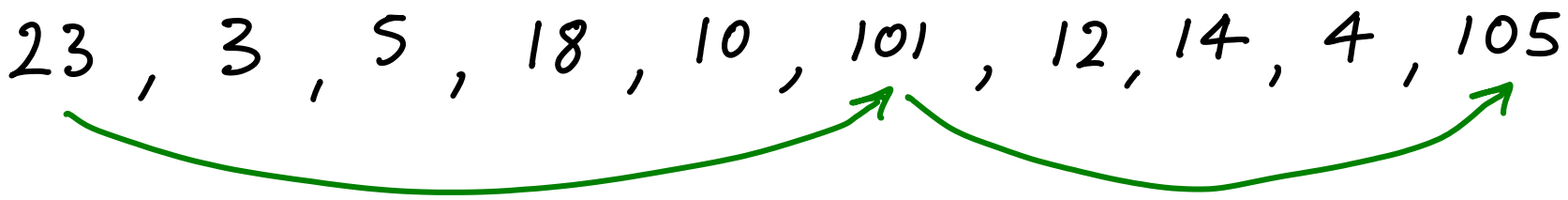
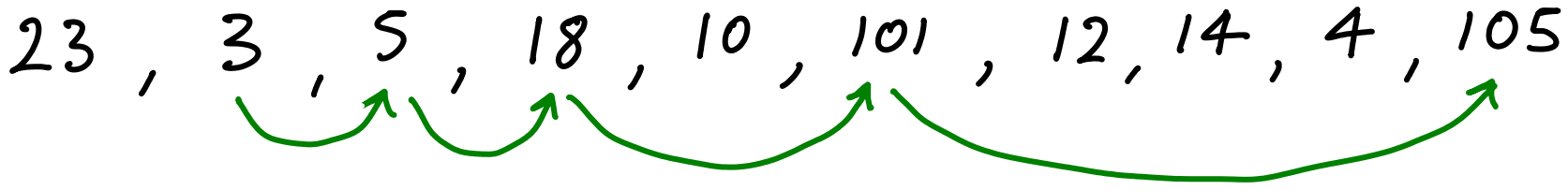
DYNAMIC PROGRAMMING – LONGEST INCREASING SUBSEQUENCE

23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4 , 105

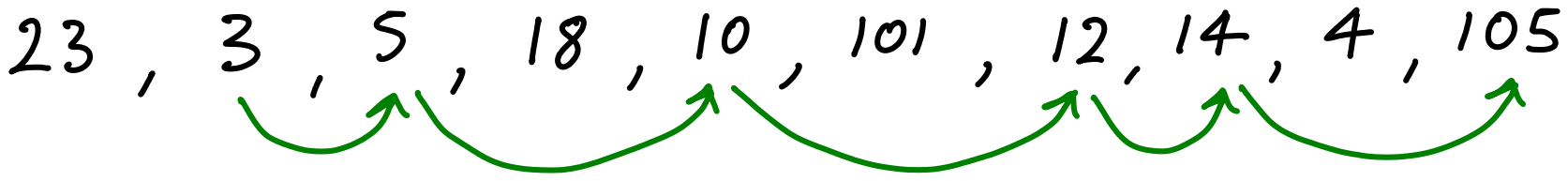
DYNAMIC PROGRAMMING - LONGEST INCREASING SUBSEQUENCE



DYNAMIC PROGRAMMING – LONGEST INCREASING SUBSEQUENCE



DYNAMIC PROGRAMMING – LONGEST INCREASING SUBSEQUENCE



DYNAMIC PROGRAMMING – LONGEST INCREASING SUBSEQUENCE

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

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

DYNAMIC PROGRAMMING – LONGEST INCREASING SUBSEQUENCE

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

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

Could try including/excluding every element:

2^n subsequences to check

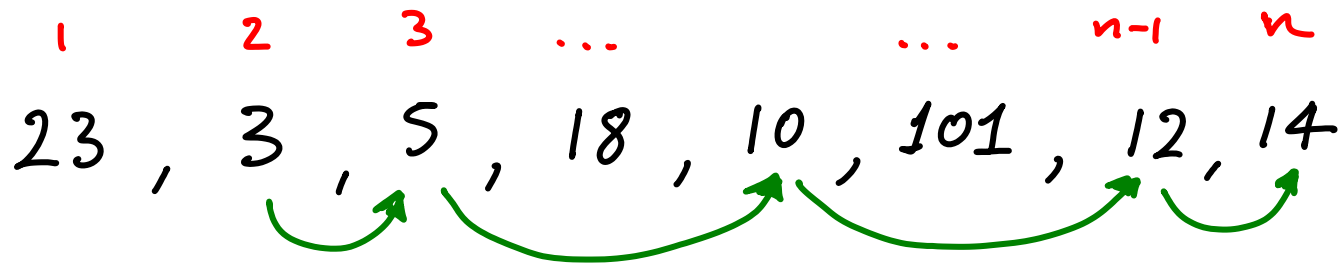
DYNAMIC PROGRAMMING – LONGEST INCREASING SUBSEQUENCE

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

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

For dynamic programming we would like

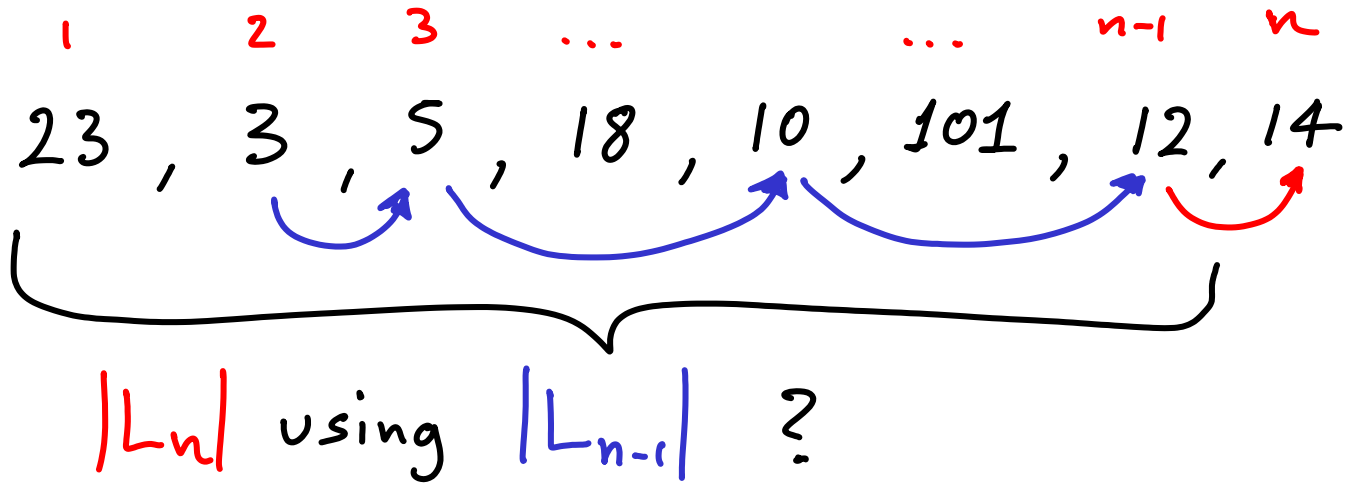
- a recursive expression w/ repeated subproblems
- an easy, fast way to use solved subproblems



