Dynamic Programming is a technique we use when our problem has:

Optimal Substructure $+$ Repeated Sub-problems

\[ \downarrow \]

solution to problem is a combination of solutions to sub-problems

\[ \downarrow \]

any recursive algorithm would solve the same sub-problems over & over
Dynamic Programming:

1. Break down problem into sub-problems
2. Solve sub-problems
   - either top-down or bottom-up
3. Store solutions to sub-problems
   - when you compute sub-problems top-down, you store them using memoization
Dynamic Programming – Longest Increasing Subsequence

S: 23 3 5 18 10 101 12 14 4 105
Dynamic Programming — Longest Increasing Subsequence

$S: 23 \ 3 \ 5 \ 18 \ 10 \ 101 \ 12 \ 14 \ 4 \ 105$

$\downarrow$

$23 \ 101 \ 105$

- increasing subsequence

Size = 3

we can do better...
Dynamic Programming – Longest Increasing Subsequence

S: 23 3 5 18 10 101 12 14 4 105
Dynamic Programming - Longest Increasing Subsequence

$S: 23 \ 3 \ 5 \ 18 \ 10 \ 101 \ 12 \ 14 \ 4 \ 105$
Dynamic Programming - Longest Increasing Subsequence

S: 23 3 5 18 10 101 12 14 4 105
Dynamic Programming – Longest Increasing Subsequence

S: 23 3 5 18 10 101 12 14 4 105

3 5 18 101 105 → Size 5

Better! But we can still do even better...
Dynamic Programming – Longest Increasing Subsequence

S: 23 3 5 18 10 101 12 14 4 105
Dynamic Programming – Longest Increasing Subsequence

\[ S: 23 \ 3 \ 5 \ 18 \ 10 \ 101 \ 12 \ 14 \ 4 \ 105 \]

Longest increasing subsequence

\[ L(S) = 3 \rightarrow 5 \rightarrow 10 \rightarrow 12 \rightarrow 14 \rightarrow 105 \]

Size = 6
Dynamic Programming - Longest Increasing Subsequence

$S: 23 \quad 3 \quad 5 \quad 18 \quad 10 \quad 101 \quad 12 \quad 14 \quad 4 \quad 105$

- Could try including/excluding every element
- $2^n$ subsequences to check
Dynamic Programming - Longest Increasing Subsequence

$S: 23 \ 3 \ 5 \ 18 \ 10 \ 101 \ 12 \ 14 \ 4 \ 105$

Try Dynamic Programming!

Which means we need:

- recursive expression with repeated subproblems
- store solutions to subproblems so that they are quickly accessible
$S: \quad 1 \quad 2 \quad 3 \quad 5 \quad 18 \quad 10 \quad 101 \quad 12 \quad 14 \quad 4 \quad 105$

$L_n(S):$ longest increasing subsequence that uses $S(n)$
$S: \quad 23 \quad 3 \quad 5 \quad 18 \quad 10 \quad 101 \quad 12 \quad 14 \quad 4 \quad 105$

$L_n(S):$ longest increasing subsequence that uses $S(n)$

ex. $L_{n-2}(S) = 3, 5, 10, 12, 14 \quad |L_{n-2}| = 5$
\[ S : \quad \frac{1}{23} \quad 3 \quad 5 \quad 18 \quad 10 \quad 101 \quad 12 \quad 14 \quad n-1 \quad n \]

\[ L_n(S) : \text{longest increasing subsequence that uses } S(n) \]

\[ \text{ex. } L_{n-2}(S) = 3, 5, 10, 12, 14 \quad |L_{n-2}| = 5 \]

\[ L_{n-1}(S) = \quad ?? \]
\[ S: \ 1 \ 23 \ 3 \ 5 \ 18 \ 10 \ 101 \ 12 \ 14 \ n^{-1} \ 4 \ n \]

\[ L_n(S) : \text{longest increasing subsequence that uses } S(n) \]

\[ \text{ex. } L_{n-2}(S) = 3, 5, 10, 12, 14 \quad |L_{n-2}| = 5 \]
\[ L_{n-1}(S) = 3, 4 \quad |L_{n-1}| = 2 \]
$S: 23 3 5 18 10 101 12 14 4 105$

