

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

INTRODUCTION TO

ALGORITHMS

THIRD EDITION

CSCE 500

Design and Analysis of Algorithms

Fall 2021

August 20, 2021

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

Textbook and Supplemental Materials:

1. Introduction to Algorithms, Third Edition, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, The MIT Press, 2009, ISBN: 978-0-262-03384-8.
2. Published articles supplementary to covered topics.

Course Description:

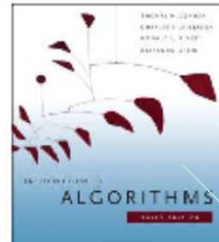
This course provides a comprehensive coverage of modern computer algorithms, aiming at in-depth treatment of algorithmic design and analysis with elementary explanation while keeping mathematical rigor. Based on the textbook of “Introduction to Algorithms”, this class covers the topics listed below in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

- (1) Foundations.
- (2) Data Structures – hash tables, trees, heaps.
- (3) Design and Analysis Techniques – dynamic programming, greedy algorithms, amortized analysis.
- (4) Graph Algorithms – spanning trees, shortest paths, maximum flow.
- (5) Selected Topics – NP-completeness, approximation algorithms, multithreaded algorithms, string matching.

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%)

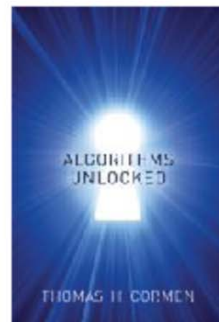


Hardcover | \$92.00
Text | £63.95 | ISBN:
 9780262033848 | 1312
 pp. | 8 x 9 in | 235 b&w
 illus. | July 2009

Paperback | \$70.00
Text | £43.95 | ISBN:
 9780262533058 | 1312
 pp. | 8 x 9 in | 235 b&w
 illus. | July 2009

Paperback edition is not
 for sale in the U.S. or
 Canada.

Of Related Interest



Algorithms Unlocked

Introduction to Algorithms, third edition

By Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein

Table of Contents

I Foundations

Introduction 3

1 The Role of Algorithms in Computing 5

1.1 Algorithms 5

1.2 Algorithms as a technology 11

2 Getting Started 16

2.1 Insertion sort 16

2.2 Analyzing algorithms 23

2.3 Designing algorithms 29

3 Growth of Functions 43

3.1 Asymptotic notation 43

3.2 Standard notations and common functions 53

4 Divide-and-Conquer 55

4.1 The maximum-subarray problem 68

4.2 Strassen's algorithm for matrix multiplication 75

4.3 The substitution method for solving recurrences 83

4.4 The recursion-tree method for solving recurrences 88

4.5 The master method for solving recurrences 93

4.6 Proof of the master theorem 97

5 Probabilistic Analysis and Randomized Algorithms 114

5.1 The hiring problem 114

5.2 Indicator random variables 118

5.3 Randomized algorithms 122

5.4 Probabilistic analysis and further uses of indicator random variables 130

II Sorting and Order Statistics

Introduction 147

6 Heapsort 151

6.1 Heaps 151

6.2 Maintaining the heap property 154

6.3 Building a heap 156

6.4 The heapsort algorithm 159

6.5 Priority queues 162

7 Quicksort 170

7.1 Description of quicksort 170

7.2 Performance of quicksort 174

7.3 A randomized version of quicksort 179

7.4 Analysis of quicksort 180

8 Sorting in Linear Time 191

- 8.1 Lower bounds for sorting 191
- 8.2 Counting sort 194
- 8.3 Radix sort 197
- 8.4 Bucket sort 200
- 9 Medians and Order Statistics 213
 - 9.1 Minimum and maximum 214
 - 9.2 Selection in expected linear time 215
 - 9.3 Selection in worst-case linear time 220

III Data Structures

- Introduction 229
- 10 Elementary Data Structures 232
 - 10.1 Stacks and queues 232
 - 10.2 Linked lists 236
 - 10.3 Implementing pointers and objects 241
 - 10.4 Representing rooted trees 246
- 11 Hash Tables 253
 - 11.1 Direct-address tables 254
 - 11.2 Hash tables 256
 - 11.3 Hash functions 262
 - 11.4 Open addressing 269
 - 11.5 Perfect hashing 277
- 12 Binary Search Trees 286
 - 12.1 What is a binary search tree? 286
 - 12.2 Querying a binary search tree 289
 - 12.3 Insertion and deletion 294
 - 12.4 Randomly built binary search trees 299
- 13 Red-Black Trees 308
 - 13.1 Properties of red-black trees 308
 - 13.2 Rotations 312
 - 13.3 Insertion 315
 - 13.4 Deletion 323
- 14 Augmenting Data Structures 339
 - 14.1 Dynamic order statistics 339
 - 14.2 How to augment a data structure 345
 - 14.3 Interval trees 348

IV Advanced Design and Analysis Techniques

- Introduction 357
- 15 Dynamic Programming 359
 - 15.1 Rod cutting 360
 - 15.2 Matrix-chain multiplication 370
 - 15.3 Elements of dynamic programming 378
 - 15.4 Longest common subsequence 390
 - 15.5 Optimal binary search trees 397
- 16 Greedy Algorithms 414
 - 16.1 An activity-selection problem 415
 - 16.2 Elements of the greedy strategy 423
 - 16.3 Huffman codes 428
 - 16.4 Matroids and greedy methods 437

- 16.5 A task-scheduling problem as a matroid 443
- 17 **Amortized Analysis** 451
 - 17.1 Aggregate analysis 452
 - 17.2 The accounting method 456
 - 17.3 The potential method 459
 - 17.4 Dynamic tables 463

