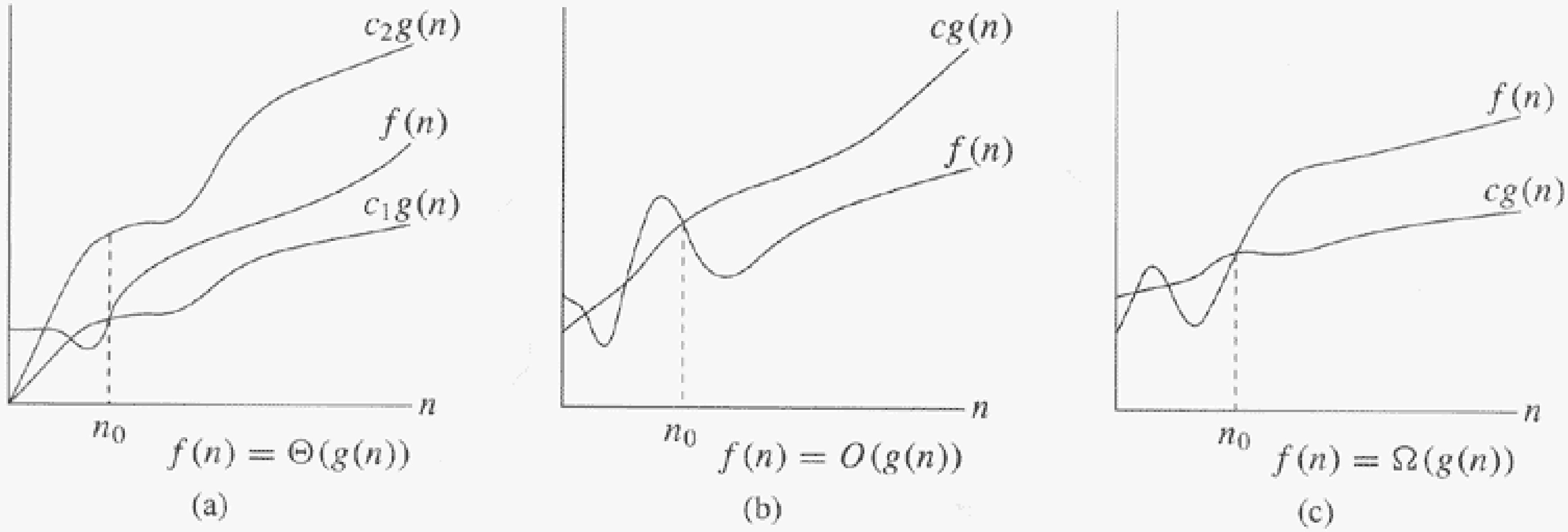
Solving Recurrence Relations

Page 62, [CLR]

Recurrence; Cond	Solution
$T(n) = T(n-1) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-1) + O(n)$	$T(n) = O(n^2)$
$T(n) = T(n-c) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-c) + O(n)$	$T(n) = O(n^2)$
$T(n) = 2T(n/2) + O(n)$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n);$ $a = b$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n);$ $a < b$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n);$ $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n);$ $f(n) = O(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \log n)$
$T(n) = aT(n/b) + f(n);$ $f(n) = \Theta(f(n))$ $af(n/b) \leq cf(n)$	$T(n) = \Omega(n^{\log_b a} \log n)$



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

$f(n) = O(g(n))$
(b)

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

Figure 3.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that to the right of n_0, the value of $f(n)$ always lies between $c_1g(n)$ and $c_2g(n)$ inclusive. **(b)** O-notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω-notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

Solving Recurrences by Substitution

- Guess the form of the solution
- (Using mathematical induction) find the constants and show that the solution works

Example

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

Guess (#1) $T(n) = O(n)$

Need **$T(n) <= cn$** for some constant $c>0$

Assume $T(n/2) <= cn/2$ Inductive hypothesis

Thus $T(n) <= 2cn/2 + n = (c+1)\,n$

Our guess was wrong!!

Solving Recurrences by Substitution: 2

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

Guess (#2) $T(n) = O(n^2)$

Need $T(n) <= cn^2$ for some constant $c>0$

Assume $T(n/2) <= cn^2/4$ Inductive hypothesis

Thus $T(n) <= 2cn^2/4 + n = cn^2/2+ n$

Works for all n as long as $c>=2$!!

