Dynamic Programming - Longest Increasing Subsequence

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

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

\{23, 101\} is an increasing subsequence; size = 2
Dynamic Programming - Longest Increasing Subsequence

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

size = 4
Dynamic Programming - Longest Increasing Subsequence

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

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

Size = 5

Could try including/excluding every element:

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

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

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

\[ |L(S)| = 5 \]

For dynamic programming we would like

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

$L(S) \sim L_{1...n}(S) \sim L_n$
\[ 23, 3, 5, 18, 10, 101, 12, 14 \]

\[ \sim \]

\[ L_i(n) \]

\[ L_{n-1}(s) \]

\[ L_n \]

\[ |L_n| \text{ using } |L_{n-1}| ? \]
$1, 2, 3, \ldots, n-1, n$

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

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

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

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

Find \(|L_n|\) using \(|L_{n-1}|\)?

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

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

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

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

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

\[ \text{find } |L_n| \]

\[ \Rightarrow \text{could be at any position in } S \]
23, 3, 5, 18, 10, 101, 12, 14

|Ln| using |Ln-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 $S[n] \leq \text{last}(L_{n-1})$

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

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

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

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

\[ \begin{cases} 
|L_n| \\
\text{find} \\
\text{if } S[n] > \text{last element in } L_{n-1} \quad \text{then } |L_n| = |L_{n-1}| + 1 \\
\text{else} \\
|L_{n-1}| \ \text{keep } |L_{n-1}| ? \\
\text{add } S[n] \text{ to suboptimal solution from } S[1...n-1] ? 
\end{cases} \]
Redefine: $L_n = \text{longest increasing subsequence that actually uses } S[n]$.

$3, 4, 5, 10, 13, 2$

$L_6 = 1$
Redefine: \( L_n = \text{longest increasing subsequence that actually uses } S[n] \)

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

\[ L_6 = 1 \]

\[ 3, 4, 5, 10, 13, 2 \]

\[ |L_n| = \ ?？ \]
Redefine: $L_n =$ longest increasing subsequence that actually uses $S[n]$

$$L_6 = 1$$

3, 4, 5, 10, 13, 2

$|L_n| =$

look at all $L_j$ ($j < n$)
Redefine: $L_n = \text{longest increasing subsequence that actually uses } S[n]$.

$L_6 = 1$

$3, 4, 5, 10, 13, 2$

$|L_n| = 1 + \ ?$

look at all $L_j \ (j < n)$.
Redefine: \( L_n = \) longest increasing subsequence that actually uses \( S[n] \)

\[ L_6 = 1 \]

\[ 3, 4, 5, 10, 13, 2 \]

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

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

Recursion:

BAD

\[ L_n \]

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

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

Recursion: \( L_n \)

**BAD**

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

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

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.
23, 3, 5, 18, 10, 101, 12, 14

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

Recursion: \( L_n \)

BAD

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

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

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

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

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

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|\\

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

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

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.
23, 3, 5, 18, 10, 101, 12, 14
1 1 2 3 3 4 4

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

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.
Recursion: \( L_n \)

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

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

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

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

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.

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

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

Recursion: $$L_n \leftarrow L_{n-1}, L_{n-2}, \ldots, L_1$$

BAD

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

\[
|L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j|
\]
\[ T(n) = \Theta(n^2) \]

space = \( \Theta(n) \)

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

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

space = \( \Theta(n) \)

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

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

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

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

\( = |L_n| ? \rightarrow \text{NO. } = \max_{j=1..n} |L_j| \)

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

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

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

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

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 $O(n^2)$ & dynamic programming.

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

$$\text{sort}$$

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

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 $O(n^2)$ & dy-co-prog.

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

sort:

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

FIND LONGEST COMMON SUBSEQUENCE!

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

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)
  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

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, so it is a candidate for LCS.