

THOMAS H. CORMEN  
CHARLES E. LEISERSON  
RONALD L. RIVEST  
CLIFFORD STEIN

INTRODUCTION TO

# ALGORITHMS

FOURTH EDITION

Over  
**1 MILLION**  
copies sold  
worldwide



# CSCE 500

## Design and Analysis of Algorithms

Fall 2022

August 22, 2022

**Instructor:** Nian-Feng Tzeng  
**Office:** Rm. OLVR 354 (x 2-6304)  
**Class meeting:** MW 10:00 – 11:15, OLVR 113

### Textbook and Supplemental Materials:

1. Introduction to Algorithms, Fourth Edition, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, The MIT Press, 2022, ISBN: 978-026204630-5.
2. Published articles supplementary to covered topics.

### Course Description:

This course provides a comprehensive coverage of modern computer algorithms, aiming at in-depth treatment of algorithmic design and analysis with elementary explanation while keeping mathematical rigor. Based on the textbook of “Introduction to Algorithms”, this class covers the topics listed below in sequence.

- (1) Foundations.
- (2) Data Structures – hash tables, trees, data structures for disjoint sets.
- (3) Design and Analysis Techniques – dynamic programming, greedy algorithms, amortized analysis.
- (4) Graph Algorithms – spanning trees, shortest paths, maximum flow, matchings in bipartite graphs.
- (5) Selected Topics – NP-completeness, approximation algorithms, parallel algorithms, on-line algorithms.

Each covered topic starts with the description of pertinent algorithms in English and/or in the pseudocode(s), followed by their careful complexity analyses.

### Course Requirements:

1. Homework (10%)
2. Midterm exams (2) (50%)
3. Final exam (comprehensive) (40%)

## **I Foundations**

	<b>Introduction</b>	<b>3</b>
<b>1</b>	<b>The Role of Algorithms in Computing</b>	<b>5</b>
	1.1 Algorithms	5
	1.2 Algorithms as a technology	12
<b>2</b>	<b><u>Getting Started</u></b>	<b>17</b>
	2.1 Insertion sort	17
	2.2 Analyzing algorithms	25
	2.3 Designing algorithms	34
<b>3</b>	<b><u>Characterizing Running Times</u></b>	<b>49</b>
	3.1 $O$ -notation, $\Omega$ -notation, and $\Theta$ -notation	50
	3.2 Asymptotic notation: formal definitions	53
	3.3 Standard notations and common functions	63
<b>4</b>	<b><u>Divide-and-Conquer</u></b>	<b>76</b>
	4.1 Multiplying square matrices	80
	4.2 Strassen's algorithm for matrix multiplication	85
	4.3 The substitution method for solving recurrences	90
	4.4 The recursion-tree method for solving recurrences	95
	4.5 The master method for solving recurrences	101
★	4.6 Proof of the continuous master theorem	107
★	4.7 Akra-Bazzi recurrences	115

# Table of Contents

<b>5</b>	<b>Probabilistic Analysis and Randomized Algorithms</b>	<b>126</b>
5.1	The hiring problem	126
5.2	Indicator random variables	130
5.3	Randomized algorithms	134
★ 5.4	Probabilistic analysis and further uses of indicator random variables	140

## II Sorting and Order Statistics

	<b>Introduction</b>	<b>157</b>
<b>6</b>	<b>Heapsort</b>	<b>161</b>
6.1	Heaps	161
6.2	Maintaining the heap property	164
6.3	Building a heap	167
6.4	The heapsort algorithm	170
6.5	Priority queues	172
<b>7</b>	<b>Quicksort</b>	<b>182</b>
7.1	Description of quicksort	183
7.2	Performance of quicksort	187
7.3	A randomized version of quicksort	191
7.4	Analysis of quicksort	193
<b>8</b>	<b>Sorting in Linear Time</b>	<b>205</b>
8.1	Lower bounds for sorting	205
8.2	Counting sort	208
8.3	Radix sort	211
8.4	Bucket sort	215
<b>9</b>	<b>Medians and Order Statistics</b>	<b>227</b>
9.1	Minimum and maximum	228
9.2	Selection in expected linear time	230
9.3	Selection in worst-case linear time	236

## III Data Structures

	<b>Introduction</b>	<b>249</b>
<b>10</b>	<b>Elementary Data Structures</b>	<b>252</b>
10.1	Simple array-based data structures: arrays, matrices, stacks, queues	252
10.2	Linked lists	258
10.3	Representing rooted trees	265

# Table of Contents

<b>11</b>	<b><u>Hash Tables</u></b>	<b>272</b>
11.1	Direct-address tables	273
11.2	Hash tables	275
11.3	Hash functions	282
11.4	Open addressing	293
11.5	Practical considerations	301
<b>12</b>	<b><u>Binary Search Trees</u></b>	<b>312</b>
12.1	What is a binary search tree?	312
12.2	Querying a binary search tree	316
12.3	Insertion and deletion	321
<b>13</b>	<b><u>Red-Black Trees</u></b>	<b>331</b>
13.1	Properties of red-black trees	331
13.2	Rotations	335
13.3	Insertion	338
13.4	Deletion	346
<hr/>		
<b>IV</b>	<b><u>Advanced Design and Analysis Techniques</u></b>	
	<b>Introduction</b>	<b>361</b>
<b>14</b>	<b><u>Dynamic Programming</u></b>	<b>362</b>
14.1	Rod cutting	363
14.2	Matrix-chain multiplication	373
14.3	Elements of dynamic programming	382
14.4	Longest common subsequence	393
14.5	Optimal binary search trees	400
<b>15</b>	<b><u>Greedy Algorithms</u></b>	<b>417</b>
15.1	An activity-selection problem	418
15.2	Elements of the greedy strategy	426
15.3	Huffman codes	431
15.4	Offline caching	440
<b>16</b>	<b><u>Amortized Analysis</u></b>	<b>448</b>
16.1	Aggregate analysis	449
16.2	The accounting method	453
16.3	The potential method	456
16.4	Dynamic tables	460

# Table of Contents

## V Advanced Data Structures

<b>Introduction</b>	<b>477</b>
<b>17 <u>Augmenting Data Structures</u></b>	<b>480</b>
17.1 Dynamic order statistics	480
17.2 How to augment a data structure	486
17.3 Interval trees	489
<b>18 <u>B-Trees</u></b>	<b>497</b>
18.1 Definition of B-trees	501
18.2 Basic operations on B-trees	504
18.3 Deleting a key from a B-tree	513
<b>19 <u>Data Structures for Disjoint Sets</u></b>	<b>520</b>
19.1 Disjoint-set operations	520
19.2 Linked-list representation of disjoint sets	523
19.3 Disjoint-set forests	527
★ 19.4 Analysis of union by rank with path compression	531

