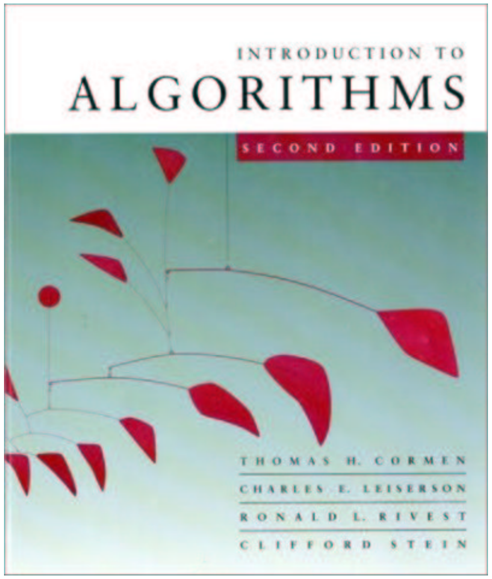
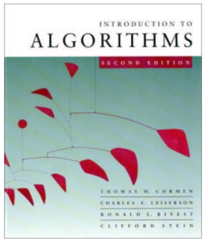
# *Introduction to Algorithms*

## 6.046J/18.401J/SMA5503



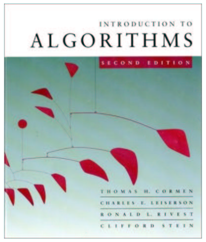

## *Lecture 3*

**Prof. Piotr Indyk**




# The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.




# Example: merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = 2\,T(n/2) + O(n)$$

*# subproblems*

*subproblem size*

*work dividing and combining*

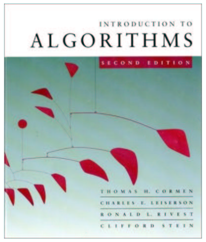
# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 8 | 9 | 12 | 15 |



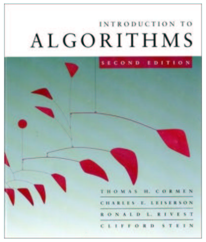

# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 8 | 9 | 12 | 15 |



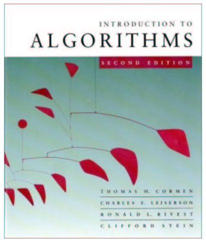

# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 9 12 15



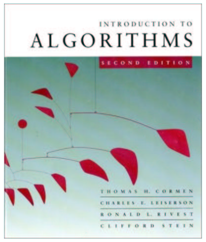

# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 9 12 15



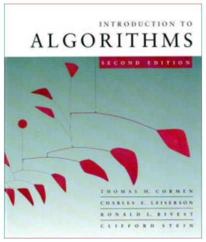

# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 9 12 15



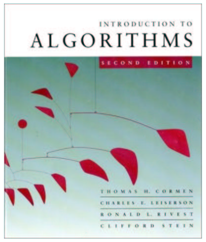

# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 9 12 15




# Recurrence for binary search

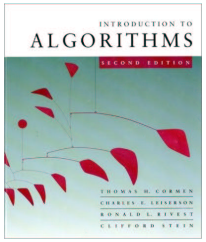
$$T(n) = 1\,T(n/2) + \Theta(1)$$

*# subproblems*

*subproblem size*

*work dividing and combining*

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$
$$\Rightarrow T(n) = \Theta(\lg n)\,.$$




# Powering a number

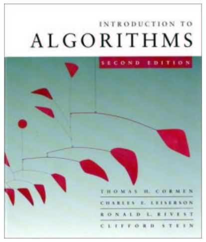
**Problem:** Compute $a^n$, where $n \in \mathbb{N}$.

