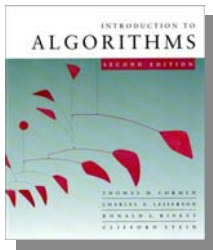


Recursive squaring

Inductive step ($n \geq 2$):

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

■

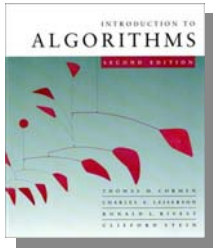


Matrix multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$ } $i, j = 1, 2, \dots, n.$
Output: $C = [c_{ij}] = A \cdot B.$

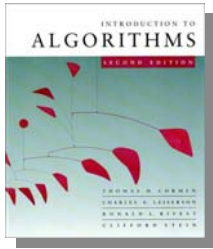
$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots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

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



Standard algorithm

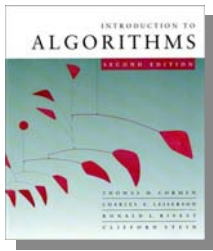
```
for  $i \leftarrow 1$  to  $n$ 
  do for  $j \leftarrow 1$  to  $n$ 
    do  $c_{ij} \leftarrow 0$ 
      for  $k \leftarrow 1$  to  $n$ 
        do  $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$ 
```



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:

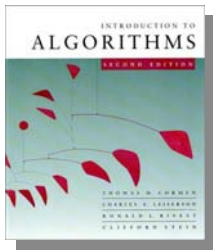
$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

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

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



Divide-and-conquer algorithm

IDEA:

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

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

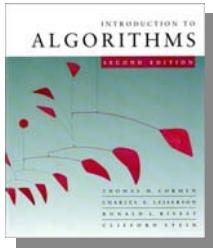
$$C = A \cdot B$$

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

recursive

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

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



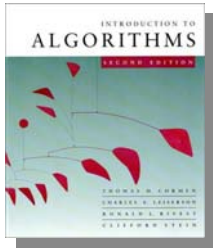
Analysis of D&C algorithm

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

submatrices

submatrix size

work adding
submatrices



Analysis of D&C algorithm

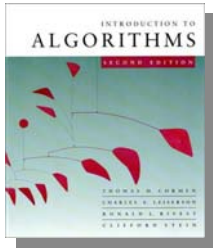
$$T(n) = 8T(n/2) + \Theta(n^2)$$

submatrices

submatrix size

work adding
submatrices

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



Analysis of D&C algorithm

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

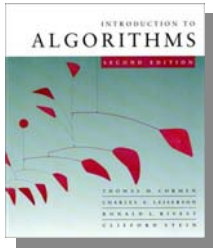
submatrices

submatrix size

work adding
submatrices

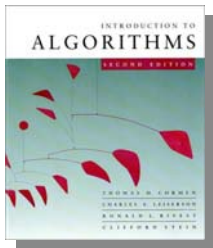
$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.



Strassen's idea

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



Strassen's idea

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

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

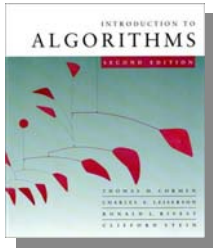
$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$



Strassen's idea

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

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

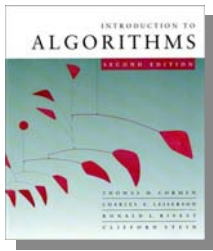
$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$s = P_1 + P_2$$

$$t = P_3 + P_4$$

$$u = P_5 + P_1 - P_3 - P_7$$



Strassen's idea

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

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

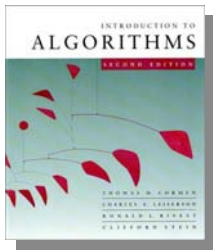
$$s = P_1 + P_2$$

$$t = P_3 + P_4$$

$$u = P_5 + P_1 - P_3 - P_7$$

7 mults, 18 adds/subs.

Note: No reliance on commutativity of mult!