## VI Graph Algorithms

<b>Introduction</b>	<b>547</b>
<b>20 <u>Elementary Graph Algorithms</u></b>	<b>549</b>
20.1 Representations of graphs	549
20.2 Breadth-first search	554
20.3 Depth-first search	563
20.4 Topological sort	573
20.5 Strongly connected components	576
<b>21 <u>Minimum Spanning Trees</u></b>	<b>585</b>
21.1 Growing a minimum spanning tree	586
21.2 The algorithms of Kruskal and Prim	591
<b>22 <u>Single-Source Shortest Paths</u></b>	<b>604</b>
22.1 The Bellman-Ford algorithm	612
22.2 Single-source shortest paths in directed acyclic graphs	616
22.3 Dijkstra's algorithm	620
22.4 Difference constraints and shortest paths	626
22.5 Proofs of shortest-paths properties	633

Table of Contents

23 **All-Pairs Shortest Paths** 646

23.1 Shortest paths and matrix multiplication 648

23.2 The Floyd-Warshall algorithm 655

23.3 Johnson’s algorithm for sparse graphs 662

24 **Maximum Flow** 670

24.1 Flow networks 671

24.2 The Ford-Fulkerson method 676

24.3 Maximum bipartite matching 693

25 **Matchings in Bipartite Graphs** 704

25.1 Maximum bipartite matching (revisited) 705

25.2 The stable-marriage problem 716

25.3 The Hungarian algorithm for the assignment problem 723

VII Selected Topics

Introduction 745

26 **Parallel Algorithms** 748

26.1 The basics of fork-join parallelism 750

26.2 Parallel matrix multiplication 770

26.3 Parallel merge sort 775

27 **Online Algorithms** 791

27.1 Waiting for an elevator 792

27.2 Maintaining a search list 795

27.3 Online caching 802

28 **Matrix Operations** 819

28.1 Solving systems of linear equations 819

28.2 Inverting matrices 833

28.3 Symmetric positive-definite matrices and least-squares approximation 838

29 **Linear Programming** 850

29.1 Linear programming formulations and algorithms 853

29.2 Formulating problems as linear programs 860

29.3 Duality 866

30 **Polynomials and the FFT** 877

30.1 Representing polynomials 879

30.2 The DFT and FFT 885

30.3 FFT circuits 894

# Table of Contents

<b>31</b>	<b>Number-Theoretic Algorithms</b>	<b>903</b>
31.1	Elementary number-theoretic notions	904
31.2	Greatest common divisor	911
31.3	Modular arithmetic	916
31.4	Solving modular linear equations	924
31.5	The Chinese remainder theorem	928
31.6	Powers of an element	932
31.7	The RSA public-key cryptosystem	936
★ 31.8	Primality testing	942
<b>32</b>	<b>String Matching</b>	<b>957</b>
32.1	The naive string-matching algorithm	960
32.2	The Rabin-Karp algorithm	962
32.3	String matching with finite automata	967
★ 32.4	The Knuth-Morris-Pratt algorithm	975
32.5	Suffix arrays	985
<b>33</b>	<b>Machine-Learning Algorithms</b>	<b>1003</b>
33.1	Clustering	1005
33.2	Multiplicative-weights algorithms	1015
33.3	Gradient descent	1022
<b>34</b>	<b><u>NP-Completeness</u></b>	<b>1042</b>
34.1	Polynomial time	1048
34.2	Polynomial-time verification	1056
34.3	NP-completeness and reducibility	1061
34.4	NP-completeness proofs	1072
34.5	NP-complete problems	1080
<b>35</b>	<b><u>Approximation Algorithms</u></b>	<b>1104</b>
35.1	The vertex-cover problem	1106
35.2	The traveling-salesperson problem	1109
35.3	The set-covering problem	1115
35.4	Randomization and linear programming	1119
35.5	The subset-sum problem	1124

## VIII Appendix: Mathematical Background

	<b>Introduction</b>	<b>1139</b>
<b>A</b>	<b>Summations</b>	<b>1140</b>
A.1	Summation formulas and properties	1140
A.2	Bounding summations	1145



# Analyzing Algorithms

## § Run Time Analysis

- Order of growth
- Worst case analysis
- Average case analysis

## § Insertion Sort for Array $A[i]$

- idea: insert  $A[i]$  into **sorted subarrays**:  $A[1 : i-1]$
- repeat insertion until  $A[i]$  is fully sorted

INSERTION-SORT( $A, n$ )	<i>cost</i>	<i>times</i>
1 <b>for</b> $i = 2$ <b>to</b> $n$	$c_1$	$n$
2 $key = A[i]$	$c_2$	$n - 1$
3       // Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$ .	0	$n - 1$
4 $j = i - 1$	$c_4$	$n - 1$
5 <b>while</b> $j > 0$ and $A[j] > key$	$c_5$	$\sum_{i=2}^n t_i$
6 $A[j + 1] = A[j]$	$c_6$	$\sum_{i=2}^n (t_i - 1)$
7 $j = j - 1$	$c_7$	$\sum_{i=2}^n (t_i - 1)$
8 $A[j + 1] = key$	$c_8$	$n - 1$

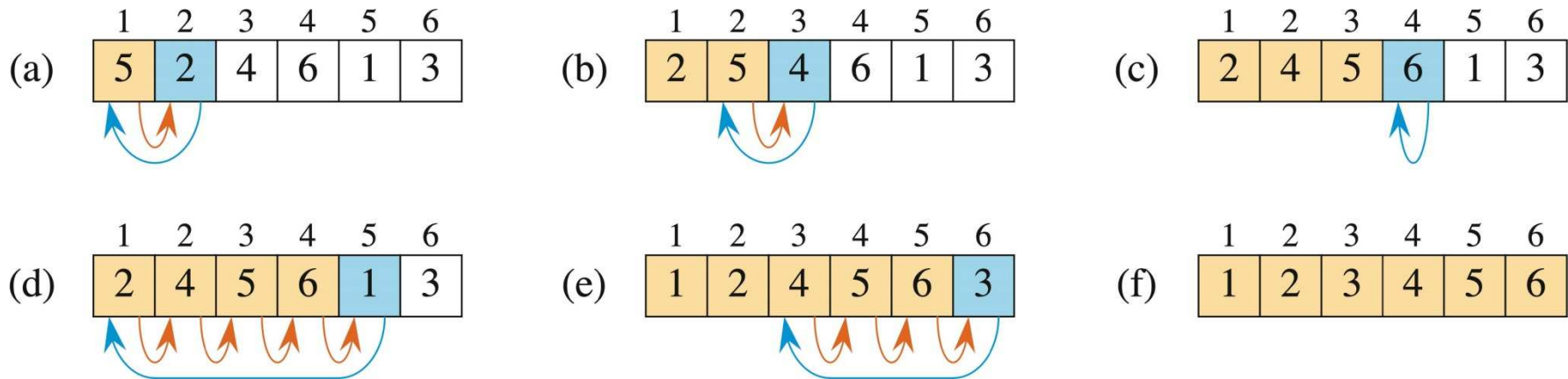
Insert  $A(i)$  properly

**Complexity:**  $T(n) = \sum_{i=1}^8 c_i$

# Analyzing Algorithms (continued)

## § Insertion Sort for Array $A[i]$

- idea: insert  $A[i]$  into **sorted subarrays**:  $A[1 : i-1]$
- repeat insertion until  $A[i]$  is fully sorted



**Worst-Case Complexity:**

$$T(n) = \sum_{i=1}^8 c_i = O(n^2)$$

INSERTION-SORT( $A, n$ )