$L_n(S)$: longest increasing subsequence that uses $S(n)$

ex. $L_{n-2}(S) = 3, 5, 10, 12, 14$ $|L_{n-2}| = 5$

$L_{n-1}(S) = 3, 4$ $|L_{n-1}| = 2$

$L_n(S) = ??$
\[ S: \quad 23 \quad 3 \quad 5 \quad 18 \quad 10 \quad 101 \quad 12 \quad 14 \quad 4 \quad 105 \]

\[ L_n(S): \quad \text{longest increasing subsequence that uses } S(n) \]

\[ \text{ex.} \quad L_{n-2}(S) = 3, 5, 10, 12, 14 \quad |L_{n-2}| = 5 \]
\[ L_{n-1}(S) = 3, 4 \quad |L_{n-1}| = 2 \]

\[ L_n(S) = 3, 5, 10, 12, 14, 105 \]
\[ S: \quad 23 \quad 3 \quad 5 \quad 18 \quad 10 \quad 101 \quad 12 \quad 14 \quad 4 \quad 105 \]

\[ L_n(S): \text{ longest increasing subsequence that uses } S(n) \]

\[ \text{ex. } \quad L_{n-2}(S) = 3, 5, 10, 12, 14 \quad |L_{n-2}| = 5 \]

\[ L_{n-1}(S) = 3, 4 \quad |L_{n-1}| = 2 \]

\[ L_n(S) = 3, 5, 10, 12, 14, 105 \quad |L_n| = 1 + |L_{n-2}| \]

\[ L_{n-2}(S) \]
\[ S: \quad 1 \quad 2 \quad 3 \quad 5 \quad 18 \quad 10 \quad 101 \quad 12 \quad 14 \quad 4 \quad 105 \]

\[ \text{Ln}(S) : \quad \text{longest increasing subsequence that uses } S(n) \]

\[ |L_n| = ?? \]
$S: \begin{array}{cccccccccccc}
1 & 2 & 3 & 5 & 18 & 10 & 101 & 12 & 14 & 4 & 105
\end{array}$

$L_n(S):$ longest increasing subsequence that uses $S(n)$

$|L_n| = ??$

look at all $L_j$ $(j < n)$
$S: \ 23 \ 3 \ 5 \ 18 \ 10 \ 101 \ 12 \ 14 \ 4 \ 105$

$L_n(S):$ longest increasing subsequence that uses $S(n)$

$|L_n| = 1 + ?$

look at all

$L_j \ (j < n)$
\[ S : \ 1 \ 2 \ 3 \ 5 \ 18 \ 10 \ 101 \ 12 \ 14 \ 4 \ 105 \]

\[ L_n(S) : \text{longest increasing subsequence that uses } S(n) \]

\[ |L_n| = 1 + \max \{|L_j| \mid \text{all } j \text{ s.t. } S(j) < S(n)\} \]

look at all \[ L_j \ (j < n) \]
$$|L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j|$$

Recursion:

BAD

\[L_{n-1} \quad L_{n-2} \quad \ldots \quad L_1\]

\[L_{n-2} \quad L_{n-3} \quad \ldots \quad \text{etc}\]
\[ |L_n| = 1 + \max_{\{ \text{all } j \text{ s.t. } S[j] < S[n] \}} |L_j| \]

**Recursion:** $L_n$

**BAD**

$L_{n-1}$ $L_{n-2}$ ... $L_1$

$L_{n-2}$ $L_{n-3}$ ...

etc

**Dyn.Prog.:** Build solutions, "bottom up"

When it's time to solve $|L_k|$ we have stored all $|L_j|$ ($j < k$) in an array.
\[ |L_n| = 1 + \max_{\{ \text{all } j \text{ s.t. } S[j] < S[n] \}} |L_j| \]

**Recursion:** \( L_n \)

**BAD**

\[ L_{n-1} L_{n-2} \ldots L_1 \]

\[ L_{n-2} L_{n-3} \ldots \text{ etc} \]

**Dyn.Prog:** Build solutions, "bottom up"

When it's time to solve \( |L_k| \) we have stored all \( |L_j| \) \((j<k)\) in an array.
23, 3, 5, 18, 10, 101, 12, 14, 4

\[ 1 \div \frac{1}{|L_2|} \]

\[ |L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j| \]

Recursion: \( L_n \)

