


# 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



The diagram illustrates a sequence of numbers: 23, 3, 5, 18, 10, 101, 12, 14, 4, 105. Two green arrows are drawn below the sequence. The first arrow starts at the number 23 and points to the number 101. The second arrow starts at the number 101 and points to the number 105. This indicates that the subsequence [23, 101, 105] is being highlighted as an increasing subsequence.

# DYNAMIC PROGRAMMING - LONGEST INCREASING SUBSEQUENCE


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



The diagram illustrates the sequence 23, 3, 5, 18, 10, 101, 12, 14, 4, 105. Green arrows are drawn below the numbers, starting from 3 and pointing to 5, then from 5 to 18, then from 18 to 101, and finally from 101 to 105. This sequence of arrows highlights the longest increasing subsequence: 3, 5, 18, 101, 105.

# DYNAMIC PROGRAMMING - LONGEST INCREASING SUBSEQUENCE


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



The diagram illustrates the sequence 23, 3, 5, 18, 10, 101, 12, 14, 4, 105. Green arrows connect the elements 3, 5, 10, 12, 14, and 105, showing an increasing subsequence. The arrows start at 3 and point to 5, then from 5 to 10, from 10 to 12, from 12 to 14, and finally from 14 to 105. The element 23 is not part of this subsequence, and the element 4 is also not part of it as it is less than 3.

# 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



$L(s) = 3, 5, 10, 12, 14, 105$        $|L(s)| = 6$

Could try including/excluding every element:

$2^n$  subsequences to check

# DYNAMIC PROGRAMMING - LONGEST INCREASING SUBSEQUENCE

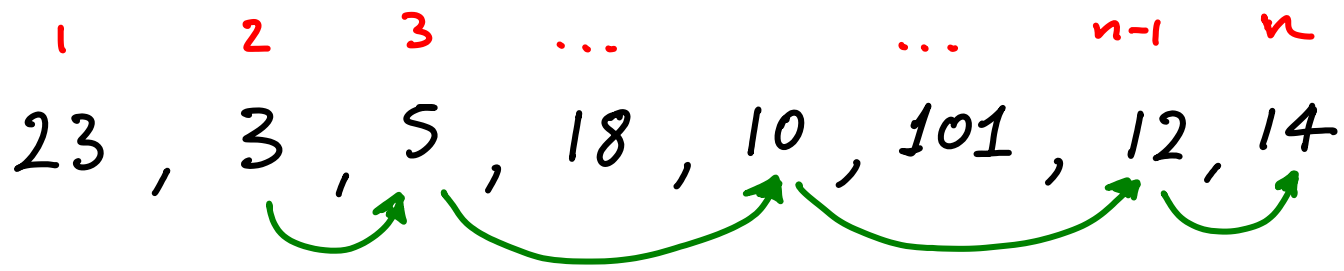
S: 23, 3, 5, 18, 10, 101, 12, 14, 4, 105

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

For dynamic programming we would like

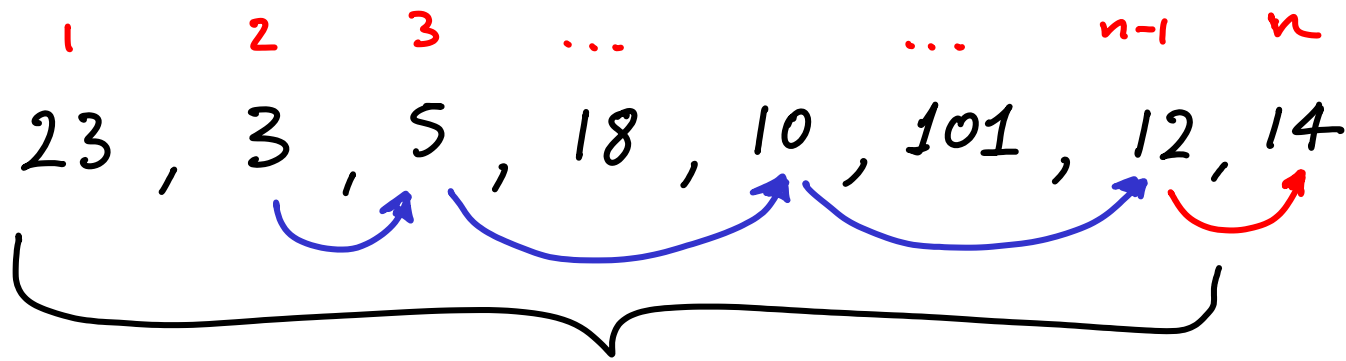
- a recursive expression w/ repeated subproblems
- an easy, fast way to use solved subproblems

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



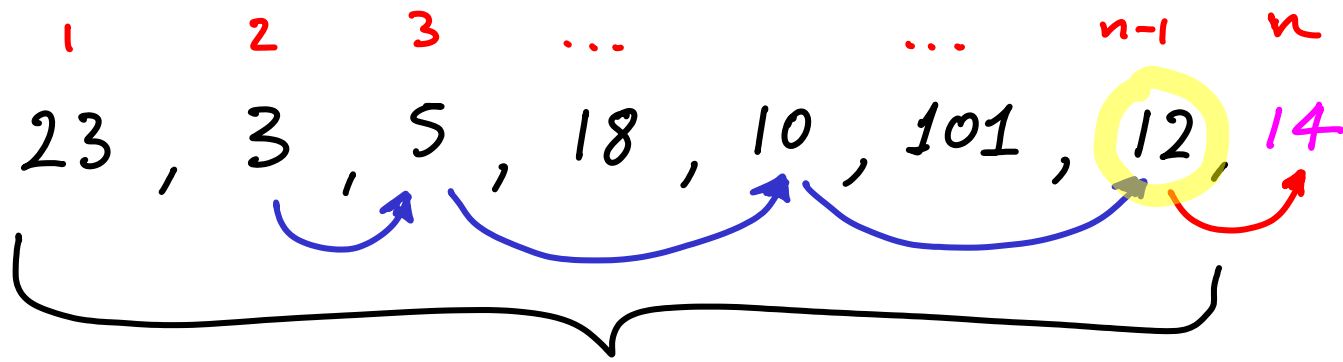
$L(s) \sim$   
 $\sim L_{1..n}(s)$   
 $\sim L_n$