$L(S) \sim$

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

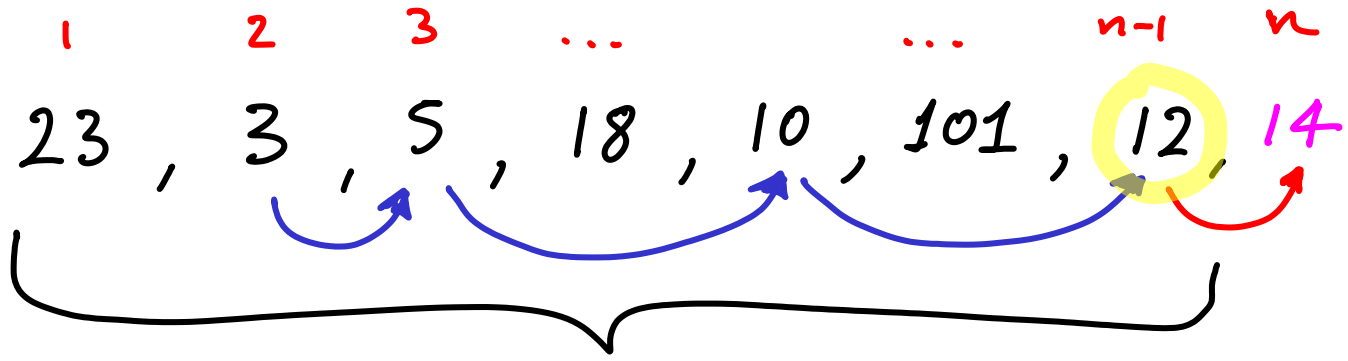
$\sim L_n$



$L(S) \sim$

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

$\sim L_n$



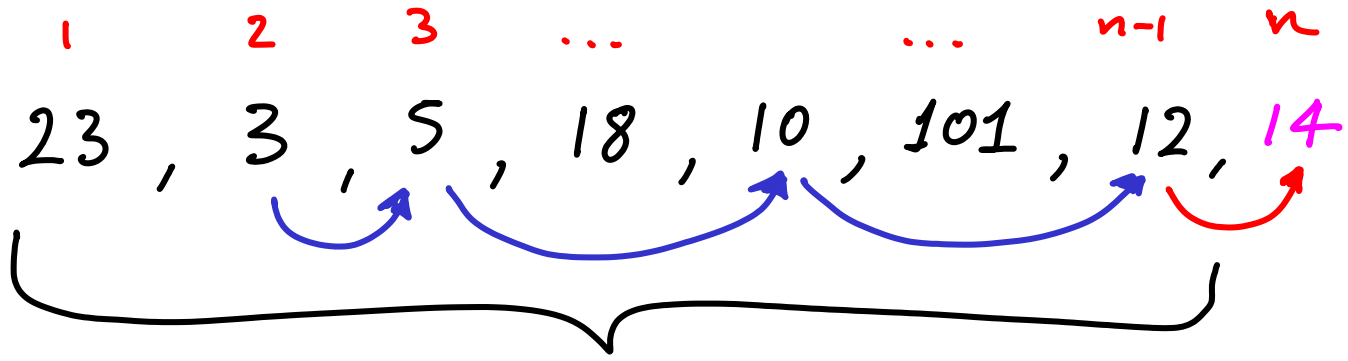
$L(S) \sim$

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

$\sim L_n$

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

if $S[n]$ > last element in L_{n-1} then ??



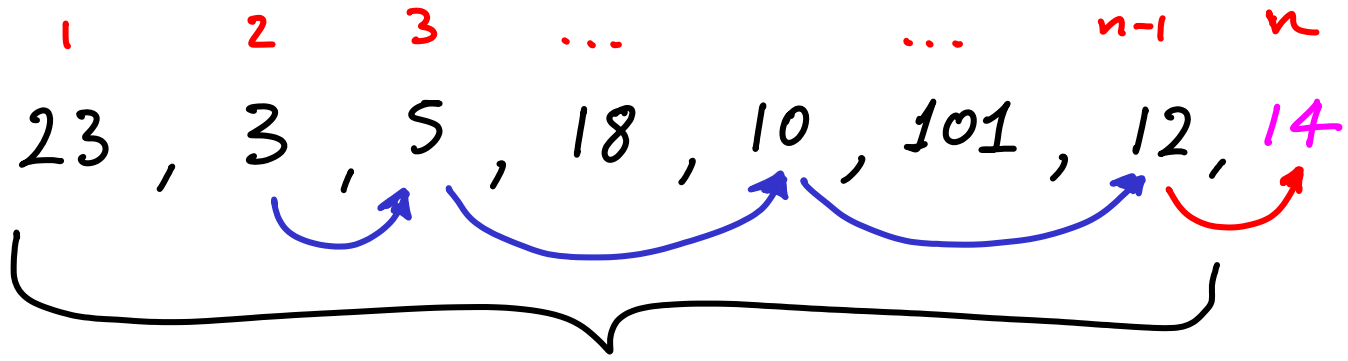
$L(S) \sim$

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

$\sim L_n$

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

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



$L(S) \sim$

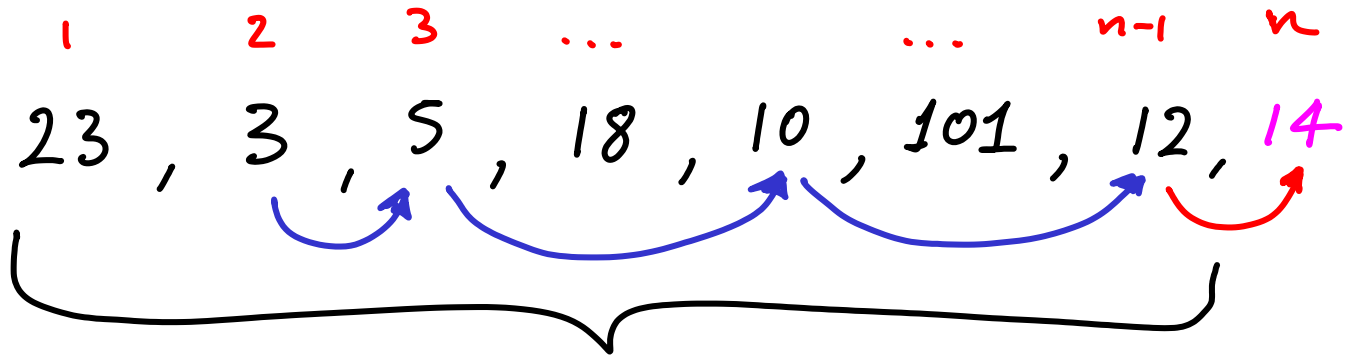
$\sim L_{1..n}(S)$

$\sim L_n$

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

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

↳ could be at any position in S



$L(S) \sim$

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

$\sim L_n$

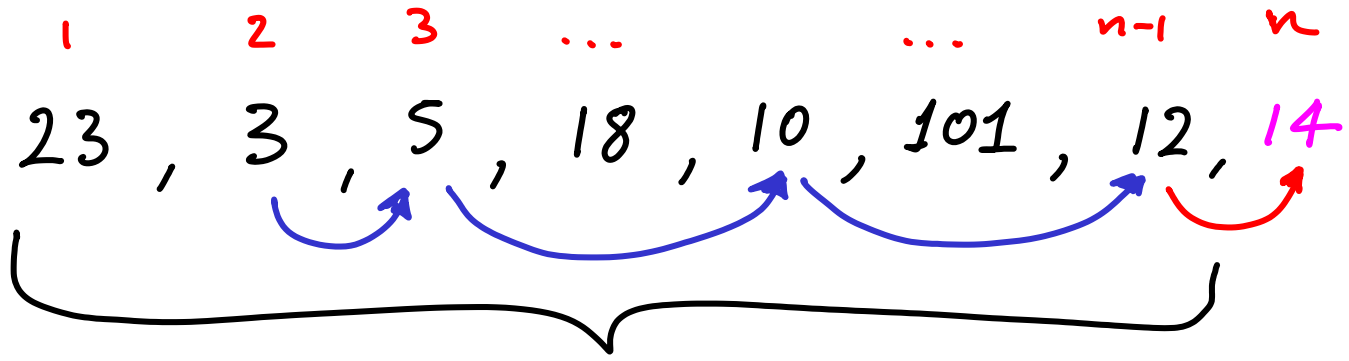
$|L_n|$ using $|L_{n-1}|$?

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

↳ could be at any position in S

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

??



