \ldots, B_m$. Give a polynomial time approximation algorithm for this problem that finds a hitting set whose total weight is at most b times the minimum possible.

(3) The difficulty in *3-SAT* comes from the fact that there are 2^n possible assignments to the input variables $x_1, x_2, \ldots, x_n$, and there's no apparent way to search this space in polynomial time. This intuitve picture, however, might create the misleading impression that the fastest algorithms for *3-SAT* actually require time 2^n. In fact, though it's somewhat counter-intuitive when you first hear it, there are algorithms for *3-SAT* that run in significantly less than 2^n time in the worst case; in other words, they determine whether there's a satisfying assignment in less time than it would take to enumerate all possible settings of

the variables.

Here we'll develop one such algorithm, which solves instances of *3-SAT* in $O(p(n) \cdot (\sqrt{3})^n)$ time for some polynomial $p(n)$. Note that the main term in this running time is $(\sqrt{3})^n$, which is approximately 1.73^n.

(a) For a truth assignment Φ for the variables $x_1, x_2, \ldots, x_n$, we use $\Phi(x_i)$ to denote the value assigned by Φ to x_i. (This can be either 0 or 1.) If Φ and Φ' are each truth assignments, we define the *distance* between Φ and Φ' to be the number of variables x_i for which they assign different values, and we denote this distance by $d(\Phi, \Phi')$. In other words, $d(\Phi, \Phi') = |\{i : \Phi(x_i) \neq \Phi'(x_i)\}|$.

A basic building block for our algorithm will be the ability to answer the following kind of question: Given a truth assignment Φ, and a distance d, we'd like to know whether there exists a satisfying assignment Φ' such that the distance from Φ to Φ' is at most d. Consider the following algorithm, *Explore*(Φ, d), that attempts to answer this question.

```
Explore(Φ,d)
  If Φ is a satisfying assignment then return ‘‘yes.’’
  Else if d = 0 then return ‘‘no.’’
  Else
    Let C_i be a clause that is not satisfied by Φ
      (i.e.  all three terms in C_i evaluate to false)
    Let Φ_1 denote the assignment obtained from Φ by
      taking the variable that occurs in the first term of clause C_i
      and negating it.
    Define Φ_2 and Φ_3 analogously in terms of the
      second and third terms of the clause C_i.
    Recursively invoke:
      Explore(Φ_1,d − 1)
      Explore(Φ_2,d − 1)
      Explore(Φ_3,d − 1)
    If any of these three calls returns‘‘yes’’ then return ‘‘yes.’’
    Else return ‘‘no.’’
```

Prove that *Explore*(Φ, d) returns "yes" if and only if there exists a satisfying assignment Φ' such that the distance from Φ to Φ' is at most d. Also, give an analysis of the running time of *Explore*(Φ, d) as a function of n and d.

(b) Clearly any two assignments Φ and Φ' have distance at most n from one another, so one way to solve the given instance of *3-SAT* would be to pick an arbitrary starting assignment Φ and then run *Explore*(Φ, n). However, this will not give us the running time we want.

Instead, we will need to make several calls to *Explore*, from different starting points Φ, and searching each out time out to more limited distances. Describe a way to do this in such a way that you can solve the instance of *3-SAT* in a running time of only $O(p(n) \cdot (\sqrt{3})^n)$.