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

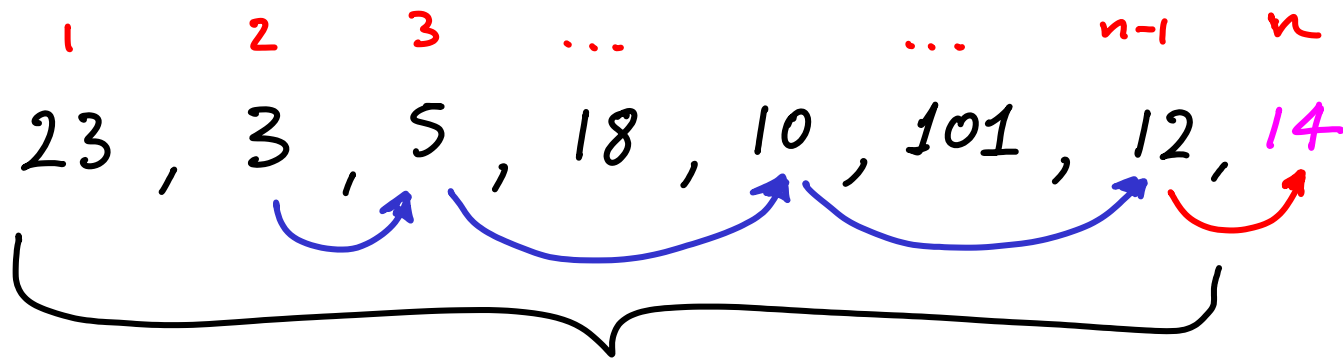
$$\begin{aligned} L(S) &\sim \\ &\sim L_{1..n}(S) \\ &\sim L_n \end{aligned}$$



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

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

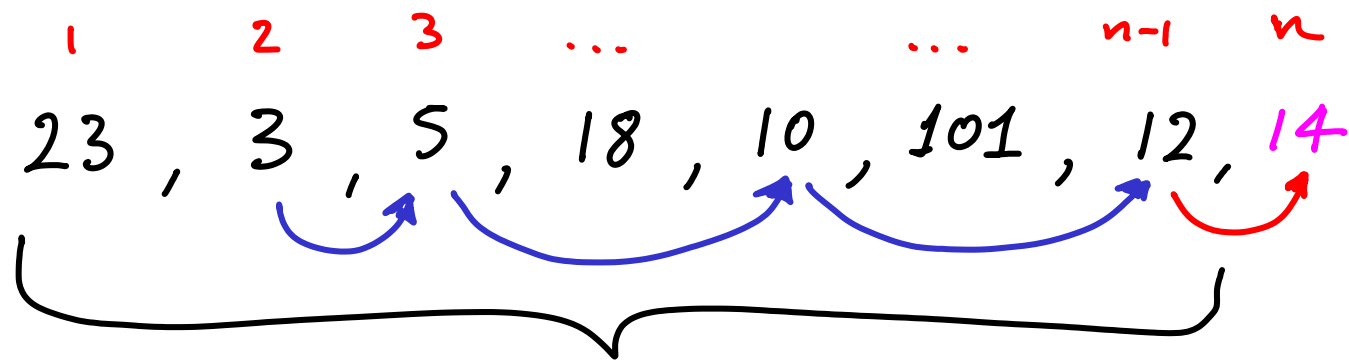
if  $S[n] > \underline{\text{last element in } L_{n-1}}$  then ??



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

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

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



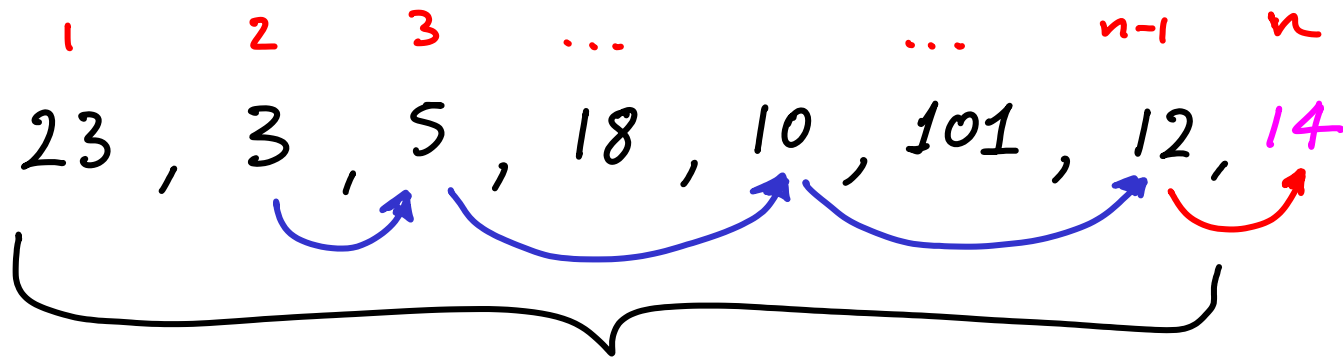
$$L(S) \sim$$

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

$$\sim L_n$$

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

if  $S[n] > \underline{\text{last element in } L_{n-1}}$  then  $|L_n| = |L_{n-1}| + 1$   
 ↳ could be at any position in  $S$



$$L(S) \sim$$

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

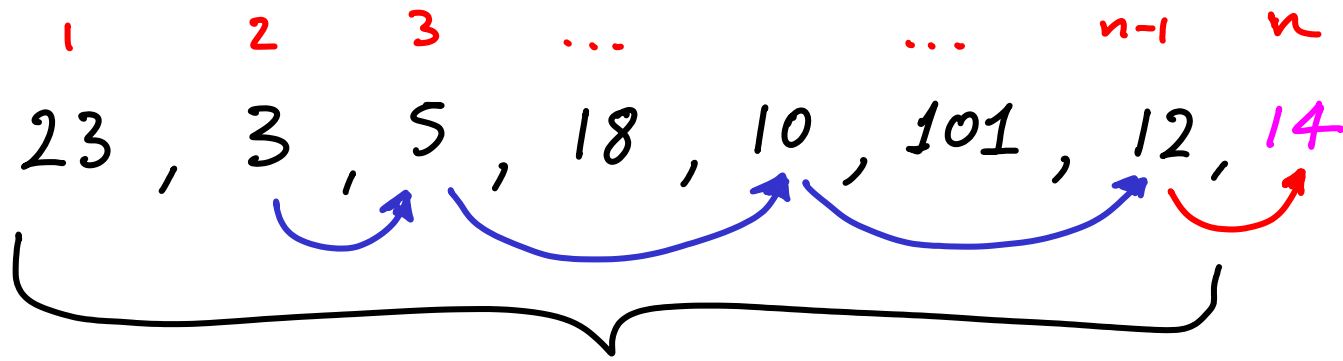
$$\sim L_n$$

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

if  $S[n] > \underline{\text{last element in } L_{n-1}}$  then  $|L_n| = |L_{n-1}| + 1$   
 $\hookrightarrow$  could be at any position in  $S$

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

??



$$L(S) \sim$$

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

$$\sim L_n$$

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

if  $S[n] > \underline{\text{last element in } L_{n-1}}$  then  $|L_n| = |L_{n-1}| + 1$   
 $\hookrightarrow$  could be at any position in  $S$

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


$\hookrightarrow$  keep  $|L_{n-1}|$  ?