$L(S) \sim$

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

$\sim L_n$

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

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

↳ could be at any position in S

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

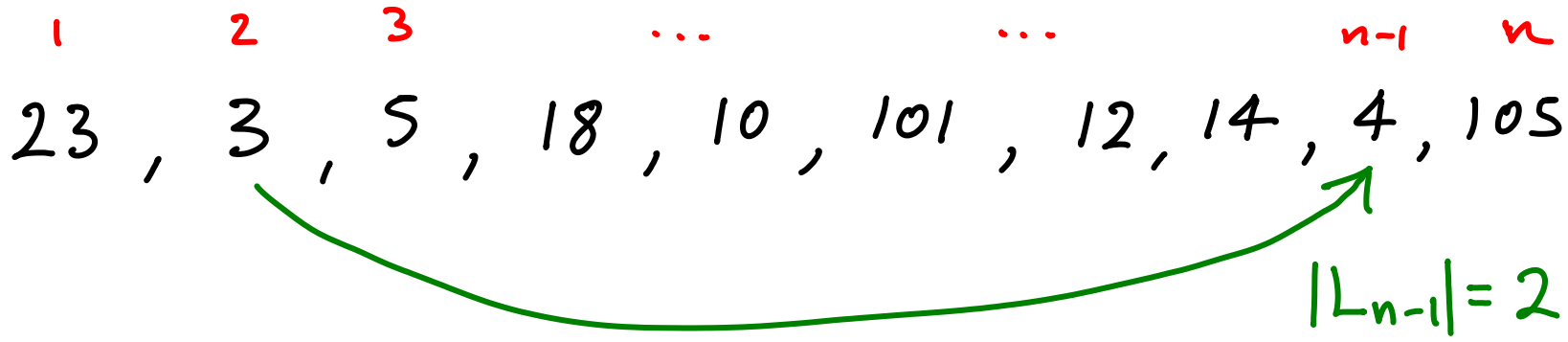
↳ keep $|L_{n-1}|$?

↳ add $S[n]$ to suboptimal solution from $S[1...n-1]$?

1 2 3 ... ... n-1 n

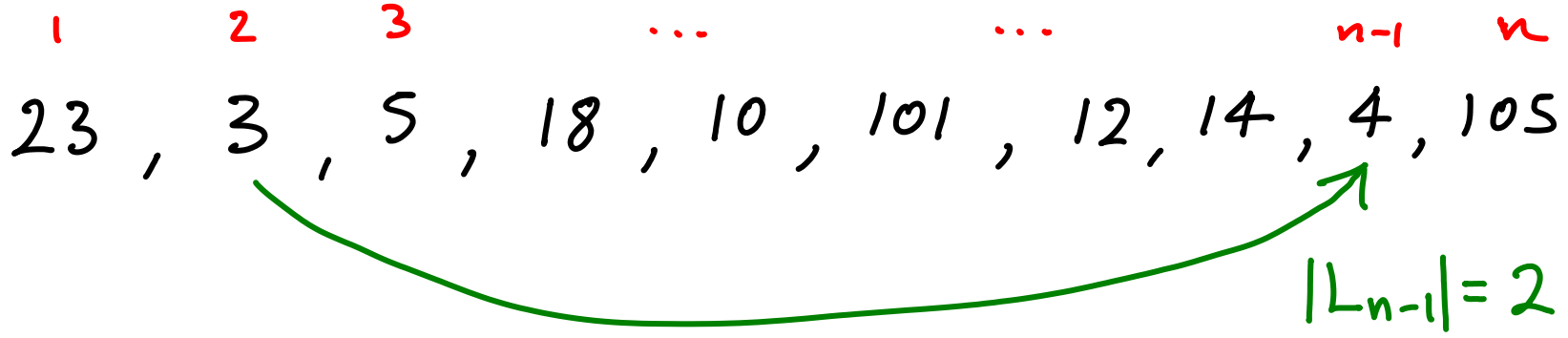
23, 3, 5, 18, 10, 101, 12, 14, 4, 105

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



Redefine: L_n = longest increasing subsequence that <u>actually uses</u> $S[n]$

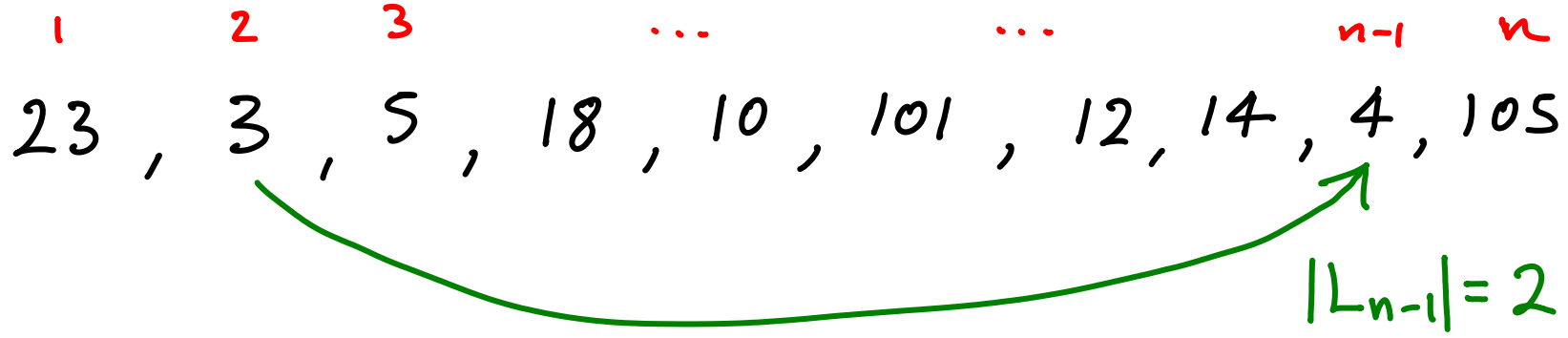
$$|L_n| = \; ??$$



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

$|L_n| =$

look at all L_j $(j<n)$



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

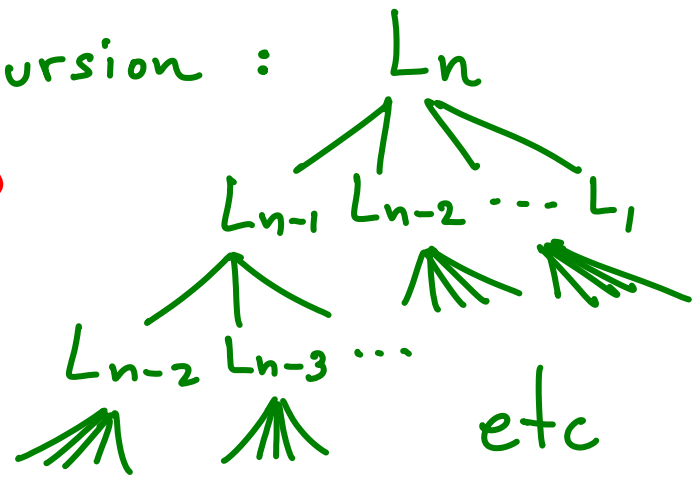
$$|L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j|$$

look at all L_j $(j<n)$

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

Recursion :

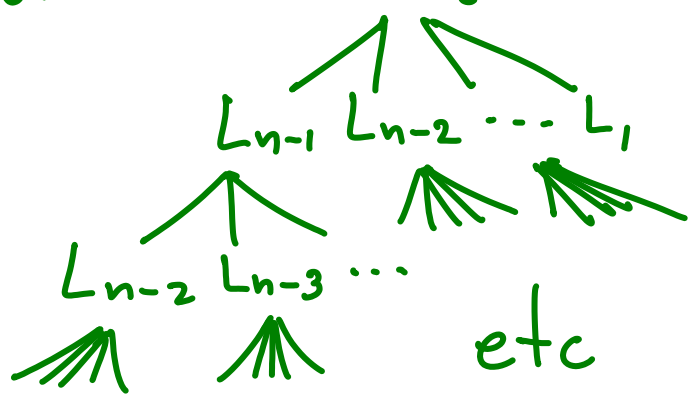
BAD



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

Recursion : L_n

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.