	<i>cost</i>	<i>times</i>
1 <b>for</b> $i = 2$ <b>to</b> $n$	$c_1$	$n$
2 $key = A[i]$	$c_2$	$n - 1$
3     // Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$ .	0	$n - 1$
4 $j = i - 1$	$c_4$	$n - 1$
5 <b>while</b> $j > 0$ and $A[j] > key$	$c_5$	$\sum_{i=2}^n t_i$
6 $A[j + 1] = A[j]$	$c_6$	$\sum_{i=2}^n (t_i - 1)$
7 $j = j - 1$	$c_7$	$\sum_{i=2}^n (t_i - 1)$
8 $A[j + 1] = key$	$c_8$	$n - 1$

# Designing Algorithms

## § Divide-and-Conquer Approaches with Recursive Nature

- Divide the problem
- Conquer subproblems recursively
- Combine solutions to subproblems

## § Example: Merge Sort

- two sorted subarrays:  $A[p \dots q]$  &  $A[q+1 \dots r]$
- merge the two sorted subarrays
- merging takes  $\Theta(n)$  time

conquer separately

```

MERGE-SORT( $A, p, r$ )
1  if  $p \geq r$                                 // zero or one element?
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$                     // midpoint of  $A[p:r]$ 
4  MERGE-SORT( $A, p, q$ )                        // recursively sort  $A[p:q]$ 
5  MERGE-SORT( $A, q + 1, r$ )                    // recursively sort  $A[q + 1:r]$ 
6  // Merge  $A[p:q]$  and  $A[q + 1:r]$  into  $A[p:r]$ .
7  MERGE( $A, p, q, r$ )
  
```

```

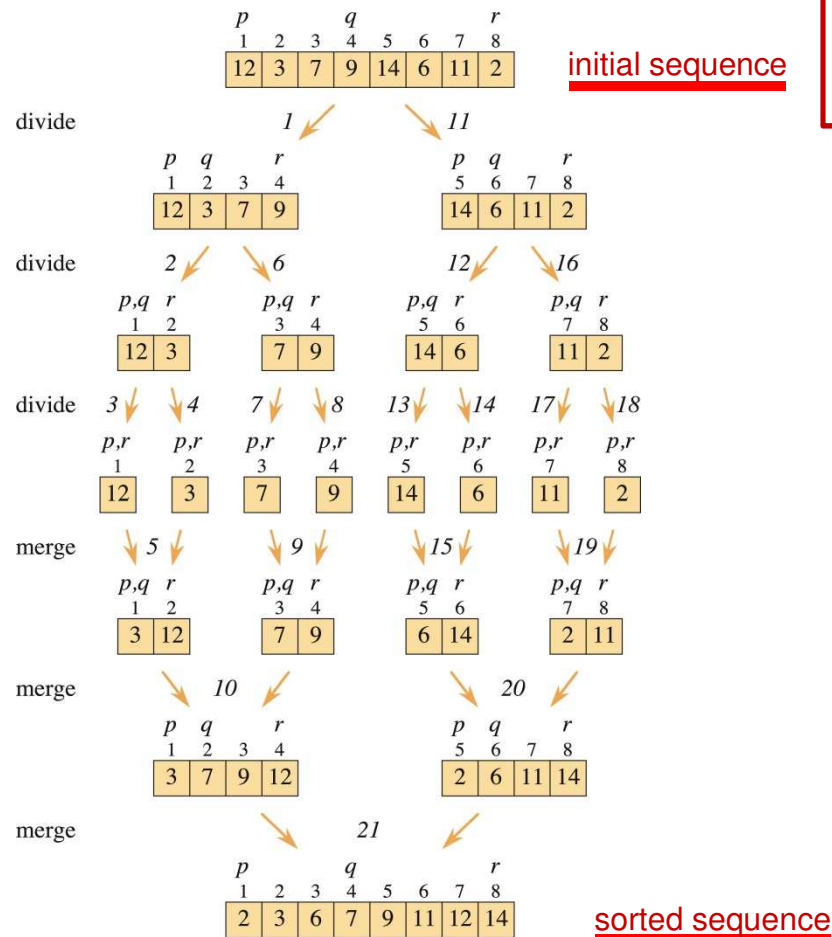
MERGE( $A, p, q, r$ )
1   $n_L = q - p + 1$  // length of  $A[p:q]$ 
2   $n_R = r - q$  // length of  $A[q + 1:r]$ 
3  let  $L[0:n_L - 1]$  and  $R[0:n_R - 1]$  be new arrays
4  for  $i = 0$  to  $n_L - 1$  // copy  $A[p:q]$  into  $L[0:n_L - 1]$ 
5       $L[i] = A[p + i]$ 
6  for  $j = 0$  to  $n_R - 1$  // copy  $A[q + 1:r]$  into  $R[0:n_R - 1]$ 
7       $R[j] = A[q + j + 1]$ 
8   $i = 0$  //  $i$  indexes the smallest remaining element in  $L$ 
9   $j = 0$  //  $j$  indexes the smallest remaining element in  $R$ 
10  $k = p$  //  $k$  indexes the location in  $A$  to fill
11 // As long as each of the arrays  $L$  and  $R$  contains an unmerged element,
    // copy the smallest unmerged element back into  $A[p:r]$ .
12 while  $i < n_L$  and  $j < n_R$ 
13     if  $L[i] \leq R[j]$ 
14          $A[k] = L[i]$ 
15          $i = i + 1$ 
16     else  $A[k] = R[j]$ 
17          $j = j + 1$ 
18      $k = k + 1$ 
19 // Having gone through one of  $L$  and  $R$  entirely, copy the
    // remainder of the other to the end of  $A[p:r]$ .
20 while  $i < n_L$ 
21      $A[k] = L[i]$ 
22      $i = i + 1$ 
23      $k = k + 1$ 
24 while  $j < n_R$ 
25      $A[k] = R[j]$ 
26      $j = j + 1$ 
27      $k = k + 1$ 
  
```

# Analyzing Algorithms

## § Analysis of Divide-and-Conquer Algorithms

- Merge Sort
- Time complexity:

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n/2) + c_2 \cdot (n) & \text{if } n > 1 \end{cases}$$



MERGE-SORT( $A, p, r$ )

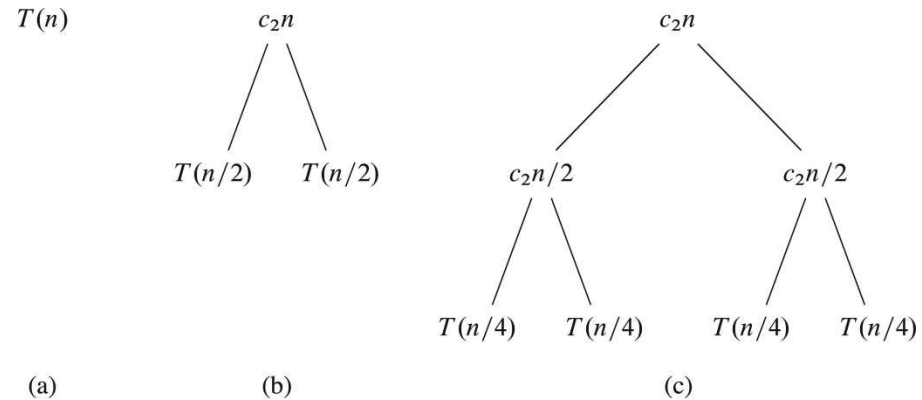
```

if  $p < r$                                 // check for base case
     $q = \lfloor (p + r) / 2 \rfloor$                 // divide
    MERGE-SORT( $A, p, q$ )                    // conquer
    MERGE-SORT( $A, q + 1, r$ )                // conquer
    MERGE( $A, p, q, r$ )                     // combine
  