Strassen's idea

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

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$= (a + d)(e + h)$$

$$+ d(g - e) - (a + b)h$$

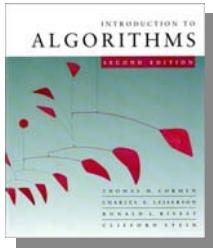
$$+ (b - d)(g + h)$$

$$= ae + ah + de + dh$$

$$+ dg - de - ah - bh$$

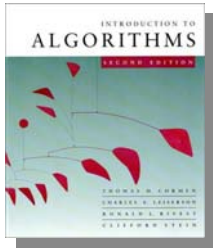
$$+ bg + bh - dg - dh$$

$$= ae + bg$$



Strassen's algorithm

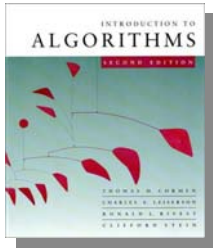
- 1. *Divide:*** Partition A and B into $(n/2) \times (n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
- 2. *Conquer:*** Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.
- 3. *Combine:*** Form C using $+$ and $-$ on $(n/2) \times (n/2)$ submatrices.



Strassen's algorithm

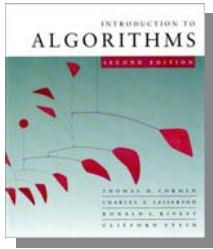
- 1. *Divide:*** Partition A and B into $(n/2) \times (n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
- 2. *Conquer:*** Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.
- 3. *Combine:*** Form C using $+$ and $-$ on $(n/2) \times (n/2)$ submatrices.

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



Analysis of Strassen

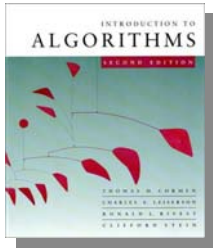
$$T(n) = 7 T(n/2) + \Theta(n^2)$$



Analysis of Strassen

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

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

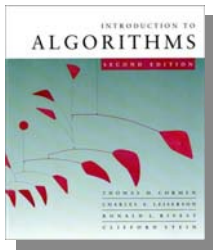


Analysis of Strassen

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

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

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 $n \geq 32$ or so.



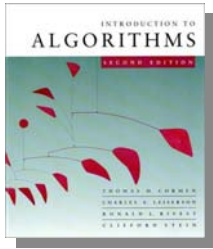
Analysis of Strassen

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

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

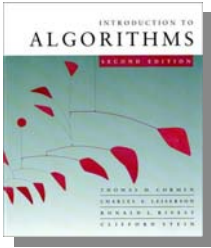
The number **2.81** may not seem much smaller than **3**, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

