# CMP448: Algorithms

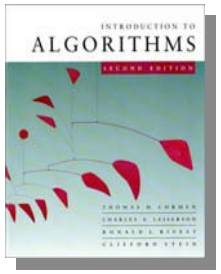

## Lecture 02: Asymptotic Analysis and Recurrences

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2013

# Agenda

- Asymptotic Notation
  - O-, Ω-, and Θ-notation
- Recurrences
  - Substitution Method
  - Recursion Tree
  - Master Theorem
- Divide-and-Conquer Examples

**Acknowledgment**
A lot of slides adapted from the slides of Erik Demaine and Charles Leiserson

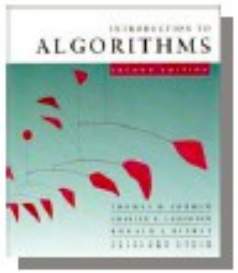



# Asymptotic notation

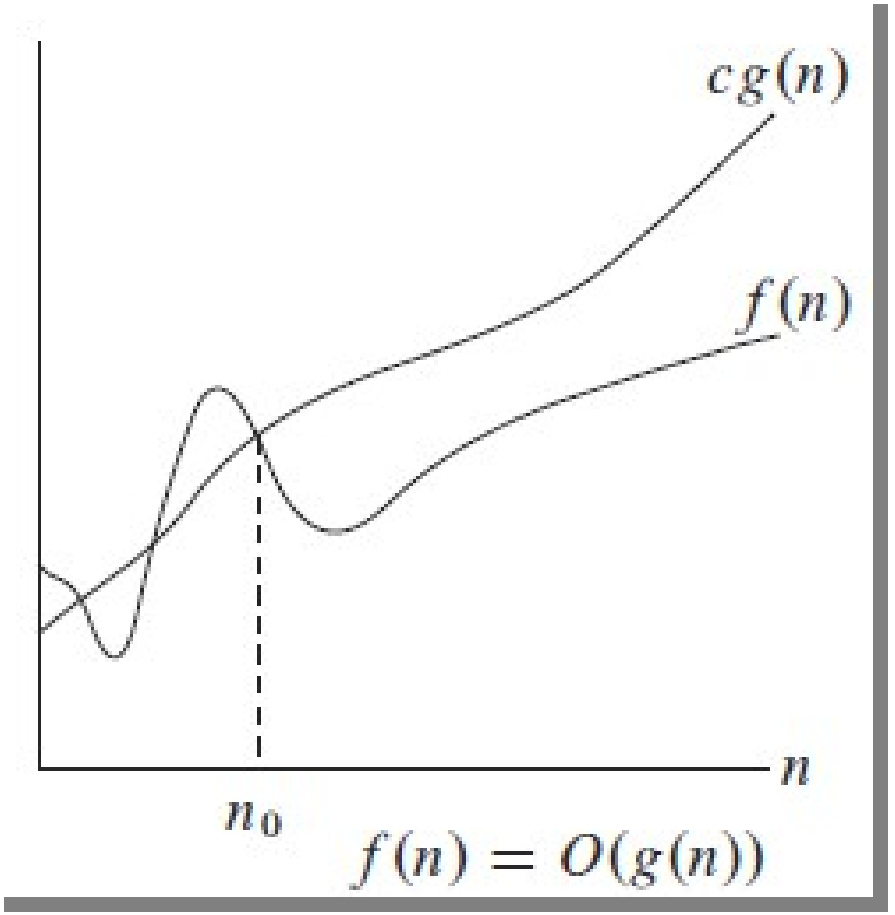
*O*-notation (upper bounds):

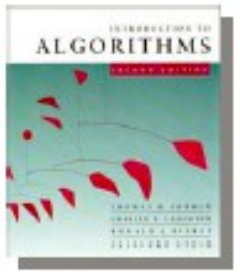
We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