**Naive algorithm:** $\Theta(n)$.

**Divide-and-conquer algorithm:**

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n) \ .$$

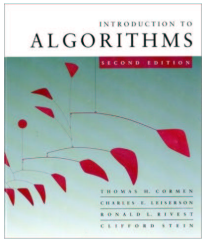



# Polynomial multiplication

**Input:** $a(x)=a_0+a_1x+\ldots+a_nx^n,$
$b(x)=b_0+b_1x+\ldots+b_nx^n,$

**Output:** $c(x)=a(x)*b(x)=c_0+c_1x+\ldots+c_{2n}x^{2n}$
$c_i=a_0b_i+a_1b_{i-1}+\ldots+a_{i-1}b_1+a_ib_0$

***Example:*** $(a_0+a_1x)*(b_0+b_1x)=$
$a_0b_0 \quad + (a_0b_1+a_1b_0)x \quad + a_1b_1x^2 =$
$c_0 \quad + c_1x \quad + c_2x^2$



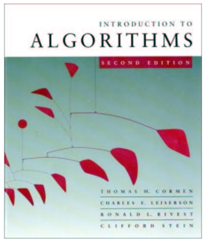

# Motivation (more in recitations)

- Essentially equivalent to multiplying large integers:

  $6046*6001 =$
  $(6*10^0 + 4*10^1 + 0*10^2 + 6*10^3) *$
  $(1*10^0 + 0*10^1 + 0*10^2 + 6*10^3) =$
  $a(10) * b(10) = c(10)$, where $c(x)=a(x)*b(x)$

  $c(10) = c_0 10^0 + c_1 10^1 + \ldots + c_6 10^6$

- The coefficients of $c$ form the "digits" of the product $c(10)$

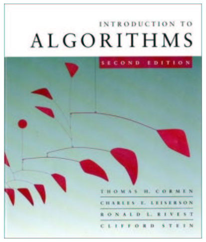



# How to multiply two polynomials

- From the definition: $\Theta(n^2)$ time
- Faster ? Use divide and conquer
  - Divide:

$$a(x) = a_0 + a_1 x + \ldots + a_n x^n =$$
$$(a_0 + \ldots + a_{n/2} x^{n/2}) + x^{n/2} (a_{n/2} x^0 + \ldots + a_n x^{n/2}) =$$
$$p(x) + x^{n/2} q(x) =$$
$$p + x^{n/2} q$$

  - In the same way: $b(x) = s + x^{n/2} t$




# Conquer

- Observe that:

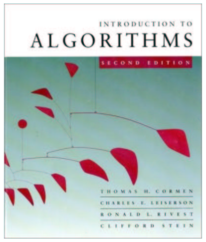
  $a*b =$

  $(p+x^{n/2}q) * (s+x^{n/2}t) =$

  $p*s + x^{n/2}(p*t+q*s) + x^n q*t$

- But $p,q,s,t$ have degree $n/2$

  $\Rightarrow$ can compute the products recursively!
  (and then perform $\Theta(n)$ additions)




# The great moment…

$$T(n) = 4\,T(n/2) + \Theta(n)$$

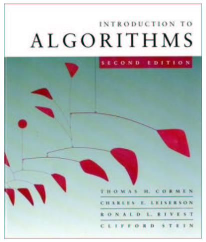
*# subproblems*

*subproblem size*

*work dividing and combining*

$$n^{\log_b a} = n^{\log_2 4} = n^2 \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^2).$$

***No better than the ordinary algorithm ???***




# Need to be more clever

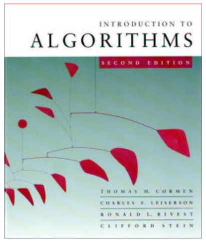
- Compute:

  $p*s$

  $q*t$

  $(p+q) * (s+t) = p*s + (p*t + q*s) + q*t$

  (all polynomials have degree $n/2$ )
- Can extract $(p*t + q*s)$ without any additional multiplications !




