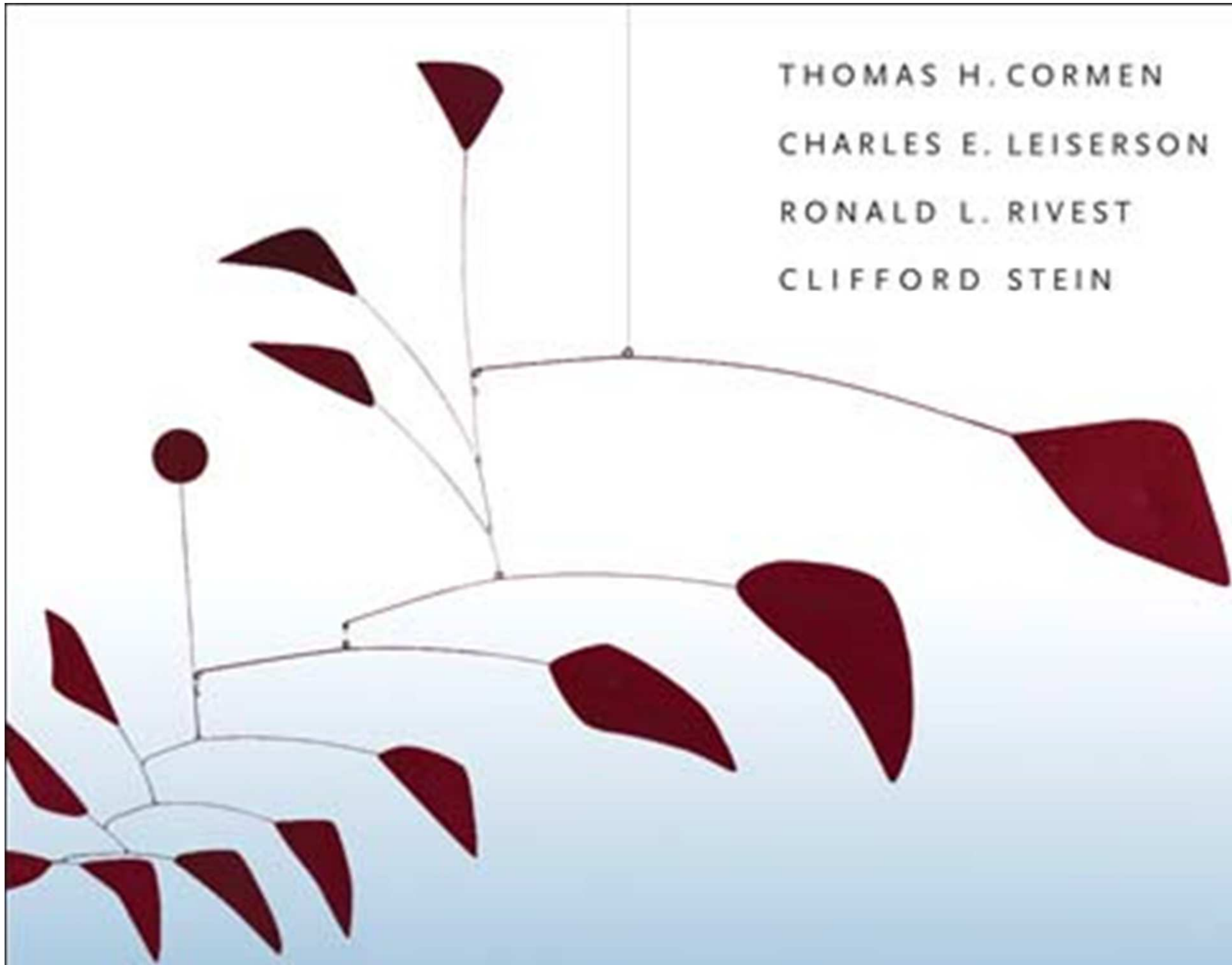
THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN



INTRODUCTION TO

ALGORITHMS

THIRD EDITION

CSCE 500

Design and Analysis of Algorithms

Fall 2017

August 21, 2017

Instructor: Nian-Feng Tzeng
Office: Rm. 454 CC (× 2-6304)
Class meeting: MW 10:30 – 11:45, 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 supplemental 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. Each covered topic starts with the description of the algorithm(s) in English and/or in the pseudocode(s), followed by a careful complexity analysis of the algorithm(s). Topics are all from the textbook and they include:

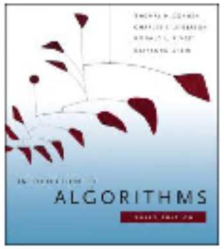
(1) foundations;
(2) data structures – hash tables, trees, heaps;
(3) design and analysis techniques – dynamic programming, greedy algorithms, amortized analysis;
(4) graph algorithms – spanning trees, shortest paths, maximum flow;
(5) selected topics – NP-completeness, approximation algorithms, multithreaded algorithms, string matching.

