DYNAMIC PROGRAMMING: EVEN MORE FUN!

David Kauchak
CS 140 – Spring 2023

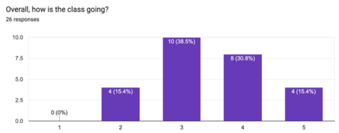
1

Admin

Assignment 5

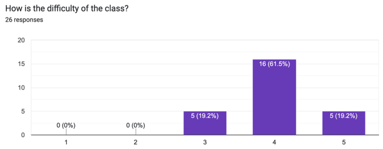
2

Overall how is the class going



3

How difficult is the class

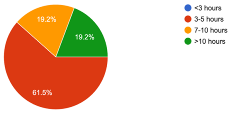


4

Time spent

About how many hours a week do you spend on this class (ignoring the DT assignment :)?

26 responses



5

What’s going well?

The short clips at the start

working with partners!

I finally get to learn DP!

the versatility of the PS because I feel like I'm practicing multiple different concepts

The topic is genuinely interesting and I love thinking of algorithms, they remind be of puzzles.

6

What could be improved?

sometimes the pace of the lectures feel a bit fast

no group sessions

late days

The content feels way too theoretical

Less proofs, less inductions pls

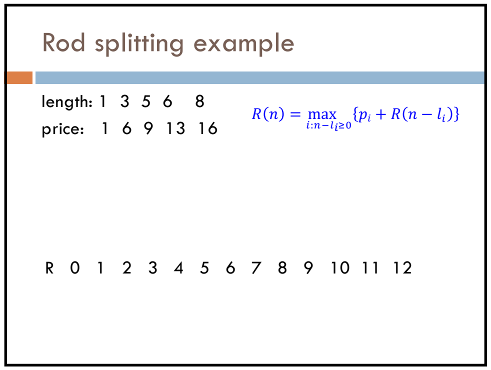
Possible Saturday mentor sessions

7

What could be improved?

It also feels like a level of background is expected from students, even though that background has not been built through previous Pomona CS classes so it feels very unfair to those of us who weren't exposed to CS beyond or before Pomona.

8



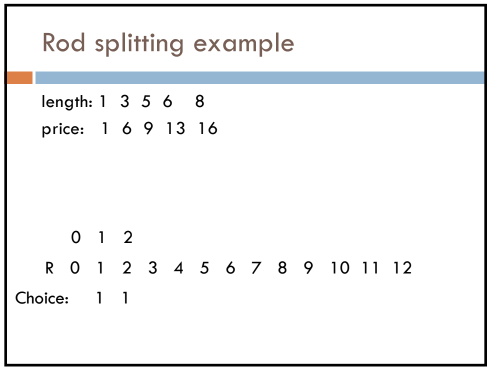
Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

$$R(n) = \max_{i:n-l_i\geq 0}\{p_i + R(n-l_i)\}$$

R	0	1	2	3	4	5	6	7	8	9	10	11	12

9

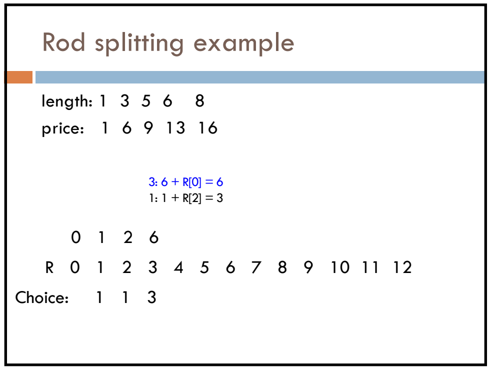


Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

	0	1	2										
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1										

10



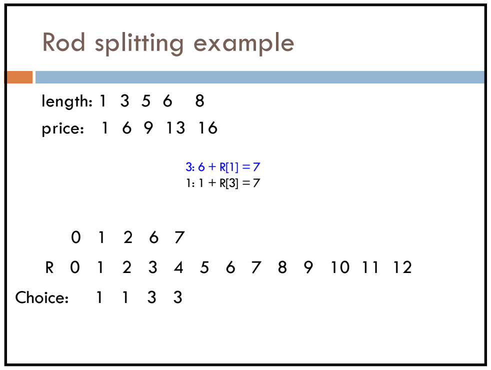
Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

3: 6 + R[0] = 6
1: 1 + R[2] = 3

	0	1	2	6									
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3									

11



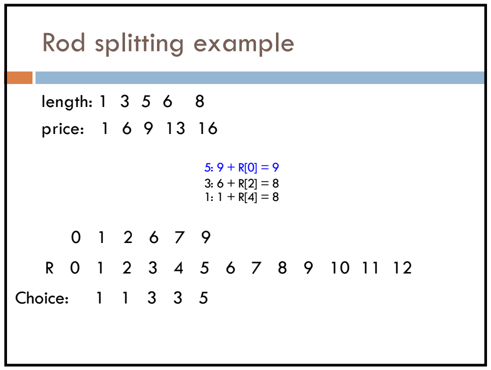
Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

3: 6 + R[1] = 7
1: 1 + R[3] = 7

	0	1	2	6	7								
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3								

12



Rod splitting example

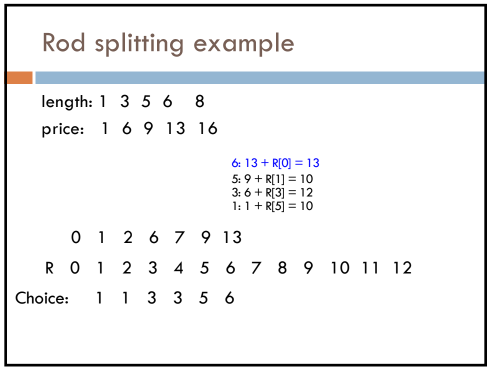
length: 1 3 5 6 8

price: 1 6 9 13 16

5: 9 + R[0] = 9
3: 6 + R[2] = 8
1: 1 + R[4] = 8

	0	1	2	6	7	9							
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5							

13



Rod splitting example

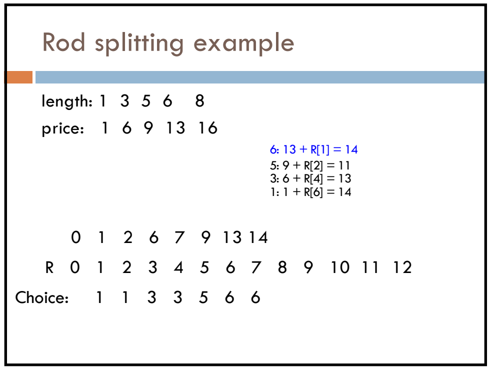
length: 1 3 5 6 8

price: 1 6 9 13 16

6: 13 + R[0] = 13
5: 9 + R[1] = 10
3: 6 + R[3] = 12
1: 1 + R[5] = 10

	0	1	2	6	7	9	13						
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5	6						

14



Rod splitting example

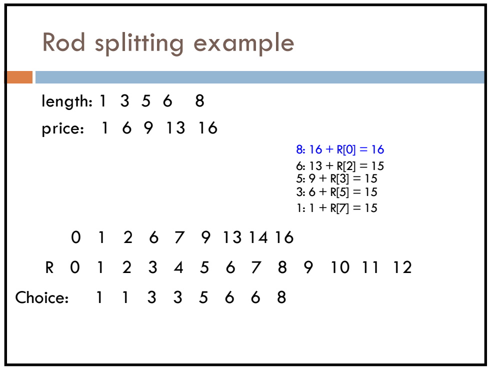
length: 1 3 5 6 8

price: 1 6 9 13 16

6: 13 + R[1] = 14
5: 9 + R[2] = 11
3: 6 + R[4] = 13
1: 1 + R[6] = 14

	0	1	2	6	7	9	13	14					
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5	6	6					

15



Rod splitting example

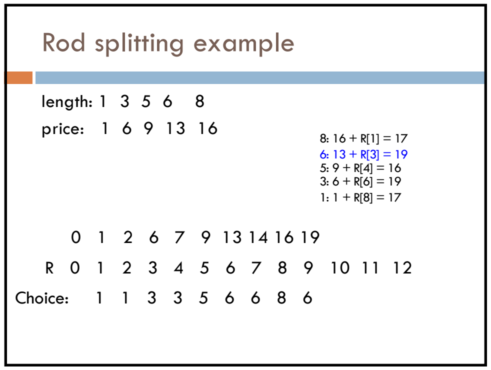
length: 1 3 5 6 8

price: 1 6 9 13 16

8: 16 + R[0] = 16
6: 13 + R[2] = 15
5: 9 + R[3] = 15
3: 6 + R[5] = 15
1: 1 + R[7] = 15

	0	1	2	6	7	9	13	14	16				
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5	6	6	8				

16



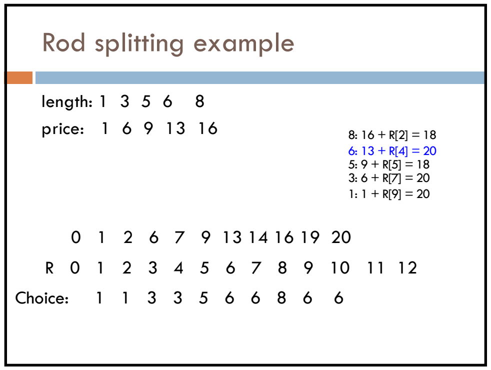
Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

8: 16 + R[1] = 17
6: 13 + R[3] = 19
5: 9 + R[4] = 16
3: 6 + R[6] = 19
1: 1 + R[8] = 17

	0	1	2	6	7	9	13	14	16	19			
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5	6	6	8	6			

17



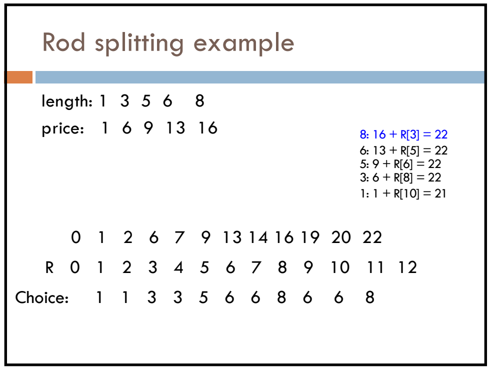
Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

