Main idea of DP

STEP 1: Recursively define the value of an optimal solution using solutions of smaller problems (optimal substructure property)

STEP 2: Compute the value of an optimal solution from the smallest to the largest problems
One Dimensional (1D) Dynamic Programming

- Solve problems of minimum size first.
- Store solutions in 1D array.
- Gradually increase problem size.
- Choose the order that you solve problems, so that you re-use solutions of smaller problems stored in the 1D array.

How to gradually increase the problem size:
- If you need to select among a set of $n$ items, order items (sometimes in arbitrary order) and allow selection among the first $1, 2, \ldots, n$ items.
- If you need to solve for an integer amount or length $n$, gradually solve for $1, 2, \ldots, n$. 
1D DP Simple Problems $\Theta(n)$

1. **Fibonacci Numbers $\Theta(n)$**

   \[ F(n) = F(n - 1) + F(n - 2), \quad F(1) = 1, \quad F(2) = 2 \]

   \[
   \begin{array}{cccc}
   F(1) & F(2) & \cdots & F(n) \\
   1 & 2 & \cdots & \\
   \end{array}
   \]

2. **Maximum Sum problem $\Theta(n)$**: Given a sequence $A$ of $n$ positive numbers $a_1, a_2, \ldots, a_n$, find a subset $S$ of $A$ that has the maximum sum, provided that if we select $a_i$ in $S$, then we cannot select $a_{i-1}$ or $a_{i+1}$.
   - Let $A_i$ be the subsequence of the first $i$ numbers ($i \leq n$): $a_1, a_2, \ldots, a_i$
   - Let $W_i$ be the sum of numbers in the optimal solution for $A_i$.
   - **Recurrence**: $W_i = \max\{W_{i-2} + a_i, \, W_{i-1}\}$

   \[
   \begin{array}{cccc}
   W_1 & W_2 & \cdots & W_n \\
   a_1 & \max(a_1, a_2) & \cdots & \\
   \end{array}
   \]
1D DP More Complex Problems

1. Rod Cutting Problem $\Theta(n^2)$: Given a rod of length $n$ and prices $p_i$ for $i = 1, \ldots, n$, where $p_i$ is the price of a rod of length $i$, cut the rod in order to maximize the total revenue.
   - $r_n$ is maximum revenue from cutting rod of length $n$
   - Recurrence: $r_n = \max_{1 \leq i \leq n} \{p_i + r_{n-i}\}$, $r_0 = 0$

2. Minimum number of Coins $\Theta(nk)$: Given amount $n$ and $k$ denominations $d_1, \ldots, d_k$, find minimum number of coins for amount $n$
   - Let $d_i$ be the last denomination used
   - Then: $C[n] = 1 + C[n-d_i]$
     - Because after using 1 coin, the amount left is $n-d_i$
   - But I have to consider all possible ($k$) denominations
   - Recurrence: $C[n] = 1 + \min\{C[n-d_i]\}$, for $d_1, \ldots, d_k$
1. **Weighted interval scheduling** $\Theta(n \log n)$.
   - Job $j$ starts at $s_j$, finishes at $f_j$, and has weight (or value) $v_j$.
   - Two jobs **compatible** if they don't overlap.
   - **Goal**: find maximum-weight subset of mutually **compatible** jobs.

Sort jobs by finishing time: $f_1 \leq f_2 \leq \cdots \leq f_n$.

