

# CS3230 Lecture 3

## “Asymptotics, Summation, Divide & Conquer Algorithms, Recurrences and Master Theorem”

### □ Lecture Topics and Readings

- ❖ Asymptotic, Summation [CLRS]-C3
- ❖ Divide and Conquer Algorithms [CLRS]-C2
- ❖ Recurrences, Master Theorem [CLRS]-C4

*Algorithms & Discrete Math  
have very happy, stable marriage!*

Hon Wai Leong, NUS

© Leong Hon Wai, 2003--

(CS3230 Outline) Page 1

## Elements of Discrete Math, 1977

“Elements of Discrete Mathematics,” C. L. Liu, McGraw-Hill, (1977),  
has the most important topics in DM for CS.

Relations & Functions

Graphs & Planar Graphs

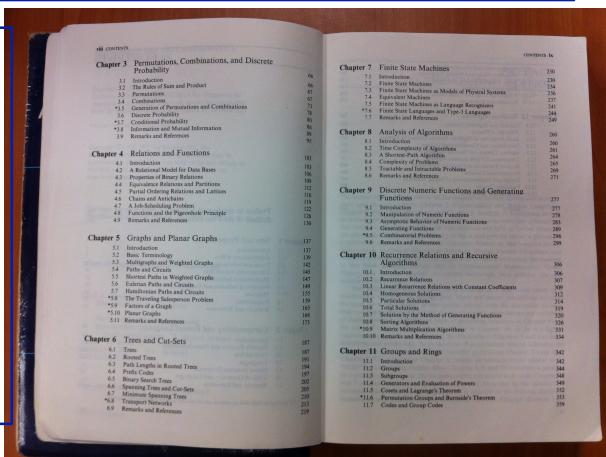
Trees and Cut-Sets

Finite State Machines

Analysis of Algorithms

Recurrence Relations

...



Hon Wai Leong, NUS

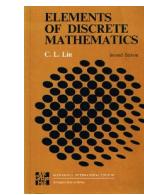
© Leong Hon Wai, 2003--

## Early pioneers of Math for CS

Don Knuth  
Stanford



C. L. Liu,  
MIT, UIUC,  
NTHU (tw)



(CS3230 Outline) Page 2

© Leong Hon Wai, 2003--

## David Plaisted



David Plaisted  
UIUC 1978-84  
UNC, 1985--

Spring 1980, LeongHW took CS373 by Plaisted;

Summer 1984, Plaisted: UIUC ----> UNC

Spring 1985 @UIUC: CS373 by Plaisted

Dave Liu recommend PhD student LeongHW  
to the Dept Head for CS373



Herbrand Award, 2010  
(for distinguished contribution  
to Automated Reasoning)

<http://www.cs.miami.edu/~geoff/Conferences/CADE/HerbrandAward.html>

© Leong Hon Wai, 2003--

# Volker Strassen



Volker Strassen

Knuth Prize, 2009

“seminal and influential contributions  
To the design and analysis  
of efficient algorithms”

[http://en.wikipedia.org/wiki/Volker\\_Strassen](http://en.wikipedia.org/wiki/Volker_Strassen)

Hon Wai Leong, NUS

(CS3230 Outline) Page 5

© Leong Hon Wai, 2003--

Thank you.

Q & A



School of Computing

Hon Wai Leong, NUS

(CS3230 Outline) Page 6

© Leong Hon Wai, 2003--

## CS3230 Lecture 3



School of Computing

“Asymptotics, Summation,  
Divide & Conquer Algorithms,  
Recurrences and Master Theorem”

### □ Lecture Topics and Readings

- ❖ Asymptotics, Summation [CLRS]-C3
- ❖ Divide and Conquer Algorithms [CLRS]-C2
- ❖ Recurrences, Master Theorem [CLRS]-C4

*Algorithms & Discrete Math  
have very happy, stable marriage!*

Hon Wai Leong, NUS

(CS3230 Algorithm Analysis) Page 1

© Leong Hon Wai, 2007--

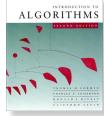
*Start by  
revising Mergesort  
(Focus on Analysis)*



Hon Wai Leong, NUS

(CS3230 Algorithm Analysis) Page 2

© Leong Hon Wai, 2007--



# Merge sort

**MERGE-SORT**  $A[1 \dots n]$

1. If  $n = 1$ , done.
2. Recursively sort  $A[1 \dots [n/2]]$  and  $A[[n/2]+1 \dots n]$ .
3. “*Merge*” the 2 sorted lists.

**Key subroutine:** MERGE

Slides from [CLRS]

Introduction to Algorithms

Page 3



# Merging two sorted arrays

20 12

13 11

7 9

2 1



# Merging two sorted arrays

20 12

13 11

7 9

2 1

1



# Merging two sorted arrays

20 12 || 20 12

13 11 || 13 11

7 9 || 7 9

2 1 || 2

1

Slides from [CLRS]

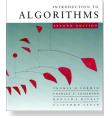
Introduction to Algorithms

Page 5

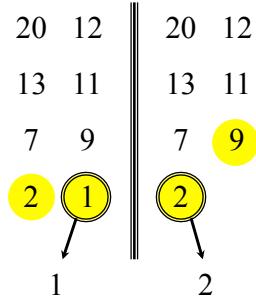
Slides from [CLRS]

Introduction to Algorithms

Page 6



## Merging two sorted arrays



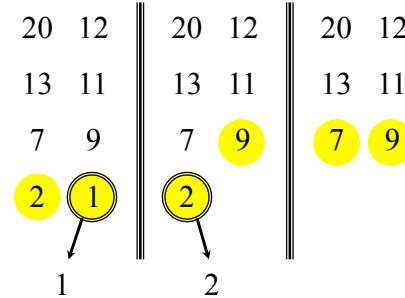
Slides from [CLRS]

Introduction to Algorithms

Page 7



## Merging two sorted arrays



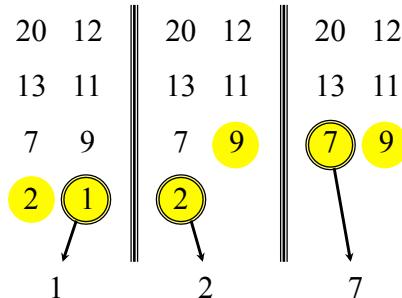
Slides from [CLRS]