Course Requirements:

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

Table of Contents

The MIT Press



Introduction to Algorithms, third edition

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

return

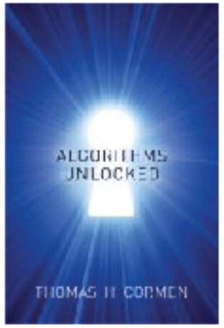
Table of Contents

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

Table of Contents

Table of Contents

Table of Contents

Table of Contents

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 .. *j*-1]
  - repeat insertion until A[*i*] is fully sorted

INSERTION-SORT(A, n)	*cost*	*times*
for $j = 2$ **to** n	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	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

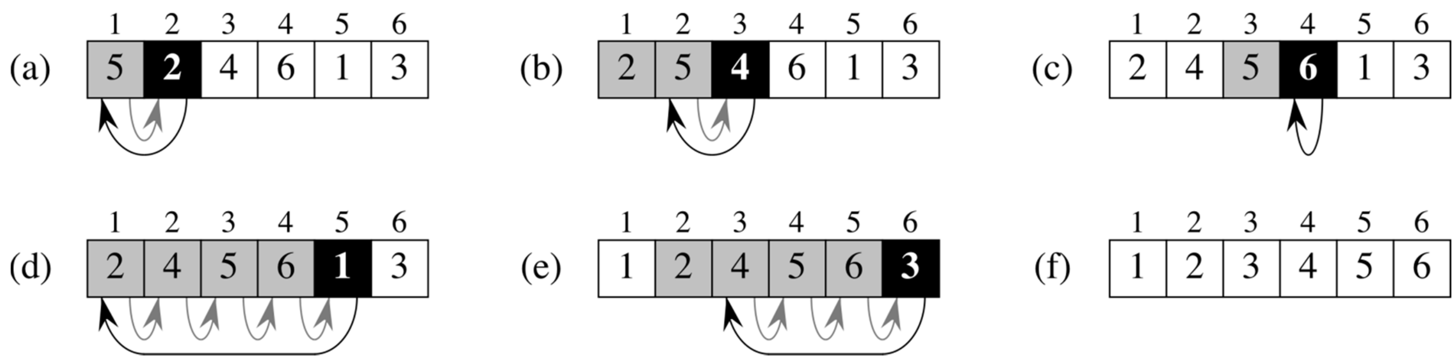
Complexity: $T(n) = \sum_{i=1}^{8} c_i$

Analyzing Algorithms (continued)

§ **Insertion Sort for Array A[*i*]**

– idea: insert A[*j*] into sorted subarrays: A[1 .. *j*-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)	*cost*	*times*
for $j = 2$ **to** n	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	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	$n-1$

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* .. *q*] & A[*q*+1 .. *r*]
  - merge the two sorted subarrays
  - merging takes Θ(*n*) time

```
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
```