$\hookrightarrow$  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  $S[n]$

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  $S[n]$

$$|L_n| = ??$$




1      2      3                    ...                    ...                    n-1      n  
23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4 , 105  
  
 $|L_{n-1}| = 2$

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

$$|L_n| =$$

look at all  $L_j$  ( $j < n$ )

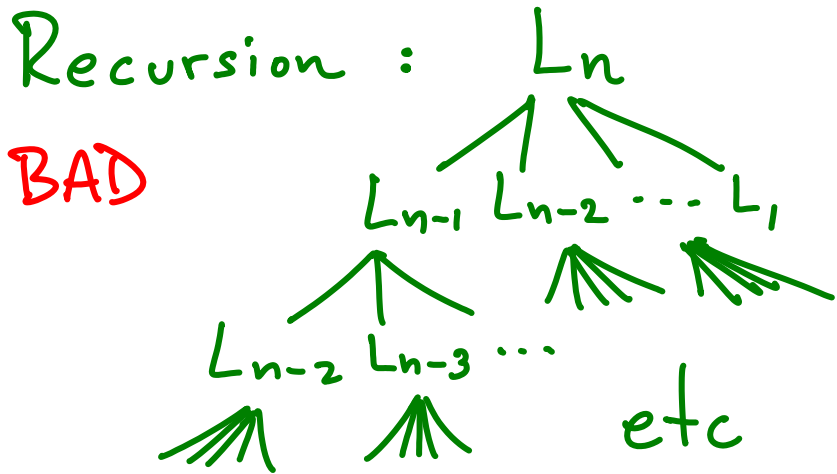
$1$      $2$      $3$         $\dots$         $\dots$         $n-1$      $n$   
 23 , 3 , 5 , 18 , 10 , 101 , 12 , 14 , 4 , 105  


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

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

look at all  $L_j$  ( $j < n$ )

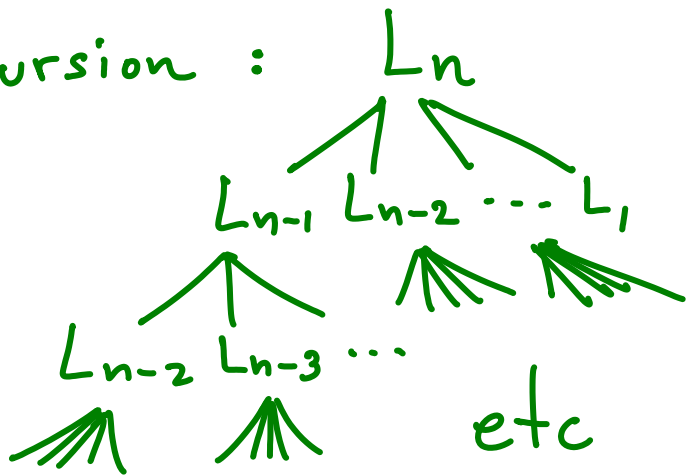
$$|L_n| = 1 + \max_{\{j \text{ s.t. } S[j] < S[n]\}} |L_j|$$



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

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.

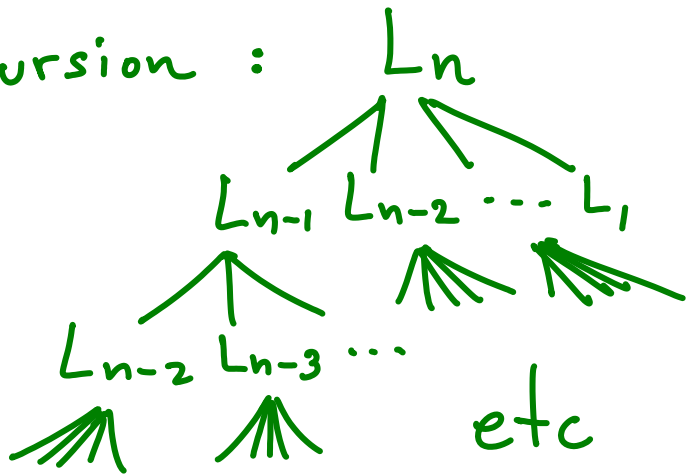
23, 3, 5, 18, 10, 101, 12, 14, 4

<sup>1</sup>  
|L<sub>1</sub>|

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

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.

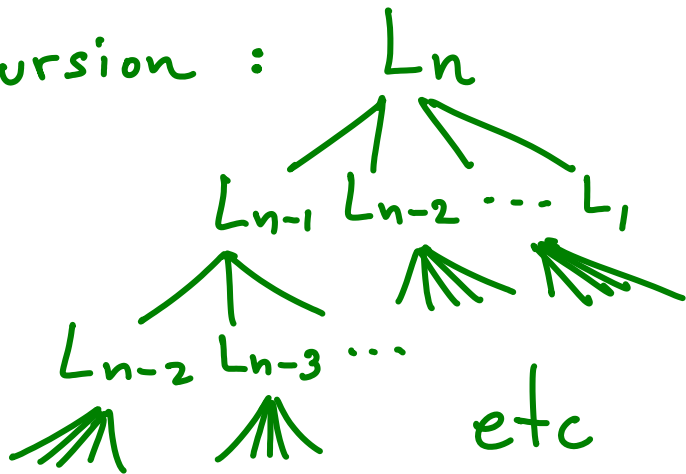
23, 3, 5, 18, 10, 101, 12, 14, 4

1      1  
         |L<sub>2</sub>|

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

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.

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

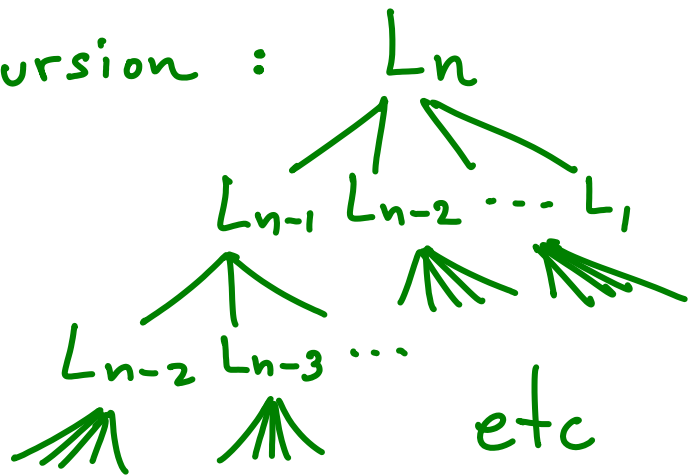
1 1 2



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

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.

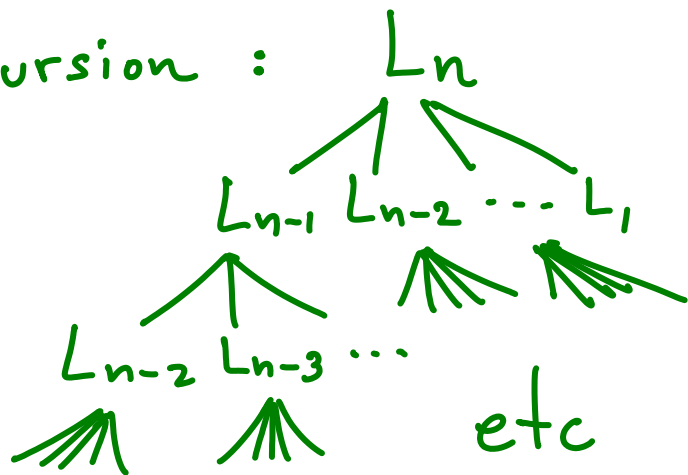
23, 3, 5, 18, 10, 101, 12, 14, 4

1    1    2    3

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

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.



