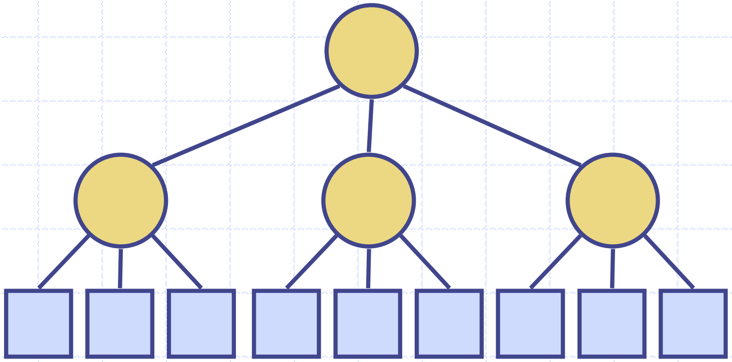
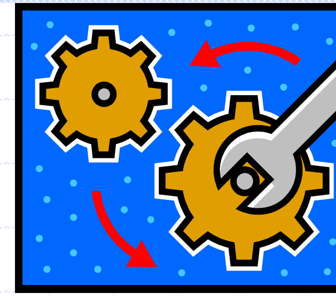
Divide-and-Conquer

- Divide-and conquer is a general algorithm design paradigm:
  - Divide: divide the input data S in two or more disjoint subsets S_1, S_2, …
  - Recur: solve the subproblems recursively
  - Conquer: combine the solutions for S_1, S_2, …, into a solution for S
- The base case for the recursion are subproblems of constant size
- Analysis can be done using **recurrence equations**



Iterative Substitution



- In the iterative substitution, or "plug-and-chug," technique, we iteratively apply the recurrence equation to itself and see if we can find a pattern:

$$
\begin{aligned}
T(n) &= 2T(n/2) + bn \\
&= 2(2T(n/2^2)) + b(n/2)) + bn \\
&= 2^2 T(n/2^2) + 2bn \\
&= 2^3 T(n/2^3) + 3bn \\
&= 2^4 T(n/2^4) + 4bn \\
&= \ldots \\
&= 2^i T(n/2^i) + ibn
\end{aligned}
$$

- Note that base, T(n)=b, case occurs when $2^i=n$. That is, i = log n.
- So,

$$T(n) = bn + bn \log n$$

- Thus, T(n) is O(n log n).

Solving Recurrences by Substitution: Guess-and-Test

- 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 **$T(n) \le cn \log n$** for some constant $c>0$

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

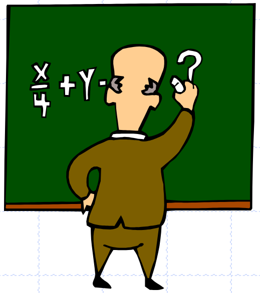
Thus $T(n) \le 2c(n/2)(\log(n/2)) + n$

$\le cn \log n - cn + n \le cn \log n$

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

This is the correct guess. WHY?

More Examples



- In the guess-and-test method, we guess a closed form solution and then try to prove it is true by induction:

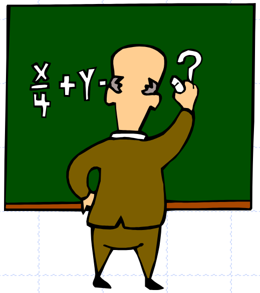
$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn\log n & \text{if } n \geq 2 \end{cases}$$

- Guess: T(n) < cn log n.

$$\begin{aligned} T(n) &= 2T(n/2) + bn\log n \\ &= 2(c(n/2)\log(n/2)) + bn\log n \\ &= cn(\log n - \log 2) + bn\log n \\ &= cn\log n - cn + bn\log n \end{aligned}$$

- Wrong: we cannot make this last line be less than cn log n

More Examples



- Recall the recurrence equation:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn\log n & \text{if } n \geq 2 \end{cases}$$

- Guess #2: T(n) < cn log² n.

$$\begin{aligned} T(n) &= 2T(n/2) + bn\log n \\ &= 2(c(n/2)\log^2(n/2)) + bn\log n \\ &= cn(\log n - \log 2)^2 + bn\log n \\ &= cn\log^2 n - 2cn\log n + cn + bn\log n \\ &\leq cn\log^2 n \end{aligned}$$

  - if c > b.
- So, T(n) is O(n log² n).
- In general, to use this method, you need to have a good guess and you need to be good at induction proofs.