But there is a lot of “slack”

Solving Recurrences by Substitution: 3

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

Guess (#3) $T(n) = O(n\log n)$

Need **$T(n) <= cn\log n$** for some constant $c>0$

Assume $T(n/2) <= c(n/2)(\log(n/2))$ Inductive hypothesis

Thus $T(n) <= 2\,c(n/2)(\log(n/2)) + n$

$<= cn\log n - cn + n <= cn\log n$

Works for all n as long as c>=1 !!

This is the correct guess. WHY?

Show $T(n) >= c'n\log n$ for some constant $c'>0$

Solving Recurrences: Recursion-tree method

- Substitution method fails when a good guess is not available
- Recursion-tree method works in those cases
  - Write down the recurrence as a tree with recursive calls as the children
  - Expand the children
  - Add up each level
  - Sum up the levels
- Useful for analyzing divide-and-conquer algorithms
- Also useful for generating good guesses to be used by substitution method

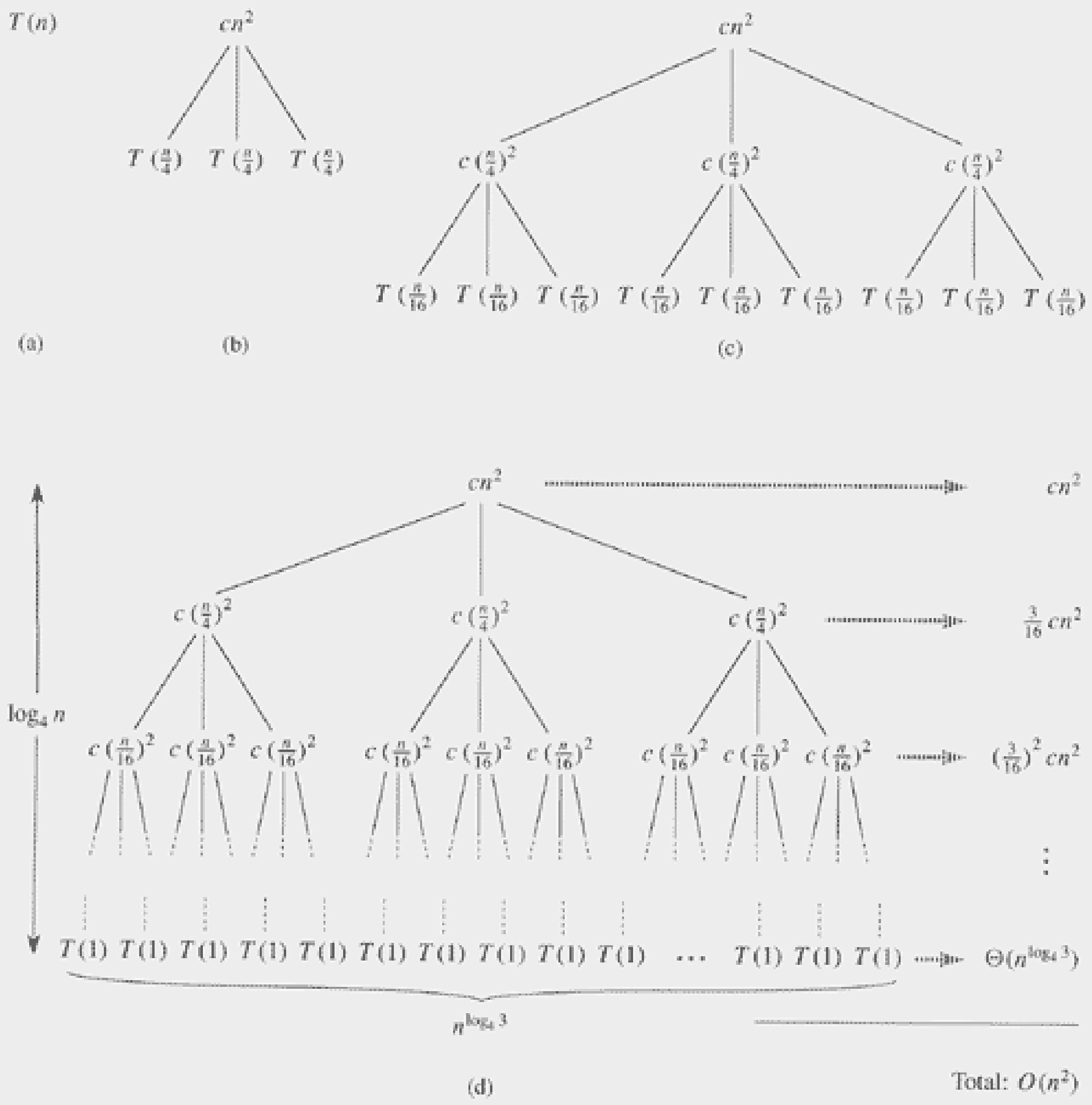


Figure 4.1 The construction of a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part **(d)** has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