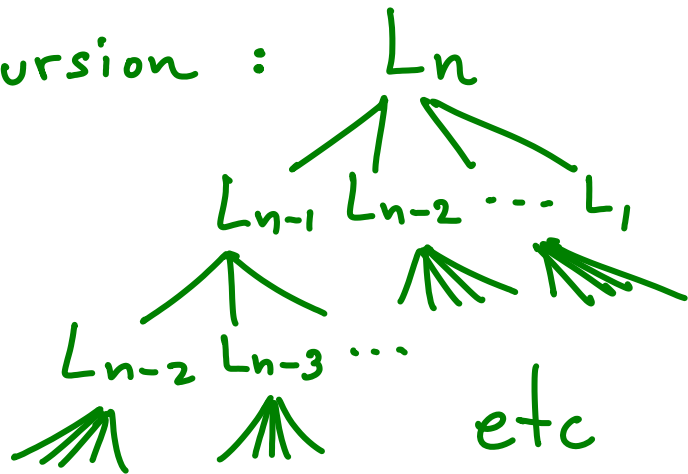
23, 3, 5, 18, 10, 101, 12, 14, 4

1 1 2 3 3

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

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.

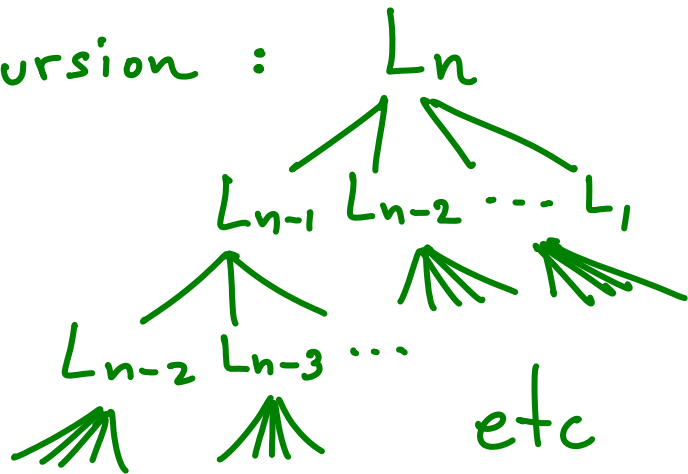
23, 3, 5, 18, 10, 101, 12, 14, 4

1, 1, 2, 3, 3, 4

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

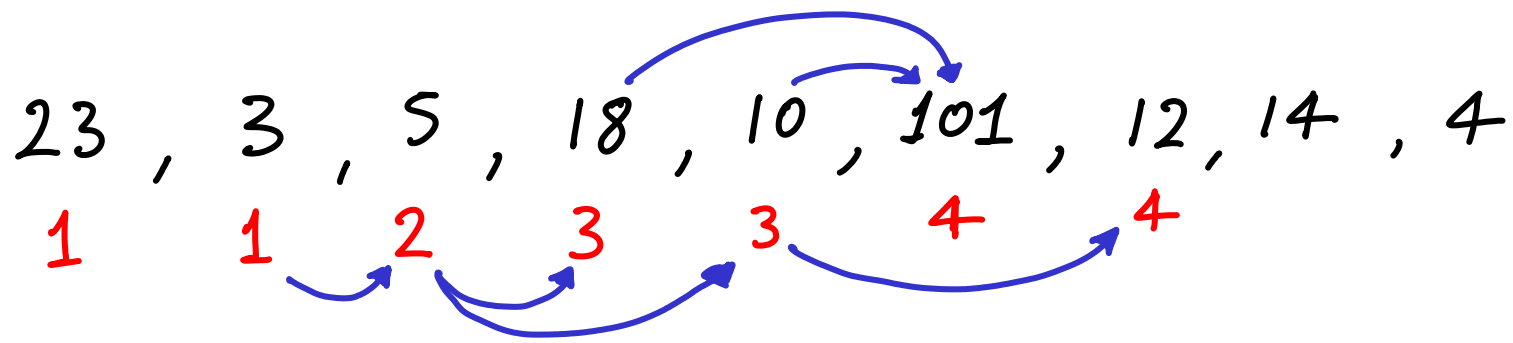
Recursion :

BAD



Dyn. Prog:

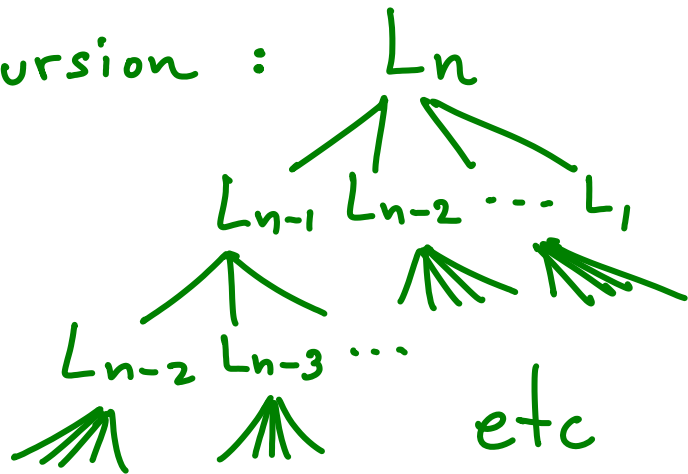
Build solutions, "bottom up"  
 When it's time to solve  $|L_k|$  we have stored all  $|L_j|$  ( $j < k$ ) in an array.



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

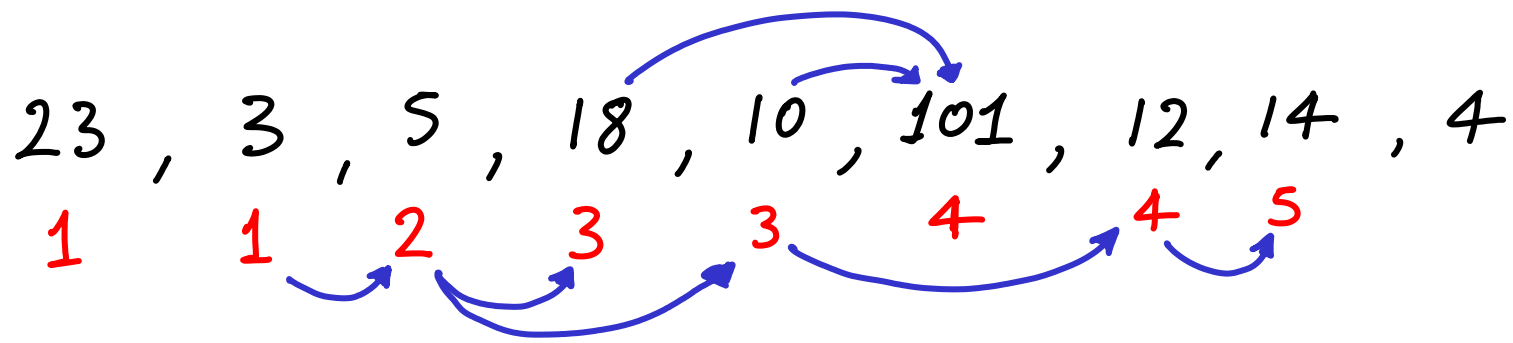
Recursion :

