Dynamic Programming (2)

Desain & Analisis Algoritma

Fakultas Ilmu Komputer

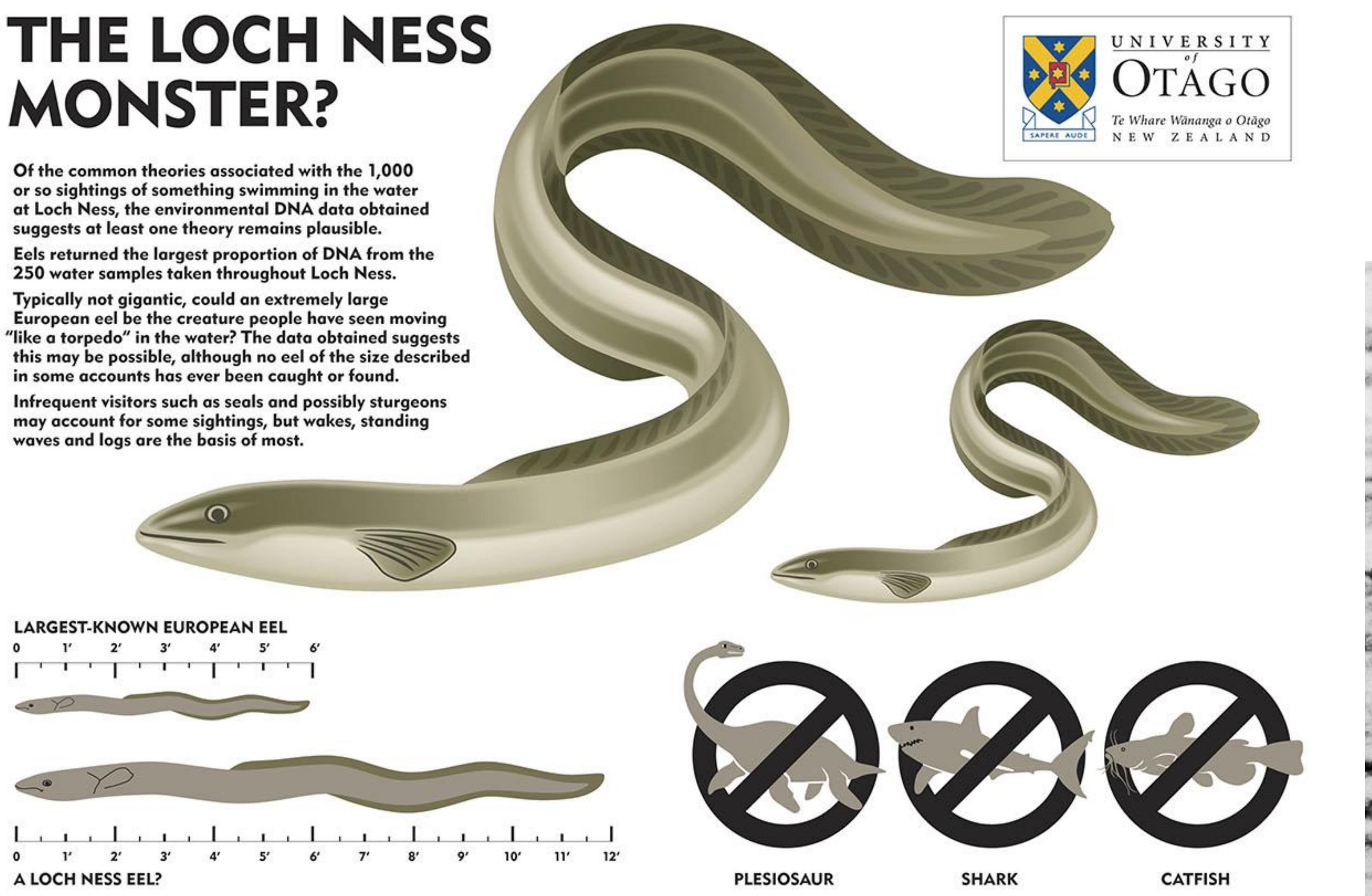
Universitas Indonesia

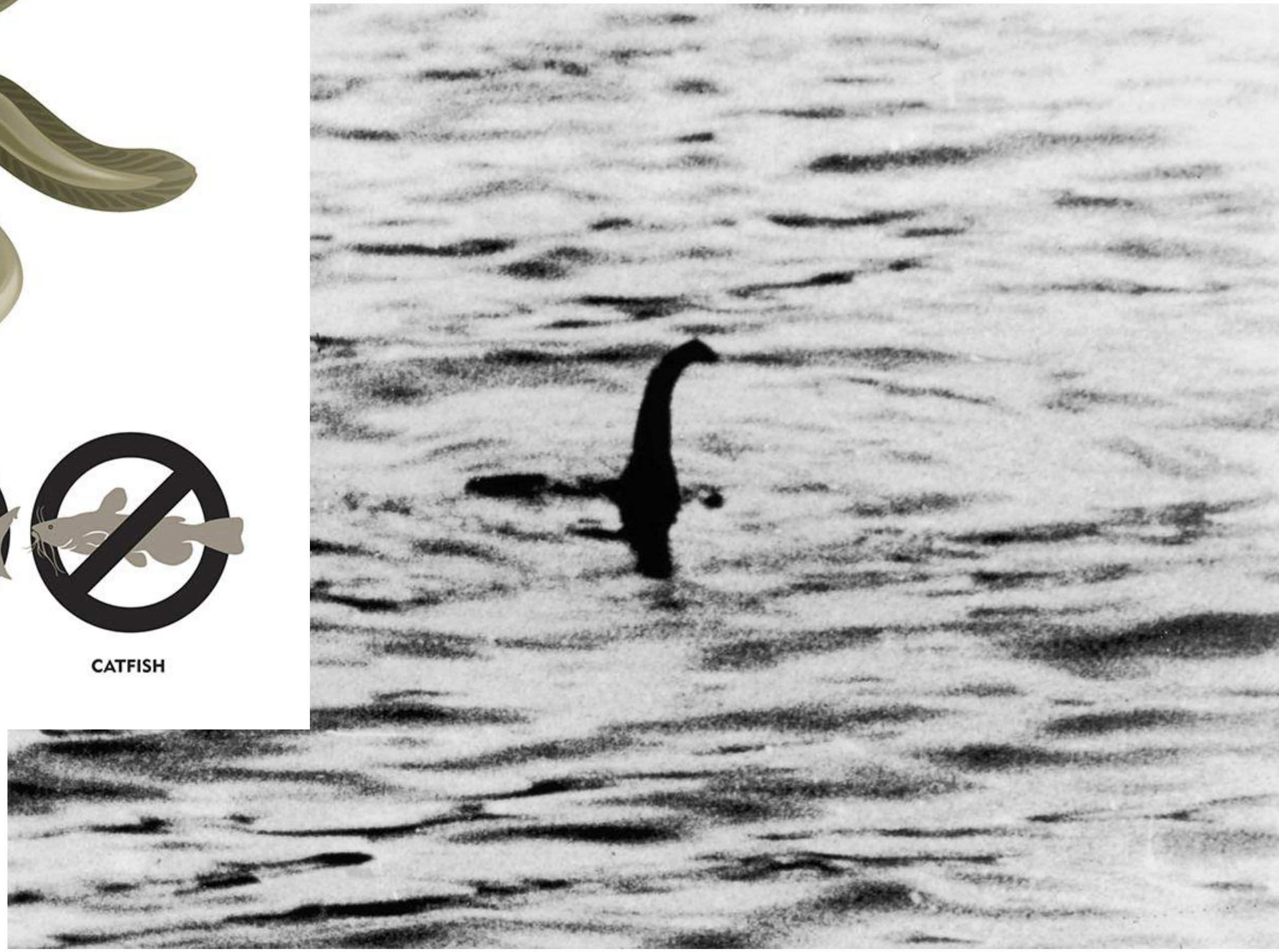
Compiled by **Alfan F. Wicaksono** from multiple sources

Credits

- Introduction to Algorithms, **Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein**
- Dynamic Programming: Weighted Interval Scheduling, CMSC 451: Lecture 10, by **Dave Mount**

Longest Common Subsequence


THE LOCH NESS MONSTER?
Of the common theories associated with the 1,000 or so sightings of something swimming in the water at Loch Ness, the environmental DNA data obtained suggests at least one theory remains plausible.
Eels returned the largest proportion of DNA from the 250 water samples taken throughout Loch Ness.
Typically not gigantic, could an extremely large European eel be the creature people have seen moving "like a torpedo" in the water? The data obtained suggests this may be possible, although no eel of the size described in some accounts has ever been caught or found.
Infrequent visitors such as seals and possibly sturgeons may account for some sightings, but wakes, standing waves and logs are the basis of most.
UNIVERSITY of OTAGO
Te Whare Wānanga o Otāgo
NEW ZEALAND
SAPERE AUDE
LARGEST-KNOWN EUROPEAN EEL
0 1' 2' 3' 4' 5' 6'
0 1' 2' 3' 4' 5' 6' 7' 8' 9' 10' 11' 12'
A LOCH NESS EEL?
PLESIOSAUR
SHARK
CATFISH



- Biological applications often need to compare the DNA of two (or more) different organisms.
- A strand of DNA consists of a string of molecules called **bases**, where the possible bases are adenine, cytosine, guanine, and thymine. Representing each of these bases by its initial letter, we can express a strand of DNA as a string over the 4-element set **{A, C, G, T}**.