V Advanced Data Structures

- Introduction** 481
- 18 **B-Trees** 484
 - 18.1 Definition of B-trees 488
 - 18.2 Basic operations on B-trees 491
 - 18.3 Deleting a key from a B-tree 499
- 19 **Fibonacci Heaps** 505
 - 19.1 Structure of Fibonacci heaps 507
 - 19.2 Mergeable-heap operations 510
 - 19.3 Decreasing a key and deleting a node 518
 - 19.4 Bounding the maximum degree 523
- 20 **van Emde Boas Trees** 531
 - 20.1 Preliminary approaches 532
 - 20.2 A recursive structure 536
 - 20.3 The van Emde Boas tree 545
- 21 **Data Structures for Disjoint Sets** 561
 - 21.1 Disjoint-set operations 561
 - 21.2 Linked-list representation of disjoint sets 564
 - 21.3 Disjoint-set forests 568
 - 21.4 Analysis of union by rank with path compression 573

VI Graph Algorithms

- Introduction** 587
- 22 **Elementary Graph Algorithms** 589
 - 22.1 Representations of graphs 589
 - 22.2 Breadth-first search 594
 - 22.3 Depth-first search 603
 - 22.4 Topological sort 612
 - 22.5 Strongly connected components 615
- 23 **Minimum Spanning Trees** 624
 - 23.1 Growing a minimum spanning tree 625
 - 23.2 The algorithms of Kruskal and Prim 631
- 24 **Single-Source Shortest Paths** 643
 - 24.1 The Bellman-Ford algorithm 651
 - 24.2 Single-source shortest paths in directed acyclic graphs 655
 - 24.3 Dijkstra's algorithm 658
 - 24.4 Difference constraints and shortest paths 664
 - 24.5 Proofs of shortest-paths properties 671
- 25 **All-Pairs Shortest Paths** 684
 - 25.1 Shortest paths and matrix multiplication 686
 - 25.2 The Floyd-Warshall algorithm 693
 - 25.3 Johnson's algorithm for sparse graphs 700
- 26 **Maximum Flow** 708

- 26.1 Flow networks 709
- 26.2 The Ford-Fulkerson method 714
- 26.3 Maximum bipartite matching 732
- 26.4 Push-relabel algorithms 736
- 26.5 The relabel-to-front algorithm 748

VII Selected Topics

Introduction 769

27 Multithreaded Algorithms

- 27.1 The basics of dynamic multithreading 774
- 27.2 Multithreaded matrix multiplication 792
- 27.3 Multithreaded merge sort 797

28 Matrix Operations 813

- 28.1 Solving systems of linear equations 813
- 28.2 Inverting matrices 827
- 28.3 Symmetric positive-definite matrices and least-squares

approximation 832

29 Linear Programming 843

- 29.1 Standard and slack forms 850
- 29.2 Formulating problems as linear programs 859
- 29.3 The simplex algorithm 864
- 29.4 Duality 879
- 29.5 The initial basic feasible solution 886

30 Polynomials and the FFT 898

- 30.1 Representing polynomials 900
- 30.2 The DFT and FFT 906
- 30.3 Efficient FFT implementations 915

31 Number-Theoretic Algorithms 926

- 31.1 Elementary number-theoretic notions 927
- 31.2 Greatest common divisor 933
- 31.3 Modular arithmetic 939
- 31.4 Solving modular linear equations 946
- 31.5 The Chinese remainder theorem 950
- 31.6 Powers of an element 954
- 31.7 The RSA public-key cryptosystem 958
- 31.8 Primality testing 965
- 31.9 Integer factorization 975

32 String Matching 985

- 32.1 The naive string-matching algorithm 988
- 32.2 The Rabin-Karp algorithm 990
- 32.3 String matching with finite automata 995
- 32.4 The Knuth-Morris-Pratt algorithm 1002

33 Computational Geometry 1014

- 33.1 Line-segment properties 1015
- 33.2 Determining whether any pair of segments intersects 1021
- 33.3 Finding the convex hull 1029
- 33.4 Finding the closest pair of points 1039

34 NP-Completeness 1048

- 34.1 Polynomial time 1053
- 34.2 Polynomial-time verification 1061

34.3	NP-completeness and reducibility	1067
34.4	NP-completeness proofs	1078
34.5	NP-complete problems	1086
35	Approximation Algorithms	1106
35.1	The vertex-cover problem	1108
35.2	The traveling-salesman problem	1111
35.3	The set-covering problem	1117
35.4	Randomization and linear programming	1123
35.5	The subset-sum problem	1128

VIII Appendix: Mathematical Background

Introduction		1143
A Summations		1145
A.1	Summation formulas and properties	1145
A.2	Bounding summations	1149
B Sets, Etc.		1158
B.1	Sets	1158
B.2	Relations	1163
B.3	Functions	1166
B.4	Graphs	1168
B.5	Trees	1173
C Counting and Probability		1183
C.1	Counting	1183
C.2	Probability	1189
C.3	Discrete random variables	1196
C.4	The geometric and binomial distributions	1201
C.5	The tails of the binomial distribution	1208
D Matrices		1217
D.1	Matrices and matrix operations	1217
D.2	Basic matrix properties	1222
Bibliography		1231
Index		

About the Authors

Thomas H. Cormen is Professor of Computer Science and former Director of the Institute for Writing and Rhetoric at Dartmouth College. He is the coauthor (with Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein) of the leading textbook on computer algorithms, *Introduction to Algorithms* (third edition, MIT Press, 2009).

Charles E. Leiserson is Professor of Computer Science and Engineering at the Massachusetts Institute of Technology.

Ronald L. Rivest is Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology.

Clifford Stein is Professor of Industrial Engineering and Operations Research at Columbia University.