```

# Analyzing Algorithms (continued)

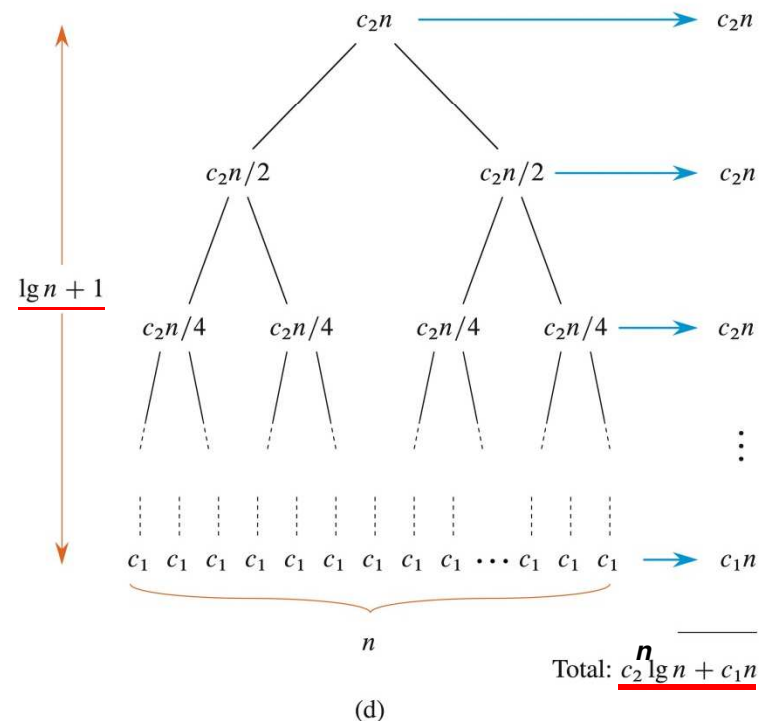
## § Evaluating: $2T(n/2) + c_2 \cdot (n)$

- Recursion tree, shown right, equal to  $c_2 n \cdot \lg(n) + c_1 n = \Theta(n \cdot \lg n)$



## Another approach for solution:

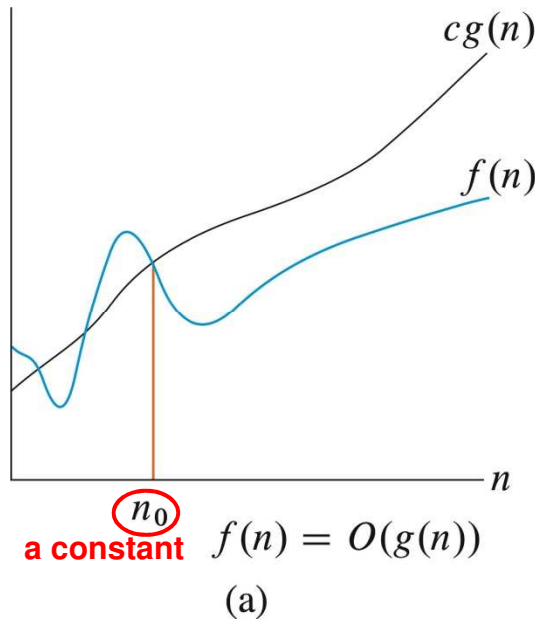
$$\begin{aligned}
 T(n) &= 2T(n/2) + c_2 \cdot (n) \\
 &= 2(2T(n/4) + c_2 \cdot (n/2)) + c_2 \cdot (n) \\
 &= 2^2 T(n/4) + 2 \cdot c_2 \cdot (n) \\
 &= 2^2 (2T(n/8) + c_2 \cdot (n/4)) + 2 \cdot c_2 \cdot (n) \\
 &= 2^3 T(n/8) + 3 \cdot c_2 \cdot (n) \\
 &\dots \\
 &= n \cdot T(1) + \lg(n) \cdot c_2 \cdot (n) \\
 &= n \cdot c_1 + c_2 \cdot (n) \lg(n)
 \end{aligned}$$



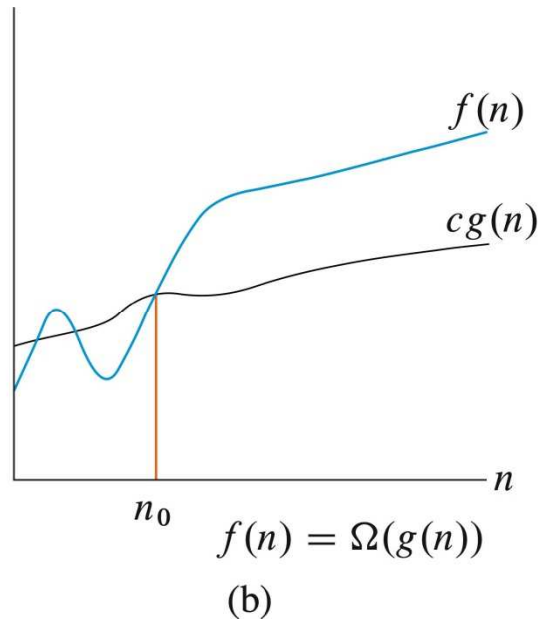
# Growth of Functions

## § Asymptotic Notations of Running Times

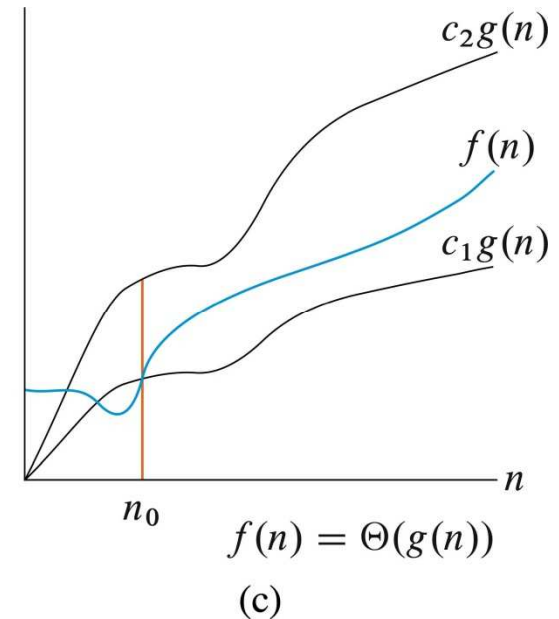
- $O$ -notation: **upper-bounding** a function to within a constant factor
- $\Omega$ -notation: **lower-bounding** a function to within a constant factor
- $\Theta$ -notation: **bounding** a function to **within** constant factors



