Dynamic Programming - Longest Increasing Subsequence

23, 3, 5, 18, 10, 101, 12, 14, 4, 105
Dynamic programming - longest increasing subsequence

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

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

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

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

\[ L(S) = 3, 5, 10, 12, 14, 105 \quad \left| L(S) \right| = 6 \]
Dynamic Programming - Longest Increasing Subsequence

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

\[ L(S) = 3, 5, 10, 12, 14, 105 \quad \mid L(S) \mid = 6 \]

Could try including/excluding every element:

2^n subsequences to check
Dynamic Programming - Longest Increasing Subsequence

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

\[ L(S) = 3, 5, 10, 12, 14, 105 \quad |L(S)| = 6 \]

For dynamic programming we would like

- a recursive expression w/ repeated subproblems
- an easy, fast way to use solved subproblems
23, 3, 5, 18, 10, 101, 12, 14

$L(S) \sim \sim L_{1\ldots n}(S) \sim L_n$
\[ L(s) \sim L_{i \ldots n}(s) \sim L_n \]

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

\[ |L_n| \text{ using } |L_{n-1}| ? \]
$23, 3, 5, 18, 10, 101, 12, 14$

$L(S) \sim L_{1\ldots n}(S) \sim L_n$

$|L_n|$ using $|L_{n-1}|$?

if $S[n] > \text{last element in } L_{n-1}$ then ??
\[ L(S) \sim L_{1 \ldots n}(s) \sim L_n \]

|L_n| using |L_{n-1}| ?

if \( S[n] > \) last element in \( L_{n-1} \) then \( |L_n| = |L_{n-1}| + 1 \)

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

\[ \text{if } S[n] > \text{last element in } L_{n-1} \text{ then } |L_n| = |L_{n-1}| + 1 \]

\[ L(S) \sim L_{1..n}(S) \sim L_n \]

Could be at any position in S
L(S) \sim L_{i\ldots n}(s) \sim L_n

|L_n| \text{ using } |L_{n-1}| \ ?

if \ S[n] > \text{ last element in } L_{n-1} \ \text{ then } |L_n| = |L_{n-1}| + 1

\Rightarrow \text{ could be at any position in } S

else \ \ \ \ S[n] \leq \text{ last } (L_{n-1})

??
\[ L(S) \sim L_{1\ldots n}(S) \sim L_n \]

\[ |L_n| \text{ using } |L_{n-1}| ? \]

if \( S[n] > \text{last element in } L_{n-1} \) then \( |L_n| = |L_{n-1}| + 1 \)

\( \Rightarrow \) could be at any position in \( S \)

else \( \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ S[n] \leq \text{last} (L_{n-1}) \)

\( \Rightarrow \) keep \( |L_{n-1}| ? \)

\( \Rightarrow \) add \( S[n] \) to suboptimal solution from \( S[1\ldots n-1] \)?
Redefine: $L_n = \text{longest increasing subsequence that actually uses } S[n]$
Redefine: $L_n =$ longest increasing subsequence that actually uses $S[n]$
Redefine: $L_n =$ longest increasing subsequence that actually uses $S[n]$

$$|L_n| =$$

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

\[ |L_{n-1}| = 2 \]

Redefine: \( L_n = \) longest increasing subsequence that actually uses \( S[n] \)

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

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

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

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

Recursion: \( L_n \)

BAD

\( L_{n-1}, L_{n-2}, \ldots, L_1, L_{n-2}, L_{n-3}, \ldots, \text{ etc} \)

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

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

Recursion: \( L_n \)

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

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

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

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

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

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

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

Recursion: \( L_n \)

BAD: \( L_{n-1}, L_{n-2}, \ldots, L_1, L_{n-2}, L_{n-3}, \ldots \)

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

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

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

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

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

Score may decrease
23, 3, 5, 18, 10, 101, 12, 14, 4
1 1 2 3 3 4 4 5 2 → Score may decrease

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

Recursion: \( L_n \)

BAD

\( L_{n-1}, L_{n-2}, \ldots, L_1, \ldots, L_{n-2}, L_{n-3}, \ldots \) 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.

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

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.

\[ T(k) = \Theta(k) \]

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

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

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

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

Space = \( \Theta(n) \)

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

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

space = \Theta(n)

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

What about |L.I.S.|? = \max_{j=1..n} |L_j|
23, 3, 5, 18, 10, 101, 12, 14, 4
1 1 2 3 3 4 4 5 2

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

What about \( |L.I.S.| \)?
\[ = \max_{j=1\ldots n} |L_j| \]

What about L.I.S.?
23, 3, 5, 18, 10, 101, 12, 14, 4
1 1 2 3 3 4 4 5 2

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

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

What about \( |L.I.S.| \)?
\[ = \max_{j=1\ldots n} |L_j| \]

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.

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

sort $\rightarrow$

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

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

Find longest common subsequence!

- any common subsequence is increasing
A quick solution for L.I.S. ... but still \(O(n^2)\) & dyn-prog.

23, 3, 5, 18, 10, 101, 12, 14 \(\Rightarrow\) \(S\)

3, 5, 10, 12, 14, 18, 23, 101 \(\Rightarrow\) \(S_2\)

**Find longest common subsequence!**

- any common subsequence is increasing
  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

sort →

3, 5, 10, 12, 14, 18, 23, 101 : S₂

FIND LONGEST COMMON SUBSEQUENCE!

- any common subsequence is increasing
  so LCS(S, S₂) qualifies as a solution
- LIS must exist in S₂
A quick solution for L.I.S. ... but still $O(n^2)$ & dyn-prog.

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

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

Find longest common subsequence!

- any common subsequence is increasing
  so $LCS(S, S_2)$ qualifies as a solution
- L.I.S. must exist in $S_2$, so it is a candidate for LCS.