- $p(j)$ = largest index $i < j$ such that job $i$ is compatible with job $j$.
- $V[j]$ = value of optimal solution to the problem on jobs $1, 2, \ldots, j$.
- **Recurrence**: $V[j] = \max\{ v_j + V[p(j)], V[j-1] \}$, $V[0] = 0$

2. **Similar problem**- **Highway Billboards** $\Theta(n \log n)$: Consider a highway from west to east. You can place billboards at locations $x_1, x_2, \ldots, x_n$. If you place a billboard at $x_i$, you receive revenue of $r_i > 0$. You wish to place billboards as to maximize your total revenue, subject to the restriction that any two billboards must be at least 5 miles apart.
   - Sort the sites in increasing order of location $\{x_1, x_2, \ldots, x_n\}$
   - Recurrence and algorithm same as Weighted interval scheduling.
Exercise: Longest Monotonically Increasing Subsequence

Give an $O(n^2)$ time dynamic programming algorithm to find the longest monotonically increasing subsequence of a sequence of $n$ numbers, i.e, each successive number in the subsequence is greater than or equal to its predecessor.

For example, if the input sequence is

\[ \langle 5, 24, 8, 17, 12, 45 \rangle, \]

the output should be either \( \langle 5, 8, 12, 45 \rangle \) or \( \langle 5, 8, 17, 45 \rangle \).

Solution:

Let $X_i = \langle x_1, \ldots, x_i \rangle$ denote the prefix of $X$ consisting of its first $i$ items.

Define $c[i] =$ the length of the longest increasing subsequence that ends at $x_i$. 

Solution

$c[i] =$ the length of the longest increasing subsequence that ends at $x_i$.

Initial Condition: $c[1] = 1$

If $i > 1$:

If all items to left of $x_i$ are > than $x_i$, answer must be 1.

Otherwise, longest increasing subsequence that ends with $x_i$ has form $\langle Z, x_i \rangle$,
where $Z$ is the longest increasing subsequence that ends with $x_r$ for some $r < i$ and $x_r \leq x_i$.

This yields the following recurrence relation:

$$c[i] = \begin{cases} 
1 & \text{if } i = 1 \\ 
1 & \text{if } x_r > x_i \text{ for all } 1 \leq r < i \\ 
\max_{1 \leq r < i} c[r] + 1 & \text{other cases}
\end{cases}$$
Solution cont.

Store the $c[i]$'s in an array in increasing order of $i$.

After computing the $c$ array, we run through all the entries to find the maximum value.

This is the length of the longest increasing subsequence in $X$.

For every $i$ it takes $O(i)$ time to calculate $c_i$.

$=>$ the running time is $O(\sum_{i=1}^{n} i) = O(n^2)$. 

Example

\[
c[i] = \begin{cases} 
1 & \text{if } i = 1 \\
1 & \text{if } x_r > x_i \text{ for all } 1 \leq r < i \\
\max_{1 \leq r < i} c[r] + 1 & \text{other cases}
\end{cases}
\]

Question:

The input sequence is \( X = \{4, 5, 7, 1, 3, 9\} \); Find the longest monotonically increasing subsequence.

Solution:

\( i = 1: c[1] = 1 \)
Question: 

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$; 
Find the longest monotonically increasing subsequence.

Solution: 

$i = 1$: $c[1] = 1$

$i = 2$: Since $x_1 \leq x_2 \Rightarrow c[2] = \max\{c[1]\} + 1 = 2$
Example

\[ c[i] = \begin{cases} 
1 & \text{if } i = 1 \\
1 & \text{if } x_r > x_i \text{ for all } 1 \leq r < i \\
\max_{1 \leq r < i} c[r] + 1 & \text{other cases}
\end{cases} \]

Question:

The input sequence is \( X = \{4, 5, 7, 1, 3, 9\} \);
Find the longest monotonically increasing subsequence.

Solution:

\( i = 1 \): \( c[1] = 1 \)

\( i = 2 \): Since \( x_1 \leq x_2 \) \( \Rightarrow \) \( c[2] = \max\{c[1]\} + 1 = 2 \)

\( i = 3 \): Since \( x_1, x_2 \leq x_3 \) \( \Rightarrow \) \( c[3] = \max\{c[1], c[2]\} + 1 = 2 + 1 = 3 \)
**Question:**

The input sequence is $X = \{4, 5, 7, 1, 3, 9\}$; Find the longest monotonically increasing subsequence.