Introduction to Algorithms

Page 8



## Merging two sorted arrays



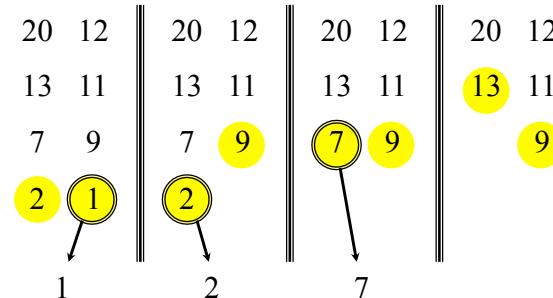
Slides from [CLRS]

Introduction to Algorithms

Page 9



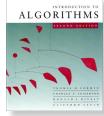
## Merging two sorted arrays



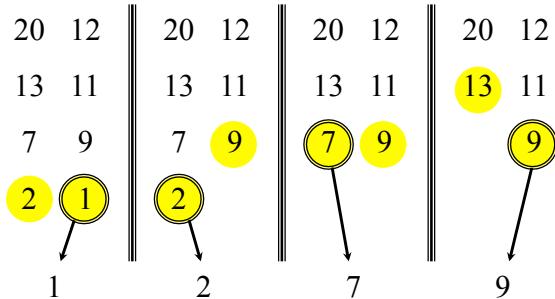
Slides from [CLRS]

Introduction to Algorithms

Page 10



## Merging two sorted arrays



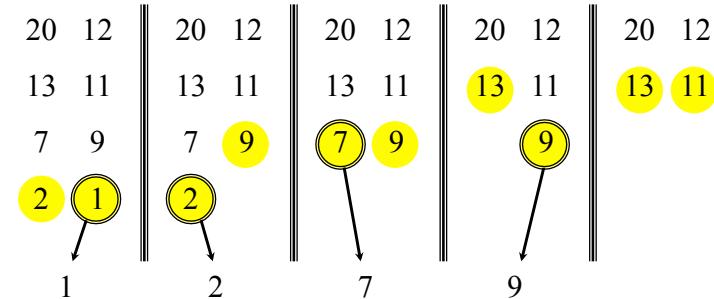
Slides from [CLRS]

Introduction to Algorithms

Page 11



## Merging two sorted arrays



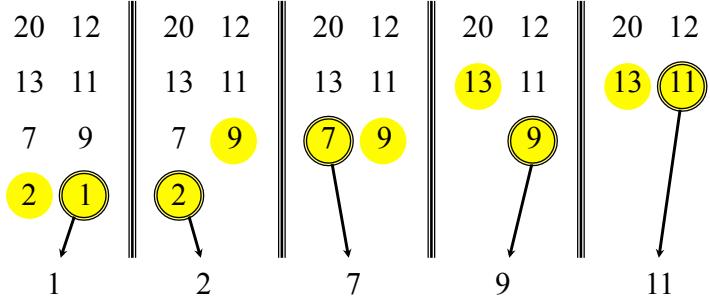
Slides from [CLRS]

Introduction to Algorithms

Page 12



## Merging two sorted arrays



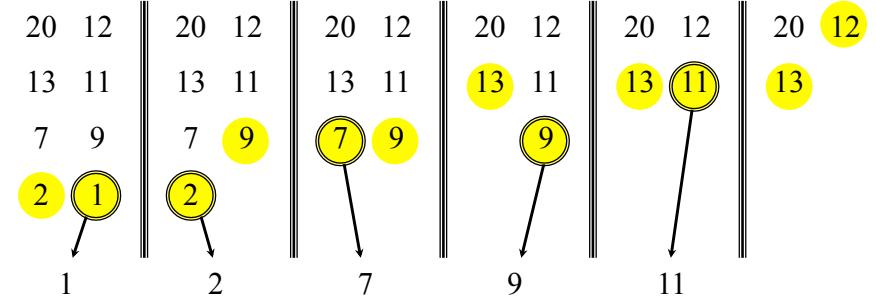
Slides from [CLRS]

Introduction to Algorithms

Page 13



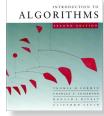
## Merging two sorted arrays



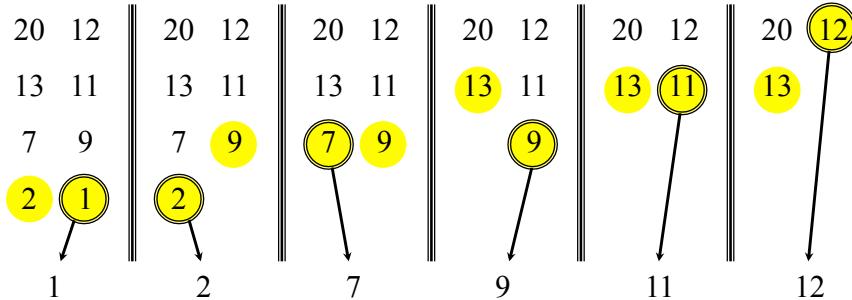
Slides from [CLRS]

Introduction to Algorithms

Page 14



## Merging two sorted arrays



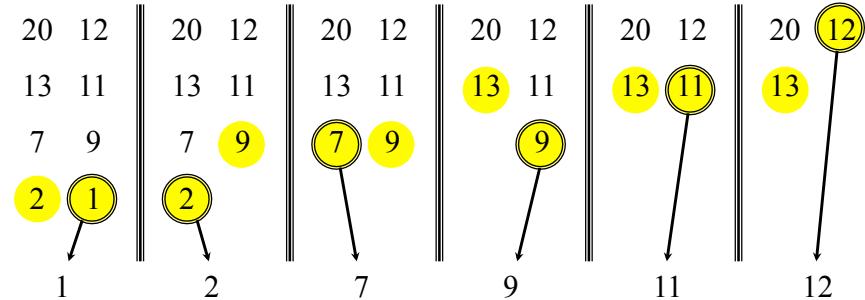
Slides from [CLRS]

Introduction to Algorithms

Page 15



## Merging two sorted arrays



Time =  $\Theta(n)$  to merge a total of  $n$  elements (linear time).

