### DESIGN AND ANALYSIS OF ALGORITHMS

Lecture 3: Divide-and-Conquer

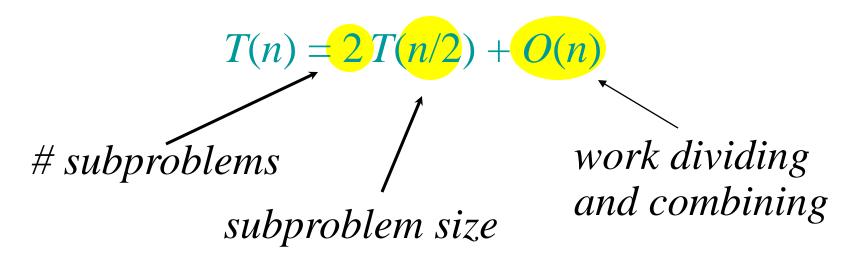
Prof. Sourav Mukhopadhyay Indian Institute of Technology Kharagpur

# 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.



# MASTER THEOREM (REPRISE)

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

$$CASE 1: f(n) = O(n^{\log_b a - \varepsilon})$$

$$\Rightarrow T(n) = \Theta(n^{\log_b a}).$$

$$CASE 2: f(n) = \Theta(n^{\log_b a} \lg^k n)$$

$$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n).$$

$$CASE 3: f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ and } af(n/b) \le cf(n)$$

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

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

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

Find an element in a sorted array:

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

Example: Find 9

Find an element in a sorted array:

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

Example: Find 9

Find an element in a sorted array:

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

Example: Find 9

Find an element in a sorted array:

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

Example: Find 9

Find an element in a sorted array:

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

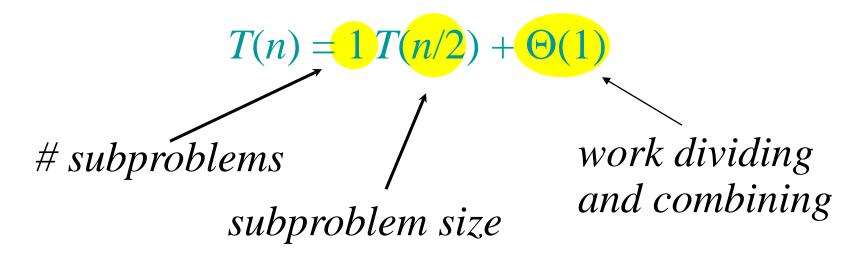
Example: Find 9

Find an element in a sorted array:

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

Example: Find 9

# RECURRENCE FOR BINARY SEARCH



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

# POWERING A NUMBER

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

Naive algorithm:  $\Theta(n)$ .

### Divide-and-conquer algorithm:

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

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

# FIBONACCI NUMBERS

### **Recursive definition:**

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

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

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  $F_0, F_1, F_2, ..., F_n$  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  $(n \ge 2)$ :

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

# MATRIX MULTIPLICATION

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

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

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

## 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:

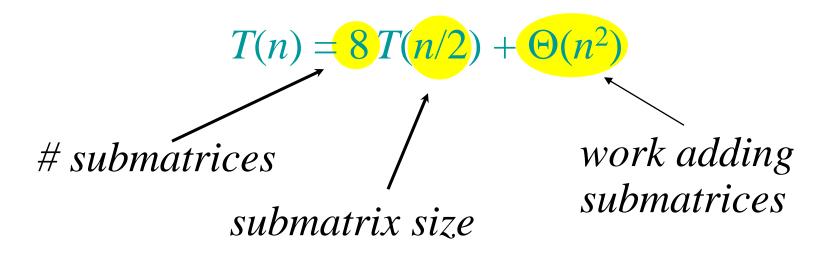
 $n \times n$  matrix =  $2 \times 2$  matrix of  $(n/2) \times (n/2)$  submatrices:

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

$$C = A \cdot B$$

$$r = ae + bg$$
  
 $s = af + bh$   
 $t = ce + dh$   
 $u = cf + dh$   
8 mults of  $(n/2) \times (n/2)$  submatrices  
4 adds of  $(n/2) \times (n/2)$  submatrices

## ANALYSIS OF D&C ALGORITHM



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

No better than the ordinary algorithm.

# 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.

$$P_{1} = a \cdot (f - h) \qquad r = P_{5} + P_{4} - P_{2} + P_{6}$$

$$P_{2} = (a + b) \cdot h \qquad = (a + d)(e + h)$$

$$P_{3} = (c + d) \cdot e \qquad + d(g - e) - (a + b)h$$

$$P_{4} = d \cdot (g - e) \qquad + (b - d)(g + h)$$

$$P_{5} = (a + d) \cdot (e + h) \qquad = ae + ah + de + dh$$

$$P_{6} = (b - d) \cdot (g + h) \qquad + dg - de - ah - bh$$

$$P_{7} = (a - c) \cdot (e + f) \qquad + bg + bh - dg - dh$$

$$= ae + bg$$

## 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.

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

# ANALYSIS OF STRASSEN

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

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