BAD



Dyn. Prog:

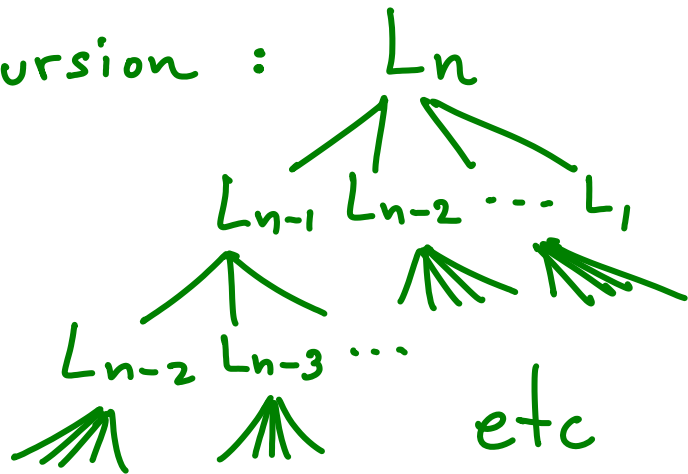
Build solutions, "bottom up"  
 When it's time to solve  $|L_k|$  we have stored all  $|L_j|$  ( $j < k$ ) in an array.



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

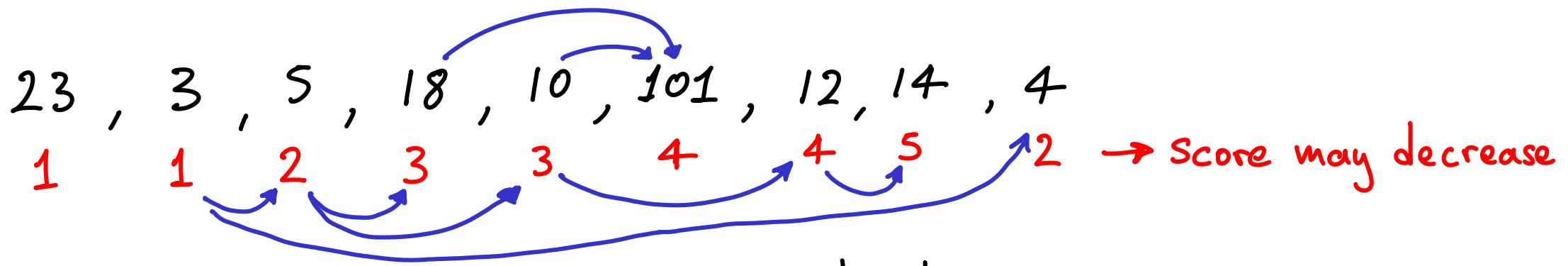
Recursion :

BAD



Dyn. Prog:

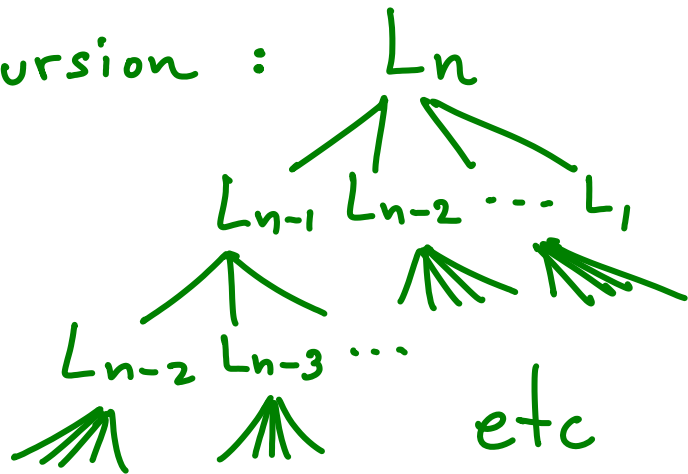
Build solutions, "bottom up"  
 When it's time to solve  $|L_k|$  we have stored all  $|L_j|$  ( $j < k$ ) in an array.



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

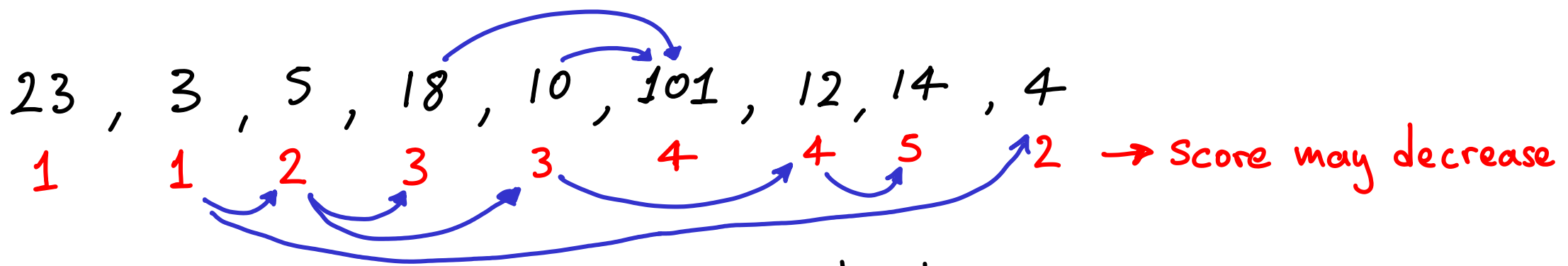
Recursion :

BAD

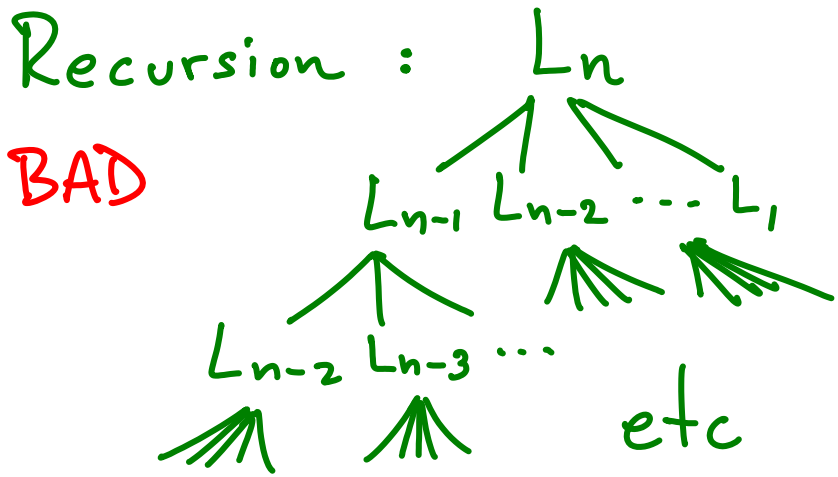


Dyn. Prog: Build solutions, "bottom up"