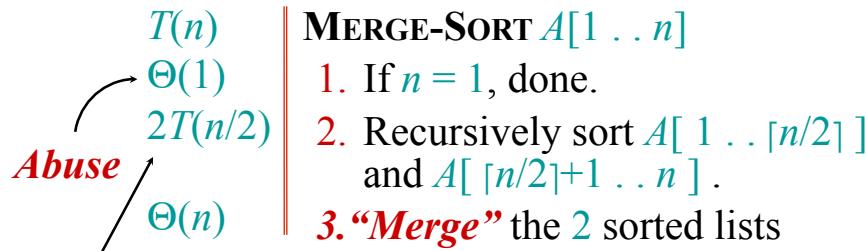
Slides from [CLRS]

Introduction to Algorithms

Page 16



## Analyzing merge sort



**Sloppiness:** Should be  $T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor)$ , but it turns out not to matter asymptotically.

Slides from [CLRS]

Introduction to Algorithms

Page 17



## Recurrence for merge sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

- We shall usually omit stating the base case when  $T(n) = \Theta(1)$  for sufficiently small  $n$ , but only when it has no effect on the asymptotic solution to the recurrence.

Slides from [CLRS]

Introduction to Algorithms

Page 18



## Solving recurrences

- The analysis of merge sort required us to solve a recurrence.
- Recurrences are like solving integrals, differential equations, etc.
  - Learn a few tricks.
- Applications of recurrences to divide-and-conquer algorithms.

Slides from [CLRS]

Introduction to Algorithms

Page 19



## Substitution method

### Recursion Tree method

basis of the Master Theorem

*The most general method:*

1. **Guess** the form of the solution.
2. **Verify** by induction.
3. **Solve** for constants.

*Not covered in CS3230; those interested  
see example and details in CLRS*

Slides from [CLRS]

Introduction to Algorithms

Page 20



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.

Slides from [CLRS]

Introduction to Algorithms

Page 21



## Recursion tree

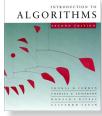
Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.

$T(n)$

Slides from [CLRS]

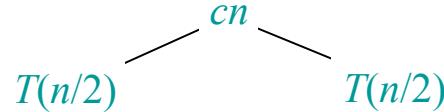
Introduction to Algorithms

Page 22



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

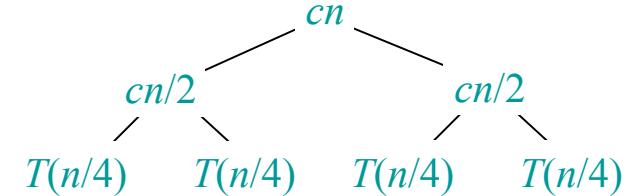
Introduction to Algorithms

Page 23



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

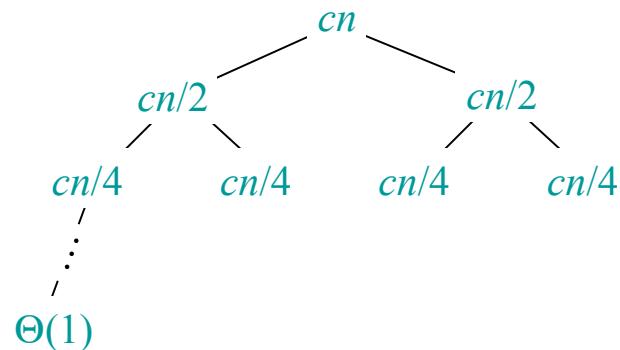
Introduction to Algorithms

Page 24



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

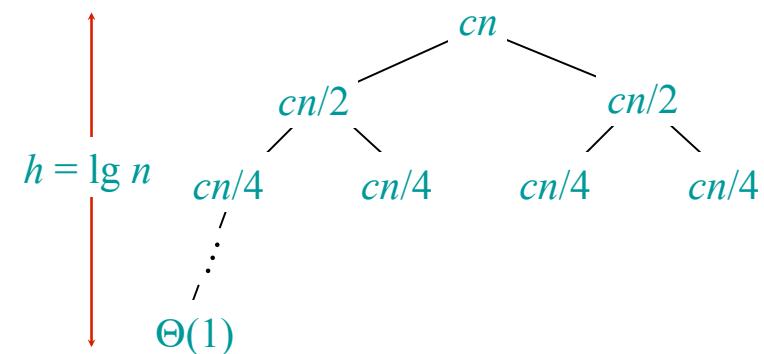
Introduction to Algorithms

Page 25



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

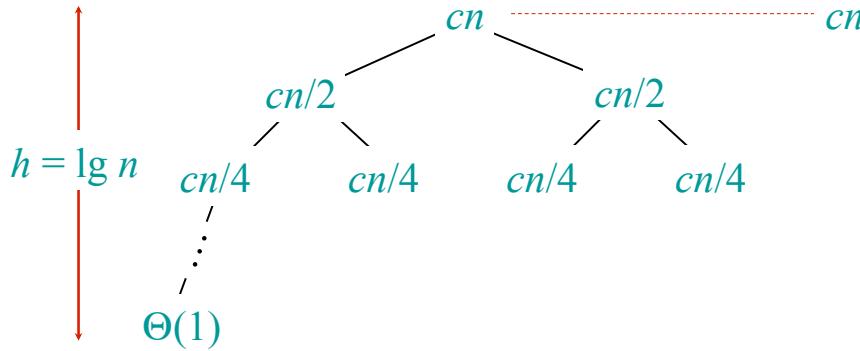
Introduction to Algorithms

Page 26



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

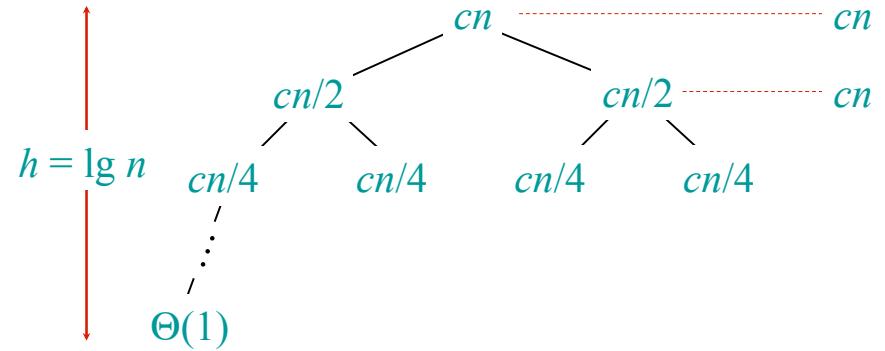
Introduction to Algorithms

Page 27



## Recursion tree

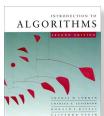
Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