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

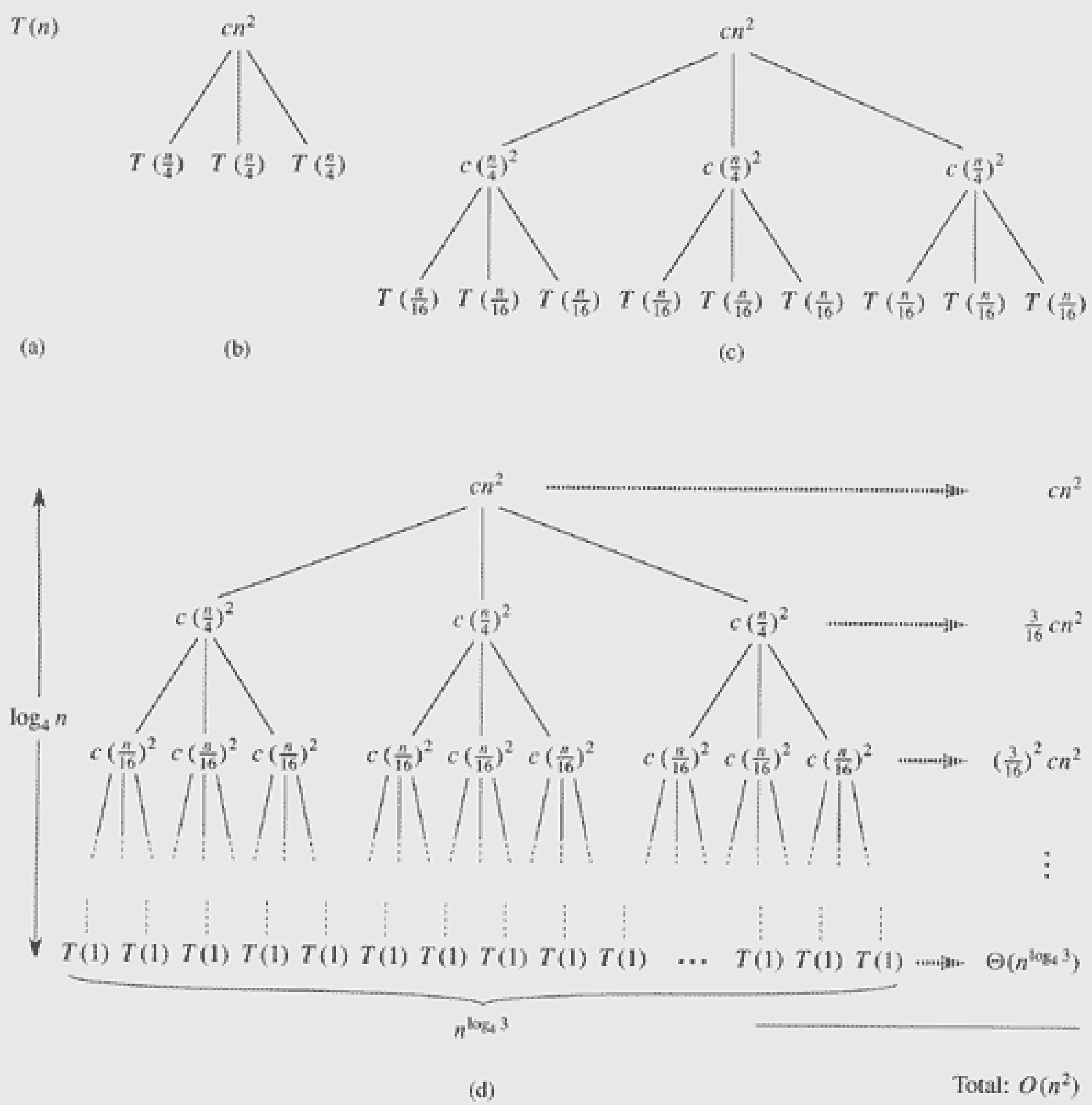


Figure 4.1 The construction of a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