8: 16 + R[2] = 18
6: 13 + R[4] = 20
5: 9 + R[5] = 18
3: 6 + R[7] = 20
1: 1 + R[9] = 20

	0	1	2	6	7	9	13	14	16	19	20		
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5	6	6	8	6	6		

18



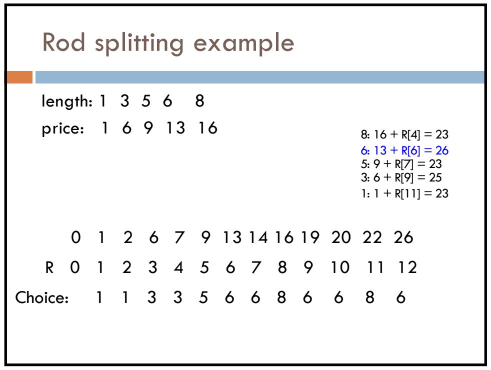
Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

8: 16 + R[3] = 22
6: 13 + R[5] = 22
5: 9 + R[6] = 22
3: 6 + R[8] = 22
1: 1 + R[10] = 21

	0	1	2	6	7	9	13	14	16	19	20	22	
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	

19



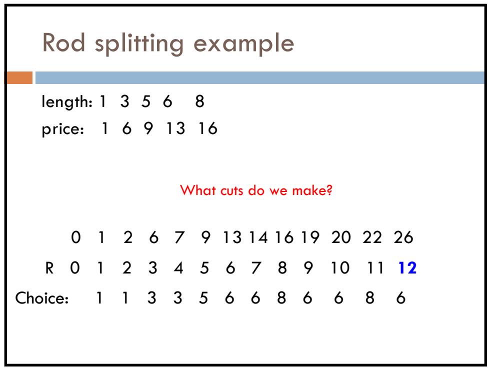
Rod splitting example

length:	1	3	5	6	8
price:	1	6	9	13	16

8: 16 + R[4] = 23
6: 13 + R[6] = 26
5: 9 + R[7] = 23
3: 6 + R[9] = 25
1: 1 + R[11] = 23

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	10	11	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

20



Rod splitting example

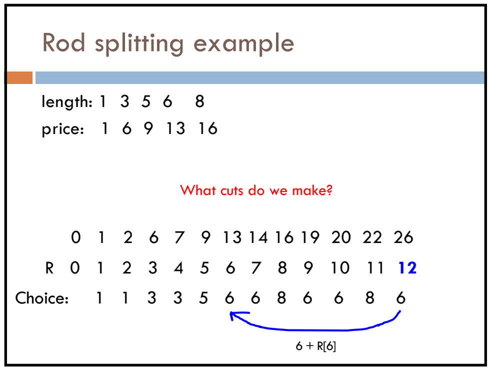
length: 1 3 5 6 8

price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	10	11	**12**
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

21



Rod splitting example

length: 1 3 5 6 8

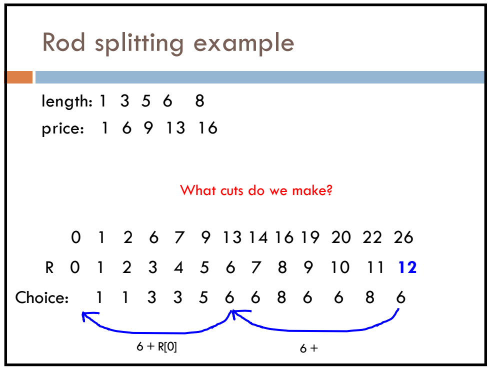
price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	10	11	**12**
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

6 + R[6]

22



Rod splitting example

length: 1 3 5 6 8

price: 1 6 9 13 16

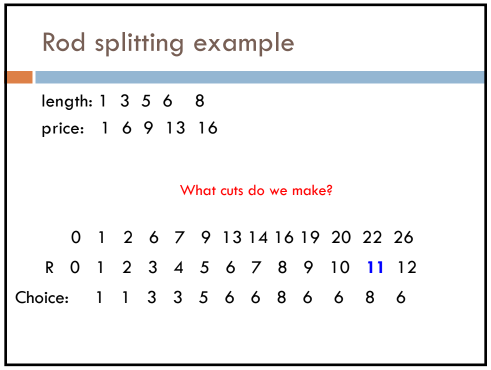
What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	10	11	**12**
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

6 + R[0]

6 +

23



Rod splitting example

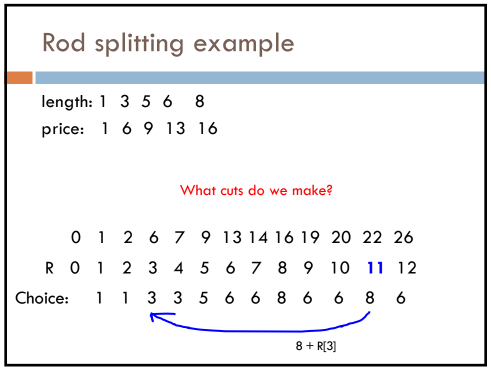
length: 1 3 5 6 8

price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	10	**11**	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

24



Rod splitting example

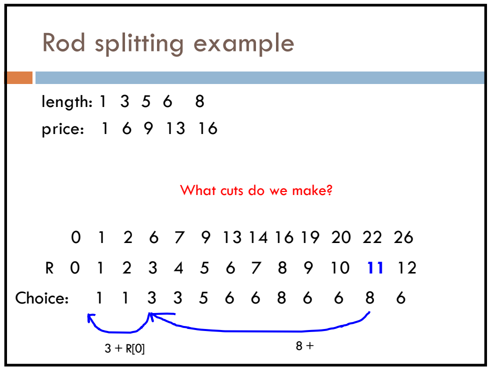
length: 1 3 5 6 8
price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	10	**11**	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

8 + R[3]

25



Rod splitting example

length: 1 3 5 6 8
price: 1 6 9 13 16

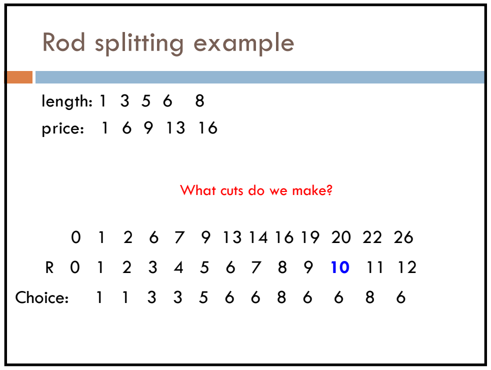
What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	10	**11**	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

3 + R[0]

8 +

26



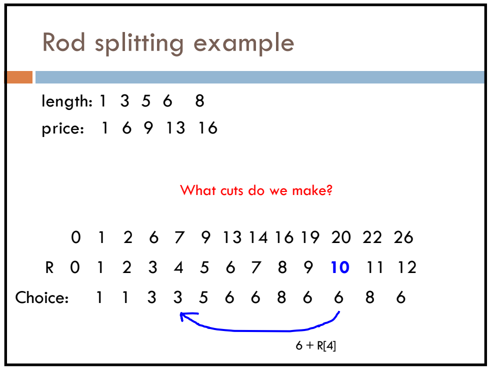
Rod splitting example

length: 1 3 5 6 8
price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	**10**	11	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

27



Rod splitting example

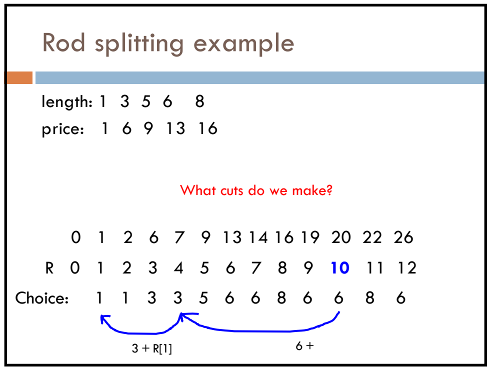
length: 1 3 5 6 8
price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	**10**	11	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

6 + R[4]

28



Rod splitting example

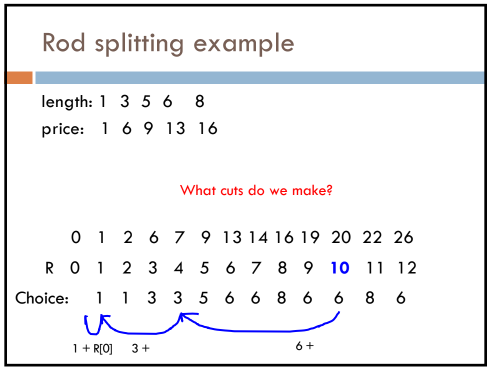
length: 1 3 5 6 8
price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	**10**	11	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

3 + R[1] 6 +

29



Rod splitting example

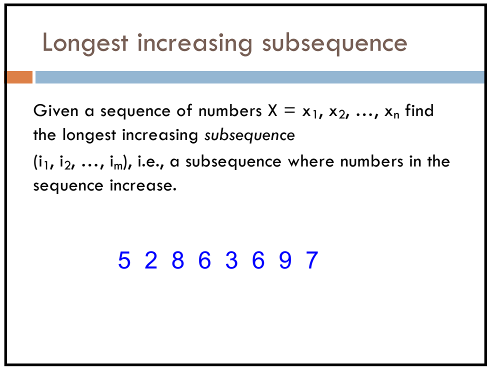
length: 1 3 5 6 8
price: 1 6 9 13 16

What cuts do we make?