merge →

```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i − 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞      ← stoppers for half arrays
  R[n2 + 1] = ∞      ←
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```

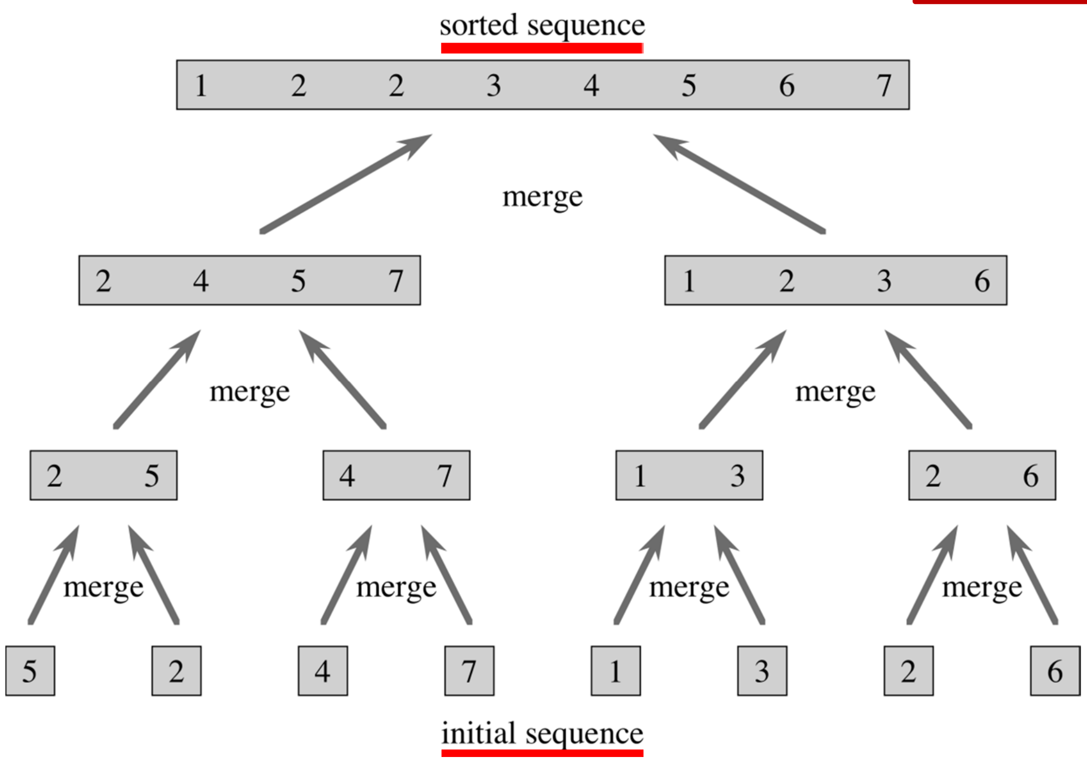
Analyzing Algorithms

§ **Analysis of Divide-and-Conquer Algorithms**

- Merge Sort
- Time complexity:

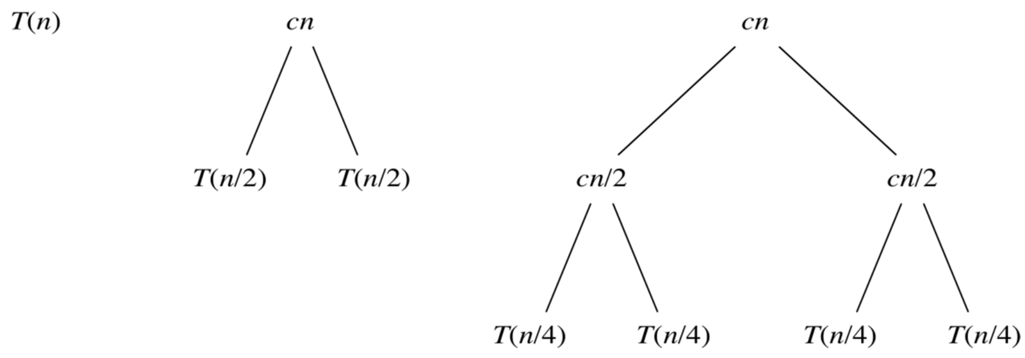
$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + c\cdot(n) & \text{if } n > 1 \end{cases}$$

```
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
- – $cn \cdot \lg(n) + cn = \Theta(n \cdot \lg n)$



(a) (b) (c)

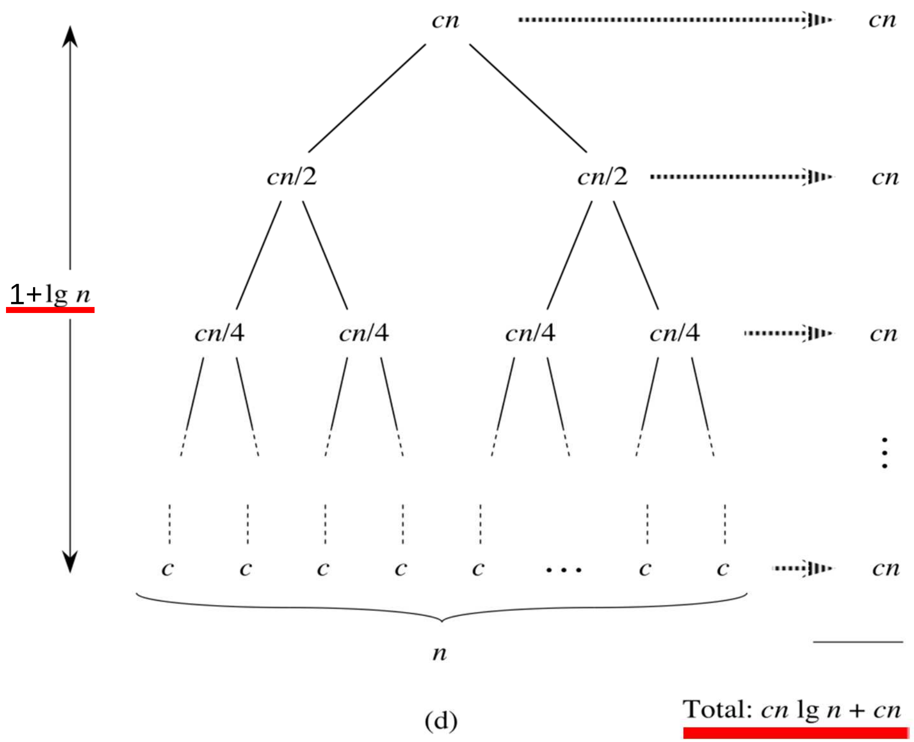
Another approach for solution:

$$T(n) = 2T(n/2) + c\cdot(n)$$
$$= 2(2T(n/4)+c\cdot(n/2)) + c\cdot(n)$$
$$= 2^2T(n/4) + 2\cdot c\cdot(n)$$
$$= 2^2(2T(n/8) + c\cdot(n/4)) + 2\cdot c\cdot(n)$$
$$= 2^3T(n/8) + 3\cdot c\cdot(n)$$

....

....

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

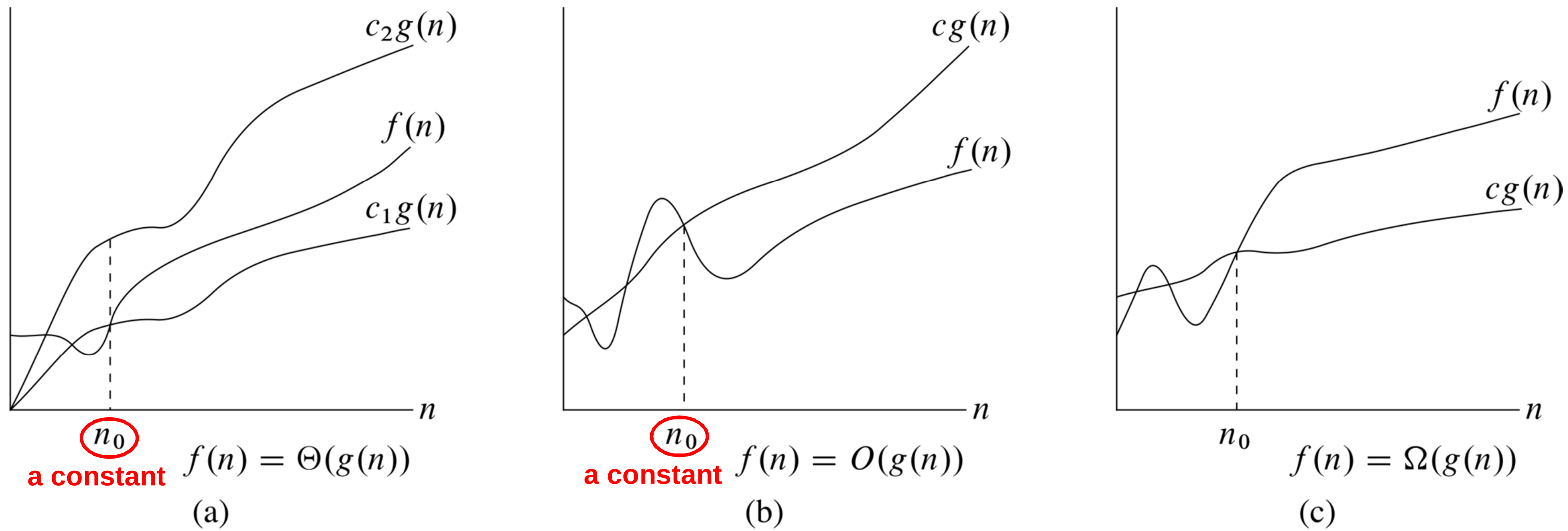


(d)

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



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

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)

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

Growth of Functions (continued)

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

o-notation

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

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

$n^{1.9999} = o(n^2)$
$n^2 / \lg n = o(n^2)$
$n^2 \ne o(n^2)$ (just like $2 \not< 2$)
$n^2/1000 \ne o(n^2)$