- For example, the DNA of one organism may be

$$S_1 = \text{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$$

- and the DNA of another organism may be

$$S_2 = \text{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$$

- One reason to compare two strands of DNA is to measure of how closely related the **two organisms** are.

- A way to measure the similarity of strands S_1 and S_2 is by finding a third strand S_3 in which the **bases** in S_3 appear in each of S_1 and S_2.
- These bases must **appear in the same order**, but **not necessarily consecutively**. The **longer** the strand S_3, the **more similar** S_1 and S_2 are.

- We call this notion as "**The longest common subsequence**". In our example,

- S_1 = **ACCGGTCGAGTGCGCGGAAGCCGGCCGAA**
- S_2 = **GTCGTTCGGAATGCCGTTGCTCTGTAAA**

- The longest common subsequence, or S_3, is:

  GTCGTCGGAAGCCGGCCGAA

- Given two sequences X and Y, a sequence Z is a **common subsequence** of X and Y if Z is a subsequence of both X and Y.

- If X = **$\langle A, B, C, B, D, A, B \rangle$** and Y = **$\langle B, D, C, A, B, A \rangle$**, then **$\langle B, C, A \rangle$** is a common subsequence of both X and Y that has length **three**.

<u>The longest-common-subsequence (LCS) problem:</u>

Given two sequence X and Y, find a **maximum-length** common subsequence of X and Y.

- **Brute-Force Approach**: enumerate all subsequences of X and check each subsequence to see if it is also a subsequence of Y, while keeping track of the longest subsequence found.

- There are **2^n** subsequences of X with **n** items. So the brute-force solution is **exponential** in the number of items in X.
- LCS can be efficiently solved using Dynamic Programming.

Notation:

Given a sequence $X = < x_1, x_2, \dots, x_m >$, we define the ***i*-th prefix** of X, for $i = 0,1,2 \dots, m$, as $\boldsymbol{pref(X, m)} = < x_1, x_2, \dots, x_m >$.

For example, if $X = < A, B, A, C, D, B >$, then $\boldsymbol{pref(X, 4)} = < A, B, A, C >$ and $\boldsymbol{pref(X, 0)} = <>$.

Step 1: a theorem (Optimal Substructure of an LCS)

Let $X =< x_1, x_2, \dots, x_m >$ and $Y =< y_1, y_2, \dots, y_n >$ be sequences, and let $Z =< z_1, z_2, \dots, z_k >$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_{\mathrm{k}} = x_m = y_n$ and $\boldsymbol{pref(Z, k-1)}$ is an LCS of $\boldsymbol{pref(X, m-1)}$ and $\boldsymbol{pref(Y, n-1)}$.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then $\boldsymbol{Z}$ and is an LCS of $\boldsymbol{pref(X, m-1)}$ and $\boldsymbol{Y}$.
3. If $x_m \neq y_n$ and $z_k \neq y_n$, then $\boldsymbol{Z}$ and is an LCS of $\boldsymbol{X}$ and $\boldsymbol{pref(Y, n-1)}$.

In general, an LCS of two sequences contains within it an LCS of prefixes of the two sequences.

Exercise:

Use “proof by contradiction” to show the truth of the optimal substructure of LCS!

Step 2: a recursive solution for LCS

Suppose $\boldsymbol{c(i,j)}$ be the **length** of an LCS of the sequences $\boldsymbol{pref(X,i)}$ and $\boldsymbol{pref(Y,i)}$.

$$c(i,j) = \begin{cases} 0 & i = 0 \text{ } or \text{ } j = 0 \\ c(i-1, j-1) + 1 & i,j > 0 \text{ } and \text{ } x_i = y_j \\ \max\{\boldsymbol{c[i, j-1]}, \quad \boldsymbol{c[i-1, j]}\} & i,j > 0 \text{ } and \text{ } x_i \neq y_j \end{cases}$$

If want to find an LCS between $X =< x_1, x_2, \dots, x_m >$ and $Y =< x_1, x_2, \dots, x_n >$, then call $\boldsymbol{c(m,n)}$

Step 2: a recursive solution for LCS

Suppose $\boldsymbol{c(i,j)}$ be the **length** of an LCS of the sequences $\boldsymbol{pref(X,i)}$ and $\boldsymbol{pref(Y,i)}$.