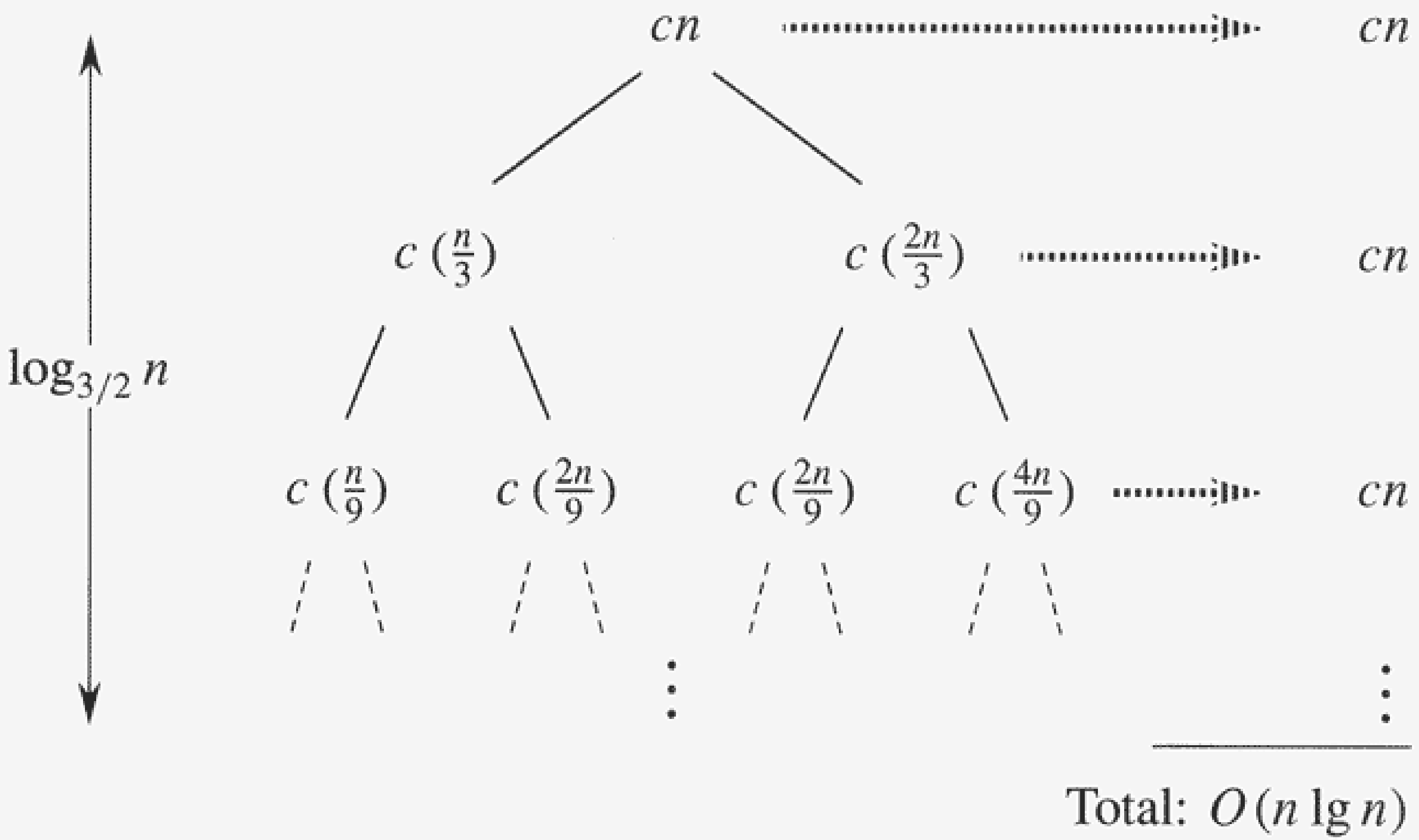
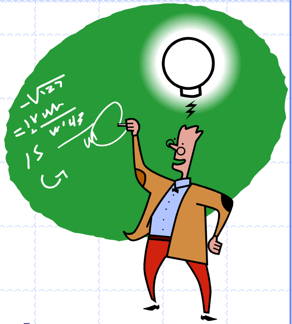


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Master Method



- Many divide-and-conquer recurrence equations have the form:

$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

- The Master Theorem:

$$1.\ \text{if } f(n) \text{ is } O(n^{\log_b a - \varepsilon}), \text{ then } T(n) \text{ is } \Theta(n^{\log_b a})$$

$$2.\ \text{if } f(n) \text{ is } \Theta(n^{\log_b a} \log^k n), \text{ then } T(n) \text{ is } \Theta(n^{\log_b a} \log^{k+1} n)$$

$$3.\ \text{if } f(n) \text{ is } \Omega(n^{\log_b a + \varepsilon}), \text{ then } T(n) \text{ is } \Theta(f(n)),$$

$$\text{provided } af(n/b) \leq \delta f(n) \text{ for some } \delta < 1.$$