BAD

\[ L_{n-1} L_{n-2} \ldots L_1 \]

\[ L_{n-2} L_{n-3} \ldots \]

etc

Dyn.Prog: Build solutions, "bottom up"

When it's time to solve \( |L_k| \) we have stored all \( |L_j| (j < k) \) in an array.
\[ |L_n| = 1 + \max_{\text{all } j \text{ s.t. } S[j] < S[n]} |L_j| \]

Recursion: \( L_n \)  

BAD  

\( L_{n-1} \ L_{n-2} \cdots L_1 \)  

\( L_{n-2} \ L_{n-3} \cdots \) etc  

Dyn. Prog: Build solutions, "bottom up"  

When it's time to solve \( |L_k| \) we have stored all \( |L_j| \) (\( j < k \)) in an array.
\[
|L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j|
\]

**Recursion:** \( L_n \)  
**BAD**  
\[ L_{n-1} \quad L_{n-2} \quad \ldots \quad L_1 \]
\[ L_{n-2} \quad L_{n-3} \quad \ldots \quad \text{etc} \]

**Dyn. Prog:** Build solutions, "bottom up"  
When it's time to solve \(|L_k|\) we have stored all \(|L_j|\) \((j<k)\) in an array.
\[ |L_n| = 1 + \max_{\{all \ j \ s.t. \ S[j] < S[n]\}} |L_j| \]

**Recursion:** \( L_n \)

**BAD**

\( L_{n-1}, L_{n-2}, \ldots, L_1 \)

\( L_{n-2}, L_{n-3}, \ldots \)

etc

**Dyn. Prog:** Build solutions, "bottom up"

When it's time to solve \(|L_k|\) we have stored all \(|L_j| (j<k)\) in an array.
\[
|L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} \left|L_j\right|
\]

Recursion: \( L_n \)

BAD

\[
L_{n-1} L_{n-2} \ldots L_1
\]

\[
L_{n-2} L_{n-3} \ldots \text{ etc}
\]

Dyn. Prog: Build solutions, "bottom up"

When it's time to solve \( |L_k| \) we have stored all \( |L_j| (j < k) \) in an array.
\[ |L_n| = 1 + \max \left\{ |L_j| : j \text{ s.t. } S[j] < S[n] \right\} \]

**Recursion:** \( L_n \)

**BAD**

\[ L_{n-1} \quad L_{n-2} \quad \cdots \quad L_1 \]

\[ L_{n-2} \quad L_{n-3} \quad \cdots \quad \text{etc} \]

**Dyn. Prog.:** Build solutions, "bottom up". When it's time to solve \( |L_k| \) we have stored all \( |L_j| (j<k) \) in an array.
\[ |L_n| = 1 + \max \{ \text{all } j \text{ s.t. } S[j] < S[n] \} |L_j| \]

Recursion: \( L_n \)

BAD

\( L_{n-1} \quad L_{n-2} \cdots L_1 \)

\( L_{n-2} \quad L_{n-3} \cdots \)

etc

Dyn. Prog: Build solutions, "bottom up"

When it's time to solve \( |L_k| \) we have stored all \( |L_j| (j < k) \) in an array.
23, 3, 5, 18, 10, 101, 12, 14, 4
1 1 2 3 3 4 4 5 2 \rightarrow \text{Score may decrease}

|L_n| = 1 + \max_{i \text{ s.t. } S[i] < S[n]} |L_j|

Recursion: \quad L_n
\quad \quad L_{n-1} L_{n-2} \ldots L_1
\quad \quad L_{n-2} L_{n-3} \ldots \text{ etc}

Dyn. Prog.: Build solutions, "bottom up"
When it's time to solve |L_k| we have stored all |L_j| (j<k) in an array.
\[|L_n| = 1 + \max_{\{j \text{ s.t. } S[j] < S[n]\}} |L_j|\]

Recursion: \(L_n\)

BAD

\(L_{n-1} L_{n-2} \ldots L_1\)

\(L_{n-2} L_{n-3} \ldots \text{ etc}\)

Dyn. Prog: Build solutions, "bottom up"

When it's time to solve \(|L_k|\) we have stored all \(|L_j|\) (j<k) in an array.