23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4

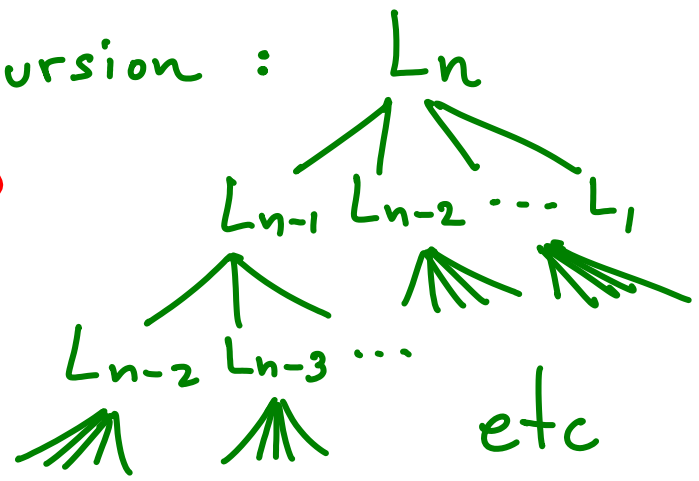
1

$|L_1|$

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

Recursion : L_n

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.

23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4

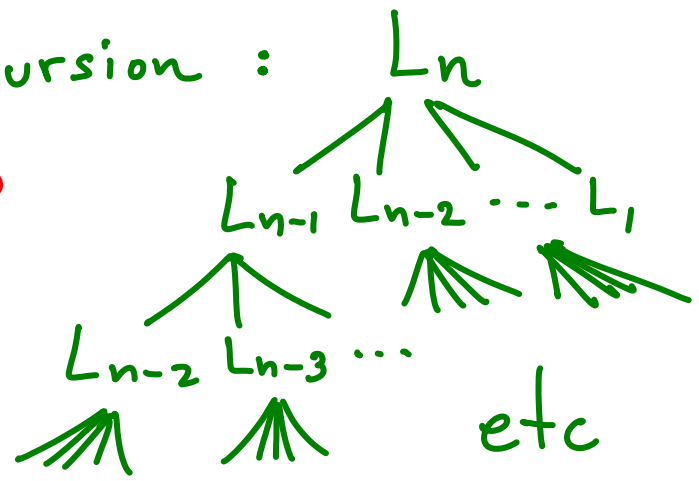
1 1

$|L_2|$

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

Recursion : L_n

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.

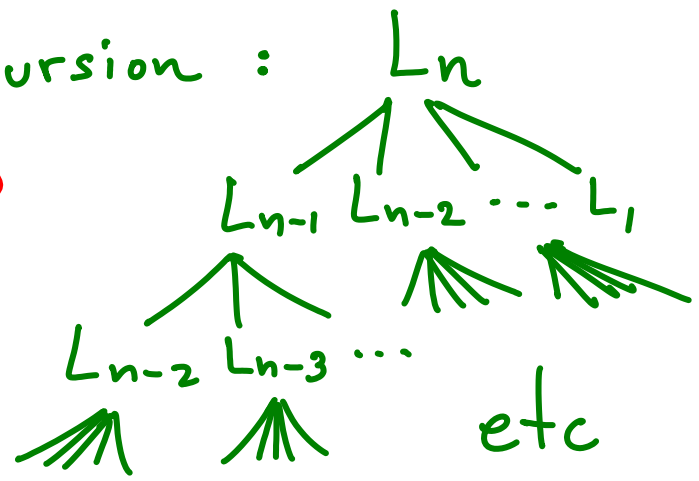
23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4

1 1 2

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

Recursion : L_n

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.

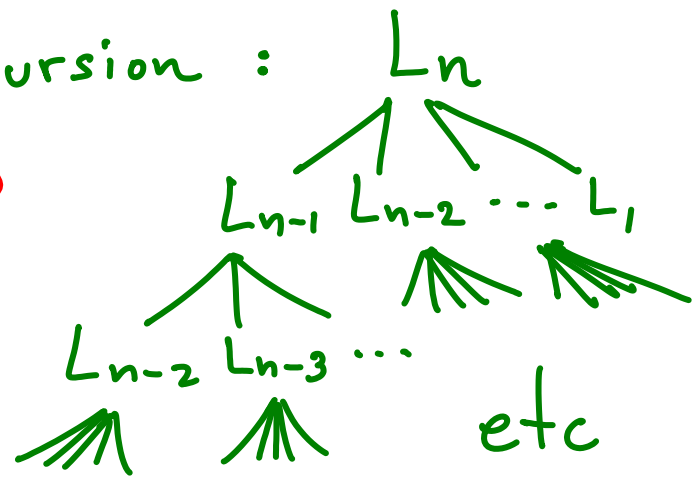
23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4

1 1 2 3

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

Recursion : L_n

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.

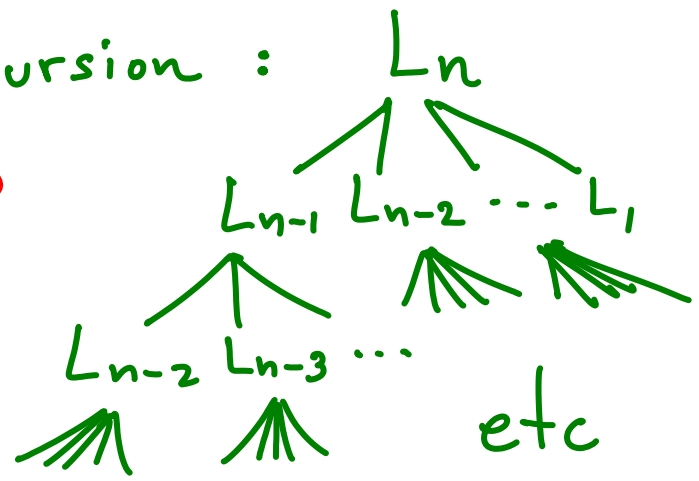
23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4

1 1 2 3 3

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

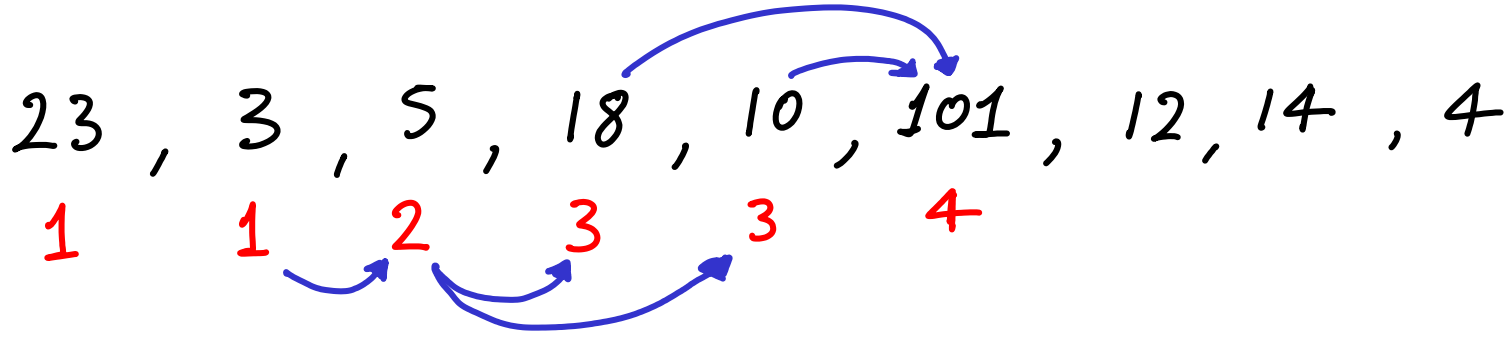
Recursion : L_n

BAD



Dyn. Prog: Build solutions, "bottom up"

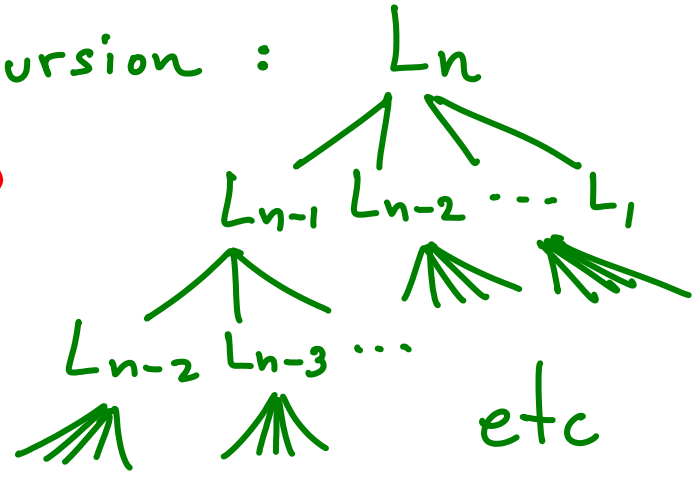
When it's time to solve $|L_k|$ we have stored all $|L_j|$ $(j<k)$ in an array.