If $x_i = y_j$ then you need to find an LCS of $pref(X, i-1)$ and $pref(Y, j-1)$. Appending $x_i = y_j$ yields an LCS of $pref(X, i)$ and $pref(Y, j)$

$$c(i,j) = \begin{cases} 0 & i = 0 \ or \ j = 0 \\ \boxed{c(i-1, j-1) + 1 \qquad i, j > 0 \ and \ x_i = y_j} & \\ \max\{\boldsymbol{c[i, j-1]}, \quad \boldsymbol{c[i-1, j]}\} & i, j > 0 \ and \ x_i \neq y_j \end{cases}$$

If want to find an LCS between $X =< x_1, x_2, \dots, x_m >$ and $Y =< x_1, x_2, \dots, x_n >$, then call $\boldsymbol{c(m, n)}$

Step 2: a recursive solution for LCS

Suppose $\boldsymbol{c(i,j)}$ be the **length** of an LCS of the sequences $\boldsymbol{pref(X,i)}$ and $\boldsymbol{pref(Y,i)}$.

If $x_i \neq y_j$ then you need to solve **two subproblems**! Whichever of these two LCSs is longer is an LCS of of $pref(X,i)$ and $pref(Y,j)$

$$c(i,j) = \begin{cases} 0 & i = 0 \; or \; j = 0 \\ c(i-1,j-1)+1 & i,j > 0 \; and \; x_i = y_j \\ \max\{\boldsymbol{c[i,j-1]}, \quad \boldsymbol{c[i-1,j]}\} & i,j > 0 \; and \; x_i \neq y_j \end{cases}$$

If want to find an LCS between $X =< x_1, x_2, \dots, x_m >$ and $Y =< x_1, x_2, \dots, x_n >$, then call $\boldsymbol{c(m,n)}$

Step 2: a recursive solution has the overlapping-subproblems

To find an LCS of X and Y, you might need to find the LCSs of X and pref(Y,n-1) and of pref(X,m-1) and Y.

Each of these subproblems has the subsubproblem of finding an LCS of pref(X,m-1) and pref(Y,n-1).

$$c(i,j) = \begin{cases} 0 & i = 0 \ or \ j = 0 \\ c(i-1, j-1) + 1 & i,j > 0 \ and \ x_i = y_j \\ \boxed{\max\{\boldsymbol{c[i, j-1]}, \quad \boldsymbol{c[i-1, j]}\} \quad i,j > 0 \ and \ x_i \neq y_j} \end{cases}$$

If want to find an LCS between $X =< x_1, x_2, \dots, x_m >$ and $Y =< x_1, x_2, \dots, x_n >$, then call $\boldsymbol{c(m, n)}$

Step 3: a DP solution (bottom-up version)

There are only $\boldsymbol{\Theta(mn)}$ distinct subproblems.

- The following LCS-LENGTH procedure takes two sequences $X = < x_1, x_2, \dots, x_m >$ and $Y =< y_1, y_2, \dots, y_n >$ as inputs, along with their lengths.
- It stores $\boldsymbol{c(i,j)}$ values the in a table $\boldsymbol{c[0{:}\,m, 0{:}\,n]}$ whose entries are computed in **row-major** order.
- The procedure also maintains the table $\boldsymbol{b[1{:}\,m, 1{:}\,n]}$ to help in constructing an optimal solution **(step 4)**.

Step 3: a DP solution (bottom-up version)

```
LCS-LENGTH(X, Y, m, n):
  let b[1:m, 1:n] and c[0:m, 0:n] be new matrix
  for i = 1 to m:
     c[i,0] = 0
  for j = 0 to n:
     c[0,j] = 0
  for i = 1 to m:                    //iterate the matrix in row-major order
     for j = 1 to n:
        if X_i = Y_j then
           c[i,j] = c[i-1, j-1] + 1
           b[i,j] = "↖"
        else if c[i-1,j] ≥ c[i,j-1] then
           c[i,j] = c[i-1, j]
           b[i,j] = "↑"
        else
           c[i,j] = c[i, j-1]
           b[i,j] = "←"
   return c and b
```

The running time is $\Theta(mn)$

Example

Create the table/matrix when computing an LCS of X = <A,B,C,B,D,A,B> and Y = <B,D,C,A,B,A>

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

Step 4: Print a solution using the table **b**

```
PRINT-LCS(b, X, i, j):      // initial call: PRINT-LCS(n,X,m,n)
   if i == 0 or j == 0 then
       return []
   if b[i,j] == "↖" then
       return PRINT-LCS(b, X, i-1, j-1) + [x_i]     // same as + [y_j]
   else if b[i,j] == "↑" then
       return PRINT-LCS(b, X, i-1, j)
   else
       return PRINT-LCS(b, X, i, j-1)
```