**EXAMPLE:** $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

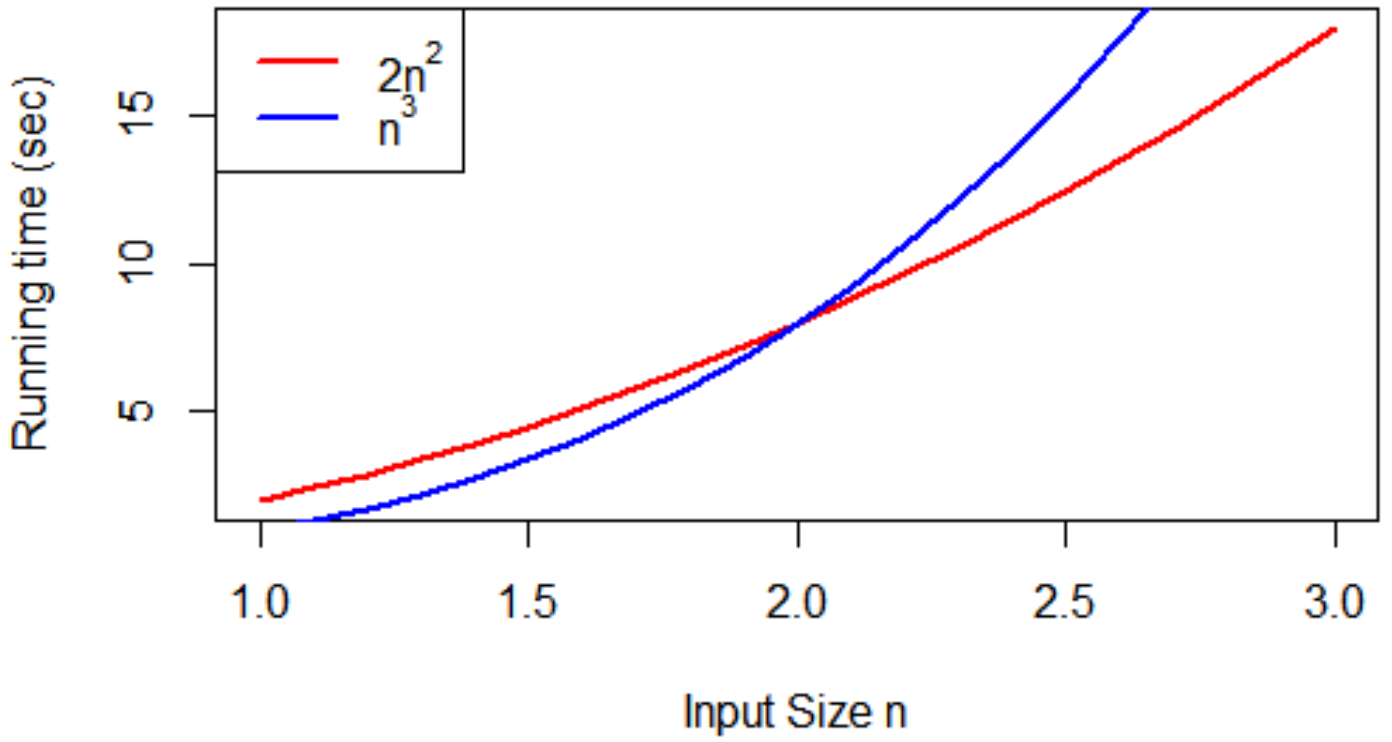



# Asymptotic notation

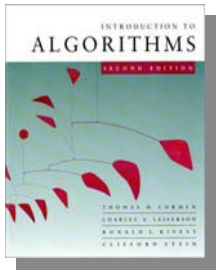





# Asymptotic notation




**EXAMPLE:** $2n^2 = O(n^3)$ $(c = 1, n_0 = 2)$



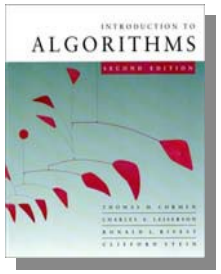

# Asymptotic notation

*O*-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

**EXAMPLE:** $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

*functions, not values*




# Asymptotic notation

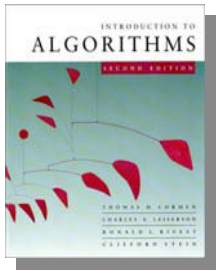
*O*-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