	0	1	2	6	7	9	13	14	16	19	20	22	26
R	0	1	2	3	4	5	6	7	8	9	**10**	11	12
Choice:		1	1	3	3	5	6	6	8	6	6	8	6

1 + R[0] 3 + 6 +

30

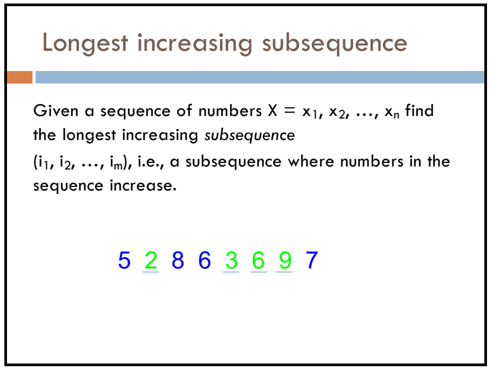


Longest increasing subsequence

Given a sequence of numbers X = $x_1, x_2, \ldots, x_n$ find the longest increasing *subsequence* ($i_1, i_2, \ldots, i_m$), i.e., a subsequence where numbers in the sequence increase.

5 2 8 6 3 6 9 7

31

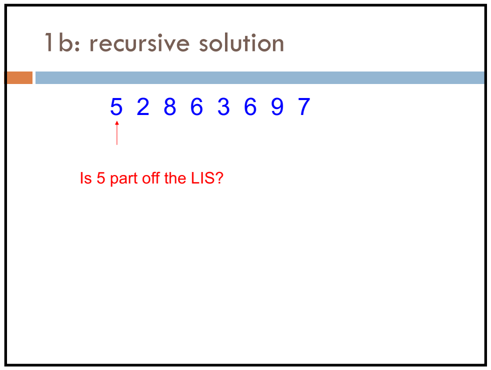


Longest increasing subsequence

Given a sequence of numbers X = $x_1, x_2, \ldots, x_n$ find the longest increasing *subsequence* ($i_1, i_2, \ldots, i_m$), i.e., a subsequence where numbers in the sequence increase.

5 2 8 6 3 6 9 7

32

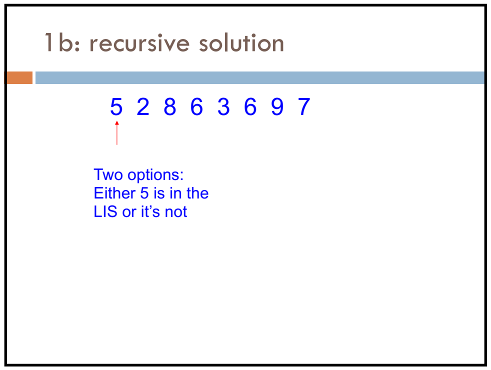


1b: recursive solution

5 2 8 6 3 6 9 7

Is 5 part off the LIS?

36

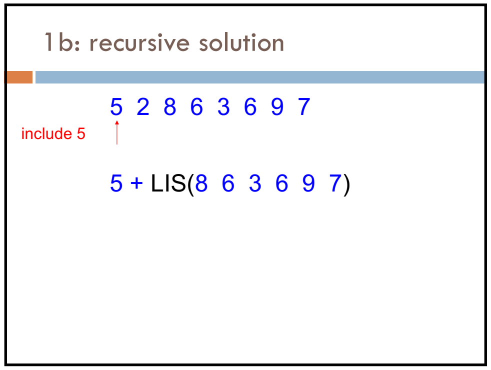


1b: recursive solution

5 2 8 6 3 6 9 7

Two options:
Either 5 is in the
LIS or it's not

37



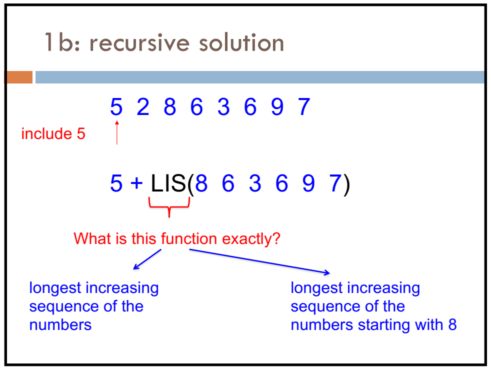
1b: recursive solution

5 2 8 6 3 6 9 7

include 5

5 + LIS(8 6 3 6 9 7)

38



1b: recursive solution

5 2 8 6 3 6 9 7

include 5

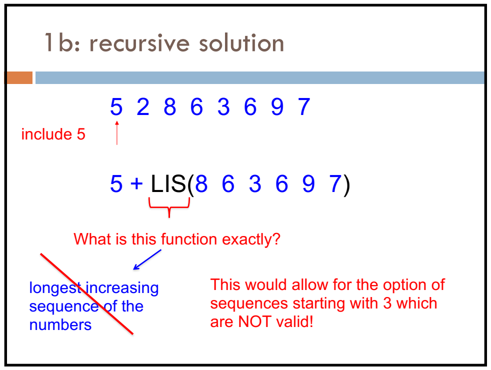
5 + LIS(8 6 3 6 9 7)

What is this function exactly?

longest increasing
sequence of the
numbers

longest increasing
sequence of the
numbers starting with 8

39



1b: recursive solution

5 2 8 6 3 6 9 7

include 5

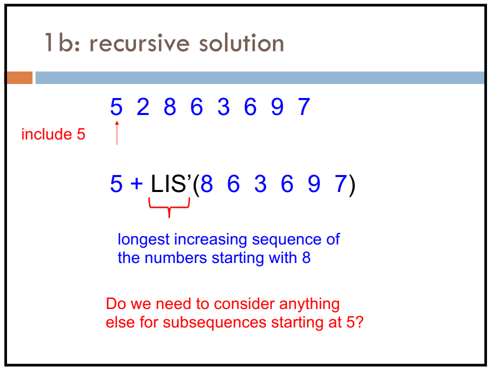
5 + LIS(8 6 3 6 9 7)

What is this function exactly?

~~longest increasing sequence of the numbers~~

This would allow for the option of sequences starting with 3 which are NOT valid!

40



1b: recursive solution

5 2 8 6 3 6 9 7

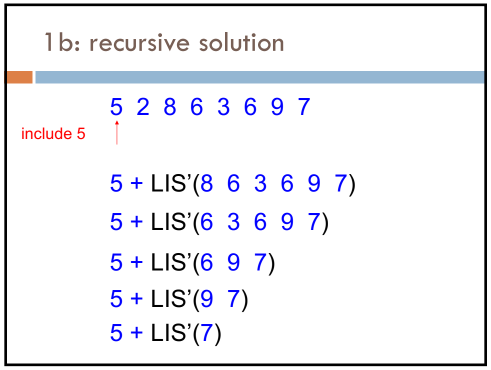
include 5

5 + LIS'(8 6 3 6 9 7)

longest increasing sequence of the numbers starting with 8

Do we need to consider anything else for subsequences starting at 5?

41



1b: recursive solution

5 2 8 6 3 6 9 7

include 5

5 + LIS'(8 6 3 6 9 7)

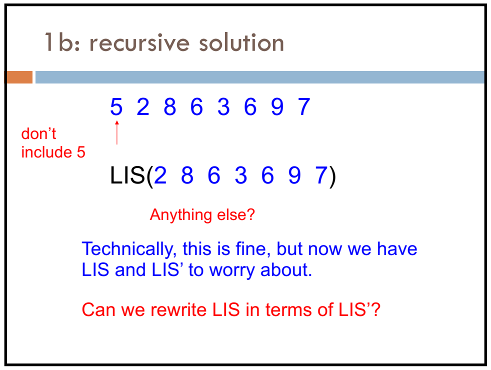
5 + LIS'(6 3 6 9 7)

5 + LIS'(6 9 7)

5 + LIS'(9 7)

5 + LIS'(7)

42



1b: recursive solution

5 2 8 6 3 6 9 7

don't include 5

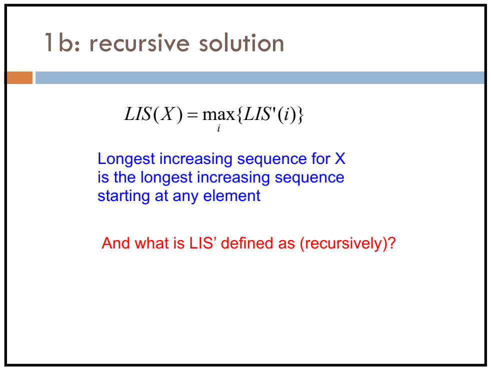
LIS(2 8 6 3 6 9 7)

Anything else?

Technically, this is fine, but now we have LIS and LIS' to worry about.

Can we rewrite LIS in terms of LIS'?

43



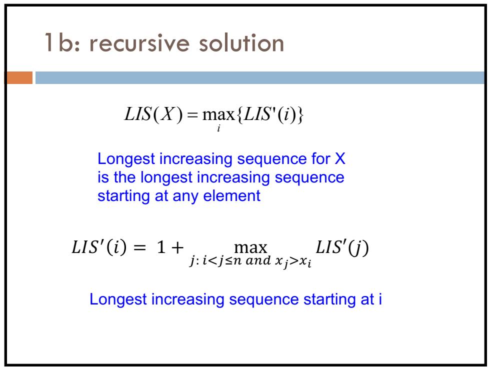
1b: recursive solution

$$LIS(X) = \max_i \{LIS'(i)\}$$

Longest increasing sequence for X is the longest increasing sequence starting at any element

And what is LIS' defined as (recursively)?