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



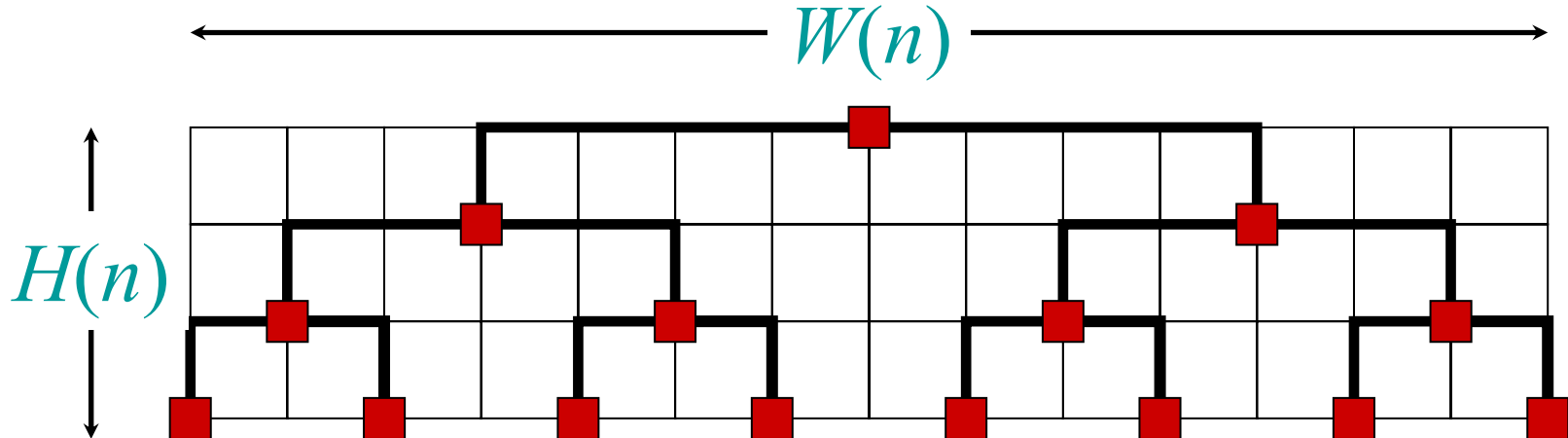
VLSI layout

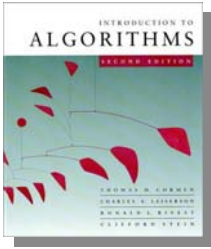
Problem: Embed a complete binary tree with n leaves in a grid using minimal area.



VLSI layout

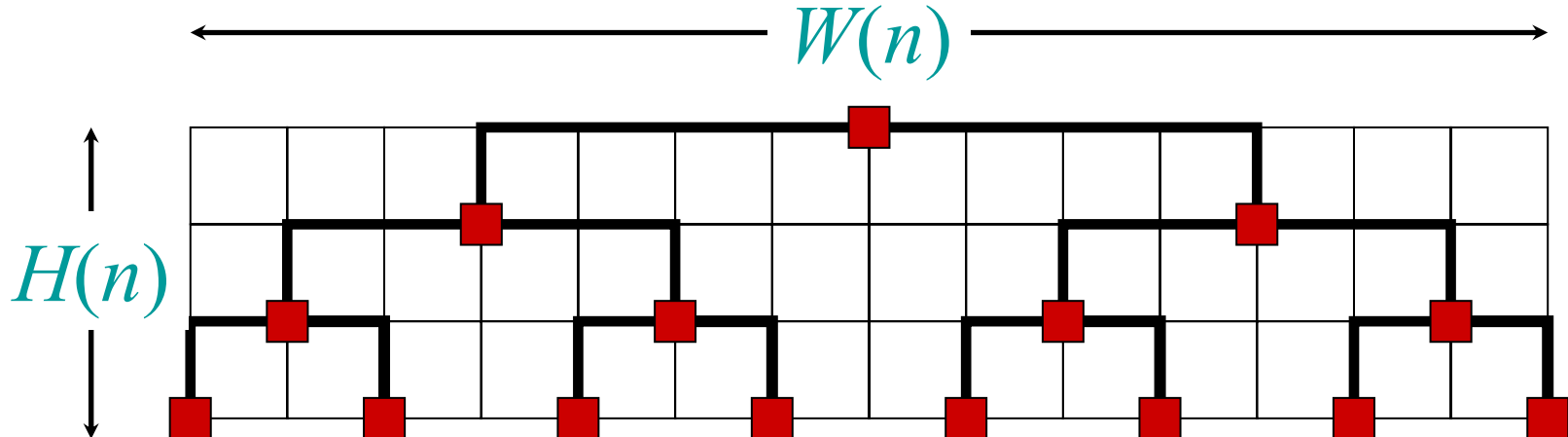
Problem: Embed a complete binary tree with n leaves in a grid using minimal area.