**EXAMPLE:** $2n^2 = O(n^3)$ $(c = 1, n_0 = 2)$

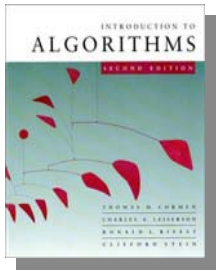
*functions, not values*

*funny, "one-way" equality*




# Set definition of O-notation

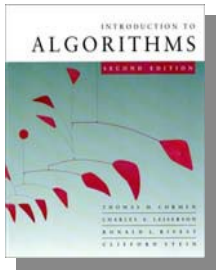
$O(g(n)) = \{ f(n) :$ there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0 \}$




# Set definition of O-notation

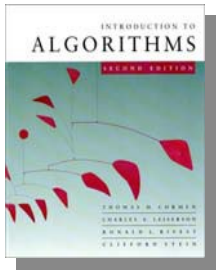
$O(g(n)) = \{ f(n) :$ there exist constants $c > 0, n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0 \}$

**EXAMPLE:** $2n^2 \in O(n^3)$




# Macro substitution

***Convention:*** A set in a formula represents an anonymous function in the set.




# Macro substitution

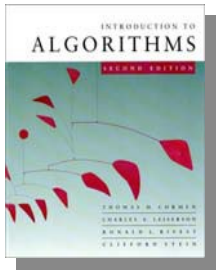
***Convention:*** A set in a formula represents an anonymous function in the set.

**EXAMPLE:** $f(n) = n^3 + O(n^2)$

means

$f(n) = n^3 + h(n)$

for some $h(n) \in O(n^2)$ .




# Macro substitution

***Convention:*** A set in a formula represents an anonymous function in the set.

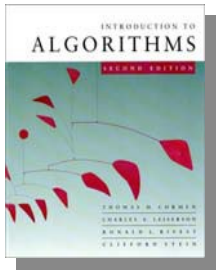
**EXAMPLE:** $n^2 + O(n) = O(n^2)$

means

for any $f(n) \in O(n)$:

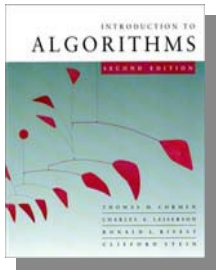
$n^2 + f(n) = h(n)$

for some $h(n) \in O(n^2)$ .




# Ω-notation (lower bounds)

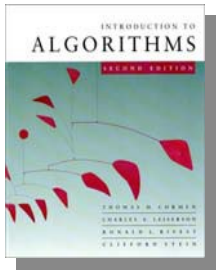
*O*-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.




# Ω-notation (lower bounds)

*O*-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$\Omega(g(n)) = \{ f(n) :$ there exist constants $c > 0, n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0 \}$