44



1b: recursive solution

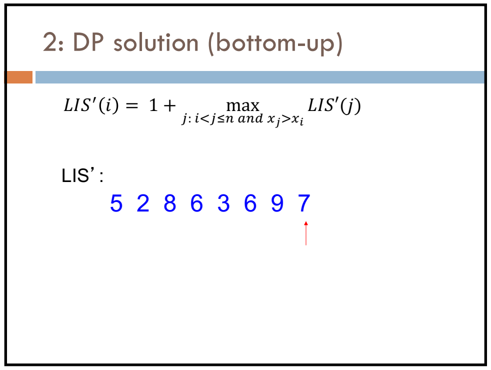
$$LIS(X) = \max_i \{LIS'(i)\}$$

Longest increasing sequence for X is the longest increasing sequence starting at any element

$$LIS'(i) = 1 + \max_{j:\, i<j\leq n \text{ and } x_j > x_i} LIS'(j)$$

Longest increasing sequence starting at i

45



2: DP solution (bottom-up)

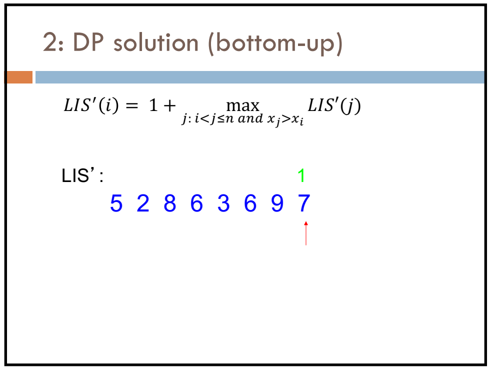
$$LIS'(i) = 1 + \max_{j:\, i<j\leq n \text{ and } x_j > x_i} LIS'(j)$$

LIS' :

5 2 8 6 3 6 9 7

↑

46



2: DP solution (bottom-up)

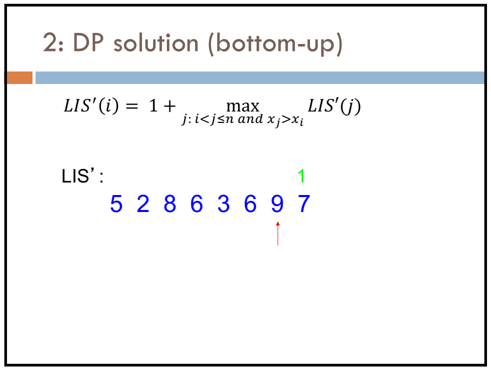
$$LIS'(i) = 1 + \max_{j:\, i<j\leq n \text{ and } x_j > x_i} LIS'(j)$$

LIS' : 1

5 2 8 6 3 6 9 7

↑

47

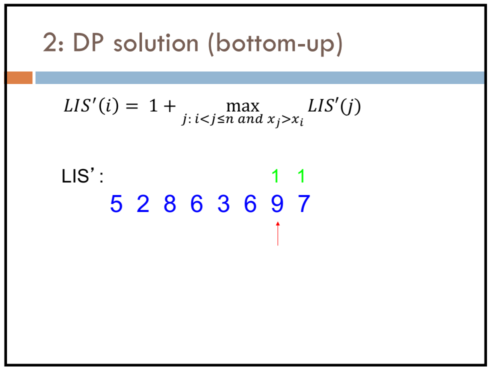


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

$$\begin{array}{lcccccccc} \text{LIS'}: & & & & & & & & 1 \\ & 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \\ & & & & & & & \uparrow & \end{array}$$

48

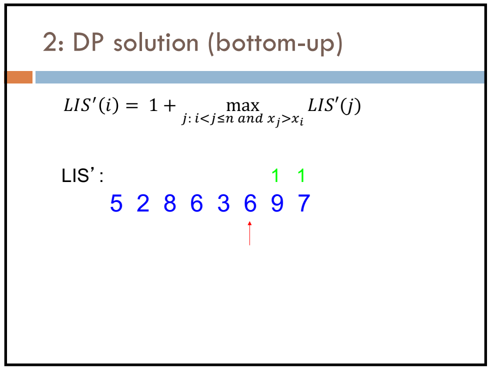


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

$$\begin{array}{lcccccccc} \text{LIS'}: & & & & & & & 1 & 1 \\ & 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \\ & & & & & & & \uparrow & \end{array}$$

49

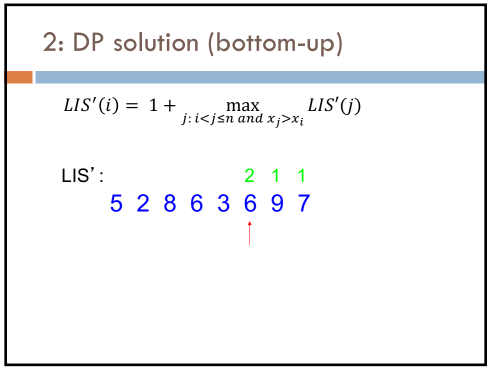


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

$$\begin{array}{lcccccccc} \text{LIS'}: & & & & & & & 1 & 1 \\ & 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \\ & & & & & & \uparrow & & \end{array}$$

50

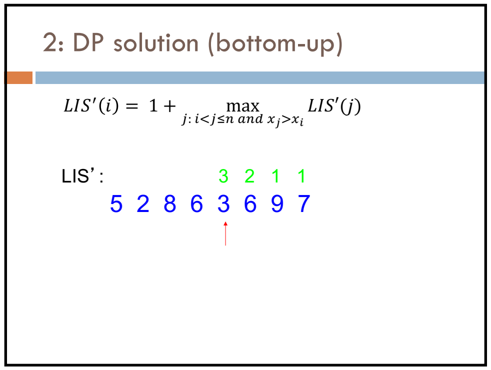


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

$$\begin{array}{lcccccccc} \text{LIS'}: & & & & & & 2 & 1 & 1 \\ & 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \\ & & & & & & \uparrow & & \end{array}$$

51

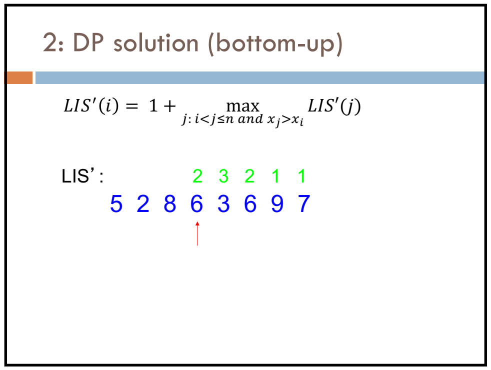


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

LIS' :					3	2	1	1
	5	2	8	6	3	6	9	7
					↑			

52

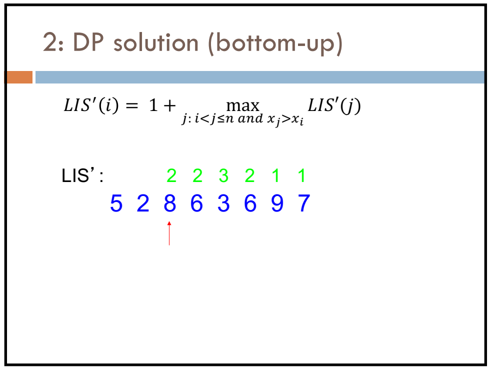


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

LIS' :				2	3	2	1	1
	5	2	8	6	3	6	9	7
				↑				

53

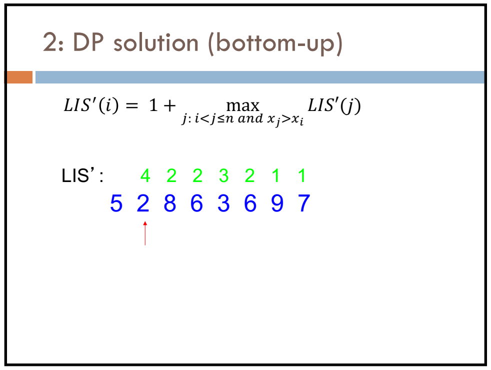


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

LIS' :			2	2	3	2	1	1
	5	2	8	6	3	6	9	7
			↑					

54

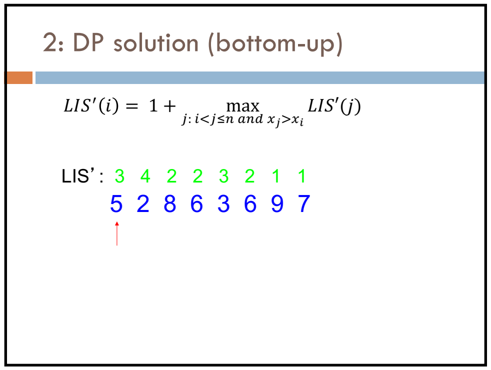


2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\le n \text{ and } x_j > x_i} LIS'(j)$$

LIS' :		4	2	2	3	2	1	1
	5	2	8	6	3	6	9	7
		↑						

55



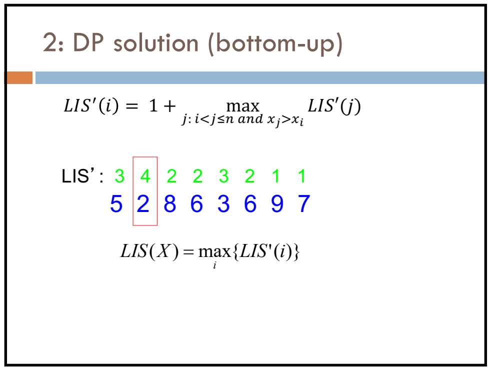
2: DP solution (bottom-up)

$LIS'(i) = 1 + \max_{j: i<j\le n \text{ and } x_j>x_i} LIS'(j)$

LIS’ : 3 4 2 2 3 2 1 1

5 2 8 6 3 6 9 7

56



2: DP solution (bottom-up)

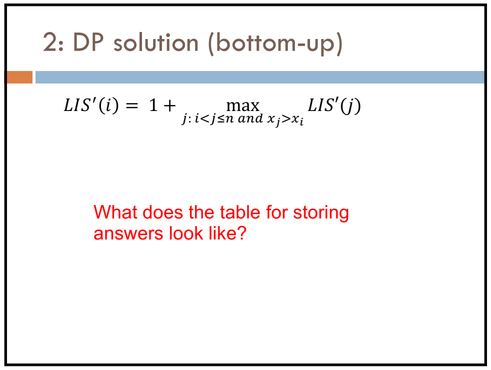
$LIS'(i) = 1 + \max_{j: i<j\le n \text{ and } x_j>x_i} LIS'(j)$

LIS’ : 3 4 2 2 3 2 1 1

5 2 8 6 3 6 9 7

$LIS(X) = \max_i \{LIS'(i)\}$

57

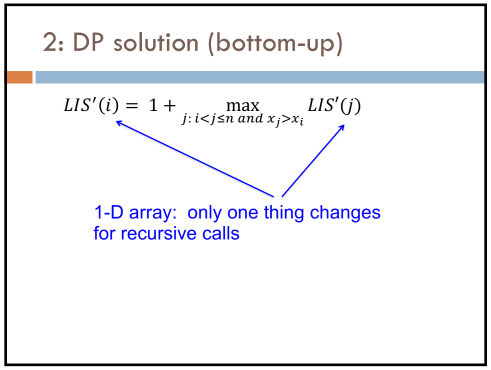


2: DP solution (bottom-up)

$LIS'(i) = 1 + \max_{j: i<j\le n \text{ and } x_j>x_i} LIS'(j)$

What does the table for storing answers look like?