**Solution:**

$i = 1$: $c[1] = 1$

$i = 2$: Since $x_1 \leq x_2 \Rightarrow c[2] = \max\{c[1]\} + 1 = 2$

$i = 3$: Since $x_1, x_2 \leq x_3 \Rightarrow c[3] = \max\{c[1], c[2]\} + 1 = 2 + 1 = 3$

$i = 4$: Since $x_1, x_2, x_3 > x_4 \Rightarrow c[4] = 1$
Example

\[
c[i] = \begin{cases} 
1 & \text{if } i = 1 \\
1 & \text{if } x_r > x_i \text{ for all } 1 \leq r < i \\
\max_{1 \leq r < i} c[r] + 1 & \text{other cases}
\end{cases}
\]

Question:

The input sequence is \( X = \{4, 5, 7, 1, 3, 9\} \);
Find the longest monotonically increasing subsequence.

Solution:

\( i = 1 \): \( c[1] = 1 \)

\( i = 2 \): Since \( x_1 \leq x_2 \) \( \Rightarrow c[2] = \max\{c[1]\} + 1 = 2 \)

\( i = 3 \): Since \( x_1, x_2 \leq x_3 \) \( \Rightarrow c[3] = \max\{c[1], c[2]\} + 1 = 2 + 1 = 3 \)

\( i = 4 \): Since \( x_1, x_2, x_3 > x_4 \) \( \Rightarrow c[4] = 1 \)

\( i = 5 \): Since \( x_4 \leq x_5 \) and \( x_1, x_2, x_3 > x_5 \) \( \Rightarrow c[5] = \max\{c[4]\} + 1 = 2 \)
Example

\[ c[i] = \begin{cases} 
1 & \text{if } i = 1 \\
1 & \text{if } x_r > x_i \text{ for all } 1 \leq r < i \\
\max_{1 \leq r < i} c[r] + 1 & \text{other cases}
\end{cases} \]

**Question:**

The input sequence is \( X = \{4, 5, 7, 1, 3, 9\} \);
Find the longest monotonically increasing subsequence.

**Solution:**

\[ i = 1: c[1] = 1 \]

\[ i = 2: \text{Since } x_1 \leq x_2 \Rightarrow c[2] = \max\{c[1]\} + 1 = 2 \]

\[ i = 3: \text{Since } x_1, x_2 \leq x_3 \Rightarrow c[3] = \max\{c[1], c[2]\} + 1 = 2 + 1 = 3 \]

\[ i = 4: \text{Since } x_1, x_2, x_3 > x_4 \Rightarrow c[4] = 1 \]

\[ i = 5: \text{Since } x_4 \leq x_5 \text{ and } x_1, x_2, x_3 > x_5 \Rightarrow c[5] = \max\{c[4]\} + 1 = 2 \]

\[ i = 6: \text{Since } x_1, x_2, x_3, x_4, x_5 \leq x_6 \Rightarrow c[6] = \max\{c[1], c[2], c[3], c[4], c[5]\} + 1 = 4 \]

**Return:** max is \( c[6] = 4 \)
Solution

\[
c[i] = \begin{cases} 
1 & \text{if } i = 1 \\
1 & \text{if } x_r > x_i \text{ for all } 1 \leq r < i \\
\max_{1 \leq r < i} c[r] + 1 & \text{other cases}
\end{cases}
\]

To report optimal subsequence, we need to store for each \( i \), not only \( c[i] \), but also value of \( r \) which achieves the maximum in the recurrence relation.

Denote this by \( r[i] \). (\( \emptyset \) means no predecessor)

Suppose \( c[k] = \max_{1 \leq i \leq n} c[i] \). Let \( S \) be optimal subsequence.

\( x_k \) is the last item in \( S \). the optimal subsequence.