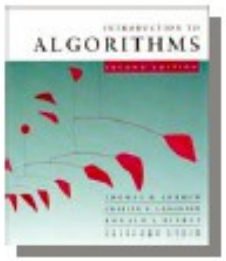



# Ω-notation (lower bounds)

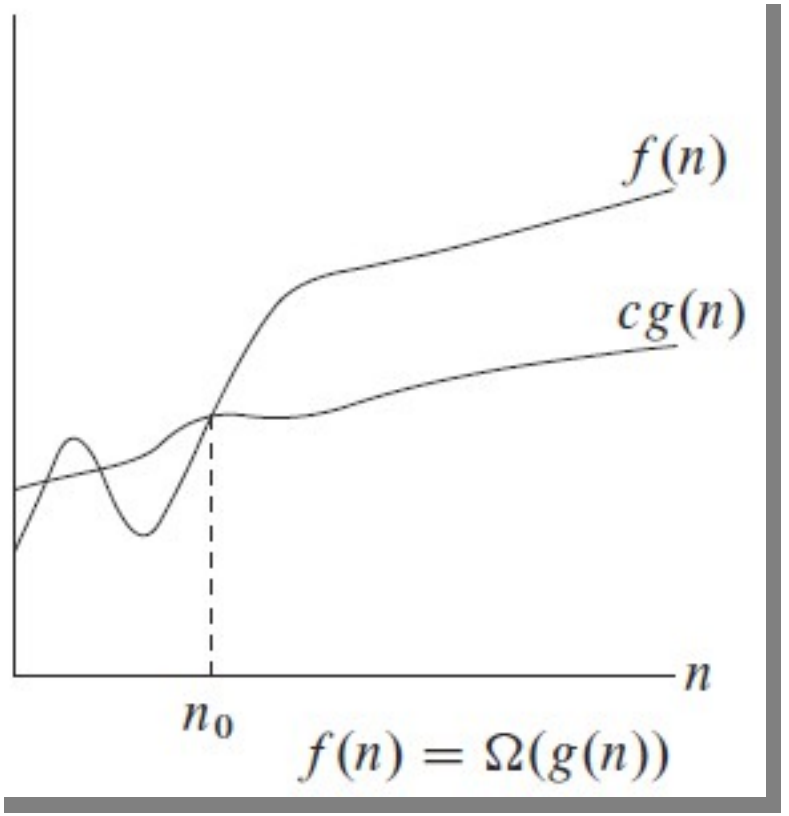
*O*-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

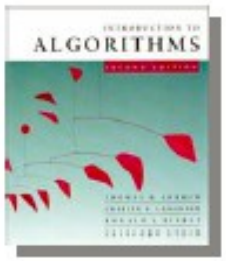
$\Omega(g(n)) = \{ f(n)$ : there exist constants $c > 0, n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0 \}$