Endorsements

"As an educator and researcher in the field of algorithms for over two decades, I can unequivocally say that the Cormen et al book is the best textbook that I have ever seen on this subject. It offers an incisive, encyclopedic, and modern treatment of algorithms, and our department will continue to use it for teaching at both the

Analyzing Algorithms

§ Run Time Analysis

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

§ Insertion Sort for Array $A[i]$

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

INSERTION-SORT(A, n)	<i>cost</i>	<i>times</i>
for $j = 2$ to n	c_1	n
$key = A[j]$	c_2	$n - 1$
// Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$.	0	$n - 1$
$i = j - 1$	c_4	$n - 1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^n t_j$
$A[i + 1] = A[i]$	c_6	$\sum_{j=2}^n (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^n (t_j - 1)$
$A[i + 1] = key$	c_8	$n - 1$

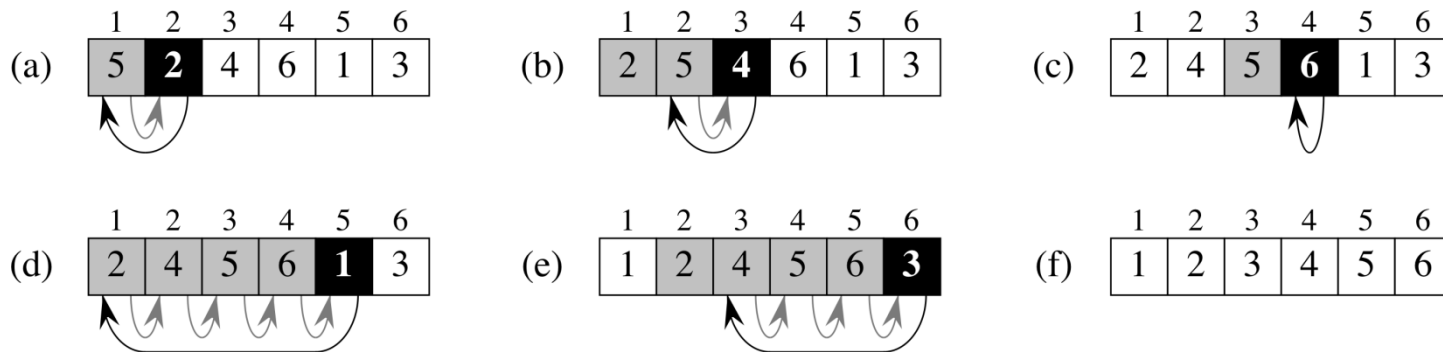
Insert $A(j)$ properly

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

Analyzing Algorithms (continued)

§ Insertion Sort for Array $A[j]$

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



Worst-Case Complexity:

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

INSERTION-SORT(A, n)

for $j = 2$ **to** n

$key = A[j]$

 // Insert $A[j]$ into the sorted sequence $A[1 \dots j-1]$.

$i = j - 1$

while $i > 0$ and $A[i] > key$

$A[i + 1] = A[i]$

$i = i - 1$

$A[i + 1] = key$

cost times

c_1 n

c_2 $n - 1$

0 $n - 1$

c_4 $n - 1$

c_5 $\sum_{j=2}^n t_j$

c_6 $\sum_{j=2}^n (t_j - 1)$

c_7 $\sum_{j=2}^n (t_j - 1)$

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

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
  
```

merge →

MERGE(A, p, q, r)

$n_1 = q - p + 1$

$n_2 = r - q$

let $L[1 \dots n_1 + 1]$ and $R[1 \dots n_2 + 1]$ be new arrays

for $i = 1$ **to** n_1

$L[i] = A[p + i - 1]$

for $j = 1$ **to** n_2

$R[j] = A[q + j]$

$L[n_1 + 1] = \infty$

$R[n_2 + 1] = \infty$ ← stoppers for half arrays

$i = 1$

$j = 1$

for $k = p$ **to** r

if $L[i] \leq R[j]$

$A[k] = L[i]$

$i = i + 1$

else $A[k] = R[j]$

$j = j + 1$

Analyzing Algorithms

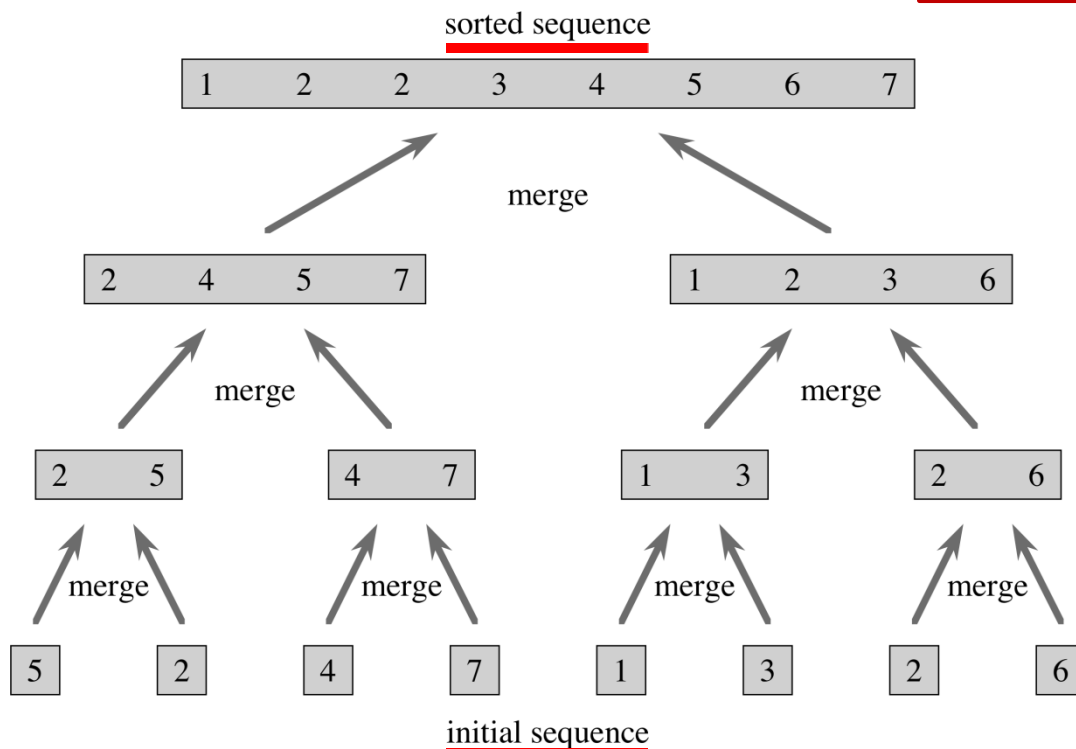
§ Analysis of Divide-and-Conquer Algorithms

- Merge Sort
- Time complexity:

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