- ω-notation: not asymptotically-tight lower-bound

ω-notation

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

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

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

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 $b > 1$

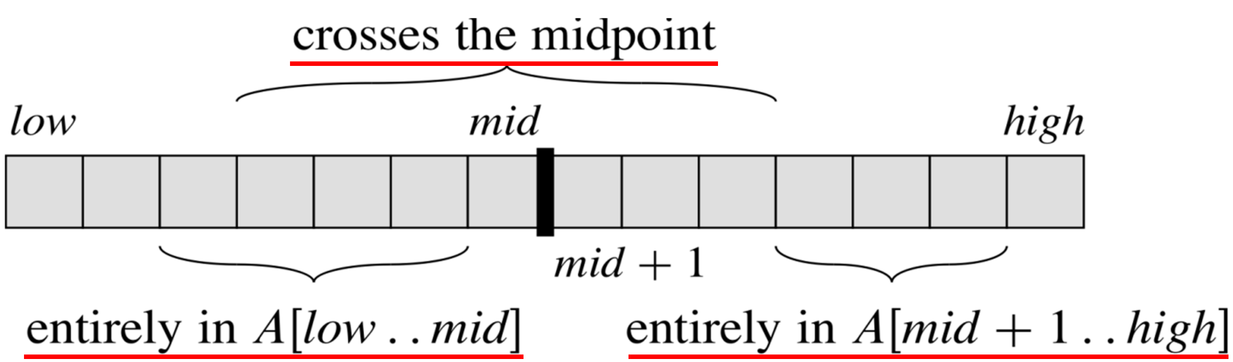
§ Example: Maximum-Subarray Problem

– divide the problem into two subarrays of (close to) 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



(a)

Divide-and-Conquer (continued)

