

CS473- Algorithms I

Lecture 4

The Divide-and-Conquer Design Paradigm

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.

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

The diagram shows the recurrence relation $T(n) = 2T(n/2) + O(n)$ with three yellow circles highlighting the terms 2, $T(n/2)$, and $O(n)$. Arrows point from descriptive text to these circles: an arrow from "# subproblems" points to the '2', an arrow from "subproblem size" points to the ' $T(n/2)$ ', and an arrow from "work dividing and combining" points to the ' $O(n)$ '.

subproblems

subproblem size

work dividing and combining

Master Theorem (reprise)

$$T(n) = aT(n/b) + f(n)$$

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

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: $f(n) = \Omega(n^{\log_b^a + \epsilon})$ and $af(n/b) \leq cf(n)$

$\Rightarrow T(n) = \Theta(f(n))$.

Merge Sort: $a = 2, b = 2 \Rightarrow n^{\log_b^a} = n$

$\Rightarrow$ **Case 2** ($k = 0$) $\Rightarrow T(n) = \Theta(n \lg n)$.

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



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 is in $\mathbb{N}$.
- **Naive algorithm:** $\Theta(n)$
- **Divide-and-conquer algorithm:**

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

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

Matrix Multiplication

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

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

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

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

$$\begin{pmatrix} \mathbf{c}_{11} & | & \mathbf{c}_{12} \\ \hline \mathbf{c}_{21} & | & \mathbf{c}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_{11} & | & \mathbf{a}_{12} \\ \hline \mathbf{a}_{21} & | & \mathbf{a}_{22} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{b}_{11} & | & \mathbf{b}_{12} \\ \hline \mathbf{b}_{21} & | & \mathbf{b}_{22} \end{pmatrix}$$

$\mathbf{C} \quad = \quad \mathbf{A} \quad \cdot \quad \mathbf{B}$

$$\left. \begin{aligned} \mathbf{c}_{11} &= \mathbf{a}_{11} \mathbf{b}_{11} + \mathbf{a}_{12} \mathbf{b}_{21} \\ \mathbf{c}_{12} &= \mathbf{a}_{11} \mathbf{b}_{12} + \mathbf{a}_{12} \mathbf{b}_{22} \\ \mathbf{c}_{21} &= \mathbf{a}_{21} \mathbf{b}_{11} + \mathbf{a}_{22} \mathbf{b}_{21} \\ \mathbf{c}_{22} &= \mathbf{a}_{21} \mathbf{b}_{12} + \mathbf{a}_{22} \mathbf{b}_{22} \end{aligned} \right\}$$

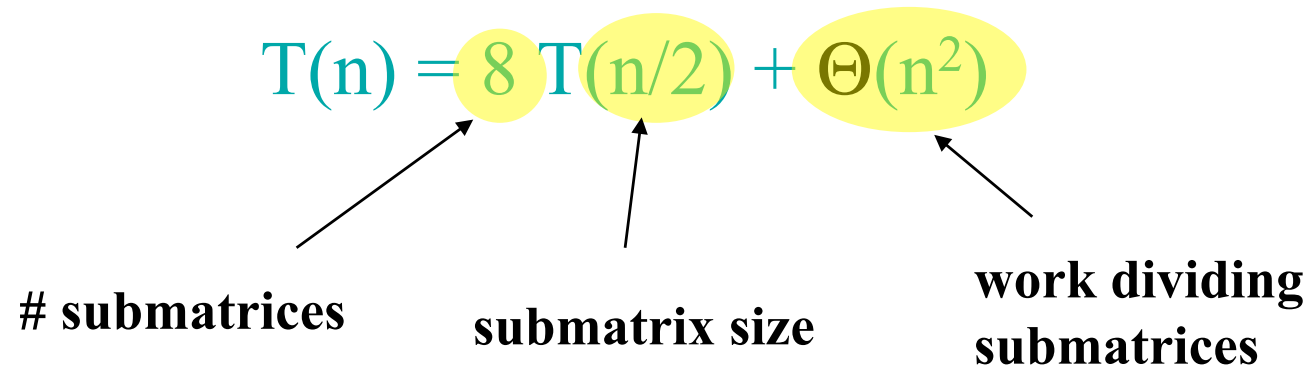
8 mults of $(n/2) \times (n/2)$ submatrices

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

Analysis of D&C Algorithm

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

submatrices submatrix size work dividing 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 **2x2** matrices with only **7** recursive mults

$$P_1 = a_{11} \times (b_{12} - b_{22})$$

$$P_2 = (a_{11} + a_{12}) \times b_{22}$$

$$P_3 = (a_{21} + a_{22}) \times b_{11}$$

$$P_4 = a_{22} \times (b_{21} - b_{11})$$

$$P_5 = (a_{11} + a_{22}) \times (b_{11} + b_{22})$$

$$P_6 = (a_{11} - a_{22}) \times (b_{21} + b_{22})$$

$$P_7 = (a_{11} - a_{21}) \times (b_{11} + b_{12})$$

$$c_{11} = P_5 + P_4 - P_2 + P_6$$

$$c_{12} = P_1 + P_2$$

$$c_{21} = P_3 + P_4$$

$$c_{22} = P_5 + P_1 - P_3 - P_7$$

7 mults **18** adds/subs.

Does not rely on
commutativity of mult.