```

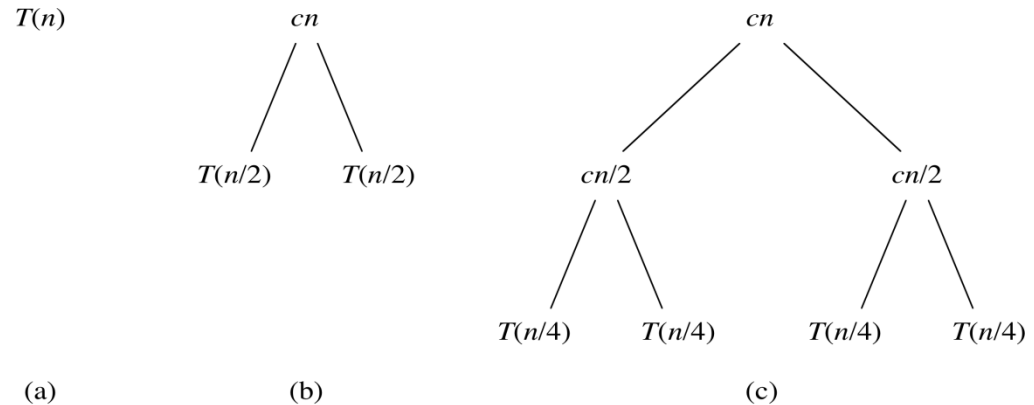
MERGE-SORT(A, p, r)
  if p < r                                // check for base case
    q = ⌊(p + r)/2⌋                       // divide
    MERGE-SORT(A, p, q)                   // conquer
    MERGE-SORT(A, q + 1, r)               // conquer
    MERGE(A, p, q, r)                     // combine
  
```



Analyzing Algorithms (continued)

§ Evaluating: $2T(n/2) + c \cdot (n)$

- Recursion tree, shown right
- $cn \cdot \lg(n) + cn = \Theta(n \cdot \lg n)$



Another approach for solution:

$$T(n) = 2T(n/2) + c \cdot (n)$$

$$= 2(2T(n/4) + c \cdot (n/2)) + c \cdot (n)$$

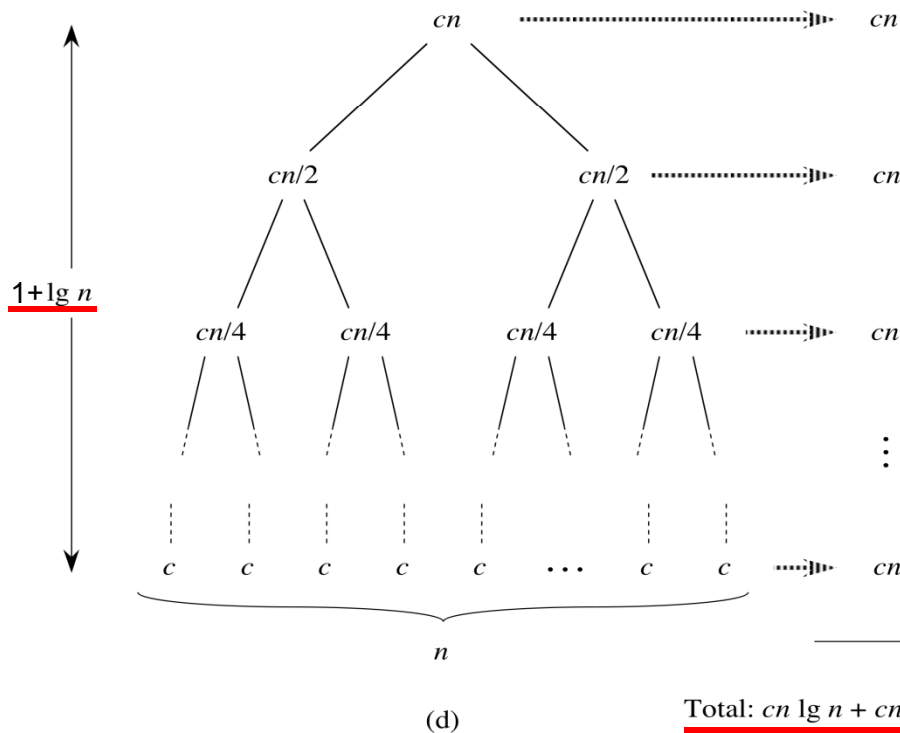
$$= 2^2 T(n/4) + 2 \cdot c \cdot (n)$$

$$= 2^2 (2T(n/8) + c \cdot (n/4)) + 2 \cdot c \cdot (n)$$

$$= 2^3 T(n/8) + 3 \cdot c \cdot (n)$$

.....
.....

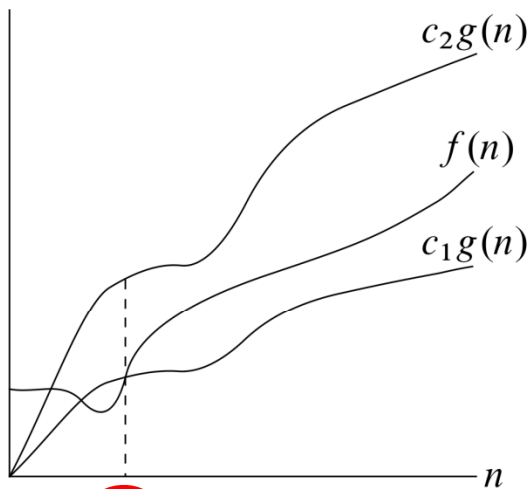
$$= nT(1) + \lg(n) \cdot c \cdot (n)$$



Growth of Functions

§ Asymptotic Notations of Running Times

- Θ -notation: **bounding** a function to **within** constant factors
- O -notation: **upper-bounding** a function to within a constant factor
- Ω -notation: **lower-bounding** a function to within a constant factor

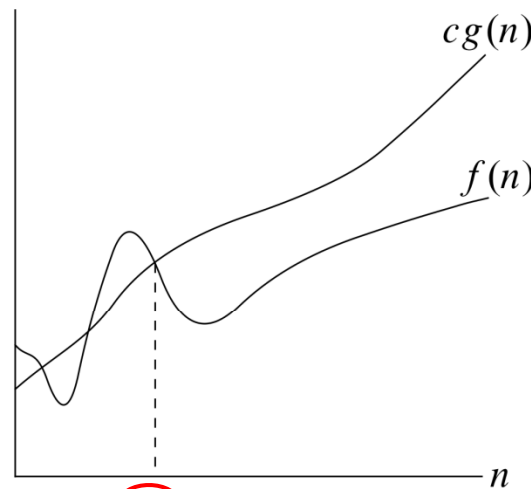


n_0
a constant $f(n) = \Theta(g(n))$

(a)

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

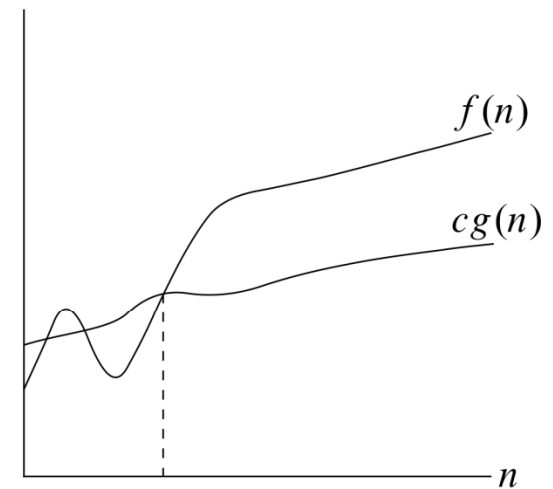
: there exist positive constants c_1 , c_2 , and n_0 s.t. $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$



n_0
a constant $f(n) = O(g(n))$

(b)