$g(n)$  is an asymptotical  
upper bound for  $f(n)$   
(may or may not be tight)



$g(n)$  is an asymptotical  
lower bound for  $f(n)$   
(may or may not be tight)



$g(n)$  is an asymptotically  
tight bound for  $f(n)$

There exist positive constants  $c_1$ ,  $c_2$ , and  $n_0$  s.t.  $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$  for all  $n \geq n_0$

# Growth of Functions (continued)

- $o$ -notation: not asymptotically-tight upper-bound

## $o$ -notation

$o(g(n)) = \{f(n) : \text{for any constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}.$

Another view, probably **easier to use**:  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$

$$n^{1.9999} = o(n^2)$$

$$n^2 / \lg n = o(n^2)$$

$$n^2 \neq o(n^2) \text{ (just like } 2 \not\leq 2)$$

$$n^2 / 1000 \neq o(n^2)$$

- $\omega$ -notation: not asymptotically-tight lower-bound

## $\omega$ -notation

$\omega(g(n)) = \{f(n) : \text{for any constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}$

Another view, again, probably **easier to use**:  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty.$

$$n^{2.0001} = \omega(n^2)$$

$$n^2 \lg n = \omega(n^2)$$

$$n^2 \neq \omega(n^2)$$



# Solutions after Divide-and-Conquer

## § Divide-and-Conquer

- leads to recurrences in various forms
- there are 3 kinds of methods for solving recurrences
  - + **substitution** methods
  - + **recursion-tree** methods
  - + **master methods** to find bounds for recurrences of  $T(n) = a \cdot T(n/b) + f(n)$ , with  $a > 0$  and  $b > 1$

## § Example: Multiplying two square matrices sized $n \times n$

- $C = A \cdot B$
- divide each  $n \times n$  matrix into four  $n/2 \times n/2$  submatrices for multiplying:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \text{ and } B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \text{ to get } C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}, \text{ with}$$

$$\begin{aligned} \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \\ &= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix}. \end{aligned}$$

Hence, we have

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}; \quad C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}; \quad C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$



# Divide-and-Conquer (continued)

## § Multiplying two square matrices

– solution involves eight **MATRIX-MULTIPLY RECURSIVE** calls

MATRIX-MULTIPLY-RECURSIVE( $A, B, C, n$ )

```

1  if  $n == 1$ 
2    // Base case.
3       $c_{11} = c_{11} + a_{11} \cdot b_{11}$ 
4      return
5    // Divide.
6    partition  $A, B$ , and  $C$  into  $n/2 \times n/2$  submatrices
        $A_{11}, A_{12}, A_{21}, A_{22}; B_{11}, B_{12}, B_{21}, B_{22};$ 
       and  $C_{11}, C_{12}, C_{21}, C_{22}$ ; respectively
7    // Conquer.
8    MATRIX-MULTIPLY-RECURSIVE( $A_{11}, B_{11}, C_{11}, n/2$ )
9    MATRIX-MULTIPLY-RECURSIVE( $A_{11}, B_{12}, C_{12}, n/2$ )
10   MATRIX-MULTIPLY-RECURSIVE( $A_{21}, B_{11}, C_{21}, n/2$ )
11   MATRIX-MULTIPLY-RECURSIVE( $A_{21}, B_{12}, C_{22}, n/2$ )
12   MATRIX-MULTIPLY-RECURSIVE( $A_{12}, B_{21}, C_{11}, n/2$ )
13   MATRIX-MULTIPLY-RECURSIVE( $A_{12}, B_{22}, C_{12}, n/2$ )
14   MATRIX-MULTIPLY-RECURSIVE( $A_{22}, B_{21}, C_{21}, n/2$ )
15   MATRIX-MULTIPLY-RECURSIVE( $A_{22}, B_{22}, C_{22}, n/2$ )

```

The time complexity of this procedure:

$$T(n) = D(n) + 8T(n/2) + \Theta(1)$$

$$= \underline{\Theta(n^3)}.$$

# Divide-and-Conquer (continued)

## § Substitution Methods: two steps involved

- guess the solution form
- mathematic induction to validate the constants

Substitution method for proving an upper bound on the recurrence of  $T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$  being  $T(n) \leq c \cdot n \cdot \lg n$  for a constant  $c > 0$ .

This is due to composing the full solution.

This is obtained by guessing its solution to be  $T(n) = O(n \cdot \lg n)$  and then substituting  $T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \cdot \lg(\lfloor n/2 \rfloor)$  into the recurrence:

$$\begin{aligned}
 T(n) &\leq 2(c \cdot \lfloor n/2 \rfloor \cdot \lg(\lfloor n/2 \rfloor)) + \Theta(n) \\
 &\leq c \cdot n \cdot \lg(\lfloor n/2 \rfloor) + \Theta(n) \\
 &\leq c \cdot n \cdot \lg(n) - c \cdot n \cdot \lg(2) + \Theta(n) \\
 &\leq c \cdot n \cdot \lg n, \text{ for } c \geq 1
 \end{aligned}$$

Similar substitution method for proving an upper bound on recurrence

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1),$$

which equals  $O(n)$ .

This is due to composing the full solution.

# Divide-and-Conquer (continued)

## § Substitution Methods

– changing variables and/or function renaming

Substitution method for proving:  $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$

Rename  $m = \lg n$  (and ignore rounding) to get  $T(n)$  (after parameter renaming):

$$T(n) = T(2^m) = 2T(2^{m/2}) + m$$

Further renaming  $T(2^m)$  as  $S(m)$ , we have (function renaming)

$S(m) = 2S(m/2) + m$ , which has the solution of

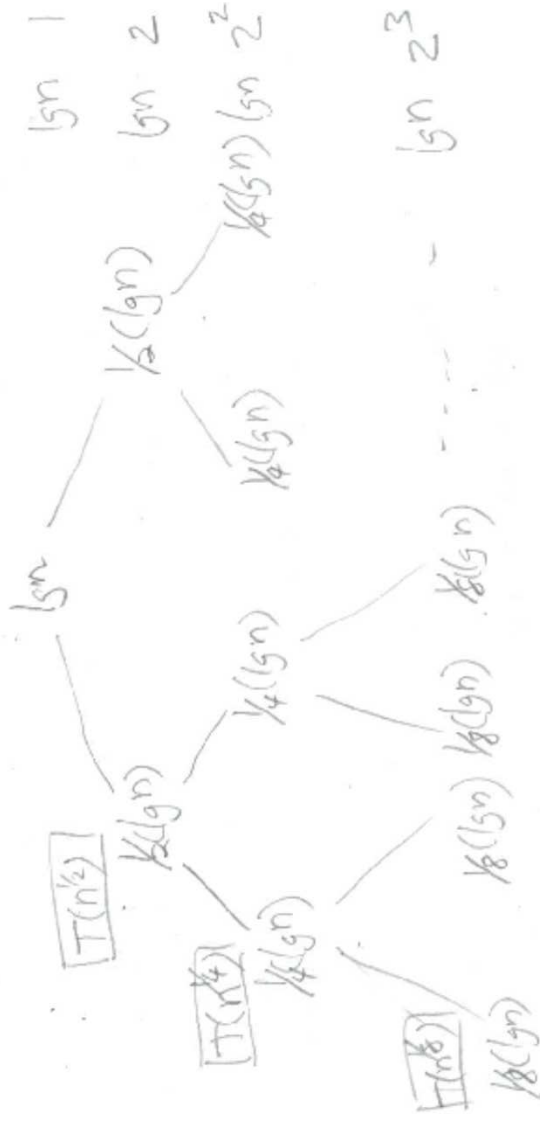
$$S(m) = O(m \cdot \lg m) \leftarrow \text{via the prior result}$$

We thus have  $T(n) = T(2^m) = S(m) = O(m \cdot \lg m)$   
 $= O(\lg n \cdot \lg \lg n)$ .