The running time is $\Theta(m + n)$

Rod Cutting Problem

The summary so far

- Weighted Interval Scheduling, Knapsack, LCS
  - There are n subproblems
  - **Two cases: e.g., include j or don't include j**
- **Rod Cutting Problem (what we're going to see)**
  - There are n subproblems
  - **Many cases …**

Rod Cutting

Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \dots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

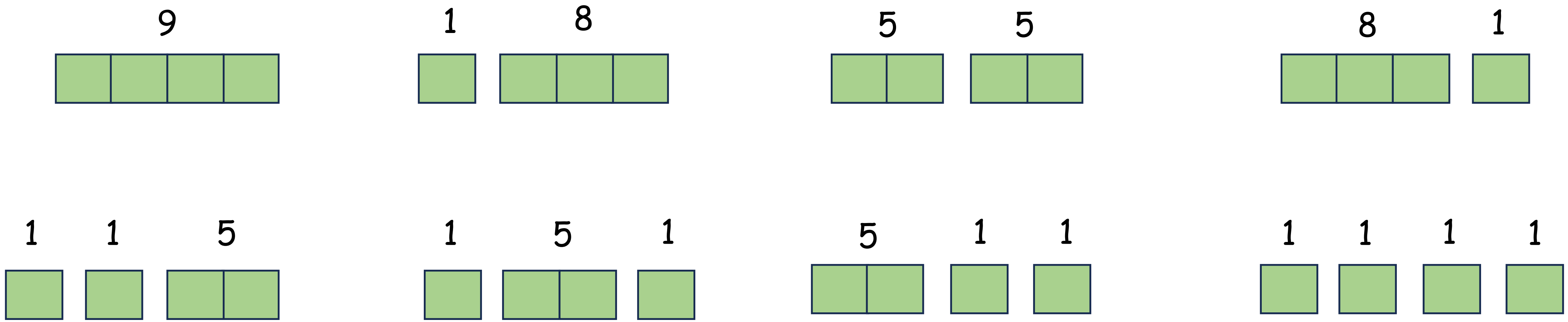
If the price p_n for a rod of length n is large enough, an optimal solution might require no cutting at all.

Rod Cutting

Consider the case when $\boldsymbol{n} = \mathbf{4}$, and the following price table for rods:

Length i	1	2	3	4	5	6	7	8	9	10
Price p_i	1	5	8	9	10	17	17	20	24	30

There are 8 possible ways of cutting up a rod of length 4:



We can determine the optimal revenue r_i for $i = 1, \ldots, 10$ by inspection:

$r_1 = 1$ from solution $r_1 = p_1$ (no cuts)
$r_2 = 5$ from solution $r_2 = p_2$ (no cuts)
$r_3 = 8$ from solution $r_3 = p_3$ (no cuts)
$r_4 = 10$ from solution $r_4 = r_2 + r_2$
$r_5 = 13$ from solution $r_5 = r_2 + r_3$
$r_6 = 17$ from solution $r_6 = p_6$ (no cuts)
$r_7 = 18$ from solution $r_7 = r_1 + r_6$ or $r_7 = r_2 + r_2 + r_3$
$r_8 = 22$ from solution $r_8 = r_2 + r_6$
$r_9 = 25$ from solution $r_9 = r_3 + r_6$
$r_{10} = 30$ from solution $r_{10} = p_{10}$ (no cuts)

Rod Cutting

We can determine the optimal revenue r_i for $i = 1, \dots, 10$ by inspection:

$r_1 = 1$ from solution $r_1 = p_1$ (no cuts)
$r_2 = 5$ from solution $r_2 = p_2$ (no cuts)
$r_3 = 8$ from solution $r_3 = p_3$ (no cuts)
$r_4 = 10$ from solution $r_4 = r_2 + r_2$
$r_5 = 13$ from solution $r_5 = r_2 + r_3$
$r_6 = 17$ from solution $r_6 = p_6$ (no cuts)
$r_7 = 18$ from solution $r_7 = r_1 + r_6$ or $r_7 = r_2 + r_2 + r_3$
$r_8 = 22$ from solution $r_8 = r_2 + r_6$
$r_9 = 25$ from solution $r_9 = r_3 + r_6$
$r_{10} = 30$ from solution $r_{10} = p_{10}$ (no cuts)

