Rod Cutting

Given integers $p_1, \ldots, n$. How to cut up rod into integer length pieces of max total cost? Cost of piece of length $\ell$ is $p[\ell]$.
Backtracking

For each possible first length, check total possible using recursion to compute profit from remains.

Max Revenue(i): total we can earn cutting up rod of length i.

Final answer is Max Revenue(n).
\text{Max Revenue}(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max_{1 \leq j \leq i} \{ p[j] + \text{Max Revenue}(i-j) \} & \text{o.w.} \end{cases}

Memoize in array

\text{Max Revenue}[0..n]

So fill from left to right ($i \in [0 \text{ to } n]$)
\text{FastRodCutting}(n, P[1 \ldots n]):

\begin{align*}
\text{MaxRevenue}[0] &\leftarrow 0 \\
\text{for } i &\leftarrow 1 \text{ to } n \\
\text{MaxRevenue}[i] &\leftarrow -\infty \\
\text{for } j &\leftarrow 1 \text{ to } i \\
\text{if } P[j] + \text{MaxRevenue}[i-j] &> \text{MaxRevenue}[i] \\
\text{MaxRevenue}[i] &\leftarrow P[j] + \text{MaxRevenue}[i-j]
\end{align*}

\text{return } \text{MaxRevenue}[n]

\text{O}(n^2) \text{ time} \\
\text{O}(n) \text{ space}

\text{Could return lists of cuts by remembering best } j \text{ for each } i.
Dynamic Programming

Erickson Section 3.4

1) Formulate problem recursively
   a) Specification: Give a precise definition of the recursive subproblems. Also, what is the real answer to the original problem?

6) Solution: Recursive alg
or recurrence.

Recommended

2) Build solutions bottom up using some appropriate data structure.
   a) Identify subproblems.
      \text{RecFibo} + \text{MaxRevenue} used i.e. \{0, \ldots, n\}$\mathbb{Z}$.

6) Choose a memoization data structure (an array?)

6) Find an evaluation order.
d) Analyze space and time.
   Space: # subproblems
   Time: at most $A \cdot \text{subproblems}$
   Time per subproblem

e) Write the algorithm.
   For loops for each order
   Copy/paste the recurrence
to fill in table
WARNING: Don't be greedy! (yet)
Longest Increasing Subsequence

Given a sequence $S$, a subsequence of $S$ comes from deleting some elements but not reordering.
Given a sequence \( A[1..n] \) of integers, find the length of the longest subsequence of \( A \) such that elements are increasing.

In other words, want \( \max \) length \( 1 \leq i_1 < i_2 < \ldots < i_n \in n \) such that \( A[i_k] < A[i_{k+1}] \) for all \( k \).
<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5?</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Take the 5?
No!

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>8?</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Take the 8?
Check both options with recursion!

Subproblems based on:
- First index of the suffix largest (previous) element that came before
- Try to use as few subproblems
parameters as possible)

\[
\text{LIS bigger}(i, j): \text{length of LIS of } AC_j \ldots n, \quad \text{s.t. all elements greater than } AC_i \text{ (the "previous" element)}
\]

Let \(A(0) = -\infty\).

Want to return \(\text{LIS bigger}(0, 1)\)

\[
\text{LIS bigger}(i, j) = \begin{cases} 
\emptyset & \text{if } j > n \\
\text{LIS bigger}(i, j + 1) & \text{if } AC_i \geq AC_j \\
\max \{\text{LIS bigger}(i, j + 1), 1 + \text{LIS bigger}(i, j + 1)\} & \text{o.w.}
\end{cases}
\]
Subproblems: $0 \leq i \leq n$
$1 \leq j \leq n+1$

Data structure:
2D array
$LIS_{bigger}[0..n, 1..n+1]$

Dependencies + eval order:

Space: $O(n^2)$
Time: $O(n^2 \cdot O(1)) = O(n^2)$
FastLIS(A[1..n]):

\( A[0] \leftarrow -\infty \) \quad \langle Add \ a \ sentinel \rangle

\( \text{for } i \leftarrow 0 \text{ to } n \quad \langle \text{Base cases} \rangle \)
\( \quad LISbigger[i, n + 1] \leftarrow 0 \)

\( \text{for } j \leftarrow n \text{ down to } 1 \)
\( \quad \text{for } i \leftarrow 0 \text{ to } j - 1 \quad \langle \ldots \text{or whatever} \rangle \)
\( \quad \keeps \leftarrow 1 + LISbigger[j, j + 1] \)
\( \quad \skips \leftarrow LISbigger[i, j + 1] \)
\( \text{if } A[i] \geq A[j] \)
\( \quad LISbigger[i, j] \leftarrow \keeps \)
\( \text{else} \)
\( \quad LISbigger[i, j] \leftarrow \max\{\keeps, \skips\} \)

return \( LISbigger[0, 1] \)