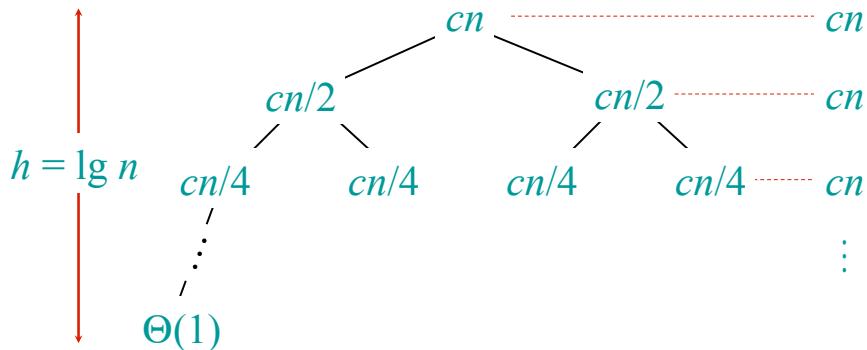
Introduction to Algorithms

Page 28



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

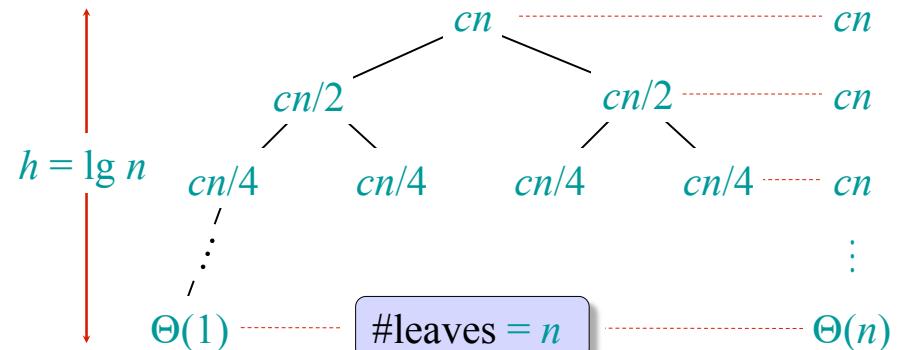
Introduction to Algorithms

Page 29



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

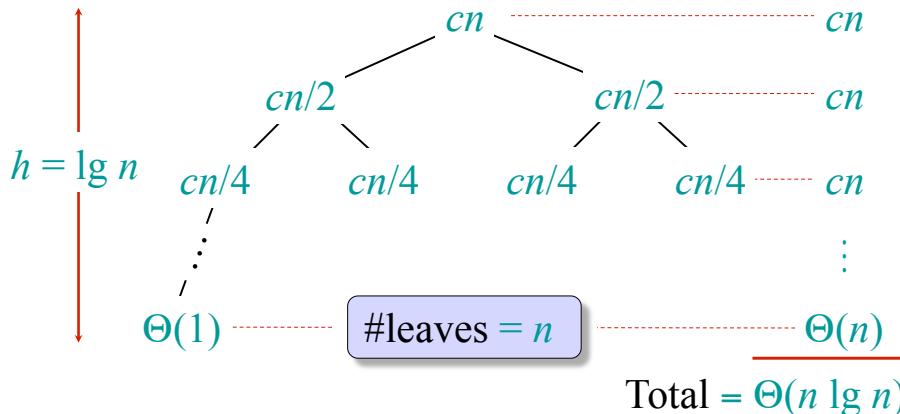
Introduction to Algorithms

Page 30



## Recursion tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



Slides from [CLRS]

Introduction to Algorithms

Page 31



## Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (...).
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method.

Slides from [CLRS]

Introduction to Algorithms

Page 32



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :

Slides from [CLRS]

Introduction to Algorithms

Page 33



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :

$$T(n)$$

Slides from [CLRS]

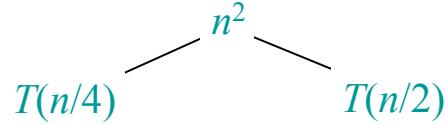
Introduction to Algorithms

Page 34



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :



Slides from [CLRS]

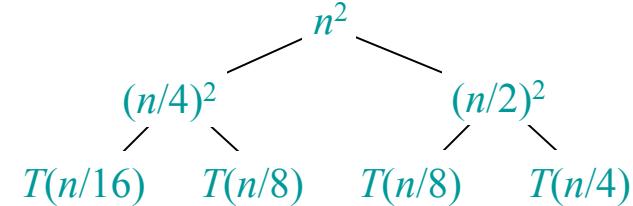
Introduction to Algorithms

Page 35



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :



Slides from [CLRS]

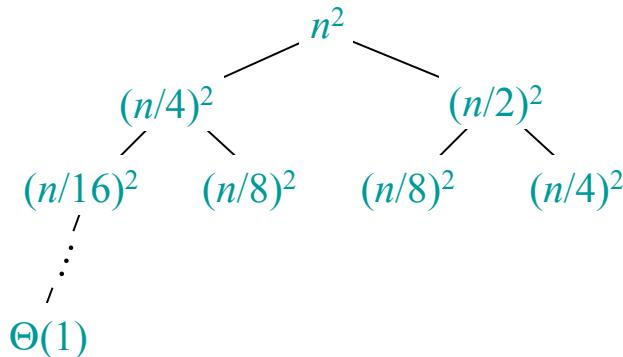
Introduction to Algorithms

Page 36



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :



Slides from [CLRS]

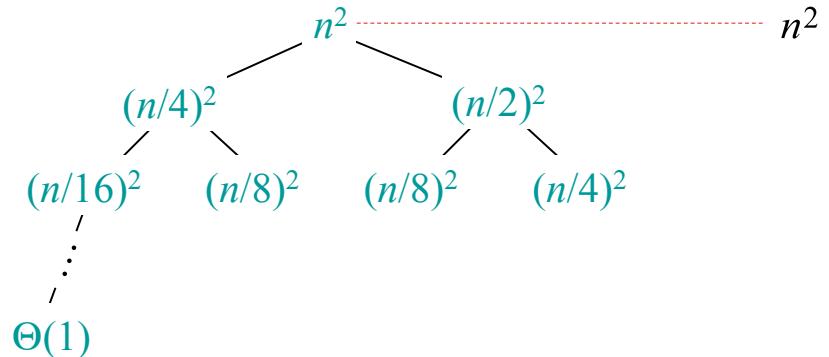
Introduction to Algorithms

Page 37



## Example of recursion tree

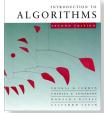
Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :



Slides from [CLRS]

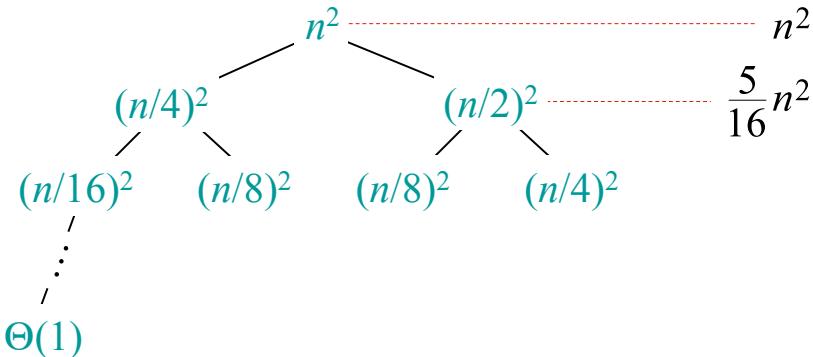
Introduction to Algorithms

Page 38



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :



Slides from [CLRS]

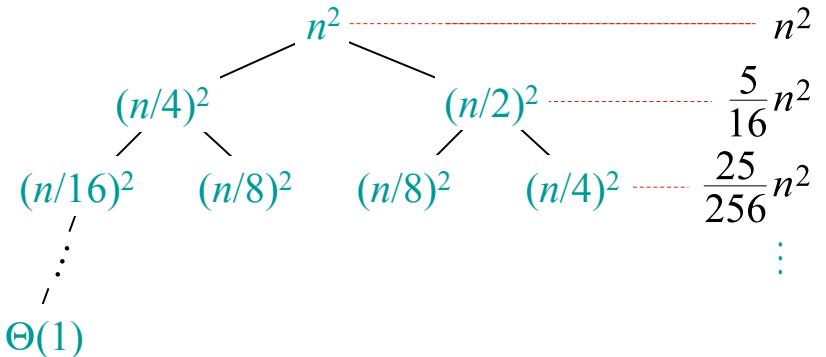
Introduction to Algorithms

Page 39



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :



Slides from [CLRS]

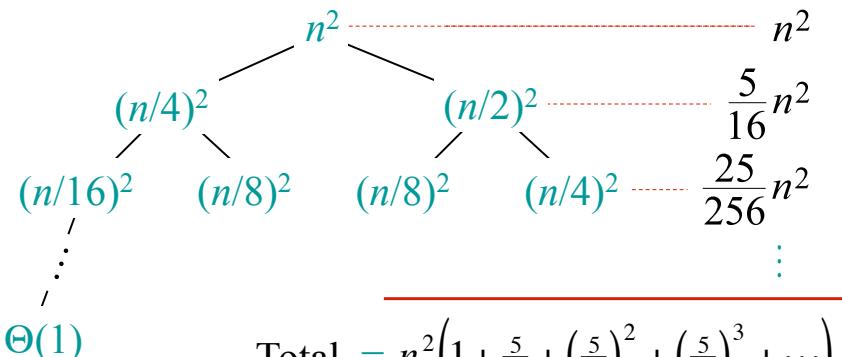
Introduction to Algorithms

Page 40



## Example of recursion tree

Solve  $T(n) = T(n/4) + T(n/2) + n^2$ :



Slides from [CLRS]

Introduction to Algorithms

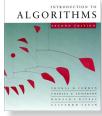
Page 41



Slides from [CLRS]

Introduction to Algorithms

Page 42



## The master method

The master method applies to recurrences of the form

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

where  $a \geq 1$ ,  $b > 1$ , and  $f$  is asymptotically positive.

Slides from [CLRS]

Introduction to Algorithms

Page 43



## Three common cases

Compare  $f(n)$  with  $n^{\log_b a}$ :

1.  $f(n) = O(n^{\log_b a - \varepsilon})$  for some constant  $\varepsilon > 0$ .

- $f(n)$  grows polynomially slower than  $n^{\log_b a}$  (by an  $n^\varepsilon$  factor).

**Solution:**  $T(n) = \Theta(n^{\log_b a})$ .

Slides from [CLRS]

Introduction to Algorithms

Page 44



## Three common cases

Compare  $f(n)$  with  $n^{\log_b a}$ :

1.  $f(n) = O(n^{\log_b a - \varepsilon})$  for some constant  $\varepsilon > 0$ .

- $f(n)$  grows polynomially slower than  $n^{\log_b a}$  (by an  $n^\varepsilon$  factor).

**Solution:**  $T(n) = \Theta(n^{\log_b a})$ .

2.  $f(n) = \Theta(n^{\log_b a} \lg^k n)$  for some constant  $k \geq 0$ .

- $f(n)$  and  $n^{\log_b a}$  grow at similar rates.

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

Slides from [CLRS]

Introduction to Algorithms

Page 45



## Three common cases (cont.)

Compare  $f(n)$  with  $n^{\log_b a}$ :

3.  $f(n) = \Omega(n^{\log_b a + \varepsilon})$  for some constant  $\varepsilon > 0$ .

- $f(n)$  grows polynomially faster than  $n^{\log_b a}$  (by an  $n^\varepsilon$  factor),

and  $f(n)$  satisfies the **regularity condition** that  $af(n/b) \leq cf(n)$  for some constant  $c < 1$ .

**Solution:**  $T(n) = \Theta(f(n))$ .

Slides from [CLRS]

Introduction to Algorithms

Page 46



## Examples

**Ex.**  $T(n) = 4T(n/2) + n$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n.$$

**CASE 1:**  $f(n) = O(n^{2-\varepsilon})$  for  $\varepsilon = 1$ .

$$\therefore T(n) = \Theta(n^2).$$

Slides from [CLRS]

Introduction to Algorithms

Page 47



## Examples

**Ex.**  $T(n) = 4T(n/2) + n$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n.$$

**CASE 1:**  $f(n) = O(n^{2-\varepsilon})$  for  $\varepsilon = 1$ .

$$\therefore T(n) = \Theta(n^2).$$

**Ex.**  $T(n) = 4T(n/2) + n^2$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2.$$

**CASE 2:**  $f(n) = \Theta(n^2 \lg^0 n)$ , that is,  $k = 0$ .

$$\therefore T(n) = \Theta(n^2 \lg n).$$

Slides from [CLRS]