# The <u>truly</u> great moment

$$T(n) = 3T(n/2) + \Theta(n)$$

*# subproblems*

*subproblem size*

*work adding and subtracting*

$$n^{\log_b a} = n^{\log_2 3} = n^{1.58496\ldots}$$

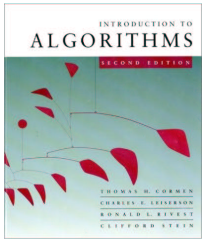
***Much better than*** $\Theta(n^2)$ ***!***

# Matrix multiplication

$$\left.\begin{array}{ll}\textbf{Input:} & A = [a_{ij}],\ B = [b_{ij}]. \\ \textbf{Output:} & C = [c_{ij}] = A \cdot B.\end{array}\right\}\ i, j = 1, 2, \ldots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$




# Standard algorithm

**for** $i \leftarrow 1$ **to** $n$

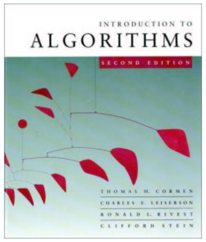
    **do for** $j \leftarrow 1$ **to** $n$

        **do** $c_{ij} \leftarrow 0$

            **for** $k \leftarrow 1$ **to** $n$

                **do** $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time $= \Theta(n^3)$




# Divide-and-conquer algorithm

**IDEA:**

$n \times n$ matrix = $2 \times 2$ matrix of $(n/2) \times (n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

8 mults of $(n/2) \times (n/2)$ submatrices
4 adds of $(n/2) \times (n/2)$ submatrices

# Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

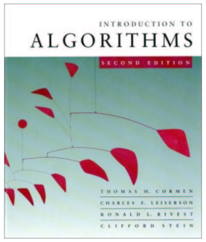
*# submatrices*

*submatrix size*

*work adding submatrices*

$$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^3).$$

***No better than the ordinary algorithm.***



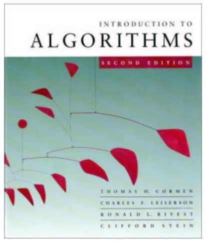

# Strassen’s idea (1969)

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs.
**Note:** No reliance on commutativity of mult!

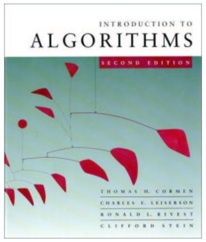



# Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

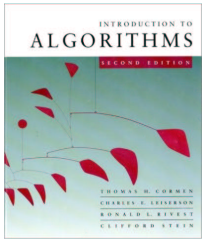
$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$




# Strassen's algorithm

1. ***Divide:*** Partition $A$ and $B$ into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$ .
2. ***Conquer:*** Perform $7$ multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form $C$ using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$



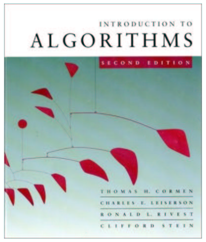

# Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^{\lg 7}).$$

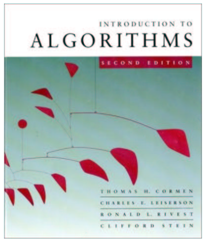
The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 30$ or so.