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

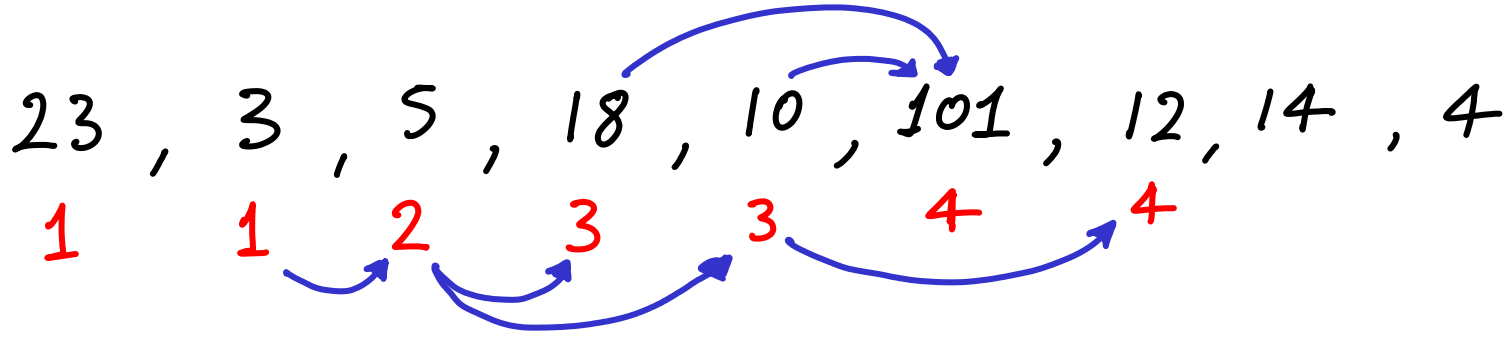
Recursion : L_n

BAD



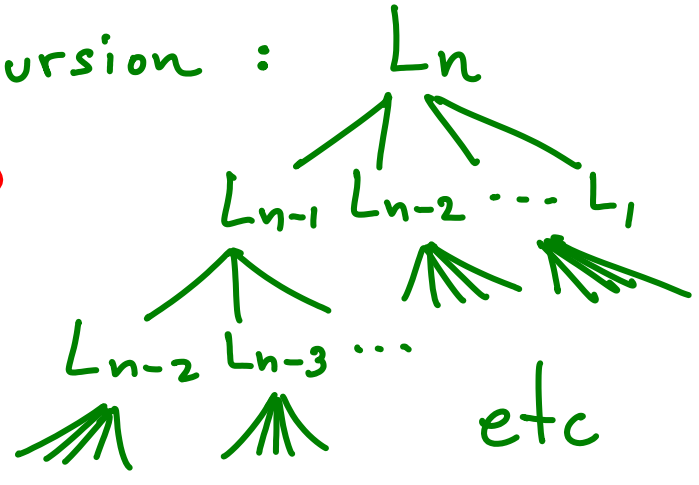
Dyn. Prog: Build solutions, "bottom up"

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



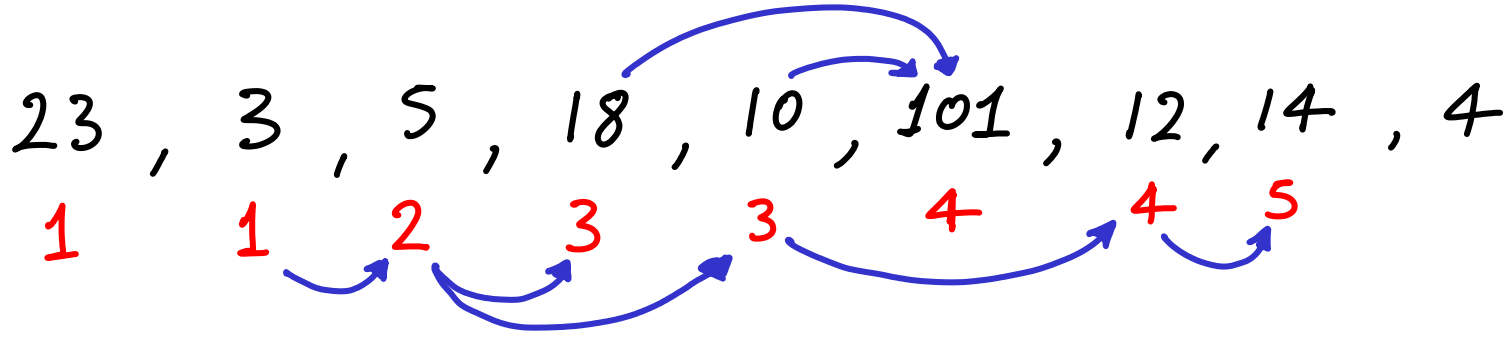
$$|L_n| = 1 + \max_{\{\text{all } j \text{ s.t. } S[j] < S[n]\}} |L_j|$$

Recursion : BAD



Dyn. Prog: Build solutions, "bottom up"

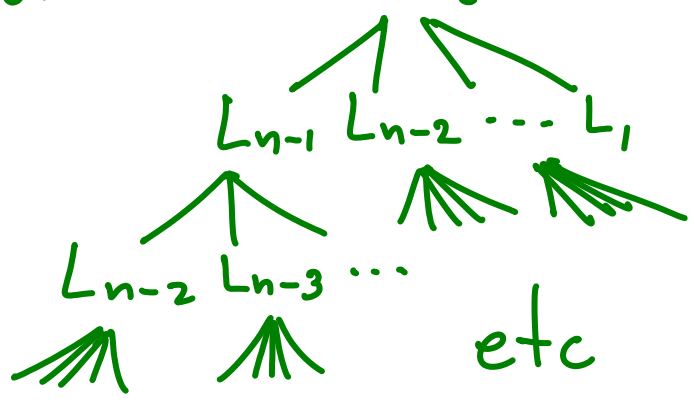
When it's time to solve $|L_k|$ we have stored all $|L_j|$ $(j<k)$ in an array.



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

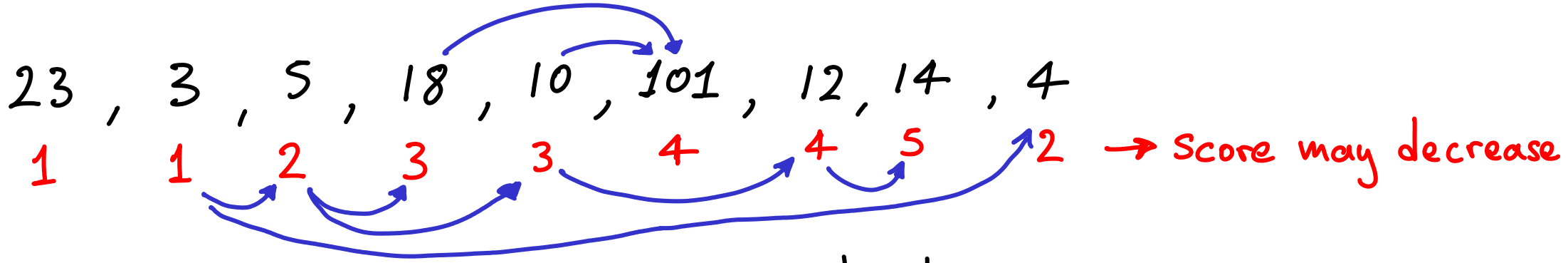
Recursion : L_n

BAD



Dyn. Prog: Build solutions, "bottom up"

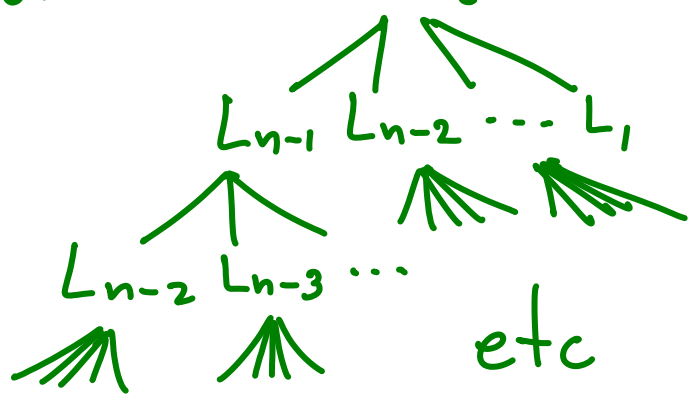
When it's time to solve $|L_k|$ we have stored all $|L_j|$ $(j<k)$ in an array.