2\textsuperscript{nd} to last item in \( S \) is \( x_{r[k]} \),

3\textsuperscript{rd} to last item in \( S \) is \( x_{r[r[k]]} \), etc.

until we have found all the items in \( S \)

<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>X</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>c[i]</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>r[i]</td>
<td>\emptyset</td>
<td>1</td>
<td>2</td>
<td>\emptyset</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Running time of this step is \( O(n) \), so entire algorithm is still \( O(n^2) \).
Solution

\[ c[i] = \begin{cases} 
1 & \text{if } i = 1 \\
1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\
\max_{1 \le r < i} c[r] + 1 & \text{other cases}
\end{cases} \]

To report optimal subsequence, we need to store for each \( i \), not only \( c[i] \), but also value of \( r \) which achieves the maximum in the recurrence relation.

Denote this by \( r[i] \). (\( \emptyset \) means no predecessor)

**Return: max is** \( c[6] = 4 \), so \( k = 6 \)

Solution is

\[ x[r[r[6]]] \leftarrow x[r[6]] \leftarrow x[r] \leftarrow x_6 \]

i.e. \( x_1 \leftarrow x_2 \leftarrow x_3 \leftarrow x_6 \)

i.e. \{4, 5, 7, 9\}

\[ r[6] = 3 \]
\[ r[r[6]] = r[3] = 2 \]
\[ r[r[r[6]]] = r[2] = 1 \]
\[ r[r[r[r[6]]]] = r[1] = \emptyset \]
Alternative Solution

This problem can also be solved using the Longest Common Subsequence (LCS) Algorithm.

Let $X = \langle x_1, \ldots, x_n \rangle$ be the original input.
Set $Y = \langle y, \ldots, y_m \rangle$ be the items from $X$ sorted.

Example: $X = \langle 5, 24, 8, 17, 12, 45, 12 \rangle$, $Y = \langle 5, 8, 12, 12, 17, 24, 45 \rangle$

Then LCS($X, Y$) is exactly the Longest Increasing Subsequence of X (why?)
Alternative Solution

This problem can also be solved using the Longest Common Subsequence (LCS) Algorithm

Let \( X = \langle x_1, \ldots, x_n \rangle \) be the original input.
Set \( Y = \langle y, \ldots, y_m \rangle \) be the items from \( X \) sorted.

Example: \( X = \langle 5, 24, 8, 17, 12, 45, 12 \rangle \), \( Y = \langle 5, 8, 12, 12, 17, 24, 45 \rangle \)

Then \( \text{LCS}(X, Y) \) is exactly the Longest Increasing Subsequence of \( X \) (why?)

Since \( \text{LCS}(X, Y) \) uses \( O(n^2) \) time, this new algorithm also uses \( O(n^2) \) time.

Surprisingly, there is also an \( O(n \log n) \) algorithm for solving the problem. See https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf
Two Dimensional (2D) Dynamic Programming

- These problems require a 2D array for the storage of solutions.
- In all the problems, we fill the 2D array row-by-row:
  - That is, first we finish the first row, then the second and so on.
- Usually, but not always, the final solution is at the bottom right corner of the array.
- In all the problems discussed during class, the running time is the same as the array size.
- However, in some cases, we do not need to keep the entire array, as algorithms only require the last two rows.
2D DP 0-1 Knapsack $\Theta(nW)$

1. **Input**: A set of $n$ items, where item $i$ has weight $w_i$ and value $v_i$, and a knapsack with capacity $W$.

   **Goal**: Find $x_1, \ldots, x_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} x_i w_i \leq W$ and $\sum_{i=1}^{n} x_i v_i$ is maximized.

   - $V[i,j]$ be the largest obtained value for a knapsack with capacity $j$, choosing only from the first $i$ items.