Introduction to Algorithms

Page 48



## Examples

**Ex.**  $T(n) = 4T(n/2) + n^3$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$$

**CASE 3:**  $f(n) = \Omega(n^{2+\varepsilon})$  for  $\varepsilon = 1$

and  $4(n/2)^3 \leq cn^3$  (reg. cond.) for  $c = 1/2$ .

$$\therefore T(n) = \Theta(n^3).$$

Slides from [CLRS]

Introduction to Algorithms

Page 49



## Examples

**Ex.**  $T(n) = 4T(n/2) + n^3$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$$

**CASE 3:**  $f(n) = \Omega(n^{2+\varepsilon})$  for  $\varepsilon = 1$

and  $4(n/2)^3 \leq cn^3$  (reg. cond.) for  $c = 1/2$ .

$$\therefore T(n) = \Theta(n^3).$$

**Ex.**  $T(n) = 4T(n/2) + n^2/\lg n$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2/\lg n.$$

Master method does not apply. In particular, for every constant  $\varepsilon > 0$ , we have  $n^\varepsilon = \omega(\lg n)$ .

Slides from [CLRS]

Introduction to Algorithms

Page 50



## Idea behind the proof of the Master Theorem



Slides from [CLRS]

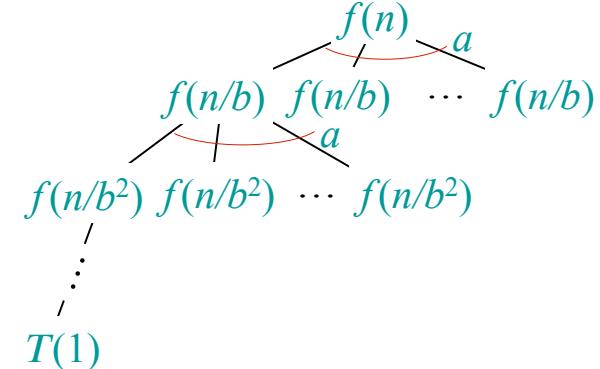
Introduction to Algorithms

Page 51



## Idea of master theorem

**Recursion tree:**



Slides from [CLRS]

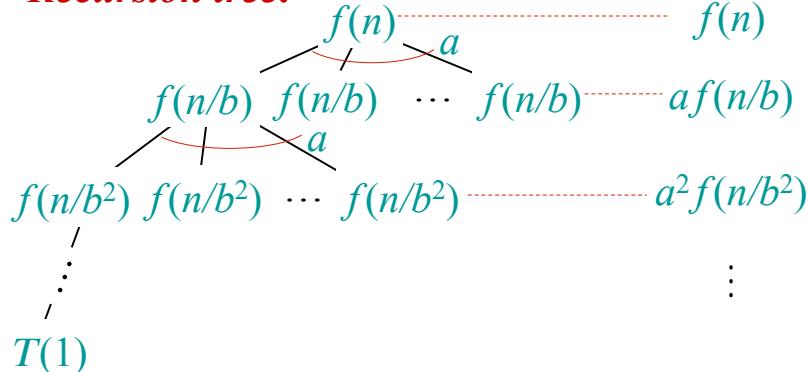
Introduction to Algorithms

Page 52



## Idea of master theorem

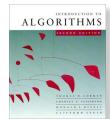
**Recursion tree:**



Slides from [CLRS]

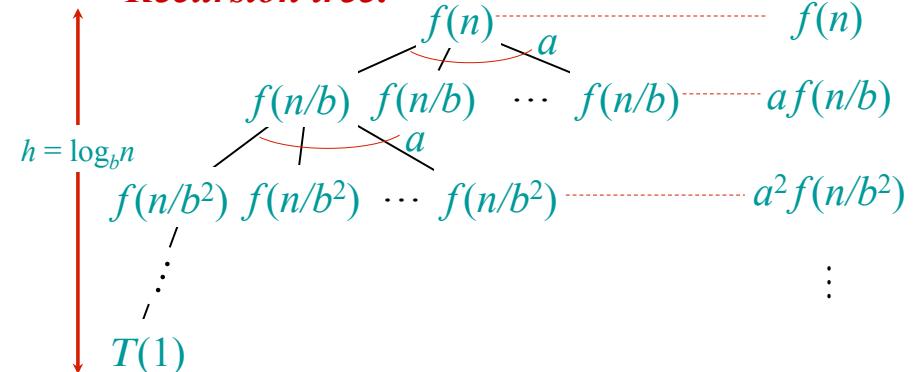
Introduction to Algorithms

Page 53



## Idea of master theorem

**Recursion tree:**



Slides from [CLRS]

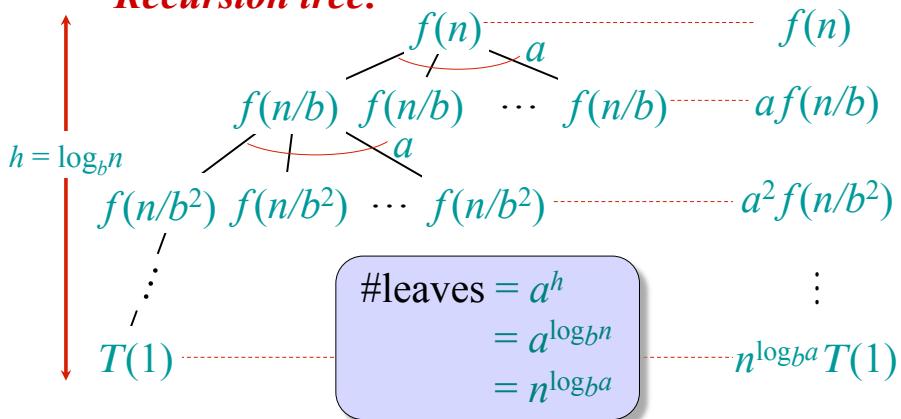
Introduction to Algorithms

Page 54



## Idea of master theorem

**Recursion tree:**



$$\begin{aligned}\# \text{leaves} &= a^h \\ &= a^{\log_b n} \\ &= n^{\log_b a}\end{aligned}$$

Slides from [CLRS]

