We have so far seen two major approaches to Dynamic programming.

1. Select a first (or last) element of a solution, and then figure out an optimal subsolution to whatever remains. This may (and usually does) require calculating many subsolutions to figure out what is best to pick first/last. For example, we present the rod-cutting problem’s recursive formula.

\[
D_p(l) = \begin{cases} 
  v(0) & l = 0 \\
  \max_{1 \leq l' \leq l} (D_p(l - l') + v(l')) & l > 0.
\end{cases}
\]

2. A ‘use it or lose it’ approach, where we order the elements of our input and consider them sequentially, either incorporating them into our solution or tossing them aside. For example, we present the Floyd-Warshall recursive formula, where we either use the new vertex \( i \) or we ignore it.

\[
D_{i,j}(k) = \begin{cases} 
  0 & i = 0 \land j = k \\
  w(j,k) & i = 0 \land (j,k) \in E \\
  \infty & i = 0 \land (j,k) \notin E \\
  \min(D_{i-1}(j,k), D_{i-1}(j,i) + D_{i-1}(i,k)) & \text{otherwise}.
\end{cases}
\]

Sometimes the use of these techniques may not be entirely obvious.

Consider a sequence of numbers \( A = a_1, \ldots, a_n \). We say \( A \) is an increasing sequence if \( a_1 \leq \ldots \leq a_n \).

A subsequence of \( A \) is a sequence \( B = a_{i_1}, \ldots, a_{i_k} \) where \( 1 \leq i_1 < \ldots < i_k \leq n \).

**Problem 1.** The longest increasing subsequence problem is in the name: to find the longest increasing subsequence of a given sequence.

When tackling a problem, it’s best to play with an example(s) to get a handle on it. Let \( D(A, i) \) be the longest increasing subsequence of \( A \) using \( a_1, \ldots, a_i \). Fill in the chart below for \( A = 0, 5, 6, 3, 4, 5 \).
Solution 1.

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0,5</td>
<td>0,5,6</td>
<td>0,5,6</td>
<td>0,5,6</td>
<td>0,5,6</td>
<td>0,3,4,5</td>
</tr>
</tbody>
</table>

In Problem 1, the abrupt and unexplained change in longest increasing subsequence from 0, 5, 6 to 0, 3, 4, 5 is a common sign that dynamic programming is not ready to be your friend. You may be missing a dimension(s) to your approach, or you may have structured your subproblems incorrectly, or your problem may simply not be amenable to dynamic programming at all.

For the longest increasing subsequence, we can think of this as either incorrectly structured or a missing dimension. We will focus on the former in this lab.

Problem 2. It’s not clear how it would help us to know that there is a first or last element of the longest increasing subsequence. Both certainly exist, but there is no readily apparent optimal substructure property we can exploit.

However, if we insist that the last element of the sequence is the last element of the increasing subsequence, then we are in business.

Let $D(A, i)$ be the longest increasing subsequence of $A$ ending in $a_i$ using $a_1, \ldots, a_i$. Fill in the chart below for $A = 0, 5, 6, 3, 4, 5, 2$.

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0,5</td>
<td>0,5,6</td>
<td>0,5,6</td>
<td>0,5,6</td>
<td>0,5,6</td>
<td>0,3,4,5</td>
</tr>
</tbody>
</table>

Solution 2.

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0,5</td>
<td>0,5,6</td>
<td>0,3</td>
<td>0,3,4</td>
<td>0,3,4,5</td>
<td>0,2</td>
<td></td>
</tr>
</tbody>
</table>

Problem 3. Using your answers to the previous problem as a guide, develop a recursive definition for $LIS_s(a_1, \ldots, a_n)$, the longest increasing subsequence of $A$ that includes the last element of $A$. You can give an English description if you like.

Solution 3. If $n < 1$, then we have the empty sequence. Otherwise, we append $a_n$ to the longest of $LIS_s(a_1, \ldots, a_i)$ for $1 \leq i < n$ for which $a_i \leq a_n$.

Problem 4. Since the longest increasing subsequence must end with something, one of the answers to $LIS_s$ must be correct. Give an algorithm that computes that longest increasing subsequence.

Solution 4.
Algorithm 1: LIS(A)

Input: A sequence \( A = \{a_1, \ldots, a_n\} (n \geq 0) \)
Output: A longest increasing subsequence of \( A \)

Let \( M \) be an array of length \( n \)
Let \( max = \bot \)

for \( i = 1 \) to \( n \) do
  // Worst-case scenario initialization
  \( M[i] = a_i \)

  // Find the longest compatible prior subsequence
  for \( j = 1 \) to \( i - 1 \) do
    if \( a_j \leq a_i \) and \( 1 + \text{len}(M[j]) > \text{len}(M[i]) \) then
      \( M[i] = M[j], a_i \)

  // Keep track of the maximum so we don’t have to find it later
  if \( \text{len}(max) < \text{len}(M[i]) \) then
    \( max = M[i] \)

return \( max \)