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



n_0
 $f(n) = \Omega(g(n))$

(c)

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

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**: $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$

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

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

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

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

- ω -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 $a \geq 1$ and $b > 1$

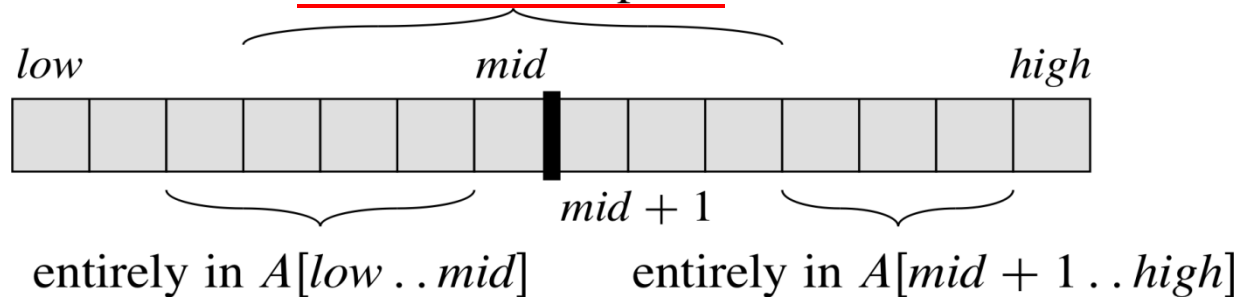
§ Example: Maximum-Subarray Problem (finding such a subarray and its sum)

- divide the problem into two subarrays of (roughly) the same size, finding *mid*
- maximum continuous subarray lies in exactly one of three places, shown below:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

maximum subarray

crosses the midpoint



(a)

Divide-and-Conquer (continued)

§ Maximum-Subarray

– solution lies in exactly one of **three places**

FIND-MAXIMUM-SUBARRAY($A, low, high$)

if $high == low$

resulting sum
↓

return ($low, high, A[low]$)

else $mid = \lfloor (low + high) / 2 \rfloor$

$(left-low, left-high, left-sum) =$

FIND-MAXIMUM-SUBARRAY(A, low, mid)

$(right-low, right-high, right-sum) =$

FIND-MAXIMUM-SUBARRAY($A, mid + 1, high$)

$(cross-low, cross-high, cross-sum) =$

FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)

if $left-sum \geq right-sum$ and $left-sum \geq cross-sum$

return ($left-low, left-high, left-sum$)