   - **Recurrence**: $V[i,j] = \max(V[i-1,j], v_i + V[i-1,j-w_i])$

     $V[i,j] = 0$ if $i = 0$ or $j = 0$

---

<table>
<thead>
<tr>
<th>$\mathbf{V[i,j]}$</th>
<th>$j=0$</th>
<th>$j=1$</th>
<th>$j=2$</th>
<th>$j=W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i=0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$i=1$</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$i=2$</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>$v_1 \text{ or } v_2$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i=n$</td>
<td>0</td>
<td></td>
<td></td>
<td>$V[n-1, W-w_n]$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>$V[n-1, W]$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>$V[n, W]$</td>
</tr>
</tbody>
</table>
2D DP Longest Common Subsequence $\Theta (mn)$

Given two sequences $X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2, \ldots, y_n)$, $Z$ is a common subsequence of $X$ and $Y$ if $Z$ has a strictly increasing sequence of indices $i$ and $j$ of both $X$ and $Y$ such that we have $x_{i_p} = y_{j_p} = z_p$ for all $p = 1, 2, \ldots, k$. Find the LCS of $X$ and $Y$.

- $c[i, j]$ is the length of the LCS of $X[1..i]$ and $Y[1..j]$
- **Recurrence:** $c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\max\{c[i - 1, j] + 1, c[i - 1, j - 1] + 1\} & \text{if } x_i = y_j \\
\max\{c[i, j - 1], c[i - 1, j]\} & \text{if } x_i \neq y_j 
\end{cases}$
2D DP Longest Common Substring Θ(mn)

Given two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$, we wish to find their longest common substring $Z$, that is, the largest $k$ for which there are indices $i$ and $j$ with $x_i x_{i+1} \ldots x_{i+k-1} = y_j y_{j+1} \ldots y_{j+k-1}$.

- $d[i, j] =$ the length of the longest common substring of $X[1..i]$ and $Y[1..j]$ that ends at $x_i$ and $y_j$.

**Recurrence:**

$$d[i, j] = \begin{cases} 
    d[i-1, j-1] + 1 & \text{if } x_i = y_j \\
    0 & \text{if } x_i \neq y_j 
\end{cases}$$

<table>
<thead>
<tr>
<th></th>
<th>$j=0$</th>
<th>$j=1$</th>
<th>$j=2$</th>
<th>$j=n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i=0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$i=1$</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>$i=2$</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$i=m$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$i=n$</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

$max d = d[m-1, n-1]$
Find the edit distance between strings $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$. Edit distance is the smallest number of operations to turn $X$ into $Y$:

1. Insertion: add a letter
2. Deletion: remove a letter
3. Substitution: replace a character with another one.

- $E[i, j] =$ edit distance of $X[1..i]$ and $Y[1..j]$

$$E[i, j] = \min \begin{cases} 
1 + E[i, j - 1] \\
1 + E[i - 1, j] \\
E[i - 1, j - 1] & \text{if } x_i = y_j, \text{or } E[i - 1, j - 1] + 1 \text{ if } x_i \neq y_j,
\end{cases}$$
Exercise Max Square Sub-Matrix with all 1s

Given a $n \times m$ binary matrix $M$ filled with 0's and 1's, find the area of the largest square containing all 1's.

Example:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- $S[i][j]$: size of the square sub-matrix with all 1's including $M[i][j]$, where $M[i][j]$ is the right bottom entry in sub-matrix.

Max Square Sub-Matrix with all 1s: Recurrence

- Recurrence:
  - If $M[i][j]$ is 1 then $S[i][j] = \min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1$
  - If $M[i][j]$ is 0 then $S[i][j] = 0$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

|     | 0   | 1   | 1   | 0   | 1   |
|-----|-----|-----|-----|-----|
| 0   | 1   | 0   | 1   | 0   |
| 1   | 1   | 1   | 1   | 0   |
| 0   | 1   | 1   | 1   | 0   |
| 1   | 1   | 2   | 2   | 0   |
| 1   | 2   | 2   | 3   | 1   |
| 0   | 0   | 0   | 0   | 0   |