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

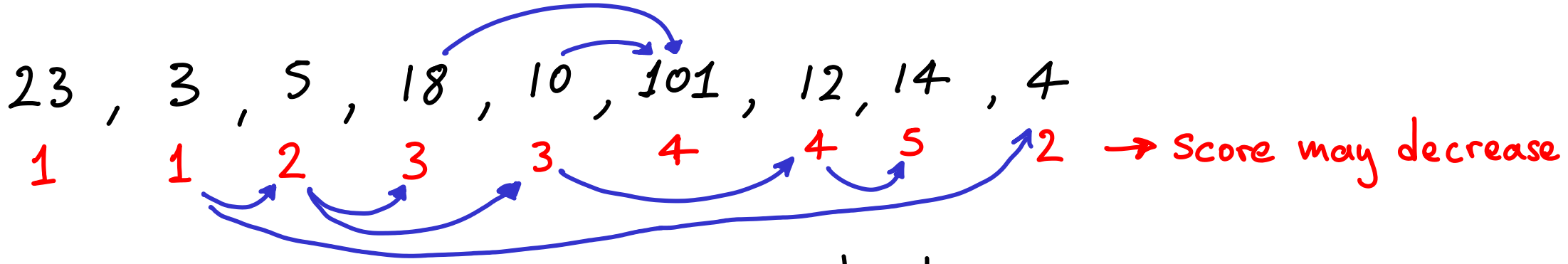
Recursion : L_n

BAD



Dyn. Prog: Build solutions, "bottom up"

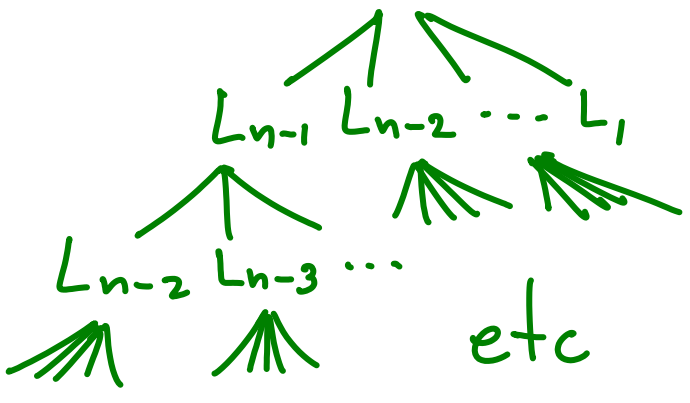
When it's time to solve $|L_k|$ we have stored all $|L_j|$ $(j<k)$ in an array.



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

Recursion : L_n

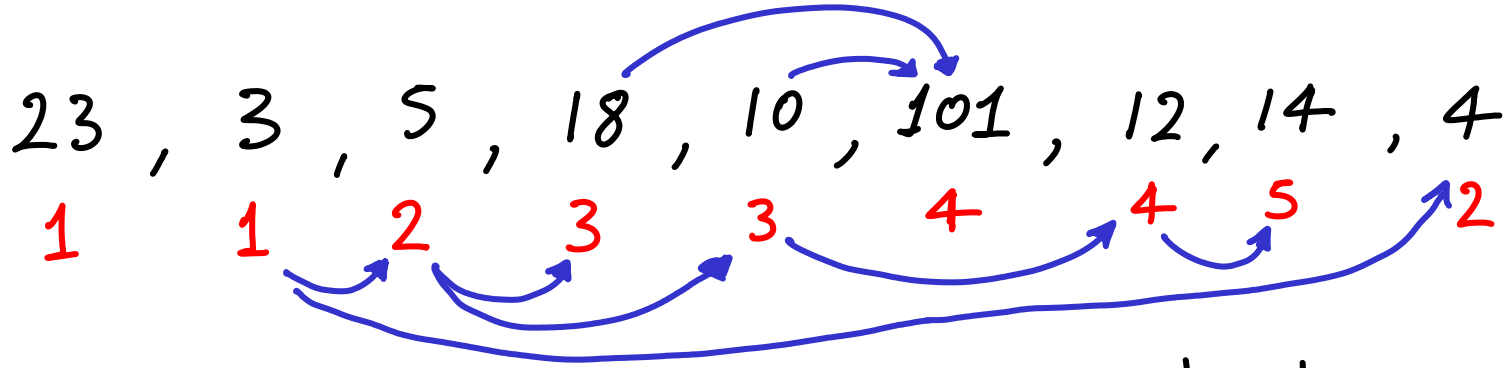
BAD



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.

time? space?

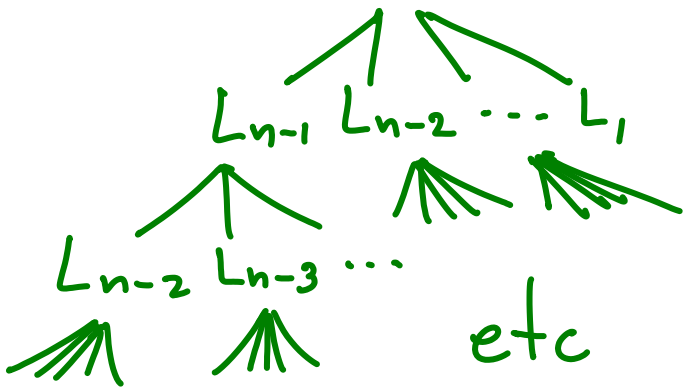


→ Score may decrease

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

Recursion : L_n

BAD



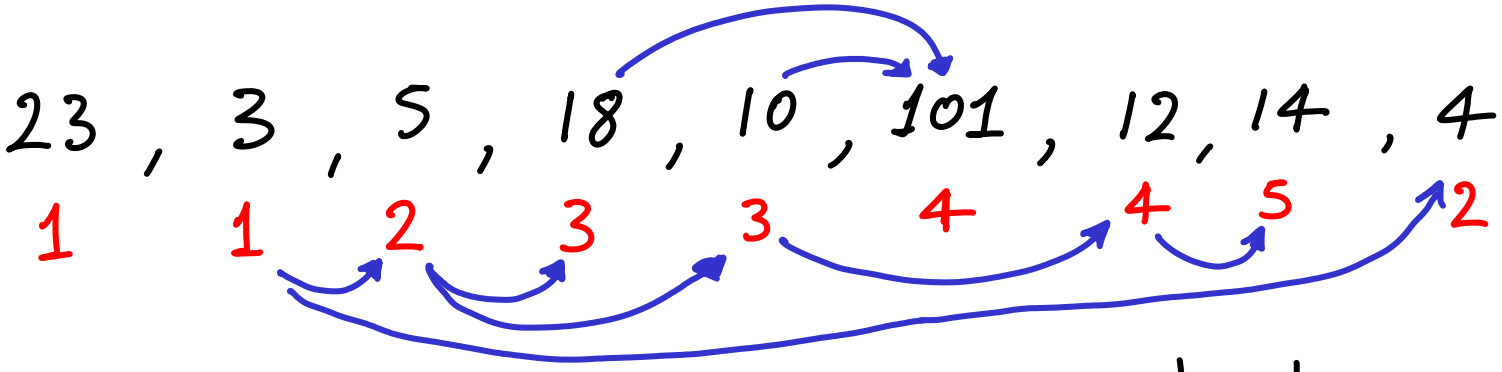
Dyn. Prog: Build solutions, "bottom up"

