Three Steps to Dynamic Programming

- Formulate the answer as a recurrence relation or recursive algorithm
- Show that number of different parameters in the recursive algorithm is "small" (e.g., bounded by a low-degree polynomial)
- Specify an order of evaluation for the recurrence so that already have the partial results ready when you need them.

Longest Increasing Run

- Given a sequence of integers $s_1, \ldots, s_n$ find a subsequence $s_i < s_{i+1} < \ldots < s_{i+k}$ so that $k > 0$ is as large as possible.
- e.g. Given $9, 5, 2, 5, 8, 7, 3, 1, 6, 9$ as input,
  - possible increasing subsequence is $1, 6$
  - better is $2, 5, 8$ or $1, 6, 9$ (either or which would be a correct output to our problem)

Longest Increasing Subsequence

- Given a sequence of integers $s_1, \ldots, s_n$ find a subsequence $s_{i_1} < s_{i_2} < \ldots < s_{i_k}$ with $i_1 < \ldots < i_k$ so that $k$ is as large as possible.
- e.g. Given $9, 5, 2, 5, 8, 7, 3, 1, 6, 9$ as input,
  - possible increasing subsequence is $2, 5, 7$
  - better is $2, 5, 6, 9$ or $2, 5, 8, 9$ (either or which would be a correct output to our problem; and there are others)
Find recursive algorithm

• Solve sub-problem on $s_1,...,s_{n-1}$ and then try to extend using $s_n$

• Two cases:
  - $s_n$ is not used
    * answer is the same answer as on $s_1,...,s_{n-1}$
  - $s_n$ is used
    * answer is $s_n$ preceded by the longest increasing subsequence in $s_1,...,s_{n-1}$ that ends in a number smaller than $s_n$

Refined recursive idea (stronger notion of subproblem)

• Suppose that we knew for each $i<n$ the longest increasing subsequence in $s_1,...,s_{n-1}$ ending in $s_i$.
  - $i=n-1$ is just the $n-1$ size sub-problem we tried before.

• Now to compute value for $i=n$ find
  - $s_n$ preceded by the maximum over all $i<n$ such that $s_i<s_n$ of the longest increasing subsequence ending in $s_i$
  - First find the best length rather than trying to actually compute the sequence itself.

Recurrence

• Let $L[i]=$length of longest increasing subsequence in $s_1,...,s_n$ that ends in $s_i$.

  $L[j]=1+\max\{L[i] : i<j \text{ and } s_i<s_j\}$
  (where max of an empty set is 0)

• Length of longest increasing subsequence:
  - $\max\{L[i] : 1\leq i \leq n\}$

Computing the actual sequence

• For each $j$, we computed $L[j]=1+\max\{L[i] : i<j \text{ and } s_i<s_j\}$
  (where max of an empty set is 0)

• Also maintain $P[j]$ the value of the $i$ that achieved that max
  - this will be the index of the predecessor of $s_j$ in a longest increasing subsequence that ends in $s_j$
  - by following the $P[j]$ values we can reconstruct the whole sequence in linear time.
Longest Increasing Subsequence Algorithm

• for \( j = 1 \) to \( n \) do
  \( L[j] \leftarrow 1 \)
  \( P[j] \leftarrow 0 \)
  for \( i = 1 \) to \( j - 1 \) do
    if \( s_i < s_j \) & \( L[i] + 1 > L[j] \) then
      \( P[j] \leftarrow i \)
      \( L[j] \leftarrow L[i] + 1 \)
  endfor
endfor

• Now find \( j \) such that \( L[j] \) is largest and walk backwards through \( P[j] \) pointers to find the sequence.

Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>90</td>
<td>50</td>
<td>20</td>
<td>80</td>
<td>70</td>
<td>30</td>
<td>10</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>( L_j )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P_j )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>