elseif $right-sum \geq left-sum$ and $right-sum \geq cross-sum$

return ($right-low, right-high, right-sum$)

else return ($cross-low, cross-high, cross-sum$)

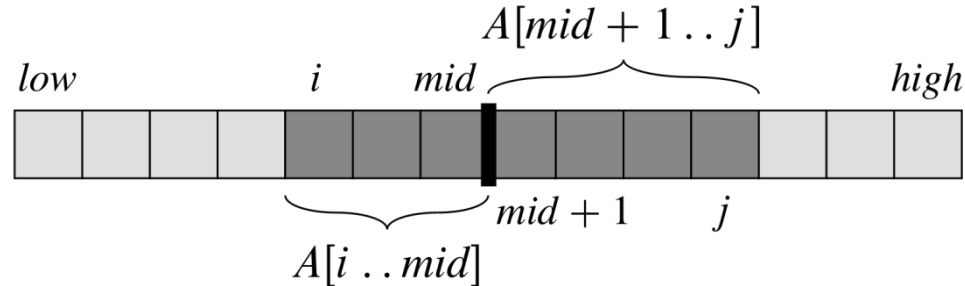
The time complexity of this procedure:

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

Solutions after Divide-and-Conquer (continued)

§ Maximum-Subarray

- crossing the midpoint, comprising two parts



(b)

FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)

// Find a maximum subarray of the form $A[i .. mid]$.

$left-sum = -\infty$

$sum = 0$

for $i = mid$ **downto** low

// This searches **all the way** down to low .

$sum = sum + A[i]$

if $sum > left-sum$

$left-sum = sum$

$max-left = i$

// Find a maximum subarray of the form $A[mid + 1 .. j]$.

$right-sum = -\infty$

$sum = 0$

for $j = mid + 1$ **to** $high$

// This searches **all the way** up to $high$.

$sum = sum + A[j]$

if $sum > right-sum$

$right-sum = sum$

$max-right = j$

// Return the indices and the sum of the two subarrays.

return ($max-left, max-right,$ $left-sum + right-sum$ $)$

This whole procedure takes $\Theta(n)$ time.

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

$$T(n) = 2T(\lfloor n/2 \rfloor) + \underline{n} \text{ being } \underline{T(n) \leq c \cdot n \cdot \lg n} \text{ for a constant } c > 0.$$

← 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:

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

Similar substitution method for proving an upper bound on recurrence

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \underline{1}$$