58



2: DP solution (bottom-up)

$LIS'(i) = 1 + \max_{j: i<j\le n \text{ and } x_j>x_i} LIS'(j)$

1-D array: only one thing changes for recursive calls

59

2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:\, i<j\leq n \text{ and } x_j>x_i} LIS'(j)$$

What are the “smallest” possible subproblems?

To calculate LIS’(n), what are all the subproblems we need to calculate? This is the “table”.

How should we fill in the table?

Where will the answer be?

60

2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

What are the “smallest” possible subproblems?
LIS’(n) and that is well-defined for this problem

To calculate LIS’(i), what are all the subproblems we need to calculate?
This is the “table”.
LIS’(1) … LIS’(n)

How should we fill in the table?
n → 1

Where will the answer be?
max(LIS’(1)…LIS’(n))

61

2: DP solution (bottom-up)

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
        max ← 1
        for j ← i + 1 to n
                if X[j] > X[i]
                        if 1 + lis[j] > max
                                max ← 1 + lis[j]
        lis[i] ← max
max ← 0
for i ← 1 to n
        if lis[i] > max
                max ← lis[i]
return max
```

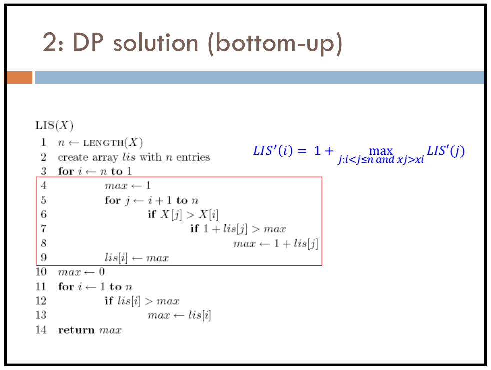
62

2: DP solution (bottom-up)

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
        max ← 1
        for j ← i + 1 to n
                if X[j] > X[i]
                        if 1 + lis[j] > max
                                max ← 1 + lis[j]
        lis[i] ← max
max ← 0
for i ← 1 to n
        if lis[i] > max
                max ← lis[i]
return max
```

start from the end (bottom)

63

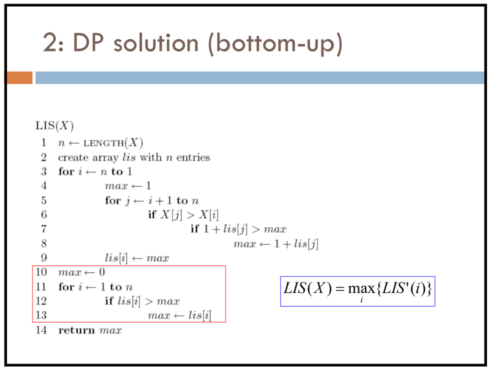


2: DP solution (bottom-up)

```
LIS($X$)
n ← LENGTH($X$)
create array $lis$ with $n$ entries
for $i ← n$ to 1
        $max ← 1$
        for $j ← i+1$ to $n$
                if $X[j] > X[i]$
                        if $1 + lis[j] > max$
                                $max ← 1 + lis[j]$
        $lis[i] ← max$
$max ← 0$
for $i ← 1$ to $n$
        if $lis[i] > max$
                $max ← lis[i]$
return $max$
```

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

64

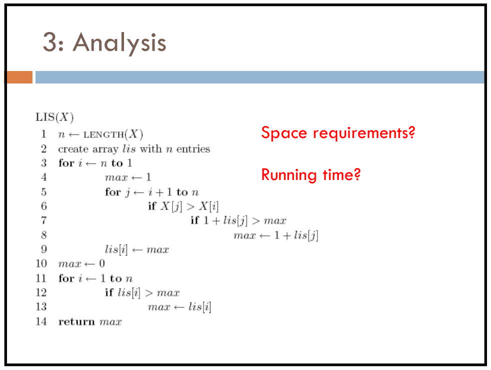


2: DP solution (bottom-up)

```
LIS($X$)
$n ← $ LENGTH($X$)
create array $lis$ with $n$ entries
for $i ← n$ to 1
        $max ← 1$
        for $j ← i+1$ to $n$
                if $X[j] > X[i]$
                        if $1 + lis[j] > max$
                                $max ← 1 + lis[j]$
        $lis[i] ← max$
$max ← 0$
for $i ← 1$ to $n$
        if $lis[i] > max$
                $max ← lis[i]$
return $max$
```

$$\boxed{LIS(X) = \max_i \{LIS'(i)\}}$$

65



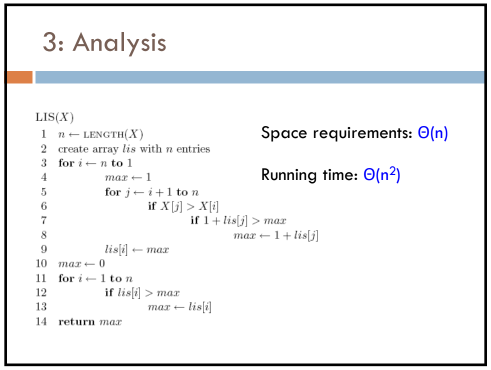
3: Analysis

```
LIS($X$)
$n ← $ LENGTH($X$)
create array $lis$ with $n$ entries
for $i ← n$ to 1
        $max ← 1$
        for $j ← i+1$ to $n$
                if $X[j] > X[i]$
                        if $1 + lis[j] > max$
                                $max ← 1 + lis[j]$
        $lis[i] ← max$
$max ← 0$
for $i ← 1$ to $n$
        if $lis[i] > max$
                $max ← lis[i]$
return $max$
```

Space requirements?

Running time?

66



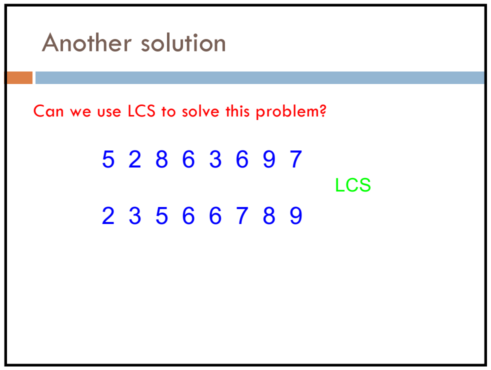
3: Analysis

```
LIS($X$)
$n ← $ LENGTH($X$)
create array $lis$ with $n$ entries
for $i ← n$ to 1
        $max ← 1$
        for $j ← i+1$ to $n$
                if $X[j] > X[i]$
                        if $1 + lis[j] > max$
                                $max ← 1 + lis[j]$
        $lis[i] ← max$
$max ← 0$
for $i ← 1$ to $n$
        if $lis[i] > max$
                $max ← lis[i]$
return $max$
```

Space requirements: $\Theta(n)$

Running time: $\Theta(n^2)$

67



Another solution

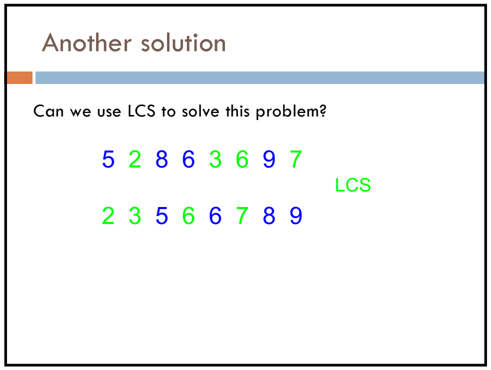
Can we use LCS to solve this problem?

5 2 8 6 3 6 9 7

LCS

2 3 5 6 6 7 8 9

68



Another solution

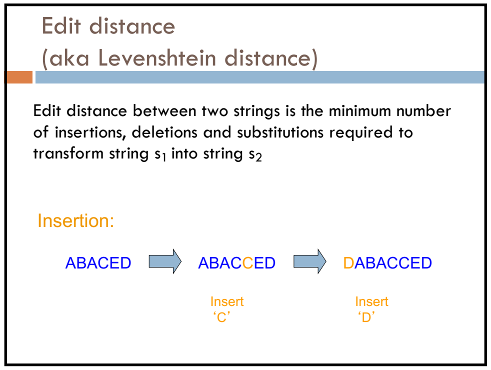
Can we use LCS to solve this problem?

