# Problem Set 4

This problem set is due **at the beginning of** class on *Thursday, April 3, 2003.*

Each problem is to be done on a separate sheet (or sheets) of paper. Mark the top of each sheet with your name, 6.046J/18.410J, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

---

**Problem 4-1. Subsequences**

Consider a sequence of numbers stored in an array $A[1..n]$. A *subsequence* of this array is just a subset of the elements of the array $A[i_1], A[i_2], \ldots, A[i_k]$ where $i_1 < i_2 < \ldots < i_k$. Note that a subsequence does not have to be a contiguous block of elements.

**(a)** A subsequence is called an *increasing subsequence* if $A[i_1] < A[i_2] < \ldots < A[i_k]$. Devise an algorithm for computing the longest increasing subsequence of an array — that is, an increasing subsequence consisting of the greatest possible number of array elements. The algorithm should run in $O(n^2)$ time, and its output should be the indices $i_1 \ldots i_k$ of the array elements forming such a sequence.

**(b)** A subsequence is called a *bitonic subsequence* if for some $t$ we have $A[i_1] < A[i_2] < \ldots < A[i_t]$ and $A[i_t] > A[i_{t+1}] > \ldots > A[i_k]$. Devise an $O(n^2)$ algorithm for computing the longest bitonic subsequence of an array.

**(c)** Let's call a subsequence *pytonic* if $A[i_1] > A[i_2]$, $A[i_2] < A[i_3]$, $A[i_3] > A[i_4]$, $A[i_4] < A[i_5]$, etc. (this is the same definition as in quiz 1). Devise an $O(n)$ algorithm for computing the longest pytonic subsequence of an array.

**(d)** *Extra Credit.* Show how to solve parts (a) and (b) in only $O(n \log n)$ time.

**Problem 4-2. The Revenge of Professor Indyk's Sock Drawer**

After the last problem set, many students volunteered to sort Prof. Indyk's socks in order to receive extra credit. However, all of this sorting has left his socks in a very poor state of repair. In particular, each of his $2n$ socks now has some number of holes, given to us in an array $h[1] \ldots h[2n]$. Prof. Indyk would like to pair these socks up, giving him one pair to wear on each of the next $n$ days. However, since it is still somewhat cold outside he would like to avoid wearing socks with too many holes on any particular day. Specifically, for a given pairing of socks let us denote by $H_1 \ldots H_n$ the total number of holes in each of the $n$ pairs of socks, and let $M = \max_i H_i$. Please design an algorithm that, given $h[1 \ldots 2n]$, computes a pairing of socks that minimizes $M$. Try to make your algorithm as efficient as possible.

**Problem 4-3. Unique Minimum Spanning Trees**

In general, a graph can have several minimum spanning trees. However, if a graph has distinct edge costs, then one can show that it will have a unique minimum spanning tree. Please prove this fact.

In order to avoid a common error for this problem, please note the following. It may be tempting to argue that the claim follows from the fact that some algorithm (for example Kruskal's algorithm) can only generate one possible minimum spanning tree if all edge costs are distinct. However, this argument does not address the possibility of there being other minimum spanning trees that are not generated by the algorithm.

**Problem 4-4. Making Change**

Consider the problem of making change for $n$ cents using the least number of coins.

**(a)** Show that the greedy algorithm yields an optimal solution for the American coin denominations (pennies, nickels, dimes and quarters).

**(b)** As mentioned in class, the greedy approach doesn't work on all possible coin denominations. Give an $O(nk)$ dynamic programming algorithm which works for any set of $k$ different coin denominations. You may assume that the set includes pennies, and that $n$ and the different coin denominations are all integers. The output of your algorithm should describe the number of coins of each denomination that comprise an optimal solution.