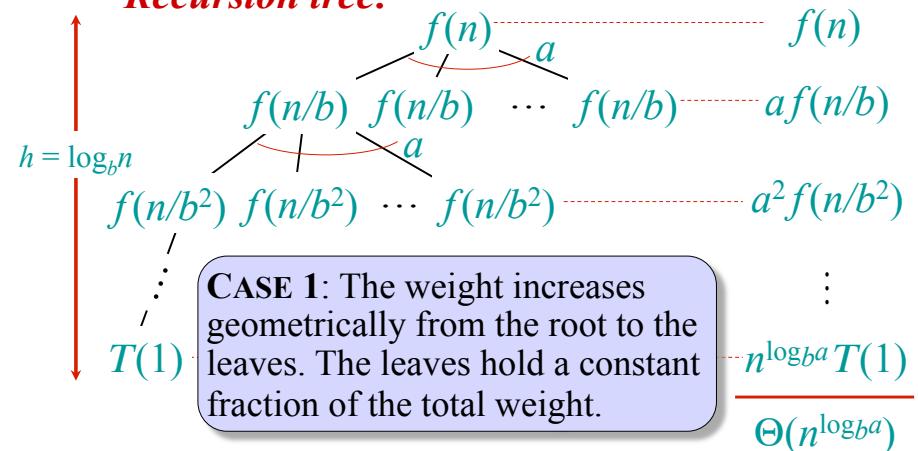
Introduction to Algorithms

Page 55



## Idea of master theorem

**Recursion tree:**



**CASE 1:** The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Slides from [CLRS]

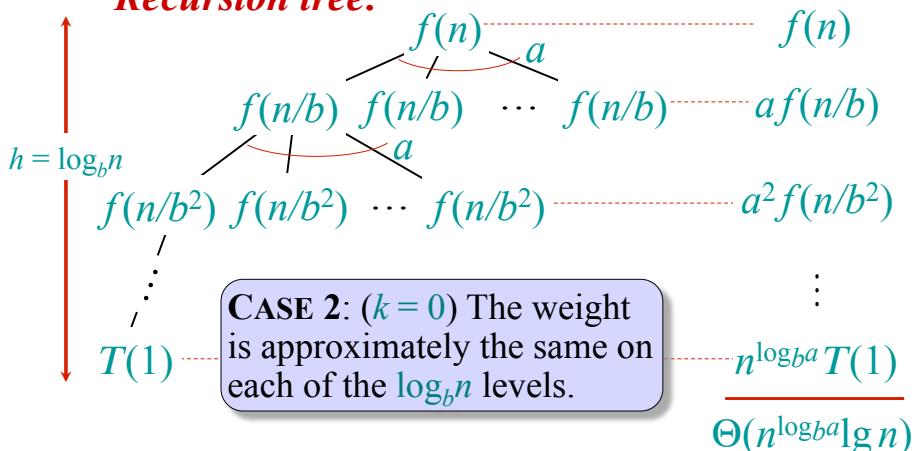
Introduction to Algorithms

Page 56



## Idea of master theorem

**Recursion tree:**



**CASE 2:** ( $k = 0$ ) The weight is approximately the same on each of the  $\log_b n$  levels.

$$\frac{n^{\log_b a} T(1)}{\Theta(n^{\log_b a} \lg n)}$$

Slides from [CLRS]

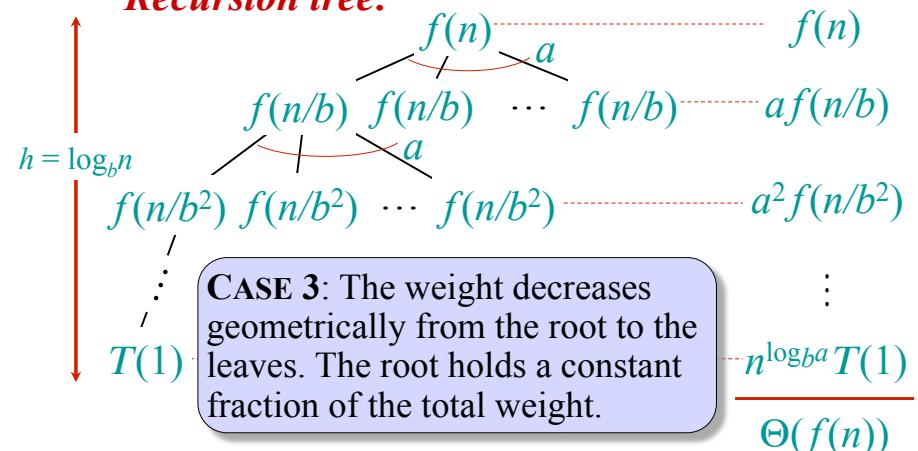
Introduction to Algorithms

Page 57



## Idea of master theorem

**Recursion tree:**



**CASE 3:** The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

Slides from [CLRS]

Introduction to Algorithms

Page 58



*Revise many  
Divide and Conquer  
Algorithms*

Slides from [CLRS]

Introduction to Algorithms

Page 59



## 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.



## Merge sort

**1. Divide:** Trivial.

**2. Conquer:** Recursively sort 2 subarrays.

**3. Combine:** Linear-time merge.

Slides from [CLRS]

Introduction to Algorithms

Page 61



## Merge sort

**1. Divide:** Trivial.

**2. Conquer:** Recursively sort 2 subarrays.

**3. Combine:** Linear-time merge.

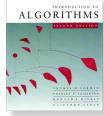
$$T(n) = 2T(n/2) + \Theta(n)$$

# subproblems      ↗      work dividing and combining  
                        ↓      subproblem size

Slides from [CLRS]

Introduction to Algorithms

Page 62



## Master theorem (Mergesort)

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

# subproblems      ↗  
subproblem size      ↙  
*work dividing  
and combining*

**Merge sort:**  $a = 2, b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$   
 $\Rightarrow$  CASE 2 ( $k = 0$ )  $\Rightarrow T(n) = \Theta(n \lg n)$ .