5 2 8 6 3 6 9 7

LCS

2 3 5 6 6 7 8 9

69



Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string s_1 into string s_2

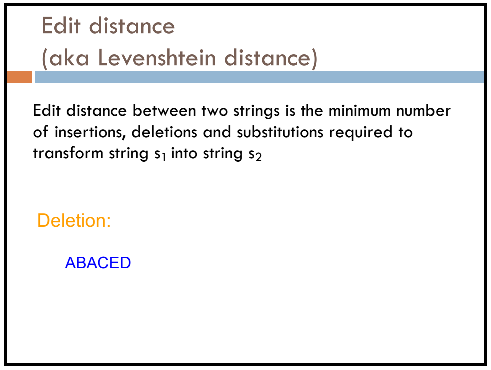
Insertion:

ABACED → ABACCED → DABACCED

Insert 'C'

Insert 'D'

70



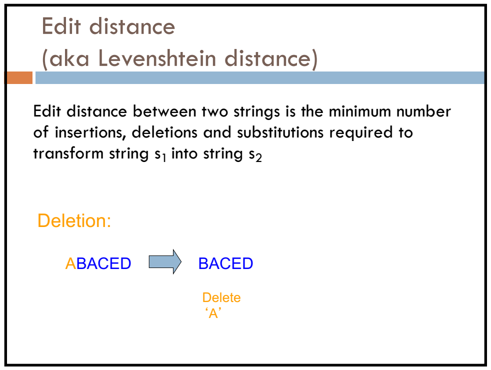
Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string s_1 into string s_2

Deletion:

ABACED

71



Edit distance (aka Levenshtein distance)

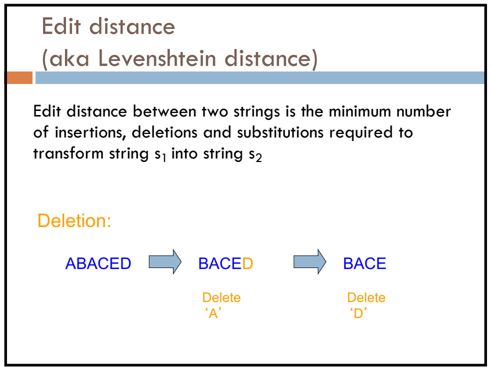
Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string s_1 into string s_2

Deletion:

ABACED → BACED

Delete 'A'

72



Edit distance (aka Levenshtein distance)

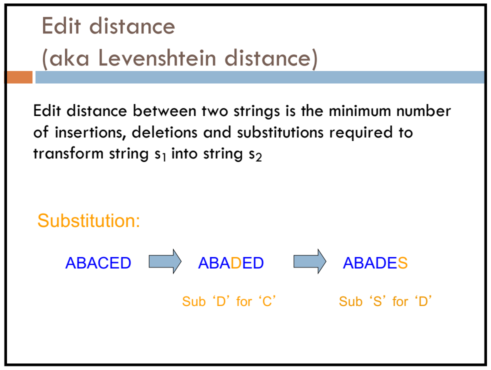
Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string s_1 into string s_2

Deletion:

ABACED → BACED → BACE

Delete 'A'   Delete 'D'

73



Edit distance (aka Levenshtein distance)

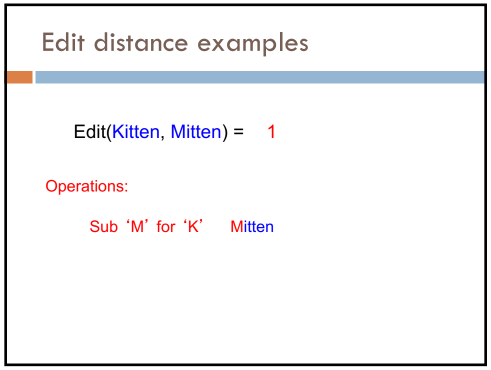
Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string s_1 into string s_2

Substitution:

ABACED → ABADED → ABADES

Sub 'D' for 'C'   Sub 'S' for 'D'

74



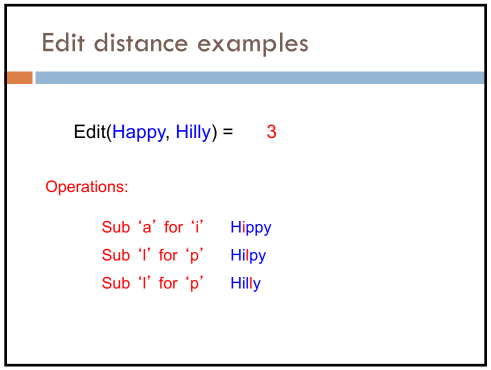
Edit distance examples

Edit(Kitten, Mitten) = 1

Operations:

Sub 'M' for 'K'   Mitten

75



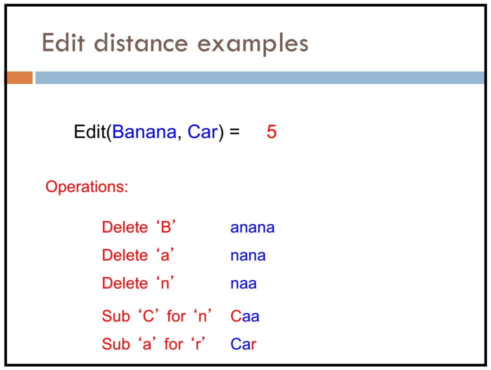
Edit distance examples

Edit(Happy, Hilly) = 3

Operations:

Sub ‘a’ for ‘i’	Hippy
Sub ‘l’ for ‘p’	Hilpy
Sub ‘l’ for ‘p’	Hilly

76



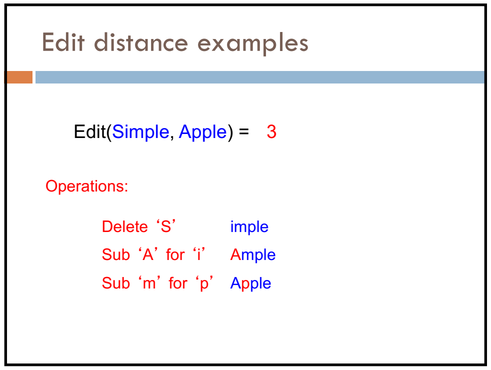
Edit distance examples

Edit(Banana, Car) = 5

Operations:

Delete ‘B’	anana
Delete ‘a’	nana
Delete ‘n’	naa
Sub ‘C’ for ‘n’	Caa
Sub ‘a’ for ‘r’	Car

77



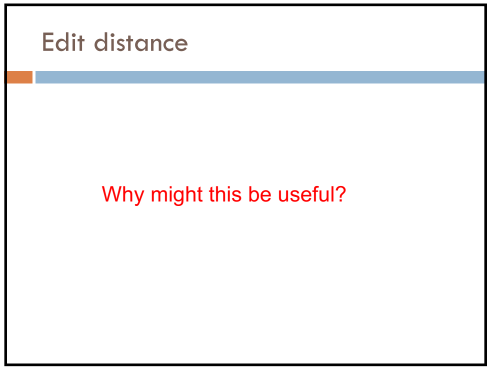
Edit distance examples

Edit(Simple, Apple) = 3

Operations:

Delete ‘S’	imple
Sub ‘A’ for ‘i’	Ample
Sub ‘m’ for ‘p’	Apple

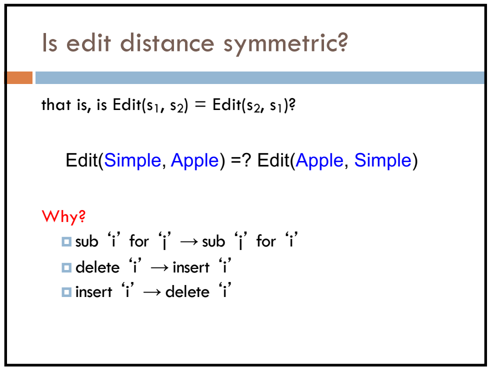
78



Edit distance

Why might this be useful?

79



Is edit distance symmetric?

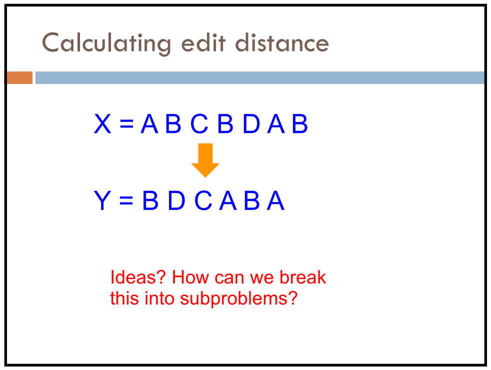
that is, is Edit(s_1, s_2) = Edit(s_2, s_1)?

Edit(Simple, Apple) =? Edit(Apple, Simple)

Why?

- sub 'i' for 'j' → sub 'j' for 'i'
- delete 'i' → insert 'i'
- insert 'i' → delete 'i'

80



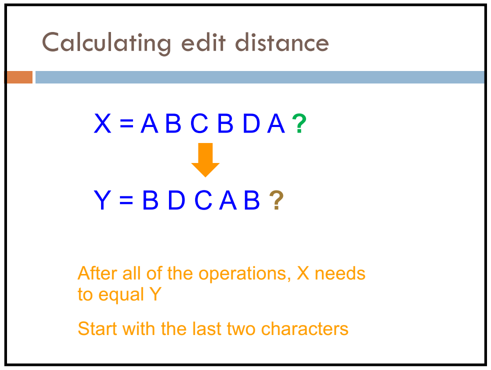
Calculating edit distance

X = A B C B D A B

Y = B D C A B A

Ideas? How can we break this into subproblems?

81



Calculating edit distance

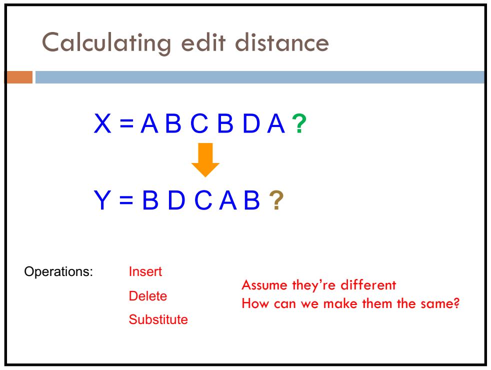
X = A B C B D A ?

Y = B D C A B ?

After all of the operations, X needs to equal Y

Start with the last two characters

82



Calculating edit distance

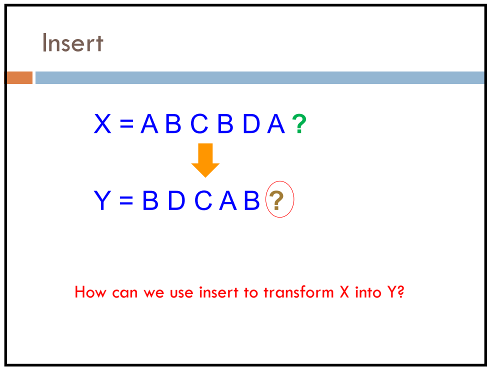
X = A B C B D A ?

Y = B D C A B ?

Operations: Insert
Delete
Substitute

Assume they're different
How can we make them the same?

83



Insert

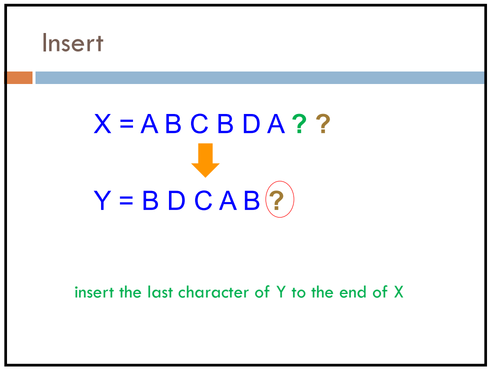
X = A B C B D A ?