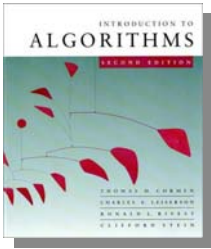


VLSI layout

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

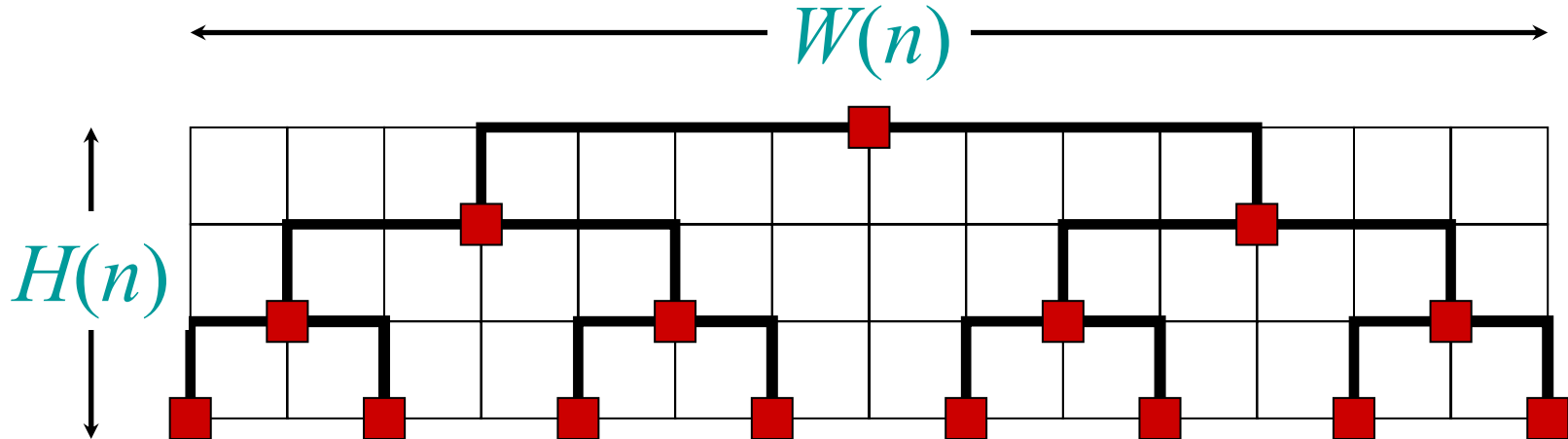


$$\begin{aligned} H(n) &= H(n/2) + \Theta(1) \\ &= \Theta(\lg n) \end{aligned}$$

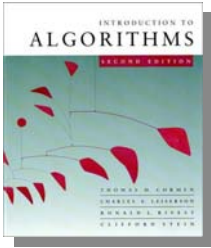


VLSI layout

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

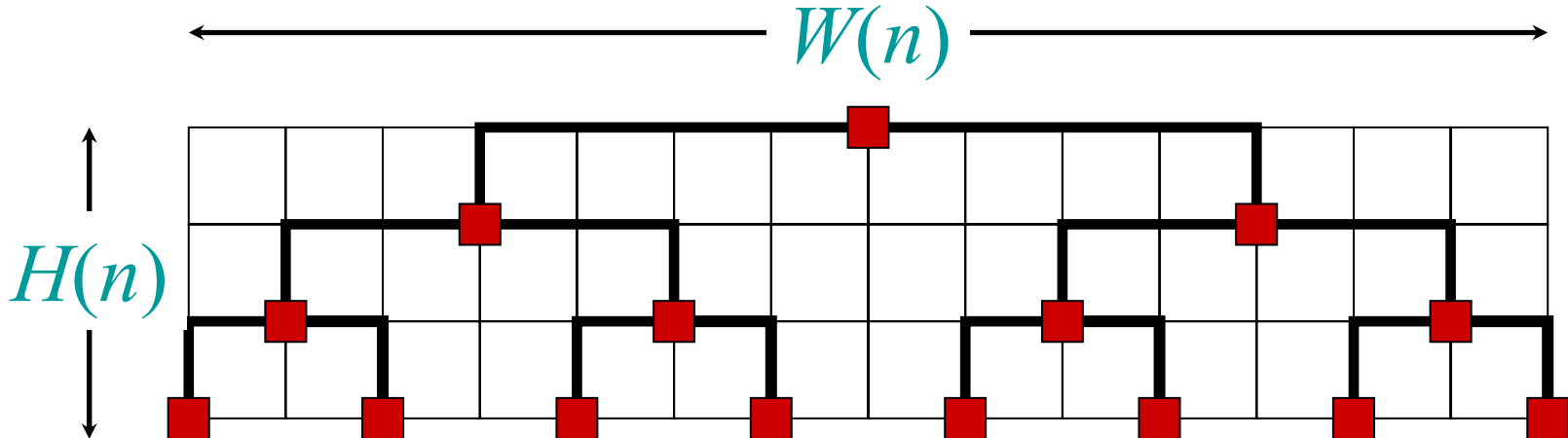


$$\begin{aligned} H(n) &= H(n/2) + \Theta(1) & W(n) &= 2W(n/2) + \Theta(1) \\ &= \Theta(\lg n) & &= \Theta(n) \end{aligned}$$