**EXAMPLE:** $\sqrt{n} = \Omega(\lg n)$ $(c = 1, n_0 = 16)$

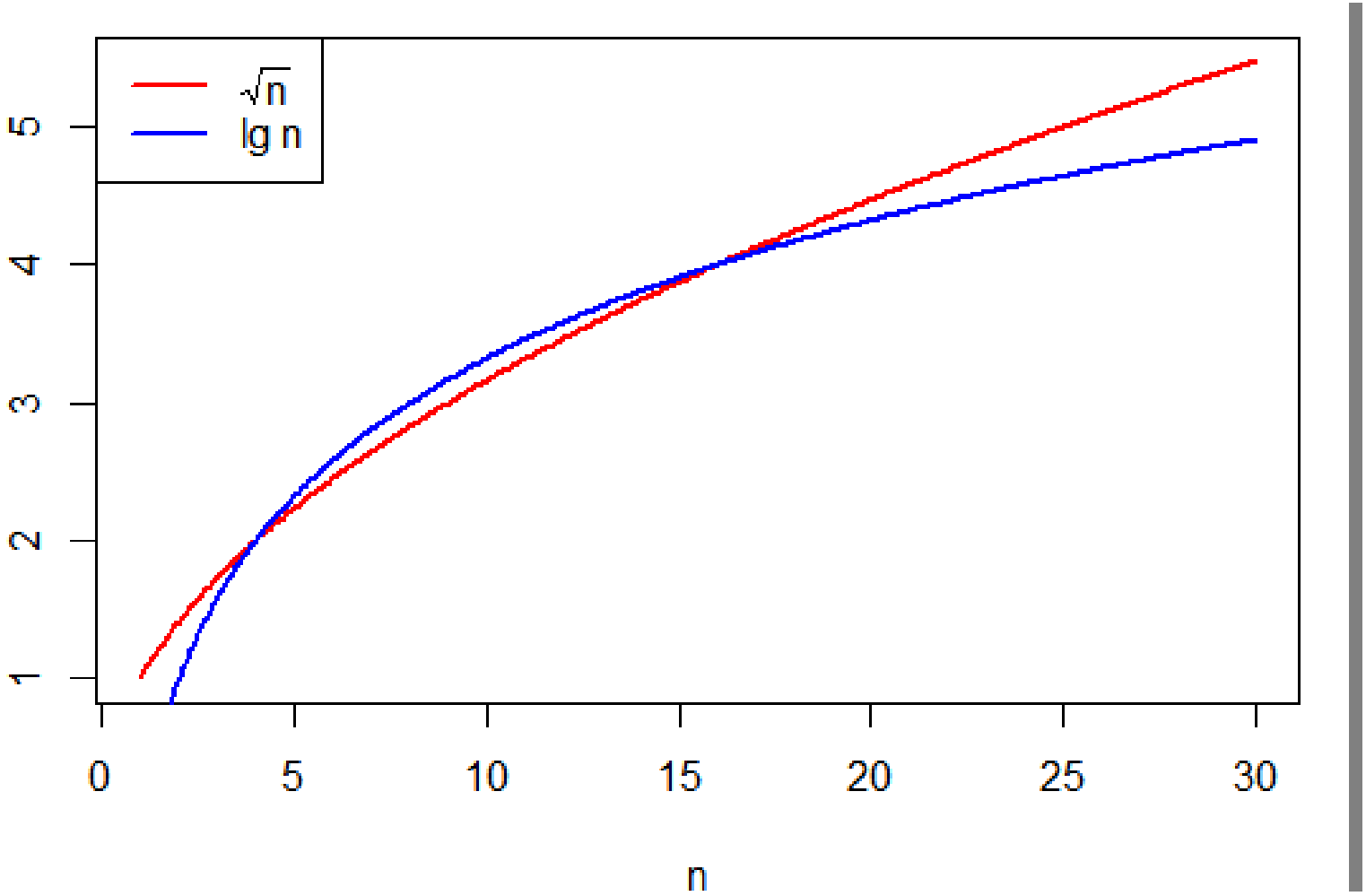



# Ω-notation (lower bounds)

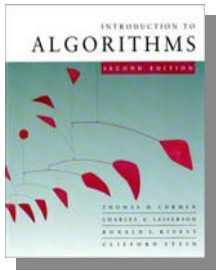





# Ω-notation (lower bounds)

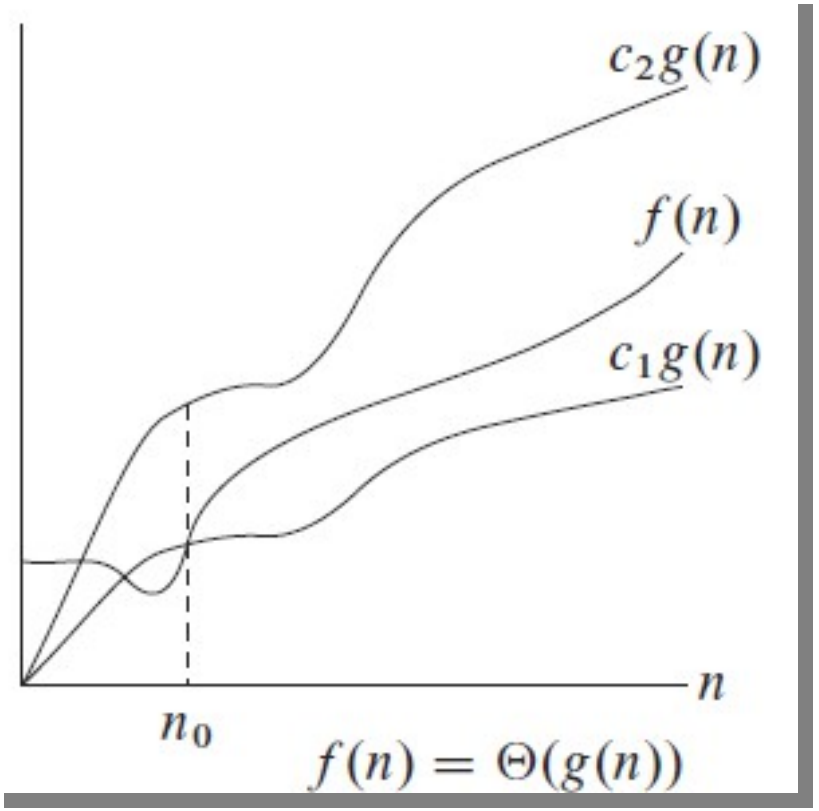


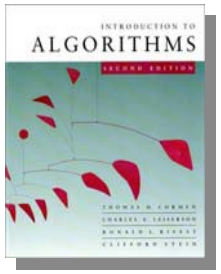

**EXAMPLE:** $\sqrt{n} = \Omega(\lg n) \quad (c = 1, n_0 = 16)$




# Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$






# Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

**EXAMPLE:** $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

# Solving Recurrences

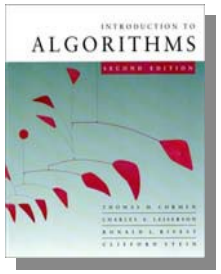
- For Merge Sort, we found that the running time was described by the recurrence

$$T(n) = 2\mathrm{T}(n/2) + cn$$

  which has a solution

$$T(n) = \Theta(n \lg n)$$

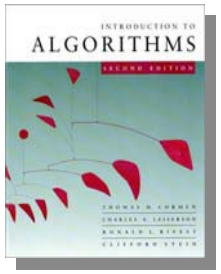
- In general, we need to solve recurrence to analyze Divide-and-Conquer algorithms




# Substitution method

*The most general method:*

1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.
3. ***Solve*** for constants.



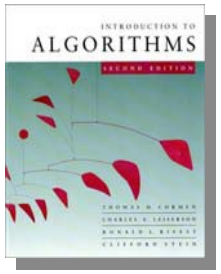