↓

Y = B D C A B ?

How can we use insert to transform X into Y?

84



Insert

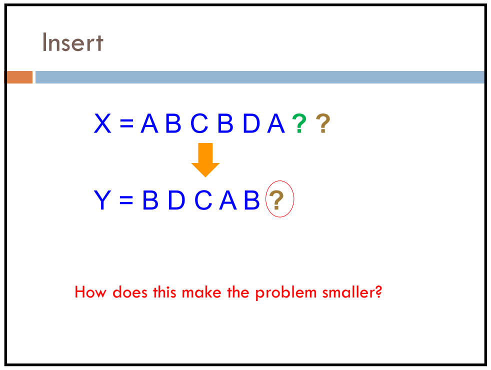
X = A B C B D A ? ?

↓

Y = B D C A B ?

insert the last character of Y to the end of X

85



Insert

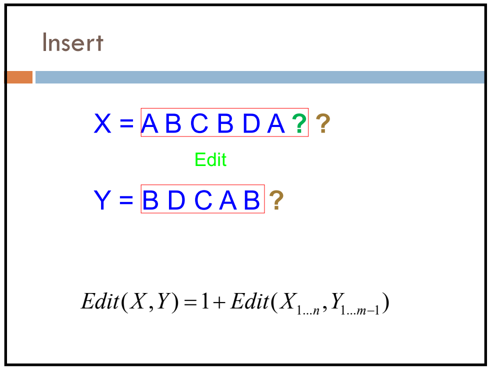
X = A B C B D A ? ?

↓

Y = B D C A B ?

How does this make the problem smaller?

86



Insert

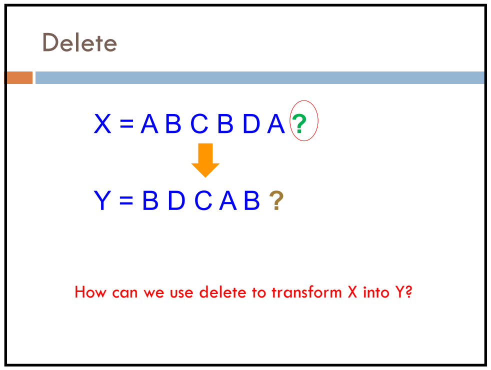
$X = \boxed{A\ B\ C\ B\ D\ A\ ?}\ ?$

Edit

$Y = \boxed{B\ D\ C\ A\ B}\ ?$

$$Edit(X,Y) = 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1})$$

87



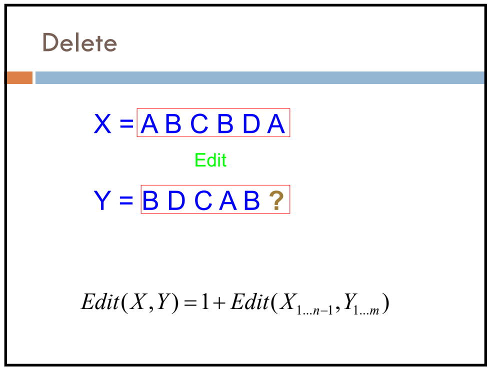
Delete

X = A B C B D A ?

Y = B D C A B ?

How can we use delete to transform X into Y?

88



Delete

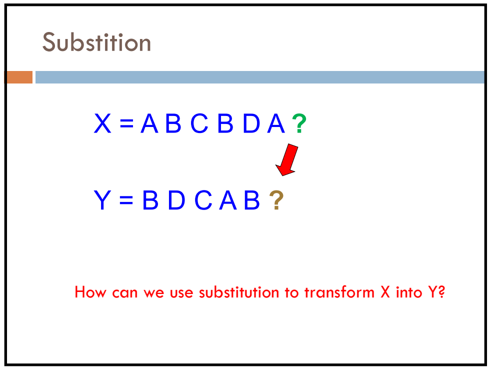
X = A B C B D A

Edit

Y = B D C A B ?

$$Edit(X,Y) = 1 + Edit(X_{1...n-1}, Y_{1...m})$$

89



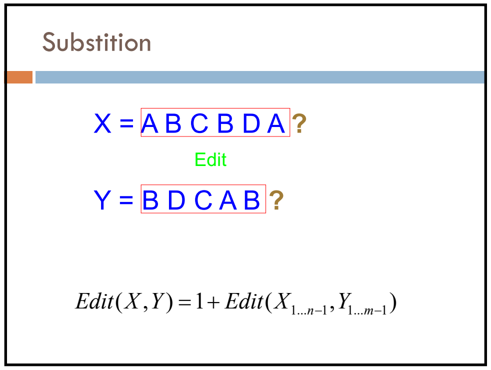
Substition

X = A B C B D A ?

Y = B D C A B ?

How can we use substitution to transform X into Y?

90



Substition

X = A B C B D A ?

Edit

Y = B D C A B ?

$$Edit(X,Y) = 1 + Edit(X_{1...n-1}, Y_{1...m-1})$$

91

Anything else?

X = A B C B D A ?

Y = B D C A B ?

92

Equal

X = A B C B D A ?

Y = B D C A B ?

What if the last characters are equal?

93

Equal

X = A B C B D A ?

Edit

Y = B D C A B ?

$$Edit(X,Y) = Edit(X_{1\ldots n-1}, Y_{1\ldots m-1})$$

94

1b: recursive solution - combining results

Insert: $Edit(X,Y) = 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1})$

Delete: $Edit(X,Y) = 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m})$

$X_n \neq Ym$

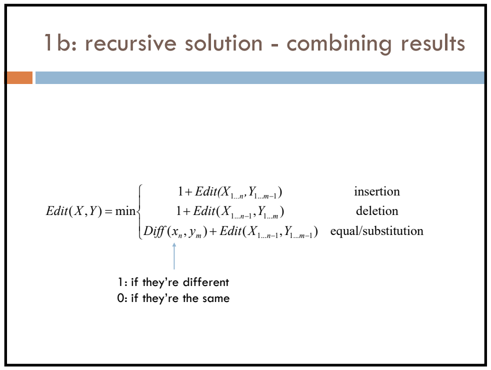
Substitute: $Edit(X,Y) = 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1})$

$X_n = Y_m$

Equal: $Edit(X,Y) = Edit(X_{1\ldots n-1}, Y_{1\ldots m-1})$

How do we decide between these?

95

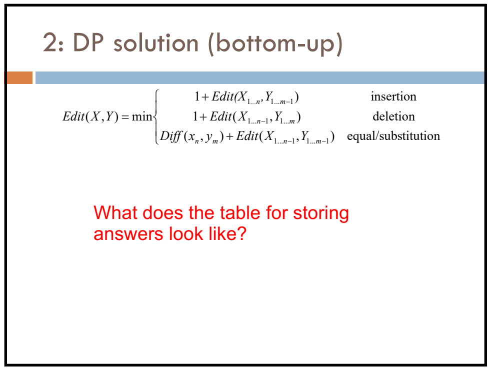


1b: recursive solution - combining results

$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ \underset{\uparrow}{Diff}(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

1: if they're different
0: if they're the same

96

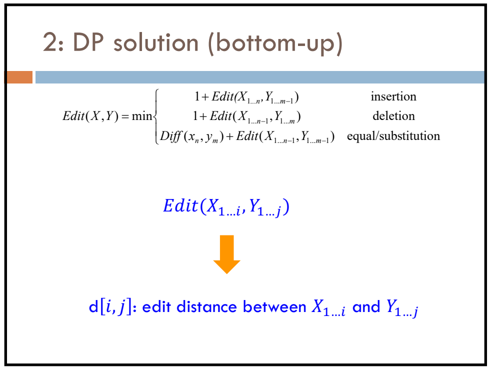


2: DP solution (bottom-up)

$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

What does the table for storing answers look like?

97



2: DP solution (bottom-up)

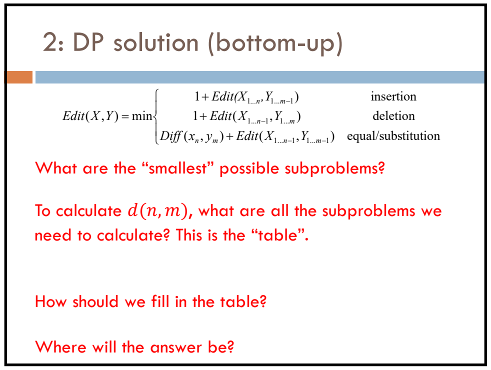
$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

$$Edit(X_{1\ldots i}, Y_{1\ldots j})$$

$$\Downarrow$$

d$[i,j]$: edit distance between $X_{1\ldots i}$ and $Y_{1\ldots j}$

98



2: DP solution (bottom-up)

$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

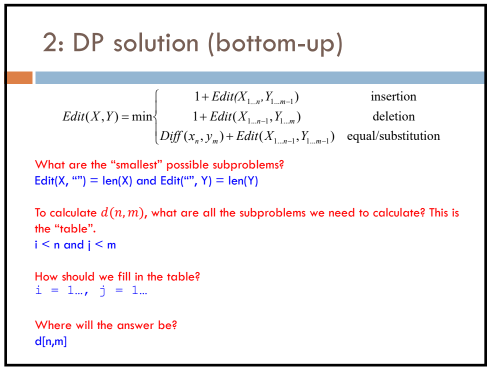
What are the "smallest" possible subproblems?

To calculate $d(n,m)$, what are all the subproblems we need to calculate? This is the "table".

How should we fill in the table?

Where will the answer be?