Slides from [CLRS]

Introduction to Algorithms

Page 63



## Binary search

Find an element in a sorted array:

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

Slides from [CLRS]

Introduction to Algorithms

Page 64



## 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

Slides from [CLRS]

Introduction to Algorithms

Page 65



## Binary search

Find an element in a sorted array:

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

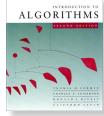
**Example:** Find 9

3    5    7    8    9    12    15

Slides from [CLRS]

Introduction to Algorithms

Page 66



# 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

Slides from [CLRS]

Introduction to Algorithms

Page 67



# 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

Slides from [CLRS]

Introduction to Algorithms

Page 68



# 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

Slides from [CLRS]

Introduction to Algorithms

Page 69



# Binary search

Find an element in a sorted array:

**1.Divide:** Check middle element.

**2.Conquer:** Recursively search 1 subarray.

**3.Combine:** Trivial.

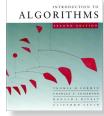
**Example:** Find 9

3    5    7    8    9    12    15

Slides from [CLRS]

Introduction to Algorithms

Page 70



## Recurrence for binary search

$$T(n) = \underbrace{1}_{\text{\# subproblems}} \underbrace{T(n/2)}_{\text{subproblem size}} + \underbrace{\Theta(1)}_{\text{work dividing and combining}}$$

Slides from [CLRS]

Introduction to Algorithms

Page 71



## Recurrence for binary search

$$T(n) = \underbrace{1}_{\text{\# subproblems}} \underbrace{T(n/2)}_{\text{subproblem size}} + \underbrace{\Theta(1)}_{\text{work dividing and combining}}$$

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

Slides from [CLRS]

Introduction to Algorithms

Page 72



## Powering a number

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

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

Slides from [CLRS]

Introduction to Algorithms

Page 73



## Powering a number

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

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

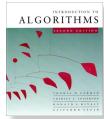
**Divide-and-conquer algorithm:**

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

Slides from [CLRS]

Introduction to Algorithms

Page 74



## Powering a number

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

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

**Divide-and-conquer algorithm:**

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

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

Slides from [CLRS]

Introduction to Algorithms

Page 75



## Matrix multiplication

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

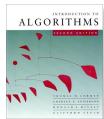
$$c_{11} = (a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + \dots + a_{1n} \cdot b_{n1})$$

$$c_{ij} = (a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + a_{i3} \cdot b_{3j} + \dots + a_{in} \cdot b_{nj})$$

Slides from [CLRS]

Introduction to Algorithms

Page 76



## Matrix multiplication

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

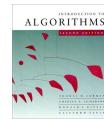
$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots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

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

Slides from [CLRS]

Introduction to Algorithms

Page 77



## Standard algorithm

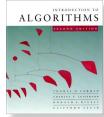
```
for  $i \leftarrow 1$  to  $n$ 
    do for  $j \leftarrow 1$  to  $n$ 
        do  $c_{ij} \leftarrow 0$ 
            for  $k \leftarrow 1$  to  $n$ 
                do  $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$ 
```

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

Slides from [CLRS]

Introduction to Algorithms

Page 78



## Divide-and-conquer algorithm

**IDEA:**

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

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

$$C = A \cdot B$$

$$\left. \begin{array}{l} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{array} \right\} \begin{array}{l} 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$

Slides from [CLRS]

Introduction to Algorithms

Page 79



## Divide-and-conquer algorithm

**IDEA:**

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

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

$$C = A \cdot B$$

$$\left. \begin{array}{l} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{array} \right\} \begin{array}{l} \text{recursive} \\ 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$

Slides from [CLRS]

Introduction to Algorithms

Page 80



## Standard algorithm

```
for i ← 1 to n
    do for j ← 1 to n
        do cij ← 0
            for k ← 1 to n
                do cij ← cij + aik · bkj
```

Slides from [CLRS]

Introduction to Algorithms

Page 81



## Analysis of D&C algorithm

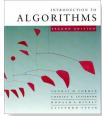
$$T(n) = 8T(n/2) + \Theta(n^2)$$

# submatrices                      submatrix size  
  ↑  
  ↑  
  work adding  
  submatrices

Slides from [CLRS]

Introduction to Algorithms

Page 82



# **Analysis of D&C algorithm**

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

# submatrices                          work adding submatrices

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

Slides from [CLRS]

Introduction to Algorithms

Page 83



## Analysis of D&C algorithm

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

$$n^{\log b a} = n^{\log 2^8} = n^3 \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^3).$$

*No better than the ordinary algorithm.*

The image shows the front cover of the second edition of "Introduction to Algorithms". The title is at the top in a large serif font. Below it, the subtitle "SECOND EDITION" is printed in a smaller, all-caps serif font. The background of the cover features a stylized graphic of red and blue shapes, possibly representing algorithmic flow or data structures.

## Strassen's idea

- Multiply  $2 \times 2$  matrices with only 7 recursive mults.

Slides from [CLRS]

Introduction to Algorithms

Page 85



## Strassen's idea

- Multiply  $2 \times 2$  matrices with only 7 recursive mults.

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



Slides from [CLRS]

Introduction to Algorithms

Page 85

Slides from [CLRS]

Introduction to Algorithms

Page 86



## Strassen's idea

- Multiply  $2 \times 2$  matrices with only 7 recursive mults.

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

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

Slides from [CLRS]

Introduction to Algorithms

Page 87



## Strassen's idea

- Multiply  $2 \times 2$  matrices with only 7 recursive mults.

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

7 mults, 18 adds/subs.

**Note:** No reliance on commutativity of mult!

Slides from [CLRS]

Introduction to Algorithms

Page 88



## Strassen's idea

- Multiply  $2 \times 2$  matrices with only 7 recursive mults.

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

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

Slides from [CLRS]

Introduction to Algorithms

Page 89



## Strassen's algorithm

**1. Divide:** Partition  $A$  and  $B$  into  $(n/2) \times (n/2)$  submatrices. Form terms to be multiplied using  $+$  and  $-$ .

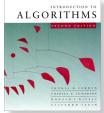
**2. Conquer:** Perform 7 multiplications of  $(n/2) \times (n/2)$  submatrices recursively.

**3. Combine:** Form  $C$  using  $+$  and  $-$  on  $(n/2) \times (n/2)$  submatrices.

Slides from [CLRS]

Introduction to Algorithms

Page 90



## 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.

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

Slides from [CLRS]

Introduction to Algorithms

Page 91



## Analysis of Strassen

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

Slides from [CLRS]

Introduction to Algorithms

Page 92



## Analysis of Strassen

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

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

Slides from [CLRS]

Introduction to Algorithms

Page 93



## 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.

Slides from [CLRS]

Introduction to Algorithms

Page 94



## Analysis of Strassen

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

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

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

Slides from [CLRS]

Introduction to Algorithms

Page 95



## 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.

Slides from [CLRS]

Introduction to Algorithms

Page 96



## Why study algorithms and performance?

- Algorithms help us to understand **scalability**.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a **language** for talking about program behavior.
- Performance is the **currency** of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

Slides from [CLRS]

Introduction to Algorithms

Page 97

---

**Thank you.**

**Q & A**



School of Computing