Note: this problem can also be solved by the recursion-tree method, described next.

Recursion tree approach for solving

$$T(n) = 2T(\sqrt{n}) + \lg n$$

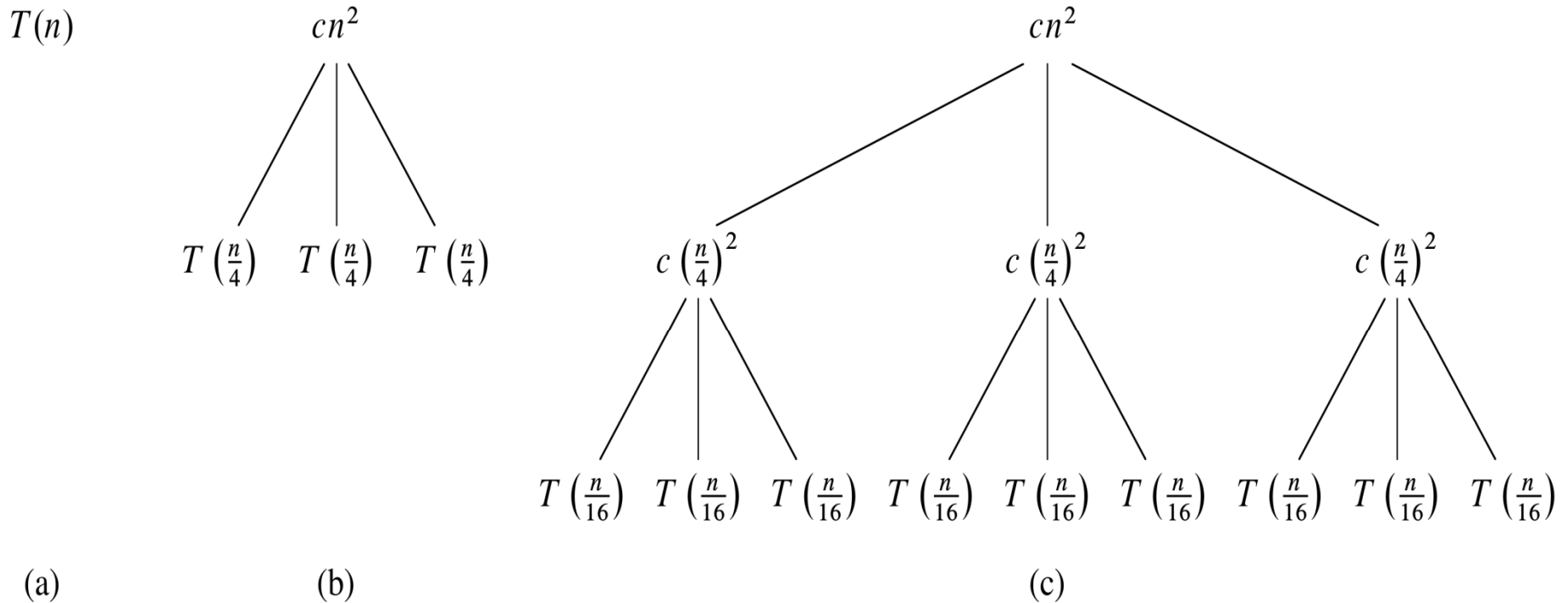


# Divide-and-Conquer (continued)

## § Recursion-Tree Methods

- best for generating good complexity bounds in general
- two examples given below

For recurrence:  $T(n) = 3T(n/4) + \Theta(n^2)$



# Divide-and-Conquer

For recurrence:  $T(n) = 3T(n/4) + \Theta(n^2)$



Total:  $O(n^2)$

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) = \frac{1}{1 - (\frac{3}{16})} cn^2 + \Theta(n^{\log_4 3}) = O(n^2). \end{aligned}$$

Use the property of:  $v^{a \cdot b} = (v^a)^b = (v^b)^a$ .

Let  $3^{\log_4 n} = x$ .

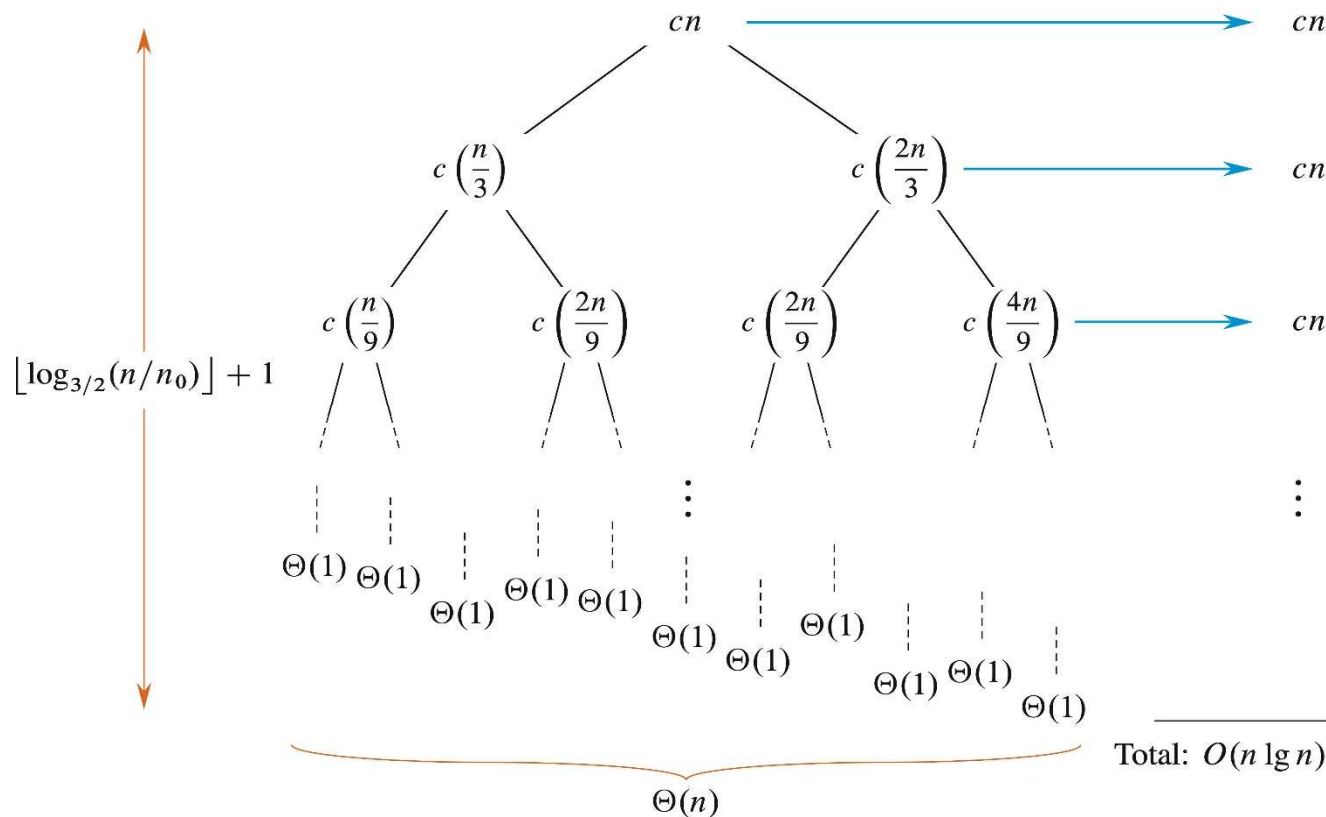
After taking  $\log_4$  on both sides, we have:  $(\log_4 n) \cdot (\log_4 3) = \log_4 x$ .