equal to $O(n)$.

← 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 $T(n) = T(2^m) = S(m) = O(m \cdot \lg m)$
 $= O(\lg n \cdot \lg \lg n)$.

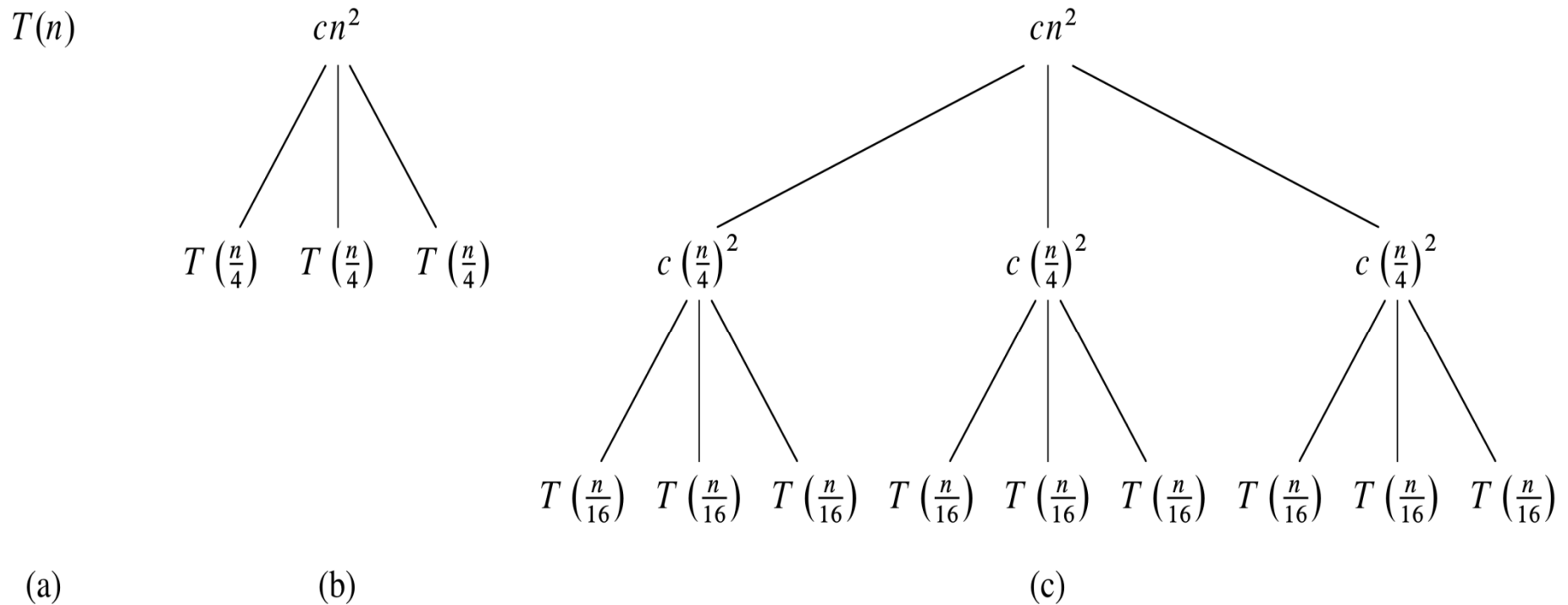
Note: this problem can also be solved by the recursion-tree method, described next.

Divide-and-Conquer (continued)

§ Recursion-Tree Methods

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

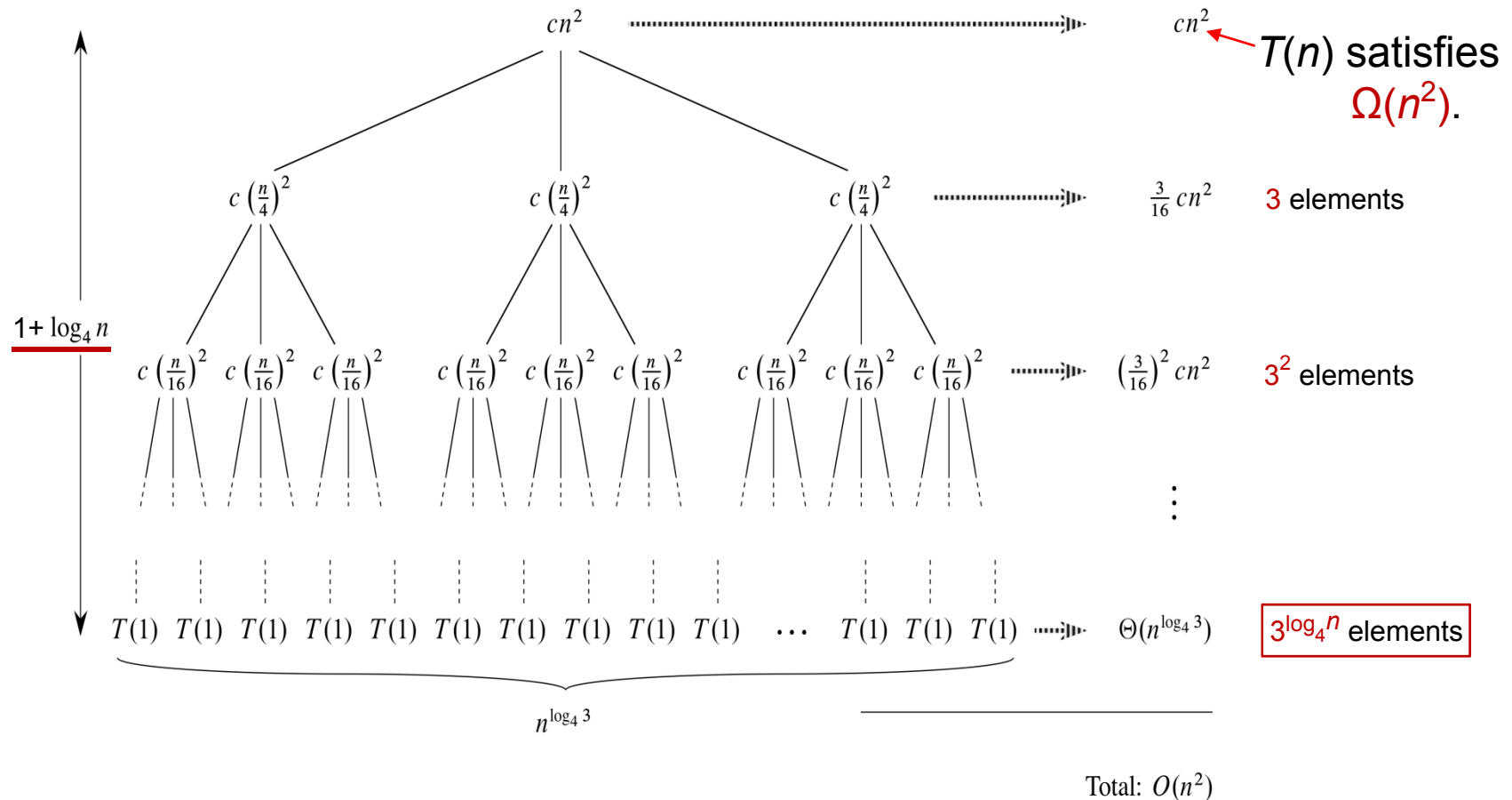
For recurrence: $T(n) = 3T(n/4) + cn^2$



Divide-and-Conquer (continued)

§ Recursion-Tree Methods

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



$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

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

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

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

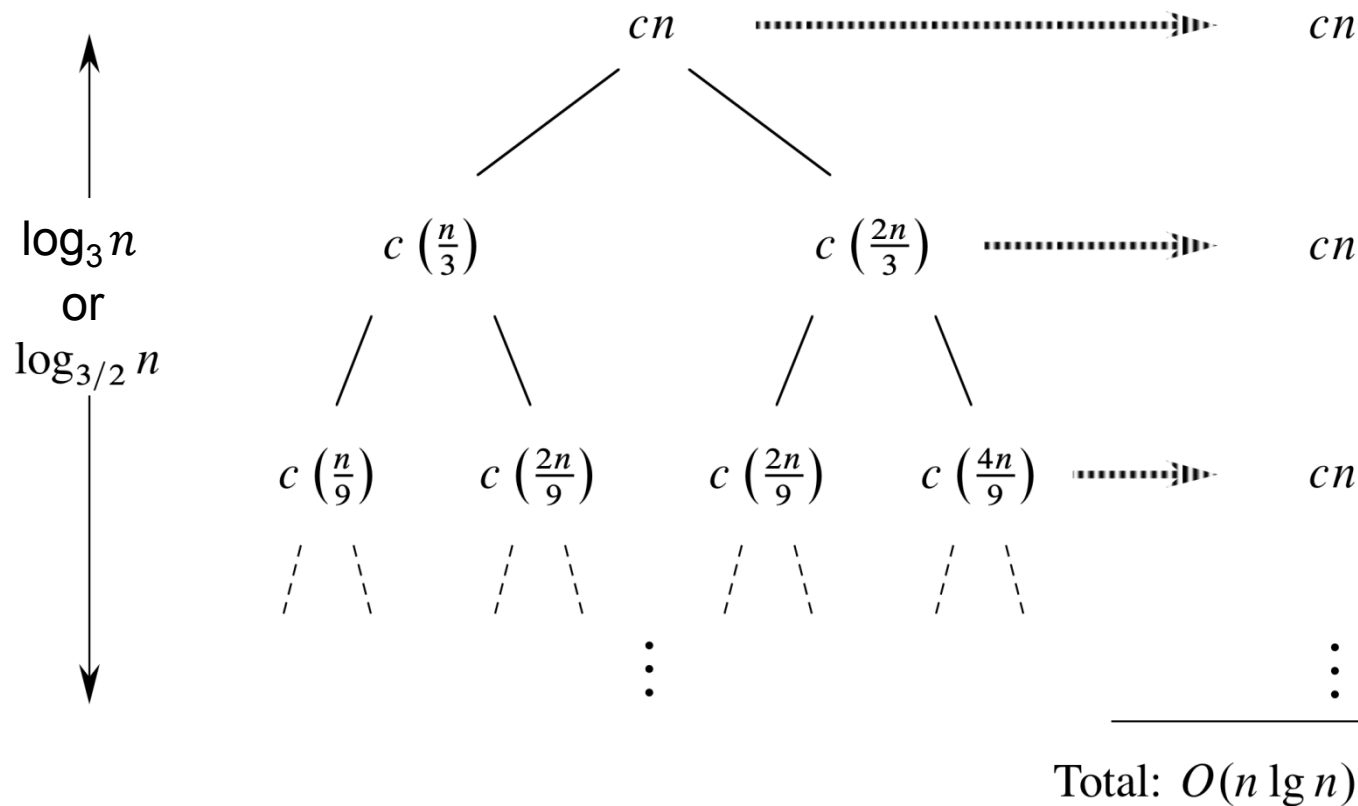
After taking \log_4 on both sides, we have: $(\log_4 n) \cdot (\log_4 3) = \log_4 x$.

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: $T(n) = T(n/3) + T(2n/3) + cn$



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

Shortest path: $cn \rightarrow c(\frac{1}{3})n \rightarrow c(\frac{1}{3})^2 n \rightarrow c(\frac{1}{3})^3 n \rightarrow \dots \rightarrow 1$ 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**.

Master Method for Solving Recurrences

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

Theorem

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