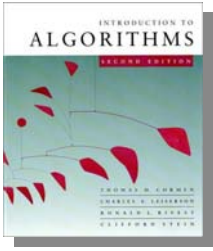
VLSI layout

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

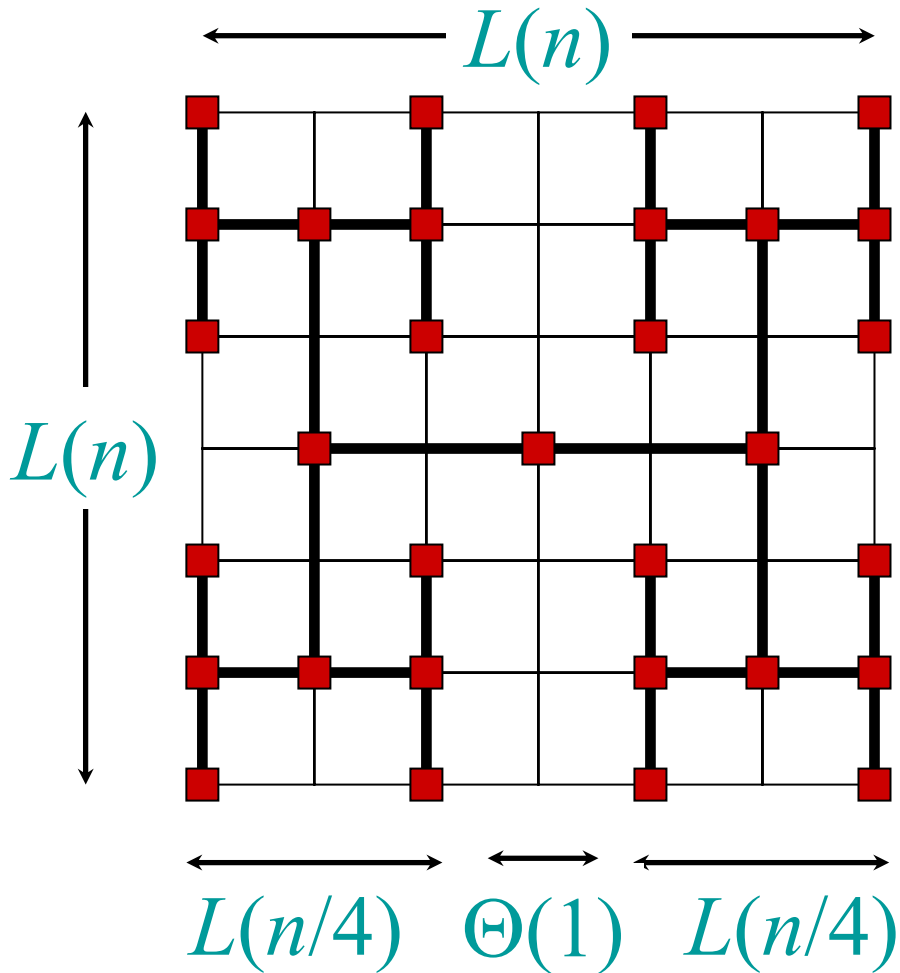


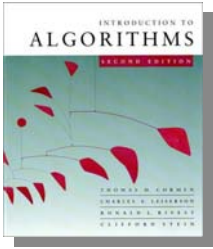
$$\begin{aligned} H(n) &= H(n/2) + \Theta(1) & W(n) &= 2W(n/2) + \Theta(1) \\ &= \Theta(\lg n) & &= \Theta(n) \end{aligned}$$

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