We then take a power of 4 on both sides to yield:  $4^{\log_4 n \cdot \log_4 3} = x$ , which becomes:

$[4^{\log_4 n}]^{\log_4 3} = [n]^{\log_4 3} = x$ . Hence,  $3^{\log_4 n} = n^{\log_4 3}$ .

# Divide-and-Conquer (continued)

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



**Longest path:**  $cn \rightarrow c\left(\frac{2}{3}\right)n \rightarrow c\left(\frac{2}{3}\right)^2 n \rightarrow c\left(\frac{2}{3}\right)^3 n \rightarrow \dots \rightarrow 1$ , we have:  $k = \log_{3/2} n$ , as  $\left(\frac{2}{3}\right)^k n = 1$

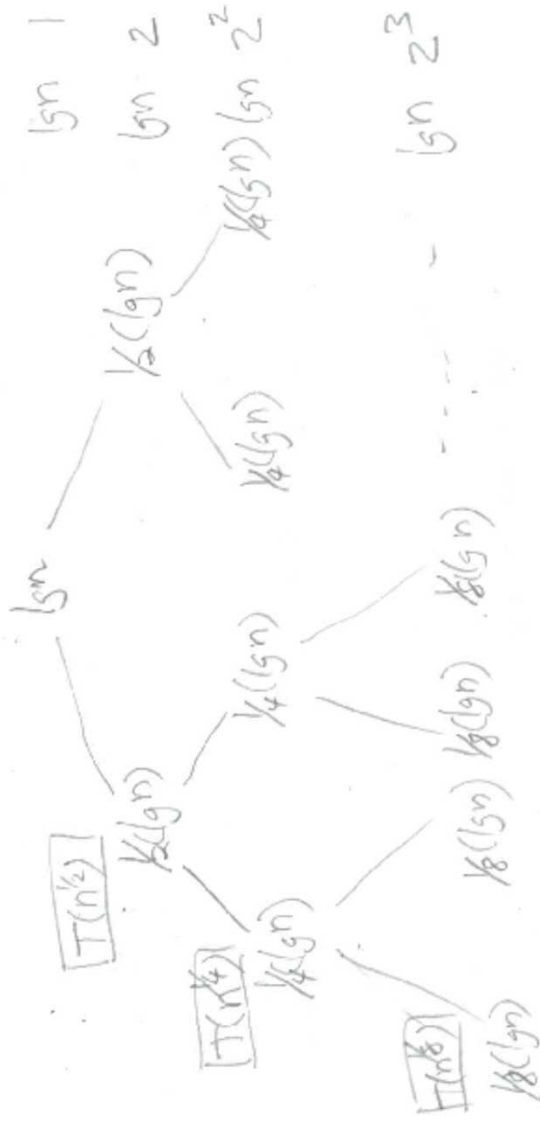
**Shortest path:**  $cn \rightarrow c\left(\frac{1}{3}\right)n \rightarrow c\left(\frac{1}{3}\right)^2 n \rightarrow c\left(\frac{1}{3}\right)^3 n \rightarrow \dots \rightarrow 1$  to get  $k = \log_3 n$ ,  $\sim (\log_{3/2} n)/2.7$

Similarly, one may show  $T(n)$  upper bounded by  $O(n \cdot \lg n)$  via **substitution method**.



Recursion tree approach for solving

$$T(n) = 2T(\sqrt{n}) + \lg n$$



# Master Method for Solving Recurrences

Recurrences of  $T(n) = a \cdot T(n/b) + f(n)$  with constants  $a > 0$  and  $b > 1$   
and  $f(n)$  nonnegative function that covers work on  
dividing the problem and on combining subproblems' results

## Theorem 4.1

$T(n) = a \cdot T(n/b) + f(n)$  has following asymptotical bounds:

1. for  $f(n) = O(n^{\log_b a - \epsilon})$  with constant  $\epsilon > 0$ , then  $T(n) = \underline{\Theta(n^{\log_b a})}$
2. for  $f(n) = \Theta(n^{\log_b a} \cdot \lg^k n)$  with constant  $k \geq 0$ , then  $T(n) = \underline{\Theta(n^{\log_b a} \cdot \lg^{k+1} n)}$
3. for  $f(n) = \Omega(n^{\log_b a + \epsilon})$  with constant  $\epsilon > 0$  and  $a \cdot f(n/b) \leq c \cdot f(n)$ , then  $T(n) = \underline{\Theta(f(n))}$

In the recursion-tree, 2<sup>nd</sup> level sums to **no more than** 1<sup>st</sup> level &  $f(n)$  is *polynomially larger*

**Note:** bound is the **larger** of the two:  $f(n)$  and  $n^{\log_b a}$

In Case 1,  $n^{\log_b a}$  is *polynomially larger* than  $f(n)$

In Case 2 with  $k = 0$  most commonly,  $n^{\log_b a}$  and  $f(n)$  are of the same size