§ **Maximum-Subarray**

– solution lies in exactly one of three places

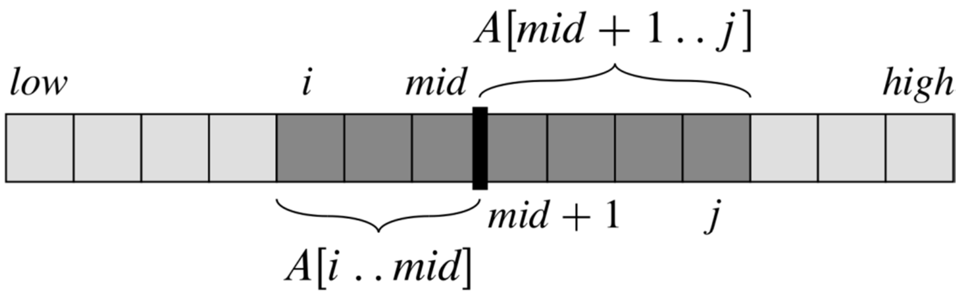
$\text{Find-Maximum-Subarray}(A, low, high)$

```
if high == low
    return (low, high, A[low])      ← resulting sum
else mid = ⌊(low + high)/2⌋
    (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 ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

The time complexity of this procedure equal to $\Theta(1)+2T(n/2)+\Theta(n)+\Theta(1) = 2T(n/2) + \Theta(n)$.

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 = -∞
  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 = -∞
  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 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
$T(n) = 2T(\lfloor n/2 \rfloor) + n$ being $T(n) \le c \cdot n \cdot \lg n$ for a constant $c > 0$.

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

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

Similar substitution method for proving an upper bound on recurrence
$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