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

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

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

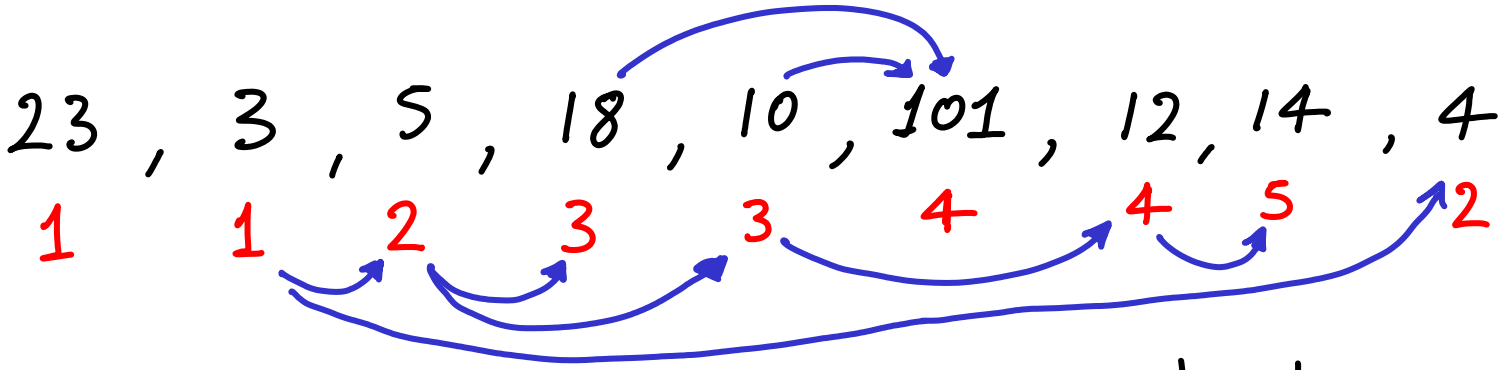
Space = $\Theta(n)$



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

space $= \Theta(n)$

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

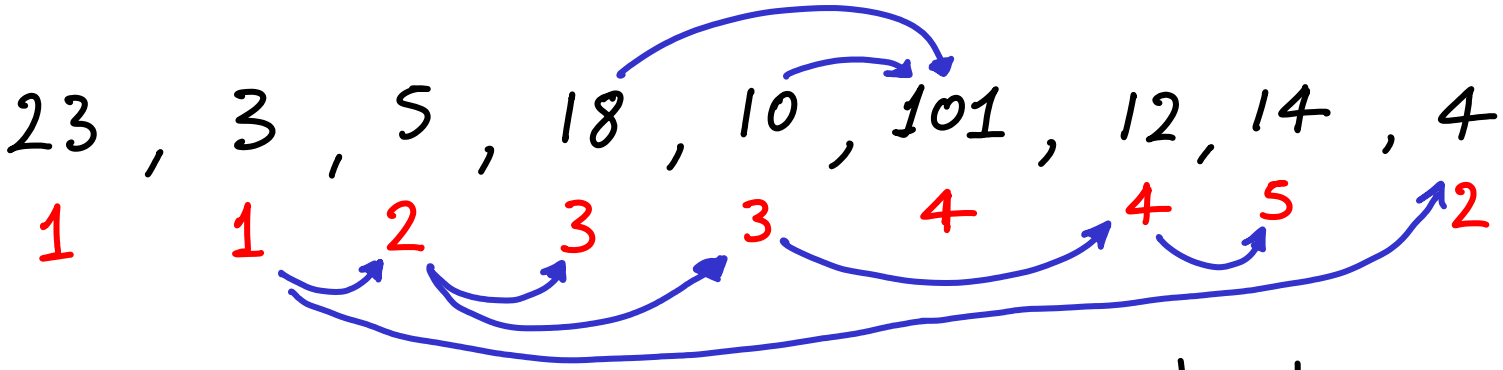


$T(n) = \Theta(n^2)$

space $= \Theta(n)$

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

What about $|L.I.S.|$?

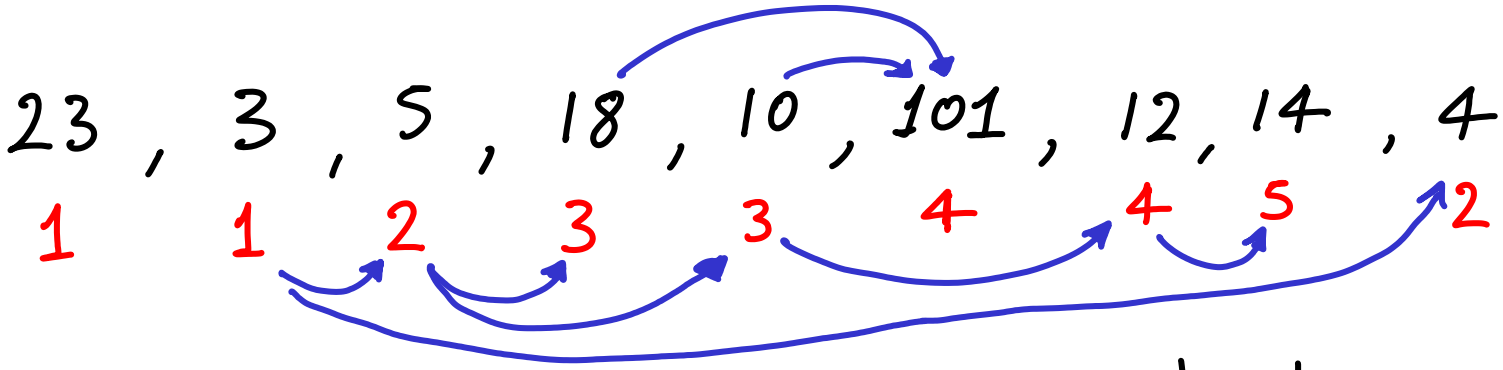


$T(n) = \Theta(n^2)$

space $= \Theta(n)$

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

What about $|L.I.S.|$? $= \max_{j=1..n} |L_j|$



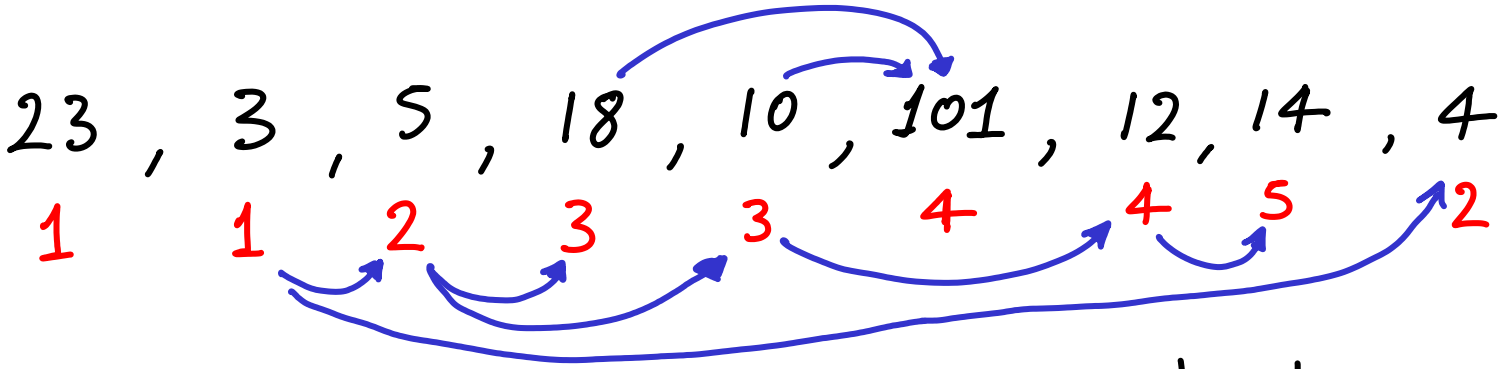
$T(n) = \Theta(n^2)$

space $= \Theta(n)$

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

What about $|L.I.S.|$? $= \max_{j=1..n} |L_j|$

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 $|L.I.S.|$? $= \max_{j=1..n} |L_j|$

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.

