Example 1: \( \text{LIS (Longest Increasing Subsequence)} \)

1. Find subproblems and transition function

   \[ \{4, 2, 3, 5, 1, 7, 10, 8\} \]

   - Focus on last element of the array
   - Is 8 in the \( \text{LIS} \)?
     - Compare \( \emptyset \): Length of \( \text{LIS} \) if 8 is not in the sequence
     - \( \{6\} \): Length of \( \text{LIS} \) if 8 is in the sequence

   - To solve 1, call \( \text{LIS} \) on \( \{4, 2, 3, \ldots, 10\} \)

   - To solve 2, not as easy
     - In this example, \( \text{LIS} \{4, 2, 3, \ldots, 10\} = \{2, 3, 5, 7, 10\} \)
     - Cannot add 8 to this sequence

2. \( \{2, 10, 3, 5, 8\} \)

   \( \text{LIS} \{2, 10, 3, 5\} = \{2, 3, 5\} \)

   - Can/should add 8 to this sequence.

- Ideas:
  1. Have a subproblem \( a_{i,j} \)
     - \( a_{i,j} = \text{LIS} \) of first \( i \) elements whose
       - last element is smaller than \( j \)

  2. Have a subproblem \( a_{i,j} \)

     \[ a_{i,j} = \text{LIS of first } i \text{ elements that ends at } i\text{-th element} \]

     \[ \begin{align*}
     & 4, 2, 3, 5, 1, 7, 10, 8 \\
     & 1, 2, 3, 4, 5, 8
     \end{align*} \]

     How to compute \( a_{i,j} \)?
     - I-th element must be in
     - If \( j < i \), \( a_{i,j} = a_{i-1,j} \) then we can add
       \( a_{i,j} \) to the end of an \( \text{LIS ending at } a_{i,j} \)
\[ a[i] = \begin{cases} 1 & \text{if } A[i] < A[j] \text{ for all } j < i \\
\max_{j < i} a[j] + 1 & \text{otherwise} \end{cases} \]

2. figure out the base cases

3. find an appropriate order \( i = 1, 2, 3, \ldots, n \)

\[
\begin{align*}
\text{for } i &= 1 \text{ to } n \\
a[i] &= 1 \\
\text{for } j &= 1 \text{ to } i-1 \\
&\quad \text{if } A[j] < A[i] \text{ and } a[j] + 1 > a[i] \text{ Then} \\
&\quad a[i] = a[j] + 1
\end{align*}
\]

Example 2: Knapsack

Look at the last item

\[ \begin{cases} 
\text{max value if last item is in Knapsack} \\
\text{if last item is not in Knapsack.}
\end{cases} \]

- to solve \( W_n, V_n \) already in my Knapsack

should try to maximize value for first \( n-1 \) items

using a Knapsack of capacity \( W - w_n \).

(2) should try to maximize value for first \( n-1 \) items

using a Knapsack of capacity \( W \).

\[ a[i, j] = \max \begin{cases} 
0 & a[n-1, W - w_n] \\
a[n-1, w] & a[n-1, W]
\end{cases} \]

\[ a[i, j] = \max \begin{cases} 
a[i-1, j - w_i] + V_i & \text{putting } i \text{th element in } j \geq w_i \\
a[i-1, j] & \text{not putting } i \text{th element}
\end{cases} \]
$a_{[0, \text{anything}]} = 0 \quad \text{(no items)}$

$a_{[i, \text{any}]} = 0 \quad \text{(no capacity)}$

Ordering $i = 1 \rightarrow n$

$j = \begin{cases} a_{[i,j]} & \text{if } j \geq w_i \text{ and } a_{[i-1,j-w_i]} + v_i \geq a_{[i-1,j]} \\ \end{cases}$

$a_{[i,j]} = a_{[i-1,j-w_i]} + v_i$

(other orderings can also work)