99



2: DP solution (bottom-up)

$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

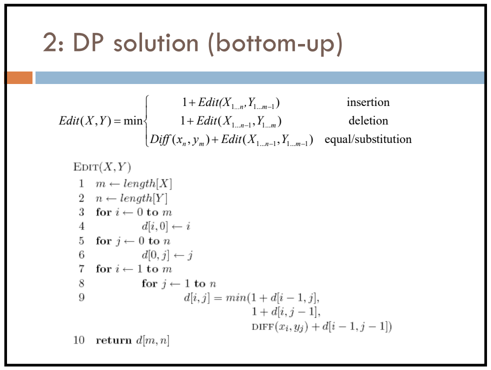
What are the “smallest” possible subproblems?
Edit(X, “”) = len(X) and Edit(“”, Y) = len(Y)

To calculate $d(n, m)$, what are all the subproblems we need to calculate? This is the “table”.
i < n and j < m

How should we fill in the table?
`i = 1…, j = 1…`

Where will the answer be?
d[n,m]

100

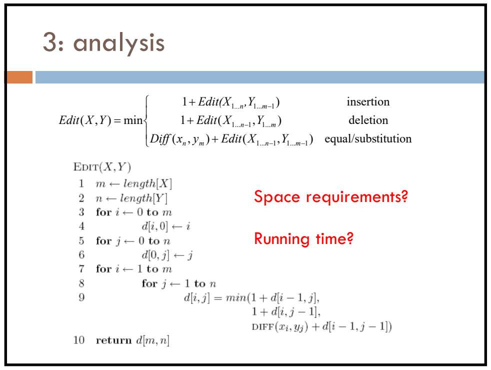


2: DP solution (bottom-up)

$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

```
EDIT($X, Y$)
$m \leftarrow length[X]$
$n \leftarrow length[Y]$
for $i \leftarrow 0$ to $m$
        $d[i,0] \leftarrow i$
for $j \leftarrow 0$ to $n$
        $d[0,j] \leftarrow j$
for $i \leftarrow 1$ to $m$
        for $j \leftarrow 1$ to $n$
                $d[i,j] = min(1 + d[i-1,j],$
                        $1 + d[i,j-1],$
                        $\text{DIFF}(x_i, y_j) + d[i-1,j-1])$
return $d[m,n]$
```

101



3: analysis

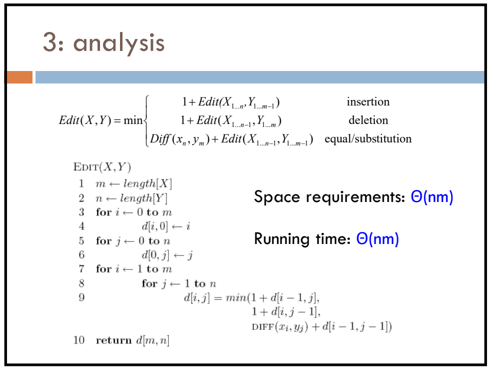
$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

```
EDIT($X, Y$)
$m \leftarrow length[X]$
$n \leftarrow length[Y]$
for $i \leftarrow 0$ to $m$
        $d[i,0] \leftarrow i$
for $j \leftarrow 0$ to $n$
        $d[0,j] \leftarrow j$
for $i \leftarrow 1$ to $m$
        for $j \leftarrow 1$ to $n$
                $d[i,j] = min(1 + d[i-1,j],$
                        $1 + d[i,j-1],$
                        $\text{DIFF}(x_i, y_j) + d[i-1,j-1])$
return $d[m,n]$
```

Space requirements?

Running time?

102



3: analysis

$$Edit(X,Y) = \min\begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

```
EDIT($X, Y$)
$m \leftarrow length[X]$
$n \leftarrow length[Y]$
for $i \leftarrow 0$ to $m$
        $d[i,0] \leftarrow i$
for $j \leftarrow 0$ to $n$
        $d[0,j] \leftarrow j$
for $i \leftarrow 1$ to $m$
        for $j \leftarrow 1$ to $n$
                $d[i,j] = min(1 + d[i-1,j],$
                        $1 + d[i,j-1],$
                        $\text{DIFF}(x_i, y_j) + d[i-1,j-1])$
return $d[m,n]$
```

Space requirements: Θ(nm)

Running time: Θ(nm)

103

Edit distance variants

- Only include insertions and deletions
  - What does this do to substitutions?
- Include swaps, i.e. swapping two adjacent characters counts as one edit
- Weight insertion, deletion and substitution differently
- Weight **specific** character insertion, deletion and substitutions differently
- Length normalize the edit distance

104

DP in practice

Simple English Wikipedia: A New Text Simplification Task

William Coster
Computer Science Department
Pomona College
Claremont, CA 91711
wpc02009@pomona.edu

David Kauchak
Computer Science Department
Pomona College
Claremont, CA 91711
dkauchak@cs.pomona.edu

Abstract

In this paper we examine the task of sentence simplification which aims to reduce the reading complexity of a sentence by incorporating more accessible vocabulary and sentence structure. We introduce a new data set that pairs English Wikipedia with Simple English Wikipedia and is orders of magnitude larger than any previously examined for sentence simplification. The data contains the full range of simplification operations including rewording, reordering, insertion and deletion. We provide an analysis of this corpus as well as preliminary results using a phrase-based translation approach for simplification.

1 Introduction

The task of text simplification aims to reduce the complexity of text while maintaining the content (Chandrasekar and Srinivas, 1997; Carroll et al., 1998; Feng, 2008). In this paper, we explore the *sentence* simplification problem: given a sentence, the goal is to produce an equivalent sentence where the vocabulary and sentence structure are simpler.

Text simplification has a number of important applications. Simplification techniques can be used to make text resources available to a broader range of readers, including children, language learners, the elderly, the hearing impaired and people with aphasia or cognitive disabilities (Carroll et al., 1998; Feng, 2008). As a preprocessing step, simplification can improve the performance of NLP tasks, including parsing, semantic role labeling, machine translation and summarization (Miwa et al., 2010; Jonnalagadda et al., 2009; Vickrey and Koller, 2008; Chandrasekar and Srinivas, 1997). Finally, models for text simplification are similar to models for sentence compression; advances in simplification can benefit compression, which has applications in mobile devices, summarization and captioning (Knight and Marcu, 2002; McDonald, 2006; Galley and McKeown, 2007; Nomoto, 2009; Cohn and Lapata, 2009).

One of the key challenges for text simplification is data availability. The small amount of simplification data currently available has prevented the application of data-driven techniques like those used in other text-to-text translation areas (Och and Ney, 2004; Chiang, 2010). Most prior techniques for text simplification have involved either hand-crafted rules (Vickrey and Koller, 2008; Feng, 2008) or learned within a very restricted rule space (Chandrasekar and Srinivas, 1997).

We have generated a data set consisting of 137K aligned simplified/unsimplified sentence pairs by pairing documents, then sentences from English Wikipedia[1] with corresponding documents and sentences from Simple English Wikipedia[2]. Simple English Wikipedia contains articles aimed at children and English language learners and contains similar content to English Wikipedia but with simpler vocabulary and grammar.

Figure 1 shows example sentence simplifications from the data set. Like machine translation and other text-to-text domains, text simplification involves the full range of transformation operations including deletion, rewording, reordering and insertion.

[1] http://en.wikipedia.org/
[2] http://simple.wikipedia.org

665

Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:shortpapers, pages 665–669,
Portland, Oregon, June 19-24, 2011. ©2011 Association for Computational Linguistics

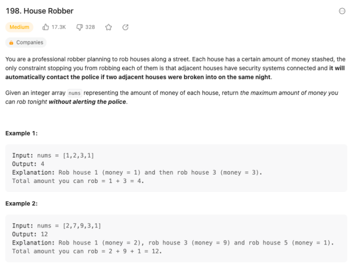
105

For each aligned paragraph pair (i.e. a simple paragraph and one or more normal paragraphs), we then used a dynamic programming approach to find that best global sentence alignment following Barzilay and Elhadad (2003). Specifically, given n normal sentences to align to m simple sentences, we find $a(n, m)$ using the following recurrence:

$$
a(i,j) =
max \begin{cases}
a(i, j-1) - skip\_penalty \\
a(i-1, j) - skip\_penalty \\
a(i-1, j-1) + sim(i,j) \\
a(i-1, j-2) + sim(i,j) + sim(i, j-1) \\
a(i-2, j-1) + sim(i,j) + sim(i-1, j) \\
a(i-2, j-2) + sim(i, j-1) + sim(i-1, j)
\end{cases}
$$

106

https://leetcode.com/problems/house-robber/



198. House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

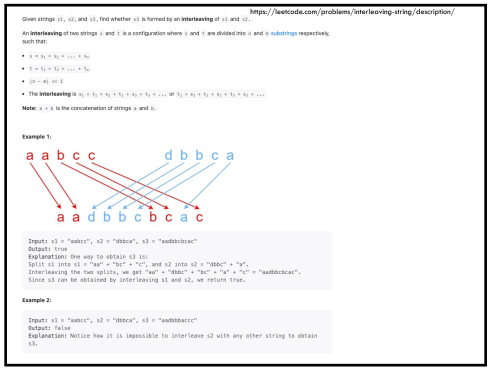
Example 1:

```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

108



https://leetcode.com/problems/interleaving-string/description/

Given strings `s1`, `s2`, and `s3`, find whether `s3` is formed by an **interleaving** of `s1` and `s2`.

An **interleaving** of two strings `s` and `t` is a configuration where `s` and `t` are divided into `n` and `m` substrings respectively, such that:

- $s = s_1 + s_2 + ... + s_n$
- $t = t_1 + t_2 + ... + t_m$
- `|n - m| <= 1`
- The **interleaving** is $s_1 + t_1 + s_2 + t_2 + s_3 + t_3 + ...$ or $t_1 + s_1 + t_2 + s_2 + t_3 + s_3 + ...$

Note: `a + b` is the concatenation of strings `a` and `b`.

Example 1:

```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
Output: true
Explanation: One way to obtain s3 is:
Split s1 into s1 = "aa" + "bc" + "c", and s2 into s2 = "dbbc" + "a".
Interleaving the two splits, we get "aa" + "dbbc" + "bc" + "a" + "c" = "aadbbcbcac".
Since s3 can be obtained by interleaving s1 and s2, we return true.
```

Example 2:

```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"
Output: false
Explanation: Notice how it is impossible to interleave s2 with any other string to obtain
s3.
```

109

Memoization?

110