equal to $O(n)$.

Divide-and-Conquer (continued)

§ Substitution Methods

– changing variables

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

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

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

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

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

$S(m) = O(m \cdot \lg m)$

(via Master Method, to be treated in detail later)

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 be solved by the recursion-tree method, described next, as well.

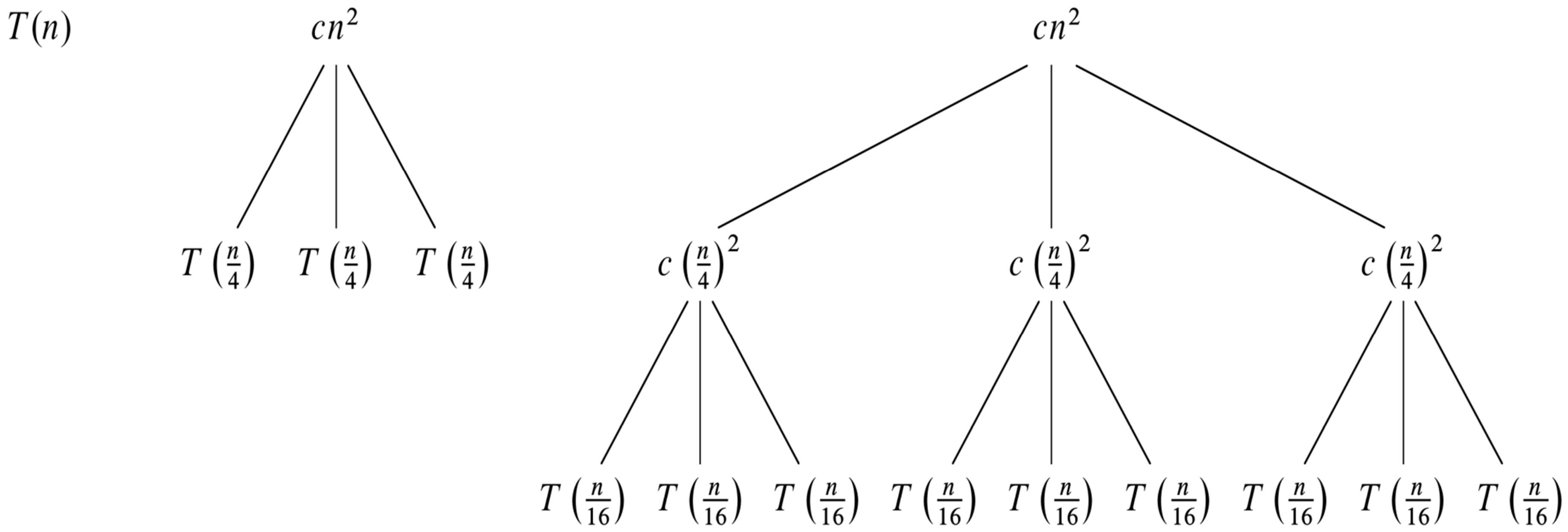
Divide-and-Conquer (continued)

§ Recursion-Tree Methods

– best used to generate good complexity bounds

– two examples given below

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

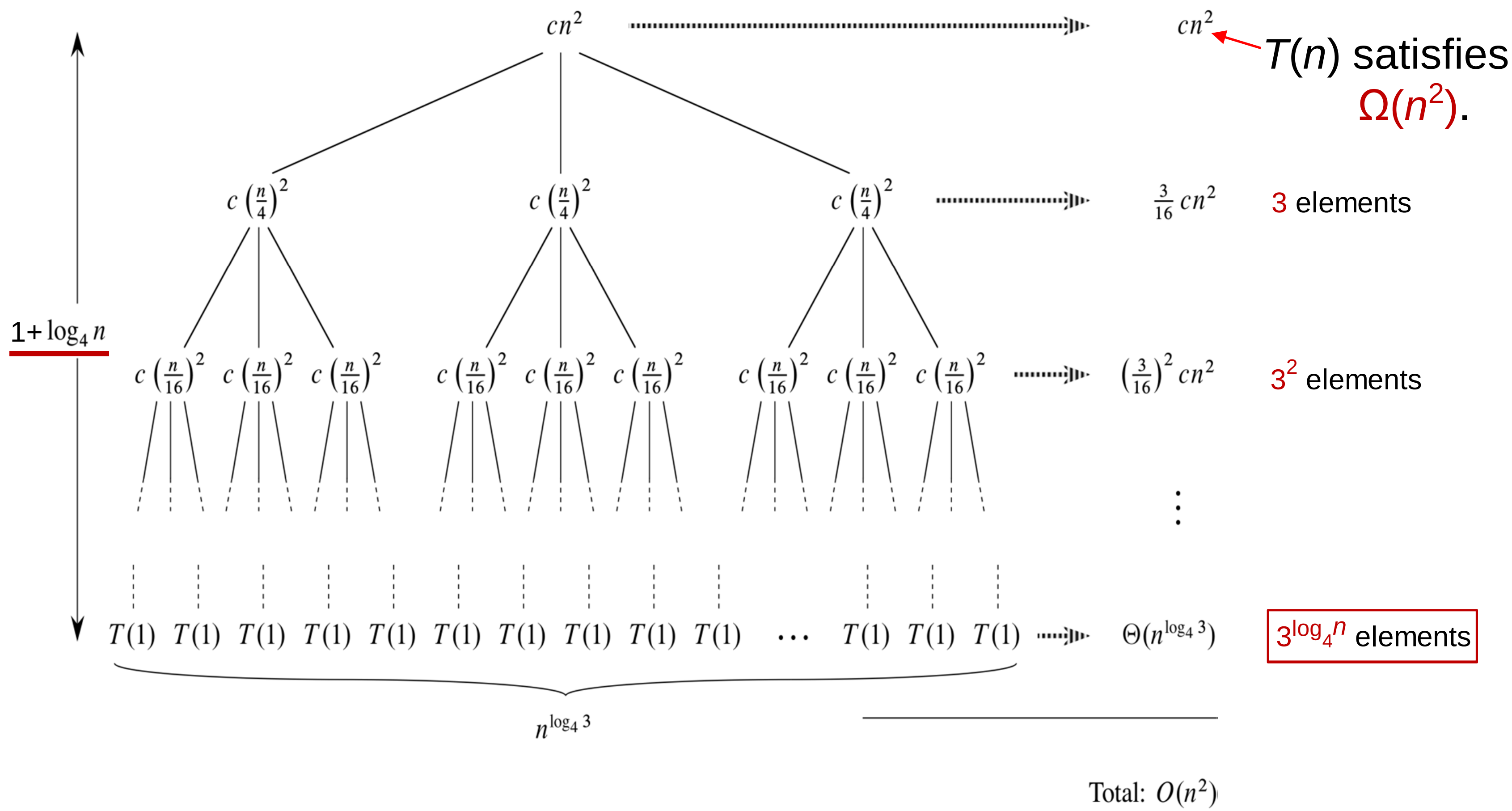


(a) (b) (c)

Divide-and-Conquer (continued)

§ Recursion-Tree Methods

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

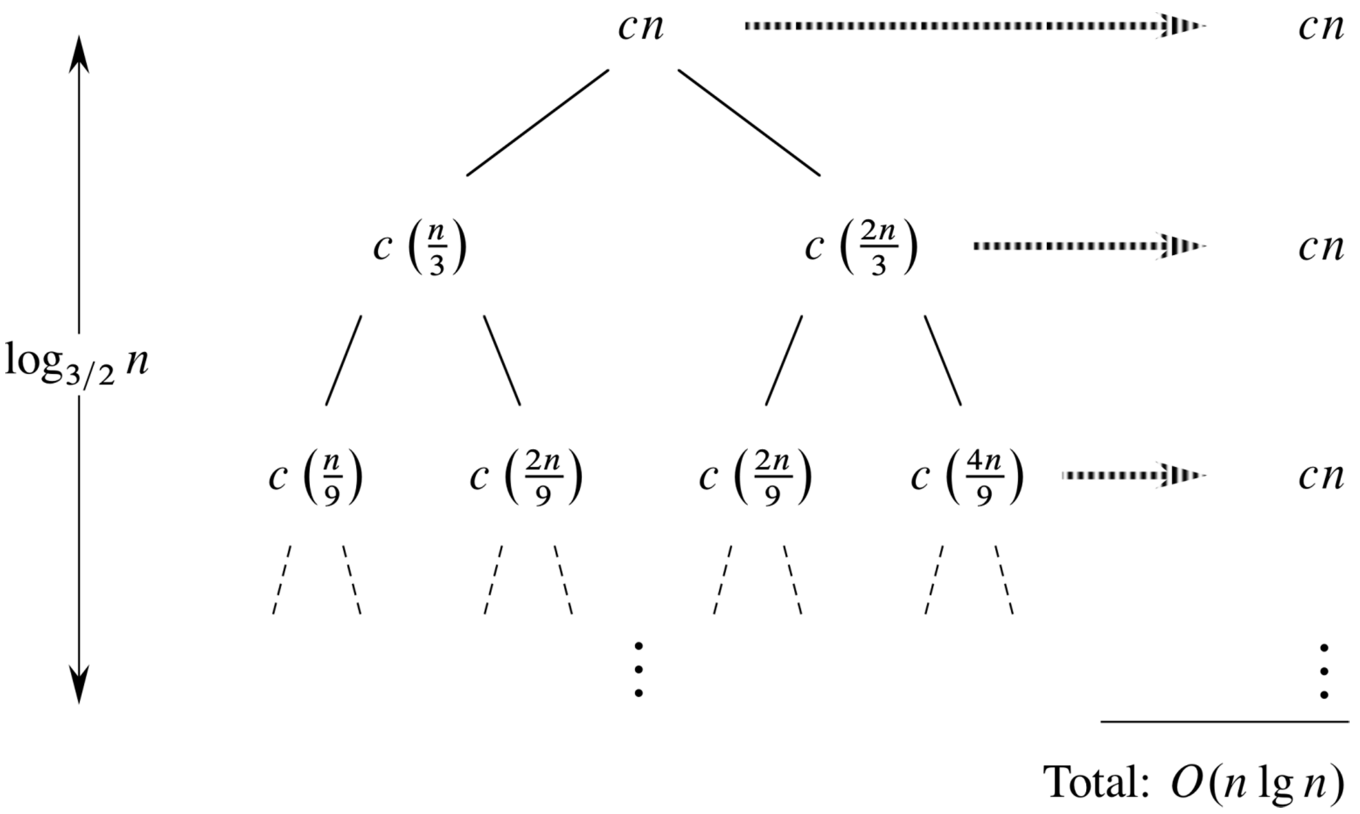


$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