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



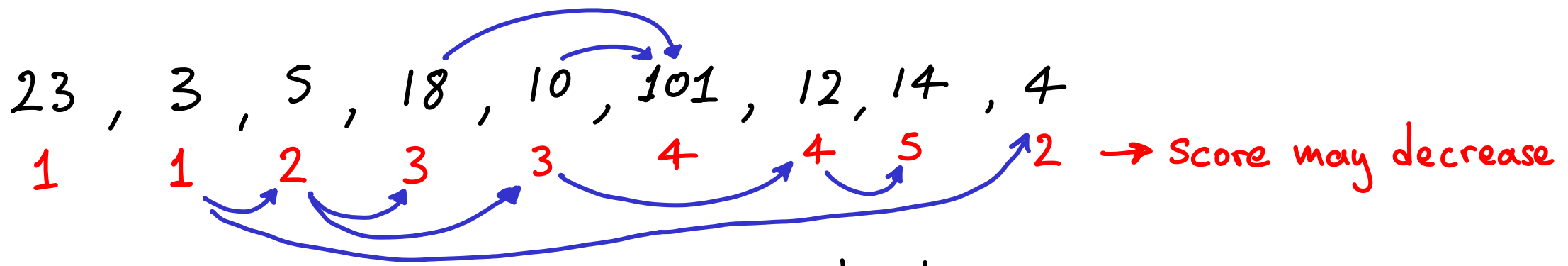
$$|L_n| = 1 + \text{MAX}_{\{ \text{all } j \text{ s.t. } S[j] < S[n] \}} |L_j|$$



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.

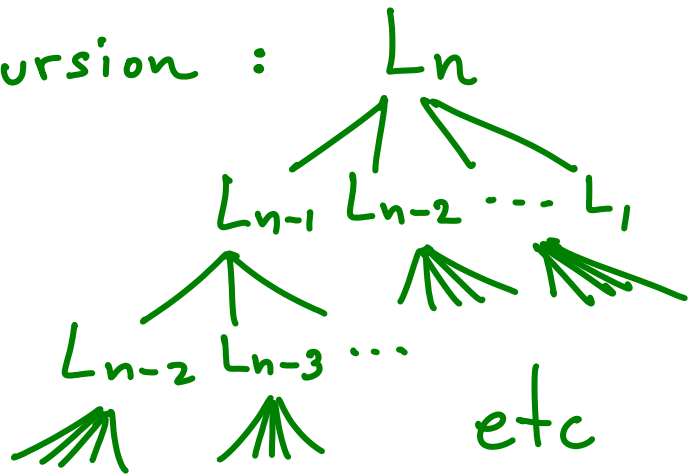
time? space?



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

Recursion :

BAD



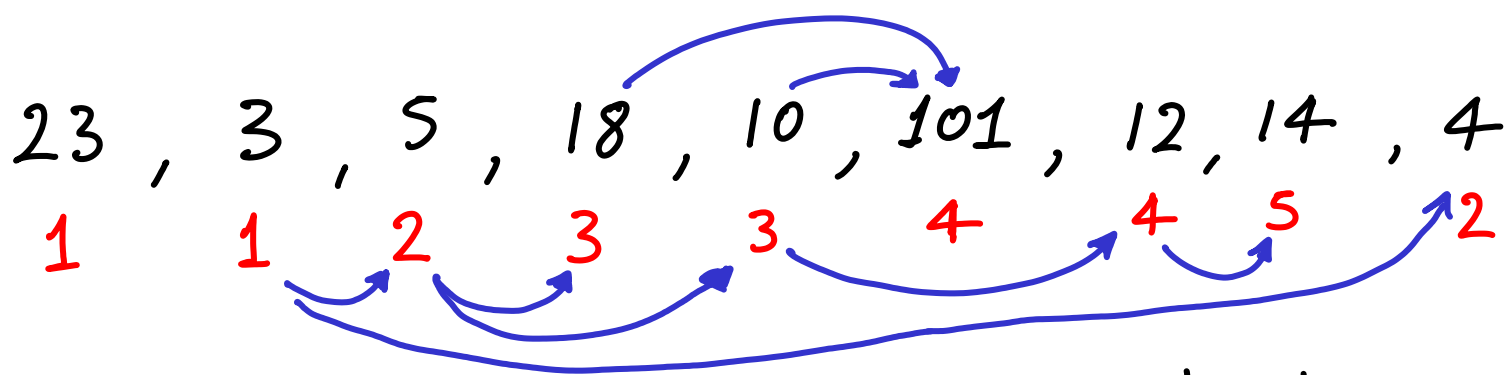
Dyn. Prog: Build solutions, "bottom up"