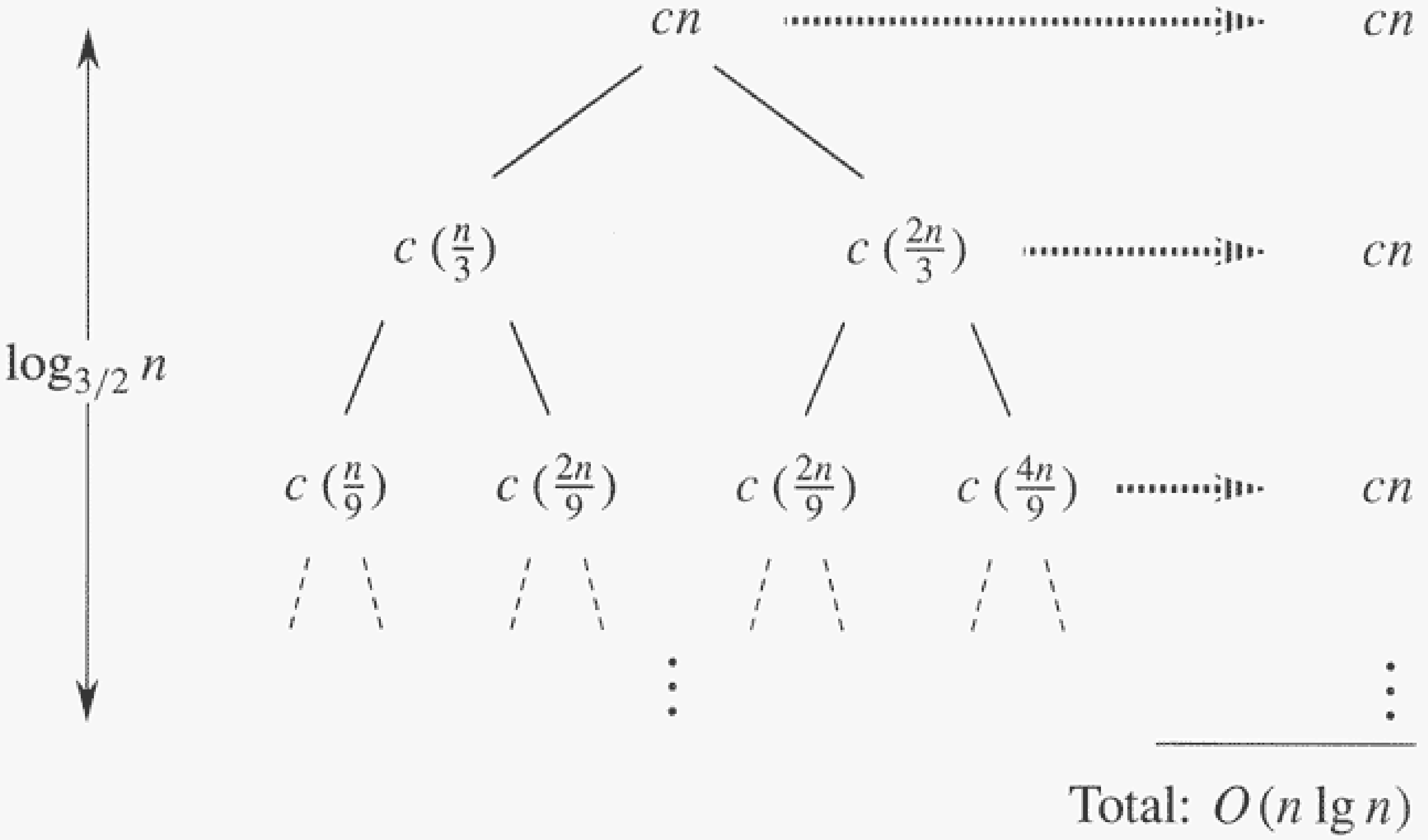


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Solving Recurrences using Master Theorem

Master Theorem:

Let a,b >= 1 be constants, let f(n) be a function, and let

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

1. If f(n) = O($n^{\log_b a-e}$) for some constant e>0, then
   T(n) = Theta($n^{\log_b a}$)
2. If f(n) = Theta($n^{\log_b a}$), then
   T(n) = Theta($n^{\log_b a}$ log n)
3. If f(n) = Omega($n^{\log_b a+e}$) for some constant e>0, then
   T(n) = Theta(f(n))