**Best to date** (of theoretical interest only): $\Theta(n^{2.376})$.



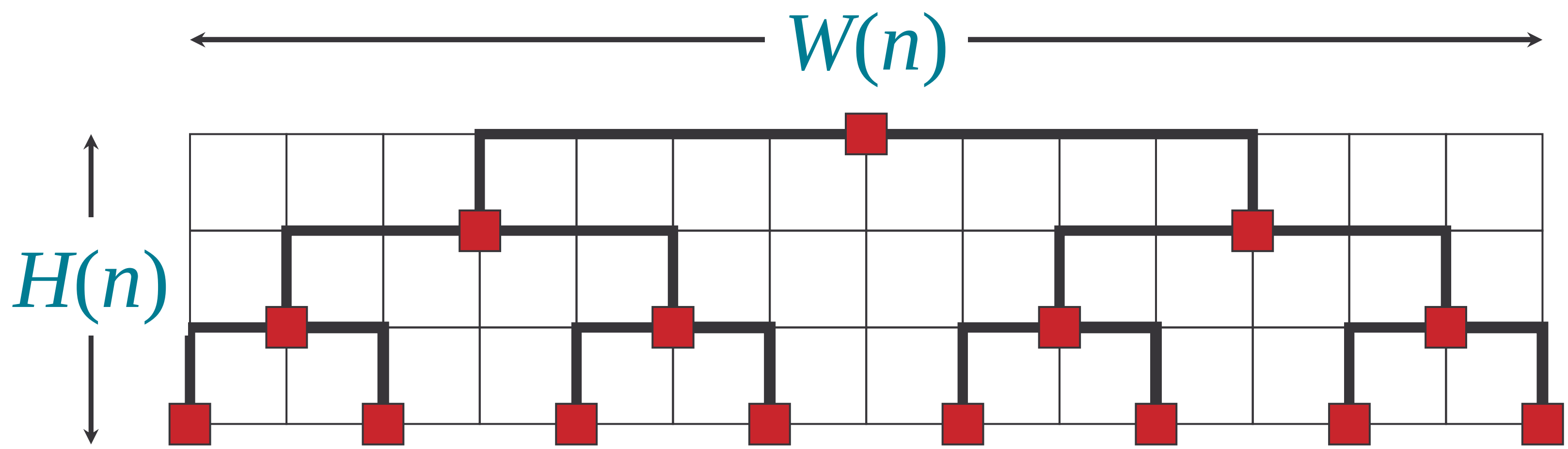

# Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method (so practice this math).
- Can lead to more efficient algorithms




# VLSI layout

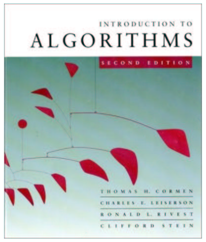
**Problem:** Embed a complete binary tree with $n$ leaves in a grid using minimal area.