This exhibits **optimal substructure!**

In General,

$$r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, r_{n-1} + r_1\}$$

$$r_n = \max\{p_i + r_{n-i} : 1 \leq i \leq n\}$$

Rod Cutting

Recursive Implementation

$$r_n = \max\{p_i + r_{n-i} : 1 \le i \le n\}$$

CUT-ROD(p,n) = max{p_i + CUT-ROD(p, n-1): 1 <= i <= n}

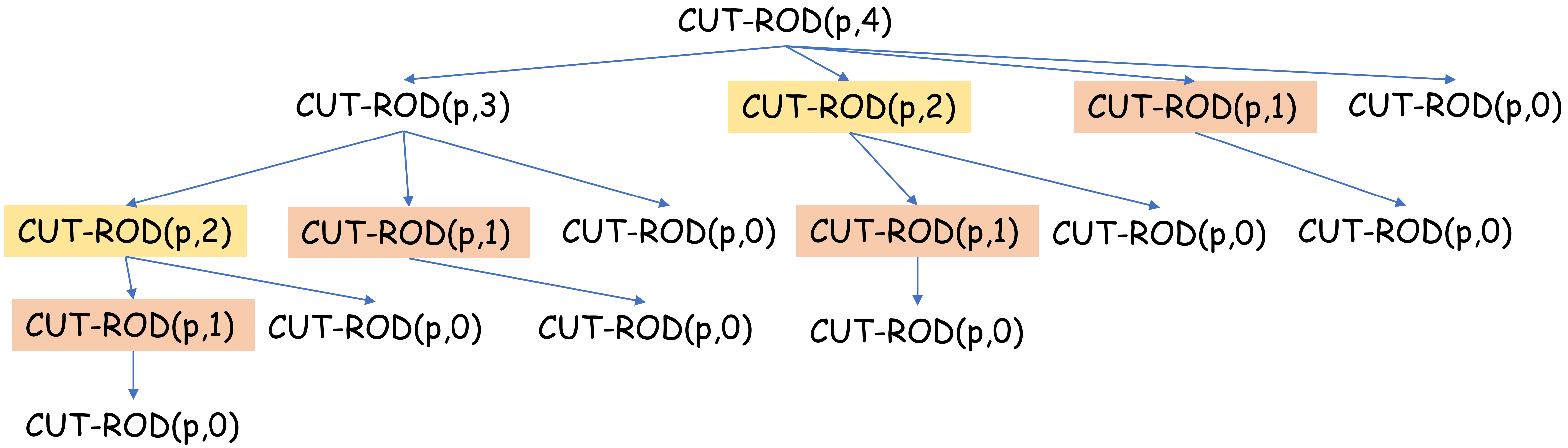
```
CUT-ROD(p, n):
    if n == 0 then
        return 0
    q = - INF
    for i = 1 to n:
        q = max {q, p[i] + CUT-ROD(p, n - i)}
    return q
```

Running Time:

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) = 2^n$$

Rod Cutting

Overlapping Subproblems? Let's inspect the recursion tree:



DP solution (bottom-up)

A subproblem of size *i* is "smaller" than a subproblem of size *j* if *i* < *j*. Thus, the procedure solves subproblems of sizes j = 0 … n in that order.

```
CUT-ROD(p, n):
   let r[0:n] be a new array      // to save the results of subproblems
   let s[1:n] be a new array      // the optimal size of the first piece to cut off (for reconstruction)
   r[0] = 0
   for j = 1 to n:
      q = -INF
      for i = 1 to j:
         temp = p[i] + r[j - i]
         if q < temp then
            q = temp
            s[j] = i
      r[j] = q
   return r and s
```

Exercise:

Running time = $\Theta(?)$

Reconstructing a solution

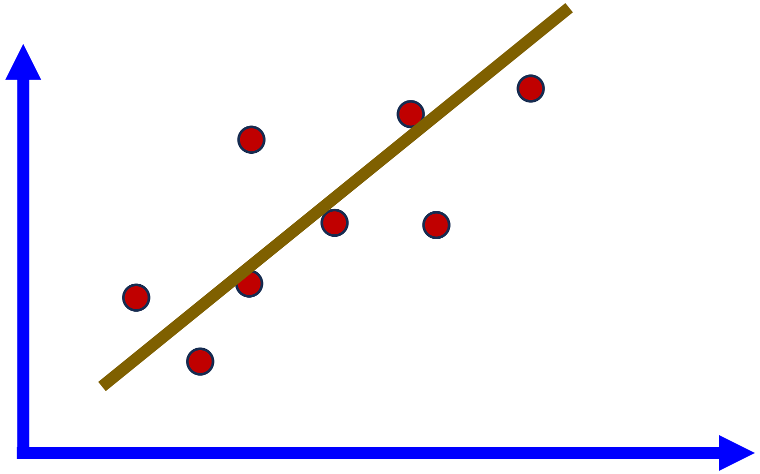
```
PRINT-CUT-ROD-SOLUTION(s, n):
    solution = []
    while n > 0:
        solution = solution + [s[n]]
        n = n - s[n]
    return solution
```

Segmented Least Squares

Ordinary Least Squares (OLS)