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

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

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

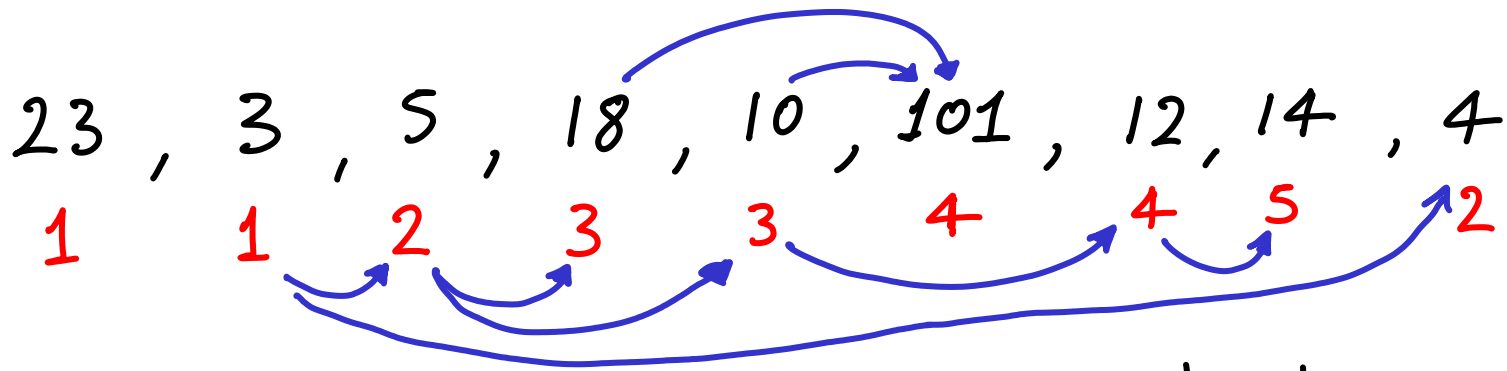
$$\text{Space} = \Theta(n)$$



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

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



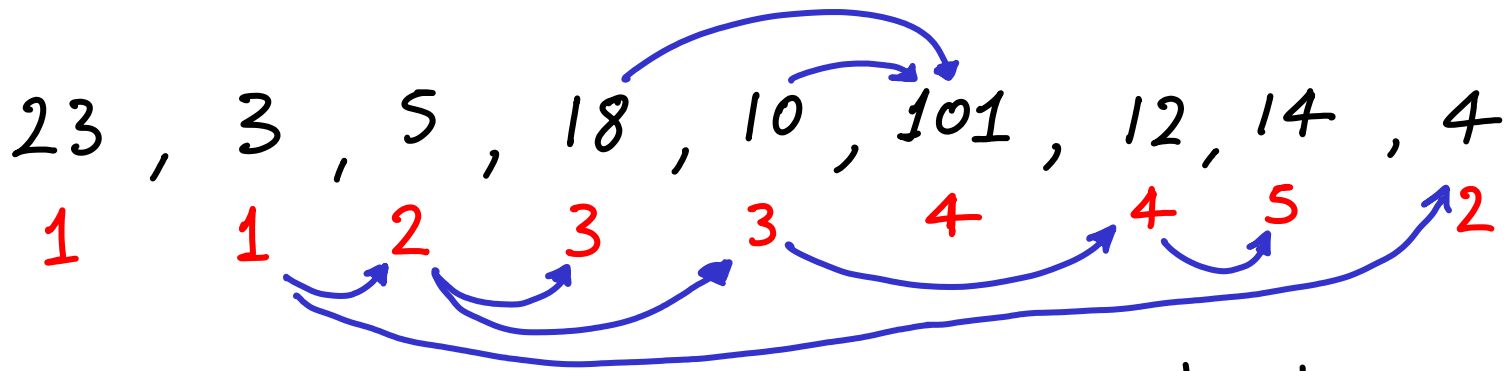


$$T(n) = \Theta(n^2)$$

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

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

What about |L.I.S.|?

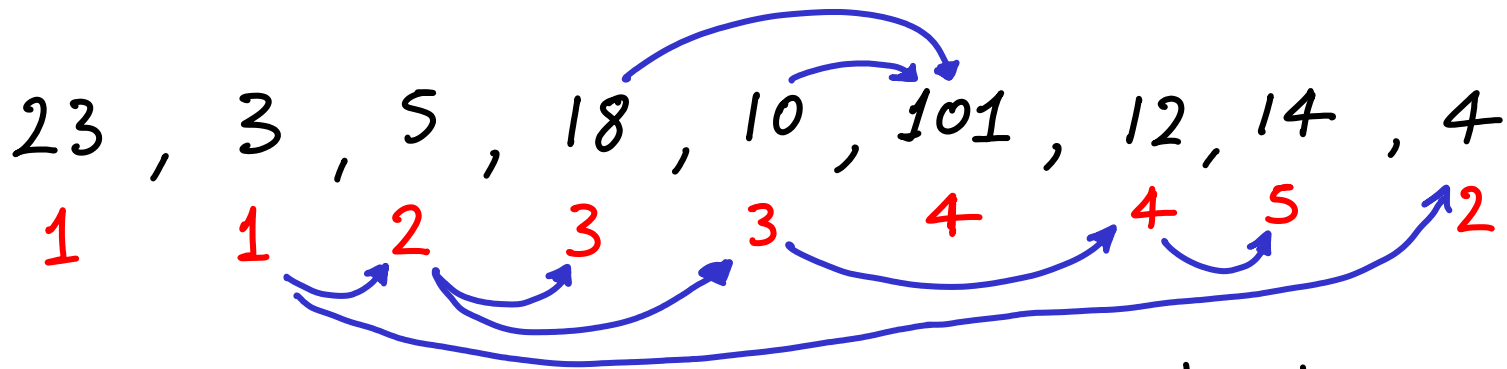


$$T(n) = \Theta(n^2)$$

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

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

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



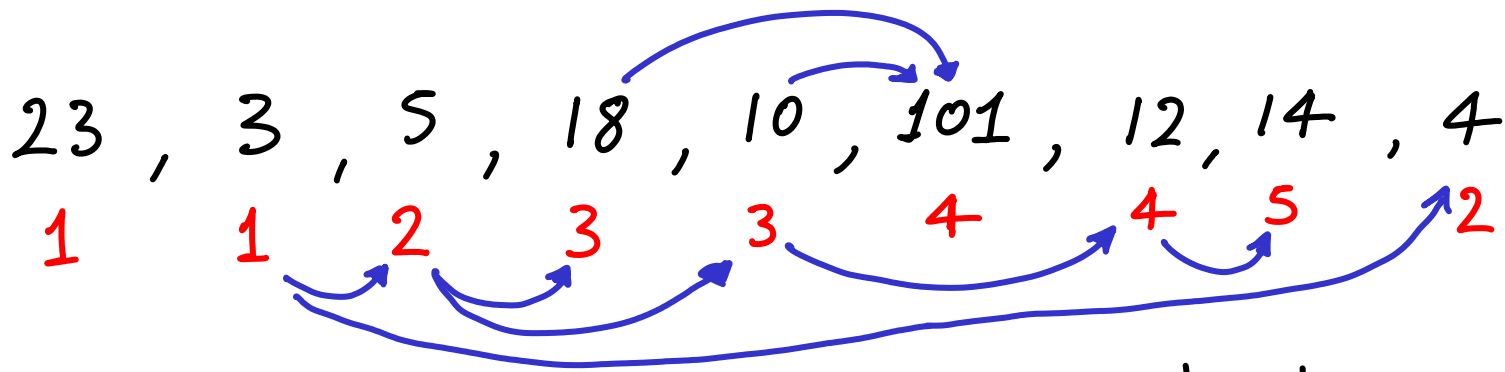
$$T(n) = \Theta(n^2)$$

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

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

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

What about L.I.S.?



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

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

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

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

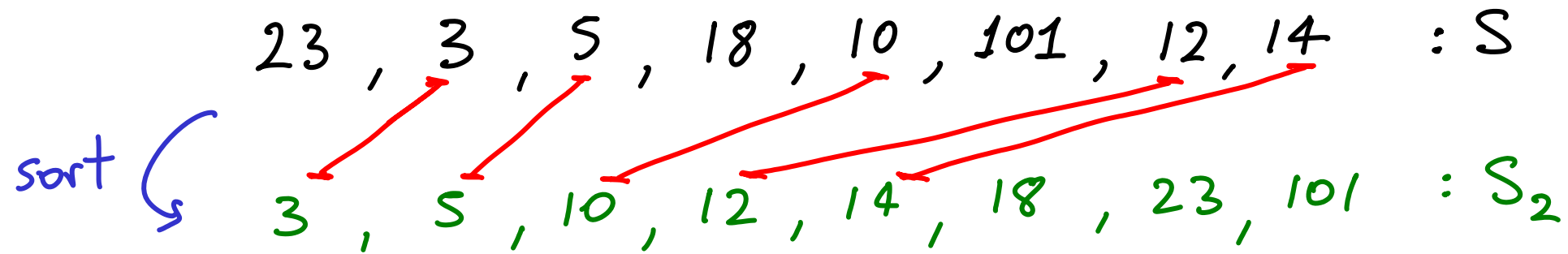
A QUICK SOLUTION FOR L.I.S. ... but still  $O(n^2)$  & dyn-prog.