Strassen's Idea

- Multiply **2x2** matrices with only **7** recursive mults

$$P_1 = a_{11} \times (b_{12} - b_{22})$$

$$P_2 = (a_{11} + a_{12}) \times b_{22}$$

$$P_3 = (a_{21} + a_{22}) \times b_{11}$$

$$P_4 = a_{22} \times (b_{21} - b_{11})$$

$$P_5 = (a_{11} + a_{22}) \times (b_{11} + b_{22})$$

$$P_6 = (a_{11} - a_{22}) \times (b_{21} + b_{22})$$

$$P_7 = (a_{11} - a_{21}) \times (b_{11} + b_{12})$$

$$c_{12} = P_1 + P_2$$

$$= a_{11}(b_{12} - b_{22}) + (a_{11} + a_{12})b_{22}$$

$$= a_{11}b_{12} - a_{11}b_{22} + a_{11}b_{22} + a_{12}b_{22}$$

$$= a_{11}b_{12} + a_{12}b_{22}$$

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

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

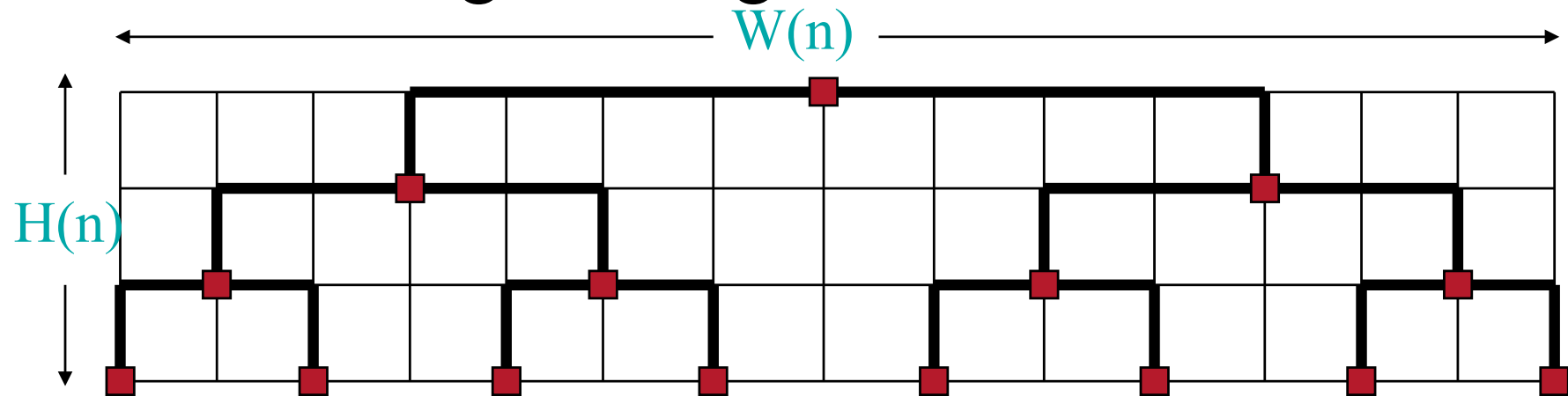
$$n^{\log_b a} = n^{\log_2 7} = n^{2.81} \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^{\lg 7}) .$$

- 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) : $\Theta(n^{2.376...})$.

VLSI Layout

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

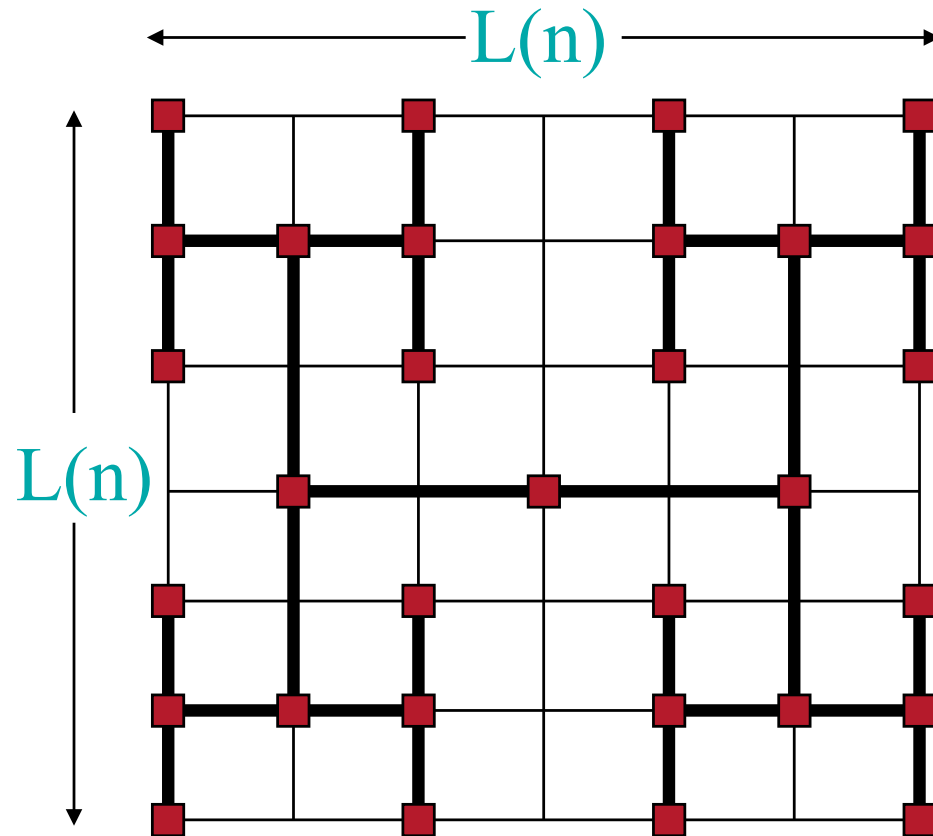


$$\begin{aligned} H(n) &= H(n/2) + \Theta(1) \\ &= \Theta(\lg n) \end{aligned}$$

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

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

H-tree Embedding



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

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

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).
- Can lead to more efficient algorithms