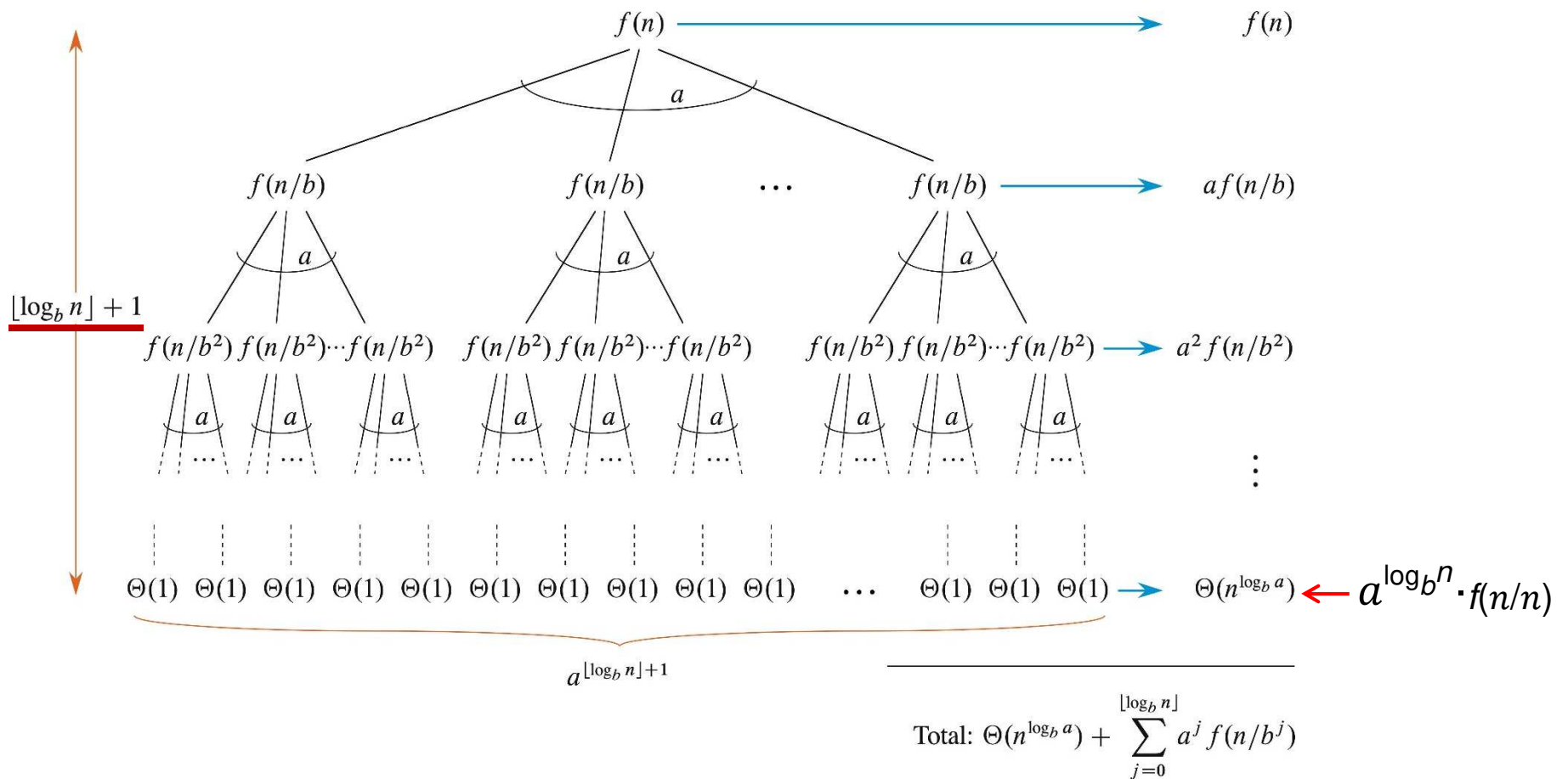
In Case 3,  $f(n)$  is *polynomially larger* than  $n^{\log_b a}$

## Master Method (continued)

Let  $T(n) = a \cdot T(n/b) + f(n)$  with constant  $a > 0$  and  $n \geq 1$  being an **exact power** of  $b (> 1)$ .  
We have:

### **Lemma 4.2**

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\lfloor \log_b n \rfloor} a^j \cdot f(n/b^j)$$



# Master Method (continued)

Example Recurrences  $T(n) = a \cdot T(n/b) + f(n)$  solved by the master method:

1. for  $f(n) = O(n^{\log_b a - \epsilon})$  with constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$
  2. for  $f(n) = \Theta(n^{\log_b a} \cdot \lg^k n)$ , then  $T(n) = \Theta(n^{\log_b a} \cdot \lg^{k+1} n)$ , with  $k = 0$  most common
  3. for  $f(n) = \Omega(n^{\log_b a + \epsilon})$  with constant  $\epsilon > 0$  and  $a \cdot f(n/b) \leq c \cdot f(n)$ , then  $T(n) = \Theta(f(n))$
- 

$$T(n) = 9T(n/3) + n$$

Here,  $a = 9$ ,  $b = 3$ , and  $f(n) = n$

From  $n^{\log_3 9} = n^2$ , we have  $f(n) = n = O(n^{\log_3 9 - 1})$  to get  $T(n) = \Theta(n^2)$

$$T(n) = T(2n/3) + 1$$

Here,  $a = 1$ ,  $b = 3/2$ , and  $f(n) = 1$

From  $n^{\log_{3/2} 1} = n^0$ , we have  $f(n) = 1 = O(n^{\log_{3/2} 1})$  to get  $T(n) = \Theta(\lg n)$

$$T(n) = 3T(n/4) + n \lg n$$

Here,  $a = 3$ ,  $b = 4$ , and  $f(n) = n \lg n$

From  $n^{\log_4 3} = O(n^{0.793})$ , we have  $f(n) = n \lg n = \Omega(n^{\log_4 3 + \epsilon})$  to get  $T(n) = \Theta(n \lg n)$

# Master Method (continued)

Example Recurrences  $T(n) = a \cdot T(n/b) + f(n)$ :

1. for  $f(n) = O(n^{\log_b a - \epsilon})$  with constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$
2. for  $f(n) = \Theta(n^{\log_b a} \cdot \lg^k n)$ , then  $T(n) = \Theta(n^{\log_b a} \cdot \lg^{k+1} n)$ , with  $k = 0$  most common
3. for  $f(n) = \Omega(n^{\log_b a + \epsilon})$  with constant  $\epsilon > 0$  and  $a \cdot f(n/b) \leq c \cdot f(n)$ , then  $T(n) = \Theta(f(n))$

---

$f(n)$  is polynomially larger

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

Here,  $a = 2$ ,  $b = 2$ , and  $f(n) = n \lg n$

From  $n^{\log_2 2} = n$ , we have  $f(n) = n \lg n > n^{\log_2 2}$  but **not polynomially**  $> n^{\log_2 2}$   
(use substitution or recursion-tree to solve this)

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

Here,  $a = 7$ ,  $b = 2$ , and  $f(n) = \Theta(n^2)$

From  $n^{\log_2 7} = n^{2.8+}$ , we have  $f(n) = O(n^{\log_2 7 - \epsilon})$  to get  $T(n) = \Theta(n^{\log_2 7})$

# Master Method (continued)

## **Lemma 4.3**

Given  $g(n) = \sum_{j=0}^{\lfloor \log_b n \rfloor} a^j \cdot f(n/b^j)$  for  $a > 0$  and  $n$  an **exact power** of  $b$  ( $> 1$ ), we have:

1. for  $f(n) = O(n^{\log_b a - \epsilon})$  with constant  $\epsilon > 0$ , then  $g(n) = \Theta(n^{\log_b a})$
2. for  $f(n) = \Theta(n^{\log_b a} \cdot \lg^k n)$ , then  $g(n) = \Theta(n^{\log_b a} \cdot \lg^{k+1} n)$ , with  $k = 0$  most common
3. if  $a \cdot f(n/b) \leq c \cdot f(n)$  for constant  $c < 1$  and for sufficiently large  $n$ ,  $g(n) = \Theta(f(n))$