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

sort ↪ 3, 5, 10, 12, 14, 18, 23, 101 : S<sub>2</sub>

and then?

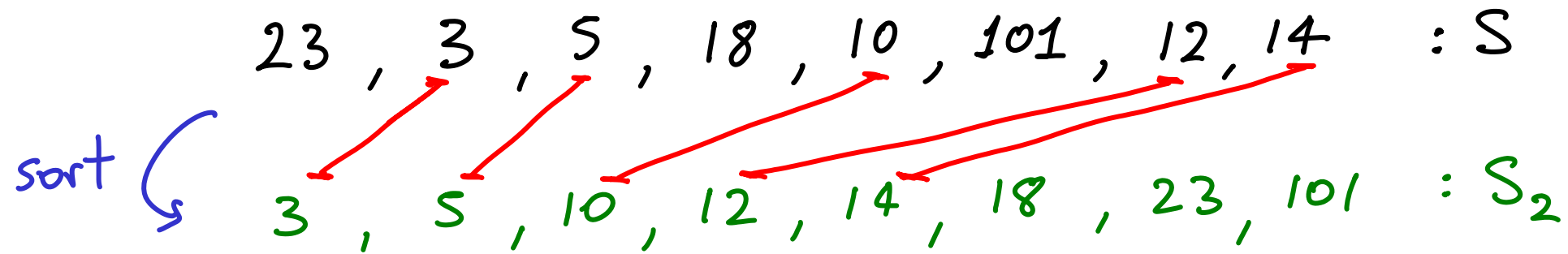
A QUICK SOLUTION FOR L.I.S. ... but still  $O(n^2)$  & dyn-prog.



FIND LONGEST COMMON SUBSEQUENCE !



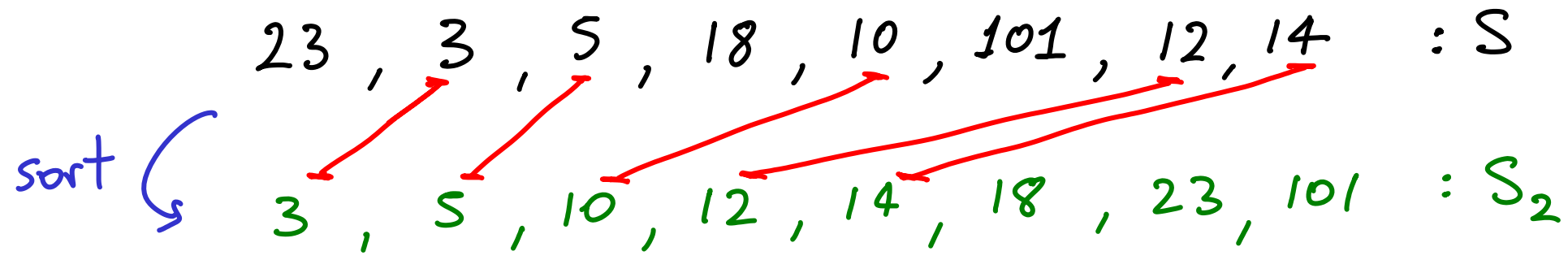
A QUICK SOLUTION FOR L.I.S. ... but still  $O(n^2)$  & dyn-prog.



FIND LONGEST COMMON SUBSEQUENCE !

- any common subsequence is increasing

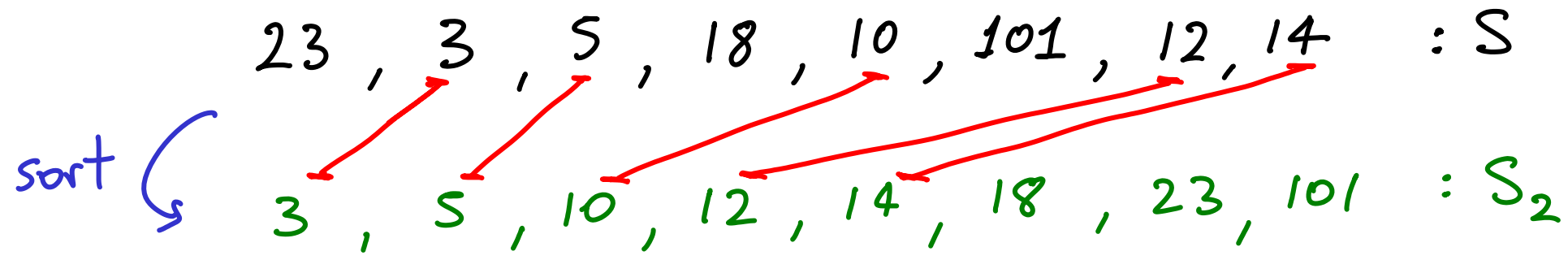
A QUICK SOLUTION FOR L.I.S. ... but still  $O(n^2)$  & dyn-prog.



FIND LONGEST COMMON SUBSEQUENCE !

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

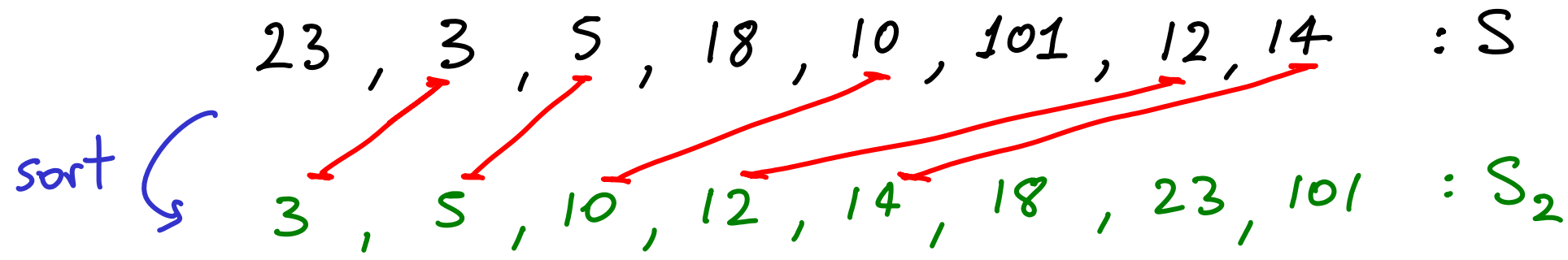
A QUICK SOLUTION FOR L.I.S. ... but still  $O(n^2)$  & dyn-prog.



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.

□