# Substitution method

*The most general method:*

1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.
3. ***Solve*** for constants.

**EXAMPLE:** $T(n) = 4T(n/2) + n$

- [Assume that $T(1) = \Theta(1)$.]
- Guess $O(n^3)$ . (Prove $O$ and $\Omega$ separately.)
- Assume that $T(k) \le ck^3$ for $k < n$ .
- Prove $T(n) \le cn^3$ by induction.

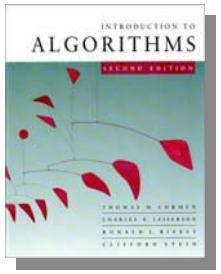



# Example of substitution

$$T(n) = 4T(n/2) + n$$
$$\le 4c(n/2)^3 + n$$
$$= (c/2)n^3 + n$$
$$= cn^3 - ((c/2)n^3 - n) \quad \leftarrow \textit{desired} - \textit{residual}$$
$$\le cn^3 \quad \leftarrow \textit{desired}$$

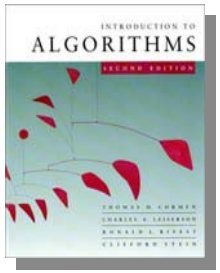
whenever $(c/2)n^3 - n \ge 0$, for example, if $c \ge 2$ and $n \ge 1$.

*residual*




# Example (continued)

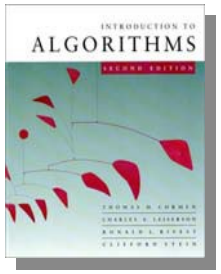
- We must also handle the initial conditions, that is, ground the induction with base cases.
- ***Base:*** $T(n) = \Theta(1)$ for all $n < n_0$, where $n_0$ is a suitable constant.
- For $1 \le n < n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick $c$ big enough.




