# Algorithms, Winter 2010-11, Homework 4
# due Friday 01/14/11, 4:00pm

## Problem 1

Given is a sequence of distinct positive numbers. We want to find a subsequence with the maximum possible sum, with the restriction that we are not allowed to take three consecutive elements from the original sequence. For example, for input $1, 6, 5, 2, 7, 9, 3, 4$, the subsequence with the maximum possible sum is $6, 5, 7, 9, 4$ (we have two pairs of consecutive elements $6, 5$ and $7, 9$ but no three consecutive elements).

Now consider the following greedy strategies:

- Start with an empty subsequence. Go through the numbers in the sequence in decreasing order. For every number, if it does not cause three consecutive elements in the subsequence, include it in the subsequence (otherwise skip the number). For our example, we include 9, 7, 6, 5, 4, then we skip 3, 2, and 1.
- Go through the numbers in the sequence in increasing order, cross them out until you get a sequence with no three consecutive elements. For our example, we cross out 1, 2, and 3.
- Split the input into groups of three elements (the last group may have one or two elements). Go through the groups from left to right. For each group find the maximum element and include it in the subsequence. Go through the remaining elements in decreasing order, include the current element if it does not cause three consecutive elements in the subsequence, otherwise skip the element. For our example, the groups are $1, 6, 5$ and $2, 7, 9$ and $3, 4$. We include 6, 9, and 4 in the subsequence. Then we include 7 and 5, and skip 3, 2, and 1.

For each greedy strategy:

- State whether the strategy works or not.
- If you say "no", include (a) a counterexample input, (b) an optimum solution and its sum, and (c) the solution produced by the greedy algorithm and its sum.
- If you say "yes", include a short paragraph explaining the intuition behind the strategy.

No coding is required for this problem.

## Problem 2

Consider the following variant of the interval scheduling problem: For a given set of $n$ intervals (unweighted), we want to compute the total number of subsets of non-overlapping intervals.

For example, for these 4 intervals: $A = (2, 4)$, $B = (3, 6)$, $C = (5, 7)$, $D = (0, 1)$, we have the following subsets of non-overlapping intervals: $\emptyset, \{A\}, \{B\}, \{C\}, \{D\}, \{A, C\}, \{A, D\}, \{B, D\}, \{C, D\}, \{A, C, D\}$. Thus, the total number is 10.

Give an $O(n^2)$ dynamic programming algorithm for this problem.

Submit the pseudocode, the "heart of the solution" (see the note below), the running time estimate, and the implementation.

## Problem 3

We are given an $n \times n$ array $A$ of zeros and ones. Give an algorithm to find the size of the largest contiguous all-ones square. Your algorithm must run in time $O(n^2)$.

Submit the pseudocode, an explanation of how your algorithm works, the running time estimate, and the implementation.

## Note: Heart of the solution

For some of these and future problems you will need to use dynamic programming. To explain how your algorithm works, describe the "heart of the algorithm" (you do not need to include any other explanation). Recall that the heart of the algorithm consists of three parts:

- Precise verbal description of the meaning of your dynamic programming array, see the slides for examples.
- A mathematical formula that describes how to compute the value of each cell of the array (using previously computed array values).
- Return value of the algorithm.

This part of your solution is worth 50% of the score for the problem.