- It's a foundational problem in statistics and numerical analysis.
- Given **n** points in the plane: **(x_1, y_1), (x_2, y_2), …, (x_n, y_n)**
- Find a line **y = ax + b** that minimizes the sum of the squared errors:

$$= \sum_{i=1}^{n} (y_i - ax_i - b)^2$$



Least Squares Solution

Use your calculus knowledge and least squares are achieved when:

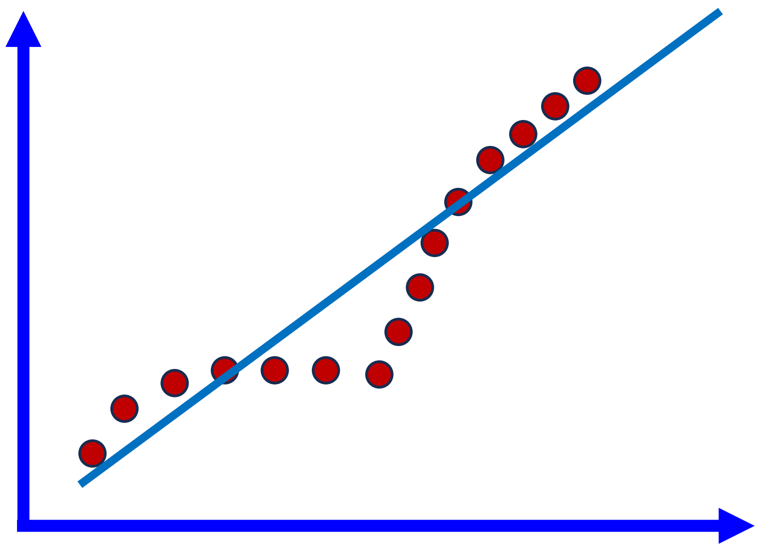
$$a = \frac{n \sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$b = \frac{\sum_{i=1}^{n} y_i - a \sum_{i=1}^{n} x_i}{n}$$

If you encapsulate this formula as a subroutine, what is the **running time?**

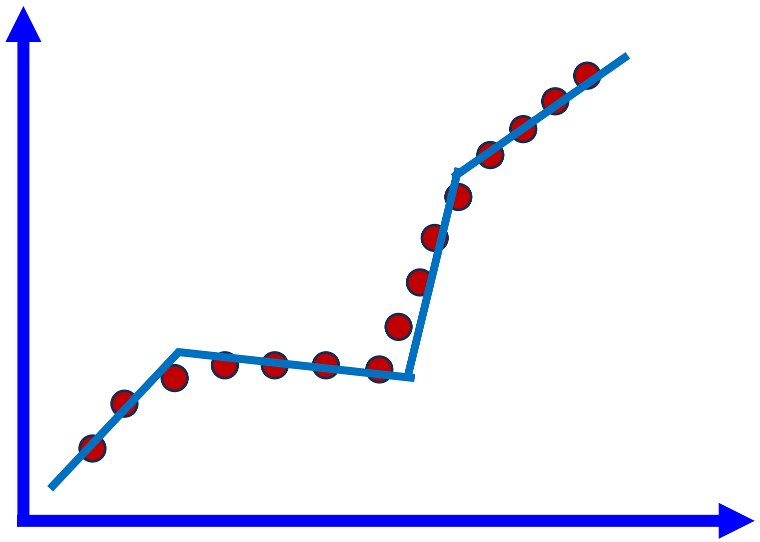
Least Squares

We sometimes think that a single line is just not enough.



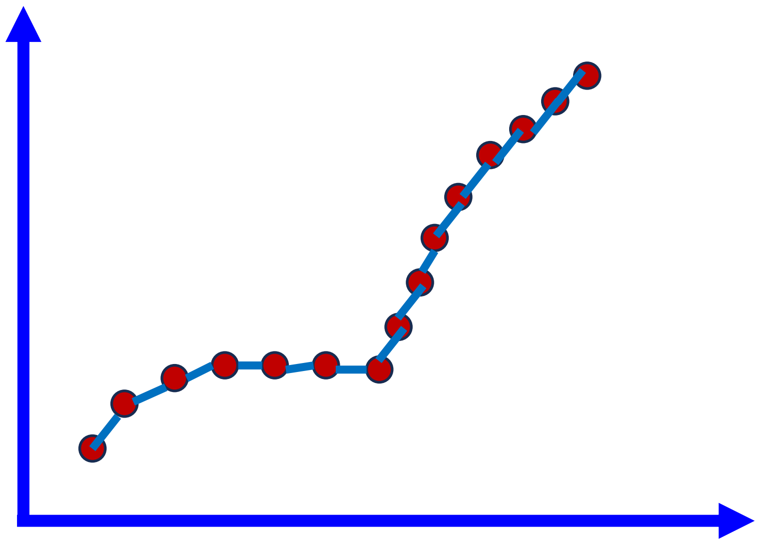
Segmented Least Squares

Given n points in the plane: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find **<u>sequence of lines</u>** that fits well.