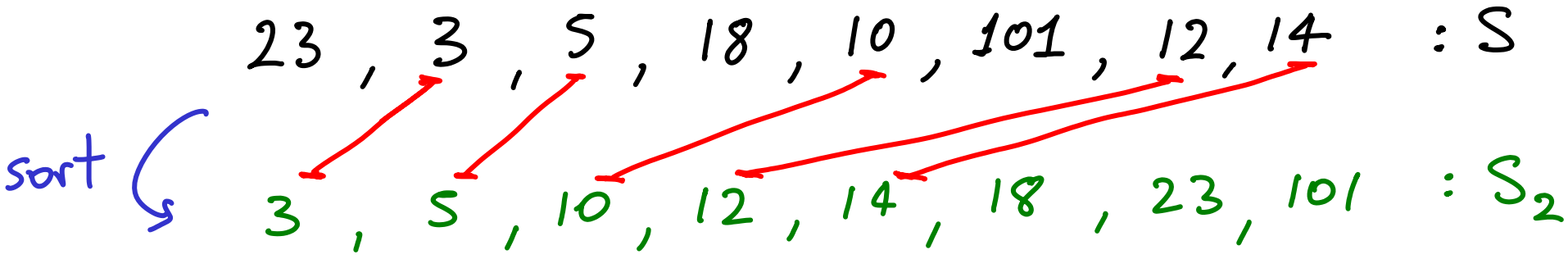
23, 3, 5, 18, 10, 101, 12, 14 : S

sort ↘

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

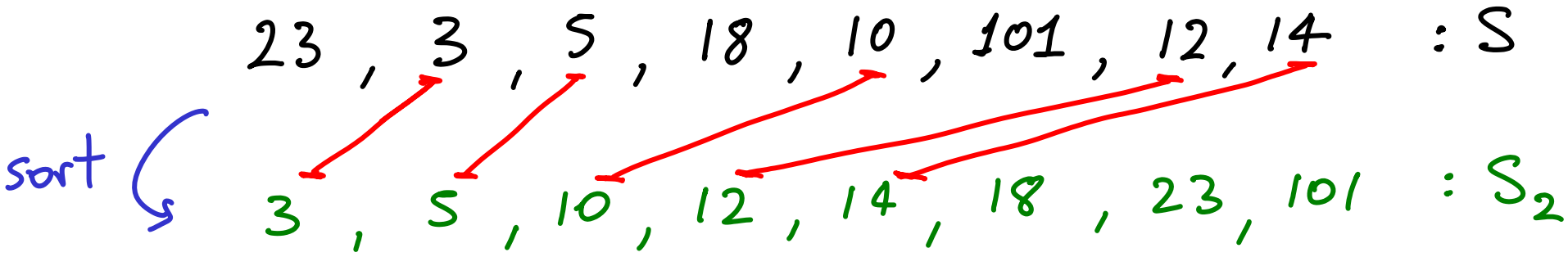
and then ?

A QUICK SOLUTION FOR L.I.S. ... but still $O(n^2)$ & dyn-prog.



FIND LONGEST COMMON SUBSEQUENCE !

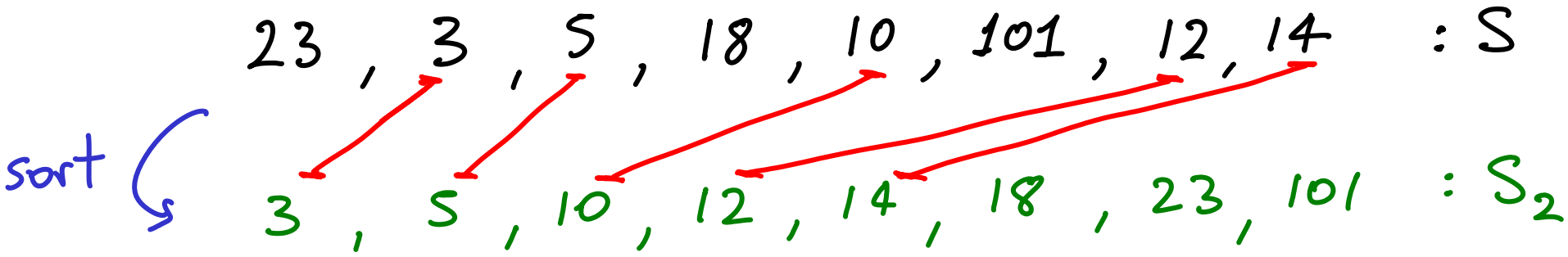
A QUICK SOLUTION FOR L.I.S. ... but still $O(n^2)$ & dyn-prog.



FIND LONGEST COMMON SUBSEQUENCE !

- any common subsequence is increasing

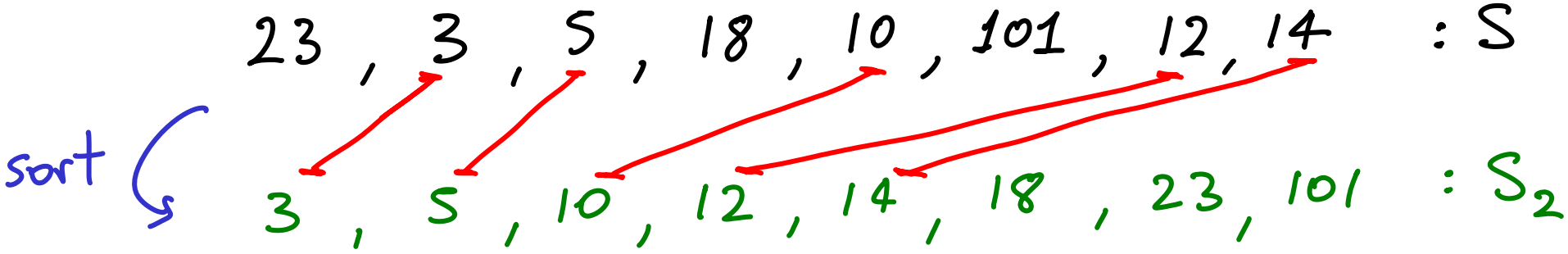
A QUICK SOLUTION FOR L.I.S. ... but still $O(n^2)$ & dyn-prog.



FIND LONGEST COMMON SUBSEQUENCE !

- any common subsequence is increasing
  so $LCS(S, S_2)$ qualifies as a solution

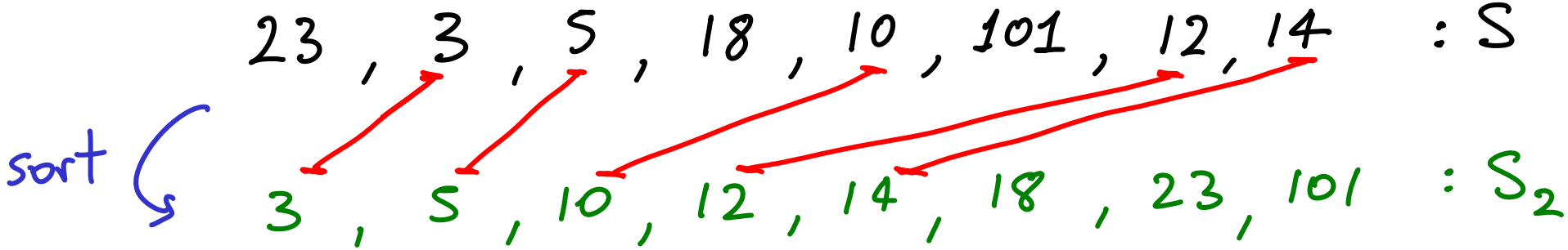
A QUICK SOLUTION FOR L.I.S. ...but still $O(n^2)$ & dyn-prog.



FIND LONGEST COMMON SUBSEQUENCE !

- any common subsequence is increasing
  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.



FIND LONGEST COMMON SUBSEQUENCE !

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

□