This exercise sheet is concerned with dynamic programming. If you have no experience, it can be very hard to apply dynamic programming directly. It may help to first design a recursive solution for understanding the problem, and then transform it into a dynamic program. An example of this transformation can be found in the sample solution for programming part 1.

A complete description of a dynamic program always consists of the following aspects (relevant for the exam!):

1) **Definition of the DP Table:** What are the dimensions of the table? What is the meaning of each entry?

2) **Computation of an Entry:** How can an entry be computed from the values of other entries? Which entries do not depend on others?

3) **Calculation order:** In which order can entries be computed so that values needed for each entry have been determined in previous steps?

4) **Extracting the solution:** How can the final solution be extracted once the table has been filled?

The running time of a dynamic program is usually easy to calculate by multiplying the size of the table with the time required to compute each entry. Sometimes, however, the time to extract the solution dominates the time to compute the entries.

**Exercise 7.1  Placing Wind Turbines.**

We want to place wind turbines along a road to produce energy. Due to geographical reasons \( n \) different positions are possible, but laws prescribe that the distance between two wind turbines has to be at least \( D \). The possible positions \( d_1, \ldots, d_n \) are given as coordinates on a line, where the leftmost position has the value 0. In other words: the distance between the \( i \)-th possible position and the first possible position is \( d_i \), we have \( d_1 = 0 \), and \( d_i < d_{i+1} \) for each \( i \in \{1, \ldots, n-1\} \).

When a wind turbine is installed at position \( i \), it produces energy \( e_i > 0 \). The task is to find a positioning of wind turbines that maximizes the energy yield.

\[
\begin{array}{ccccccc}
d_1 &=& 0 & d_2 &=& 4 & d_3 &=& 6 & d_4 &=& 10 & d_5 &=& 13 \\
e_1 &=& 6 & e_2 &=& 9 & e_3 &=& 5 & e_4 &=& 15 & e_5 &=& 11 \\
\times & \times & \times & \times & \times & \\
1 & 2 & 3 & 4 & 5 \end{array}
\]

\[\text{D=5} \quad \text{D=5}\]

**Example:** The image above shows a situation for \( n = 5 \) possible positions. For example, if a wind turbine is installed at position 3, then no wind turbines can be installed at the positions 2 or 4. When the wind turbines are placed on the positions 1, 3 and 5, then they produce \( 6+5+11 = 22 \) units of energy. This solution is not optimal: An installation of wind turbines on the positions 2 and 4 produces \( 9+15 = 24 \) units of energy.
a) Provide an example (as simple as possible) that shows that the following greedy strategy does not necessarily lead to an optimal solution: “Select a possible position with maximal energy yield until no other wind turbines can be placed.”

b) Describe a dynamic programming algorithm that computes the maximal producible energy as efficiently as possible.

c) Specify the running time of your solution.

d) Describe in detail how the above algorithm has to be modified to compute an optimal placement of wind turbines that produces a maximum amount of energy.

**Exercise 7.2  Ascending Sequences.**

In this exercise, we consider a two-dimensional array $A$ with $n$ rows and $m$ columns. The element $A[i][j]$ is adjacent to the elements $A[i−1][j]$, $A[i][j−1]$, $A[i+1][j]$ and $A[i][j+1]$, if these elements exist (elements at the borders of the array are adjacent to correspondingly fewer elements).

A sequence $x_1, x_2, ..., x_k$ of elements in the array is called ascending sequence if it satisfies the following conditions:

- the elements in the sequence are sorted in ascending order, and
- for every $i \in \{1, ..., k−1\}$, the elements $x_i$ and $x_{i+1}$ are adjacent in the array.

We search for a longest ascending sequence in a given two-dimensional array. In the example below, a possible sequence would be 4, 6, 28, 29, 47, 49. Design the most efficient algorithm that finds such longest ascending sequence using dynamic programming. Describe the algorithm, and specify its running time.

**Example array:**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>27</td>
<td>42</td>
<td>41</td>
<td>48</td>
</tr>
<tr>
<td>35</td>
<td>39</td>
<td>8</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>49</td>
<td>2</td>
<td>38</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>47</td>
<td>29</td>
<td>28</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>25</td>
<td>33</td>
<td>10</td>
</tr>
</tbody>
</table>

**Hand-in:** until Wednesday, 17th April 2013.