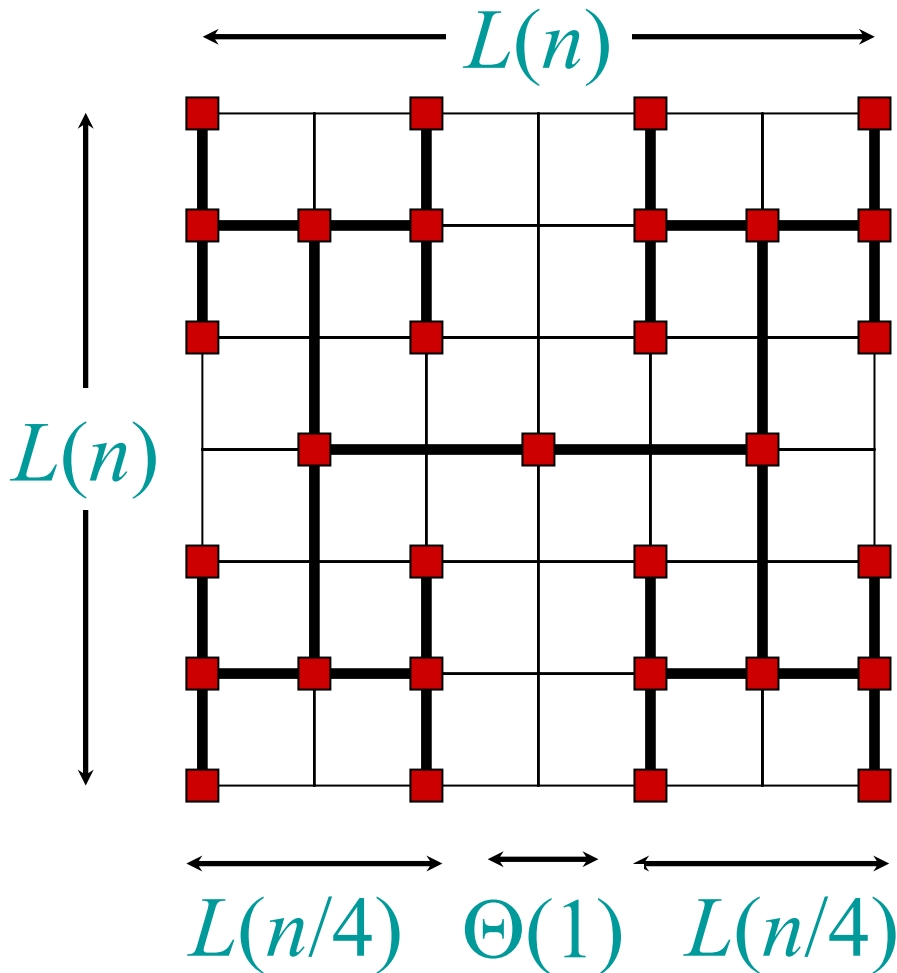


H-tree embedding



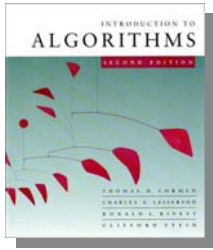


H-tree embedding



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

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



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).
- The divide-and-conquer strategy often leads to efficient algorithms.