1. for $f(n) = O(n^{\log_b a - \epsilon})$ with constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. for $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$
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 $T(n) = \Theta(f(n))$

In the recursion-tree, 2nd level sums to **no more than** 1st 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, $n^{\log_b a}$ and $f(n)$ are of the same size

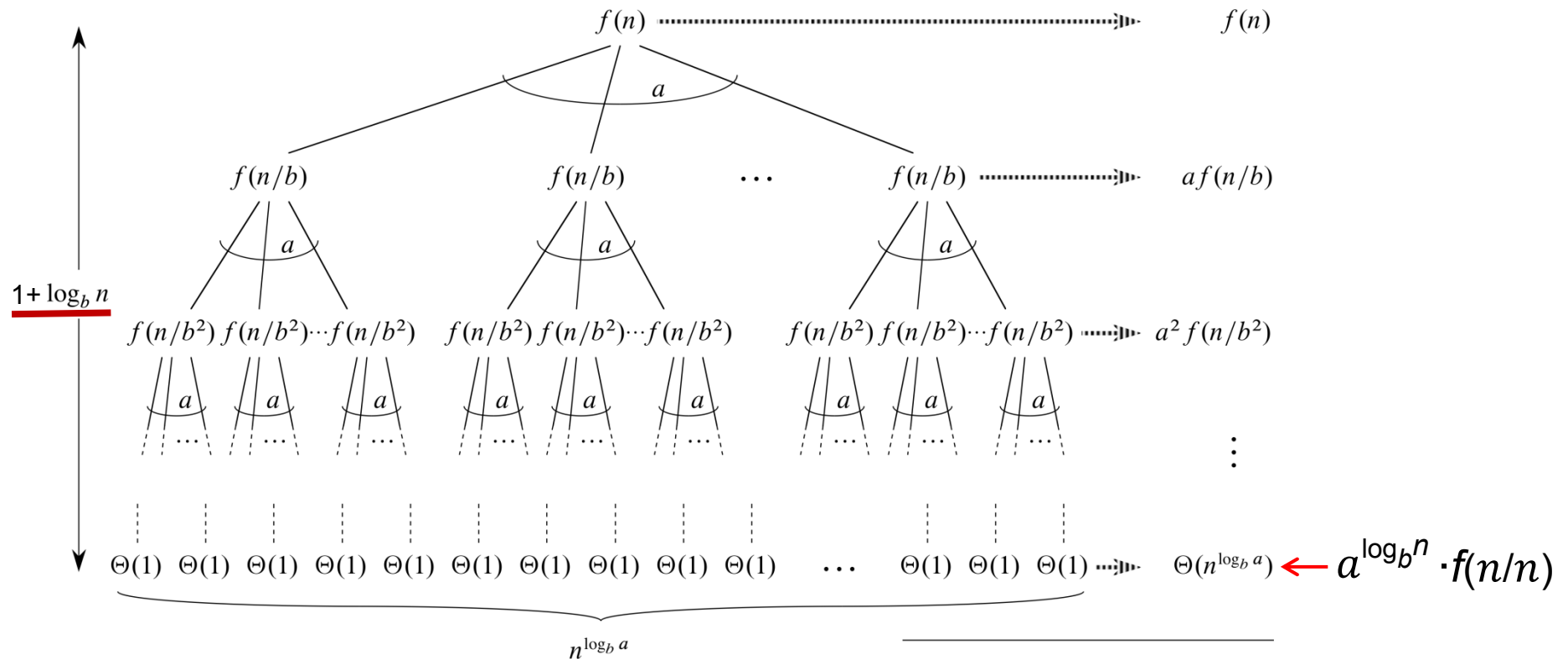
In Case 3, $f(n)$ is *polynomially larger* than $n^{\log_b a}$

Master Method (continued)

Let $T(n) = a \cdot T(n/b) + f(n)$ with constants $a \geq 1$ and n being an **exact power** of b (> 1)

Lemma 4.2

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j \cdot f(n/b^j)$$



$$\text{Total: } \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

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 $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$
 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 $T(n) = \Theta(f(n))$
-

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

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 $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$
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 $T(n) = \Theta(f(n))$

$f(n)$ 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 $g(n) = \sum_{j=0}^{\log_b n - 1} a^j \cdot f(n/b^j)$ for $a \geq 1$ 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 $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \cdot \lg n)$
 3. if $a \cdot f(n/b) \leq c \cdot f(n)$ for constant $c < 1$ and for sufficiently large n , $g(n) = \Theta(f(n))$
-