**time? space?**
\begin{align*}
|L_n| &= 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j| \\
\text{Recursion:} & \quad L_n \\
\text{BAD} & \quad L_{n-1}, L_{n-2}, \ldots, L_1 \\
\text{etc} & \quad L_{n-2}, L_{n-3}, \ldots \\
\text{Dyn. Prog:} & \quad \text{Build solutions, “bottom up”}
\end{align*}

When it’s time to solve $|L_k|$ we have stored all $|L_j|$ ($j < k$) in an array.

$T(k) = \Theta(k)$

$T(n) = \sum_{i=1}^{n} T(k) = \Theta(n^2)$

Space = $\Theta(n)$
\[ T(n) = \Theta(n^2) \]

\[ \text{space} = \Theta(n) \]

\[ |L_n| = 1 + \max_{\{j \text{ s.t. } S[j] < S[n]\}} |L_j| \]
23, 3, 5, 18, 10, 101, 12, 14, 4
1 1 2 3 3 4 4 5 2

\[ |L_n| = 1 + \max_{\{\forall j \text{ s.t. } S[j] < S[n]\}} |L_j| \]

What about \(|L.I.S.|\)? \(= |L_n|\)?

\(T(n) = \Theta(n^2)\)

\(\text{space} = \Theta(n)\)
\[ |L_n| = 1 + \max \{ \text{all } j \text{ s.t. } S[j] < S[n] \} |L_j| \]

What about \( |L.I.S.| \)? \( = |L_n|? \rightarrow \text{NO.} \)

\[ = \max_{j=1..n} |L_j| \]

\[ T(n) = \Theta(n^2) \]

\[ \text{space} = \Theta(n) \]
\[ T(n) = \Theta(n^2) \]

\[ \text{space} = \Theta(n) \]

\[ |L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j| \]

What about \(|L.I.S.|\)?  \(= |L_n|\)?  \(\rightarrow \text{NO.}\)  \(= \max_{j=1..n} |L_j|\)

What about L.I.S.?
$T(n) = \Theta(n^2)$

$\text{space} = \Theta(n)$

$$|L_n| = 1 + \max_{\text{all } j \text{ s.t. } S[j] < S[n]} |L_j|$$

What about $|L.I.S.|$? $= |L_n|$? → NO. $= \max_{j=1..n} |L_j|$.

What about L.I.S.? Keep the pointers: for each $S[j]$ store any $S[i]$ pointer that generated $|L_j|$.
A quick solution for L.I.S. ... but still \( O(n^2) \) & dyn-prog.
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

23, 3, 5, 18, 10, 101, 12, 14 : S
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

23, 3, 5, 18, 10, 101, 12, 14 : S

\[ \text{sort} \]

3, 5, 10, 12, 14, 18, 23, 101 : S_2

and then?
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

\[23, 3, 5, 18, 10, 101, 12, 14 : S\]

\[\text{sort} \leftarrow\]
\[3, 5, 10, 12, 14, 18, 23, 101 : S_2\]

Find longest common subsequence!
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

$$23, 3, 5, 18, 10, 101, 12, 14 : S$$

$$3, 5, 10, 12, 14, 18, 23, 101 : S_2$$

**Find longest common subsequence!**

- any common subsequence is increasing (assume no duplicates)
  or remove them
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

$$23, 3, 5, 18, 10, 101, 12, 14 : S$$

$$3, 5, 10, 12, 14, 18, 23, 101 : S_2$$

FIND LONGEST COMMON SUBSEQUENCE!

- Any common subsequence is increasing (assume no duplicates) or remove them.

so $\text{LCS}(S, S_2)$ qualifies as a solution.
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

$$23, 3, 5, 18, 10, 101, 12, 14 : S$$

find longest common subsequence!

- any common subsequence is increasing (assume no duplicates)
  so $LCS(S, S_2)$ qualifies as a solution

- LIS must exist in $S_2$
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

\[
23, 3, 5, 18, 10, 101, 12, 14 : S
\]
\[
3, 5, 10, 12, 14, 18, 23, 101 : S_2
\]

\[\text{sort} \rightarrow \]

\text{FIND LONGEST COMMON SUBSEQUENCE !}

- any common subsequence is increasing (assume no duplicates)
  so $\text{LCS}(S, S_2)$ qualifies as a solution

- LIS must exist in $S_2$, so it is a candidate for LCS.