Segmented Least Squares

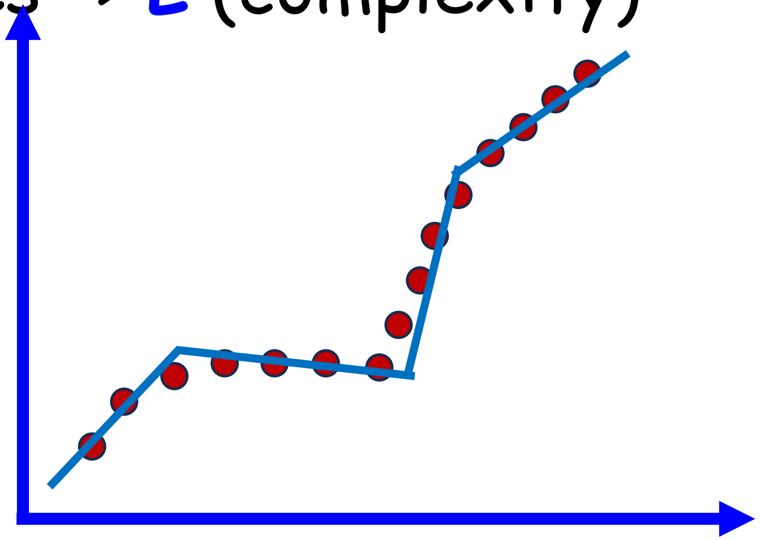
Too many lines: perfect solution, but too complex (prone to overfitting)



Segmented Least Squares

Goal: Find a sequence to minimize some combination of

- The total error from each segment -> **err** (fitness)
- The number of lines -> **L** (complexity)



Trade-off function:

cost = err + cL,

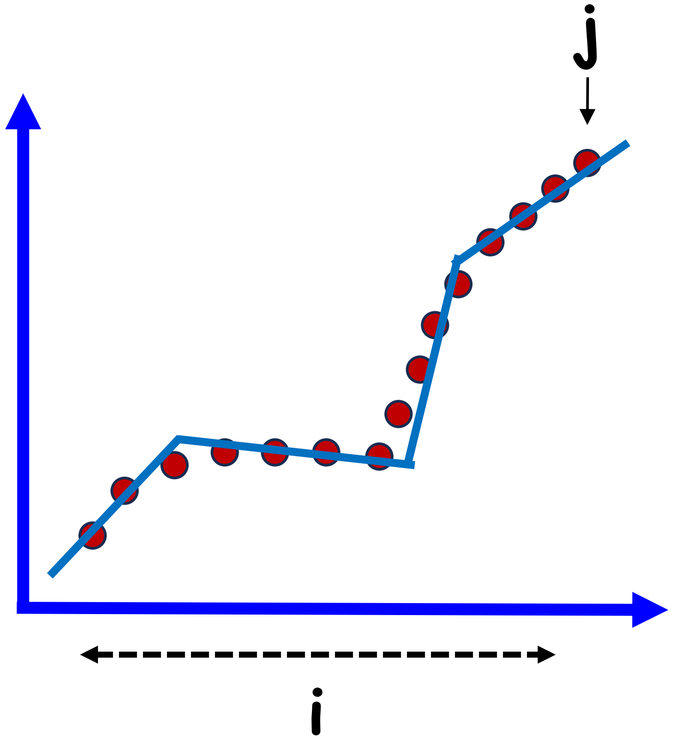
For some constant **c > 0**

Segmented Least Squares

- **OPT(j)** = minimum cost for points p_1, p_2, …, p_j
- **e(i,j)** = minimum sum of squared errors for points p_i, p_{i+1}, …, p_j

$$\boldsymbol{Cost = e(i,j) + c + OPT(i-1)}$$

$$OPT(j) = \begin{cases} 0 & j = 0 \\ \min_{1 \le i \le j} \{e(i,j) + c + OPT(i-1)\} & j > 0 \end{cases}$$



```
Segmented-LS(n, p_1, ..., p_n, c):
    let M[0:n] be a new array
    let e[1:n, 1:n] be a new matrix

    M[0] = 0
    for j = 1 to n:
        for i = 1 to j:
            compute least square error e[i,j] for the segment p_i, ..., p_j

    for j = 1 to n:
        M[j] = INF
        for i = 1 to j:
            temp = e[i,j] + c + M[i-1]
            if temp < M[j] then
                M[j] = temp
```

What is the running time?