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

Divide-and-Conquer (continued)

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



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

Shortest path: $cn \rightarrow c(\frac{1}{3})n \rightarrow c(\frac{1}{3})^2 n \rightarrow c(\frac{1}{3})^3 n \rightarrow \cdots \rightarrow 1$ to get $k = \log_3 n$, ~ $(\log_{3/2} n)/2.3$

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 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)

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 $a \cdot f(n/b) \le c \cdot f(n)$, 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 $f(n) = \Theta(n^2)$

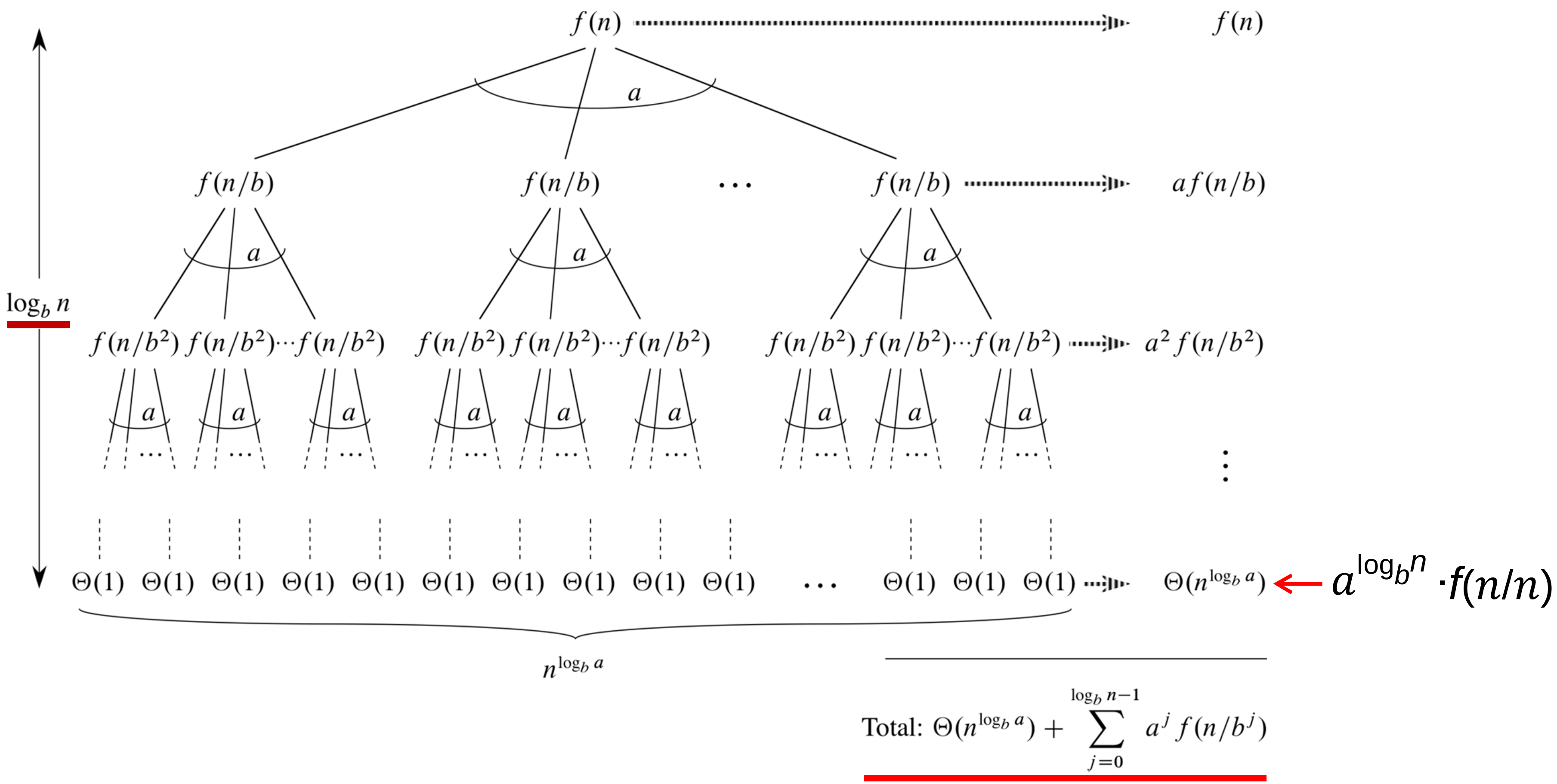
From $n^{\log_2 7} = n^{2.8^+}$, we have $f(n) = O(n^{\log_2 7 - \epsilon})$ to get $T(n) = \Theta(n^{\log_2 7})$

<u>Master Method</u> (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)

<u>Lemma 4.2</u>

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



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, $g(n) = \Theta(f(n))$