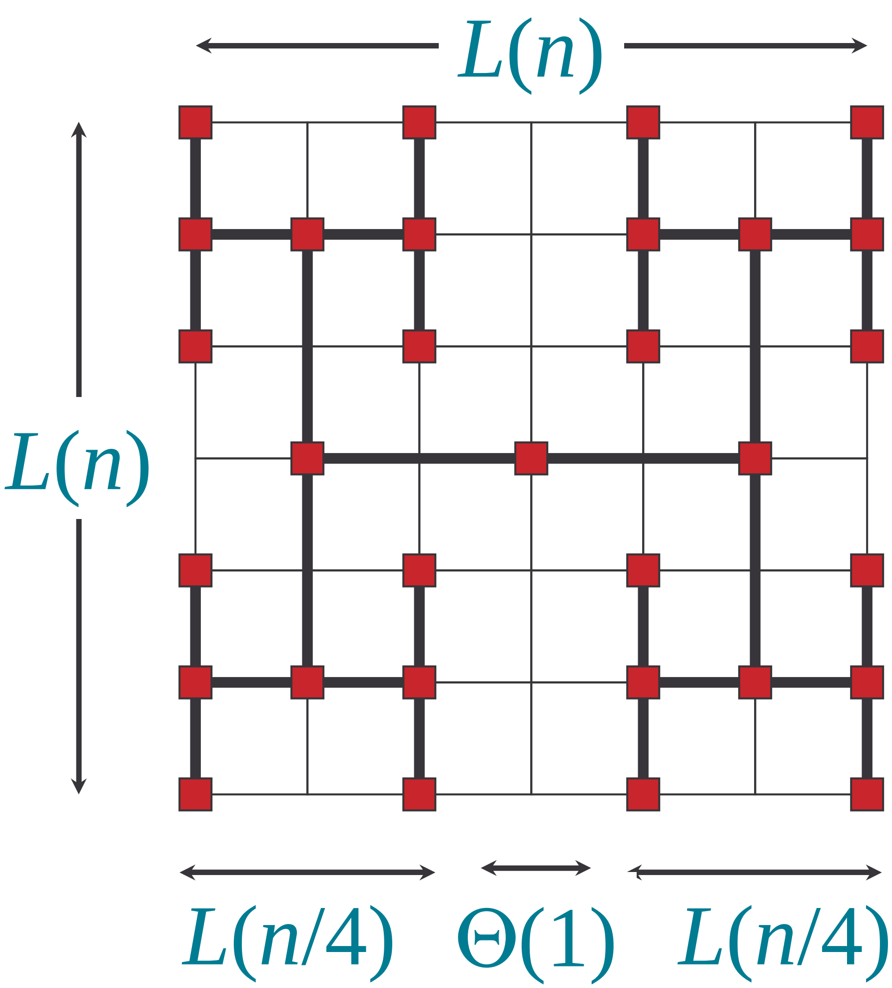

$$H(n) = H(n/2) + \Theta(1) = \Theta(\lg n)$$

$$W(n) = 2W(n/2) + \Theta(1) = \Theta(n)$$

$$\text{Area} = \Theta(n \lg n)$$

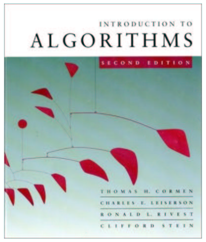



# H-tree embedding




$$
\begin{aligned}
L(n) &= 2L(n/4) + \Theta(1) \\
&= \Theta(\sqrt{n})
\end{aligned}
$$

Area $= \Theta(n)$




# Master theorem (reprise)

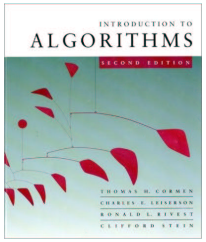
$$T(n) = a\,T(n/b) + f(n)$$

**CASE 1:** $f(n) = O(n^{\log_b a - \varepsilon})$
$\Rightarrow T(n) = \Theta(n^{\log_b a})$ .

**CASE 2:** $f(n) = \Theta(n^{\log_b a} \lg^k n)$
$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .

**CASE 3:** $f(n) = \Omega(n^{\log_b a + \varepsilon})$ and $a\,f(n/b) \le c\,f(n)$
$\Rightarrow T(n) = \Theta(f(n))$ .

***Merge sort:*** $a = 2,\ b = 2 \ \Rightarrow\ n^{\log_b a} = n$
$\Rightarrow$ CASE 2 $(k = 0)$ $\Rightarrow\ T(n) = \Theta(n \lg n)$ .



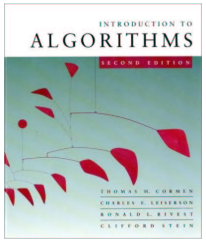

# Fibonacci numbers

**Recursive definition:**

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

$$0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad 34 \quad \cdots$$

**Naive recursive algorithm:** $\Omega(\phi^n)$ (exponential time), where $\phi = (1+\sqrt{5})/2$ is the ***golden ratio***.




# Computing Fibonacci numbers

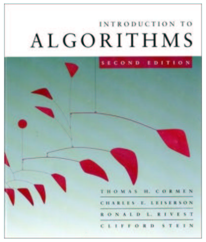
**Naive recursive squaring:**

$F_n = \phi^n / \sqrt{5}$ rounded to the nearest integer.

- Recursive squaring: $\Theta(\lg n)$ time.
- This method is unreliable, since floating-point arithmetic is prone to round-off errors.

**Bottom-up:**

- Compute $F_0, F_1, F_2, \ldots, F_n$ in order, forming each number by summing the two previous.
- Running time: $\Theta(n)$.




# Recursive squaring

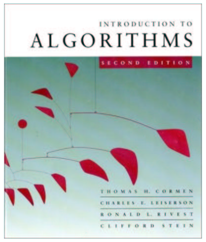
**Theorem:** $$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n .$$

**Algorithm:** Recursive squaring.

Time = $\Theta(\lg n)$ .

*Proof of theorem.* (Induction on $n$.)

Base ($n = 1$): $$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 .$$




# Recursive squaring

Inductive step ($n \geq 2$):

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n} \qquad \blacksquare$$