# Example (continued)

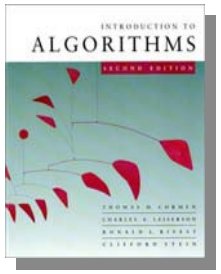
- We must also handle the initial conditions, that is, ground the induction with base cases.
- ***Base:*** $T(n) = \Theta(1)$ for all $n < n_0$, where $n_0$ is a suitable constant.
- For $1 \le n < n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick $c$ big enough.

***This bound is not tight!***




# A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

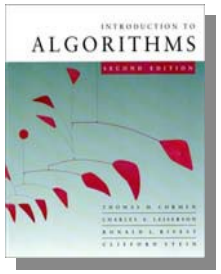



# A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= O(n^2) \end{aligned}$$



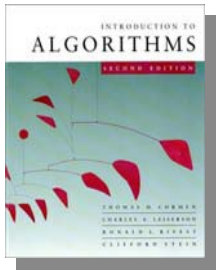

# A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^2 + n \\
&= cn^2 + n \\
&= O(n^2)
\end{aligned}
$$

***Wrong!*** We must prove the I.H.

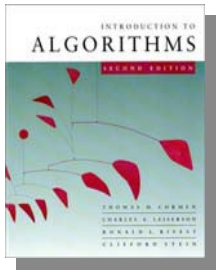



# A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^2 + n \\
&= cn^2 + n \\
&= O(n^2) \quad \textbf{\textit{Wrong!}}\ \text{We must prove the I.H.} \\
&= cn^2 - (-n) \quad [\ \text{desired} - \text{residual}\ ] \\
&\le cn^2 \quad \text{for } \textbf{\textit{no}} \text{ choice of } c > 0. \ \text{Lose!}
\end{aligned}
$$

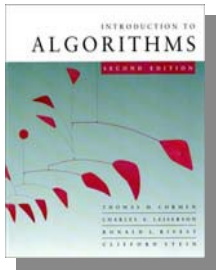



# A tighter upper bound!

**IDEA:** Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

*Inductive hypothesis*: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.



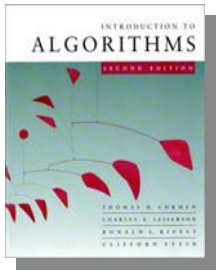

# A tighter upper bound!

**IDEA:** Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

*Inductive hypothesis*: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\le c_1 n^2 - c_2 n \quad \text{if } c_2 \ge 1.
\end{aligned}
$$




# A tighter upper bound!

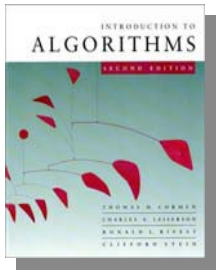
**IDEA:** Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

*Inductive hypothesis*: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.

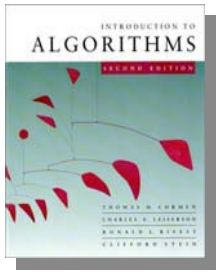
$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4(c_1(n/2)^2 - c_2(n/2)) + n \\ &= c_1 n^2 - 2c_2 n + n \\ &= c_1 n^2 - c_2 n - (c_2 n - n) \\ &\le c_1 n^2 - c_2 n \quad \text{if } c_2 \ge 1. \end{aligned}$$

Pick $c_1$ big enough to handle the initial conditions.



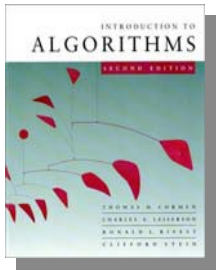

# Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method.




# Example of recursion tree

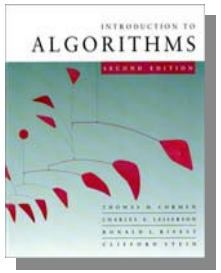
Solve $T(n) = T(n/4) + T(n/2) + n^2$:




# Example of recursion tree

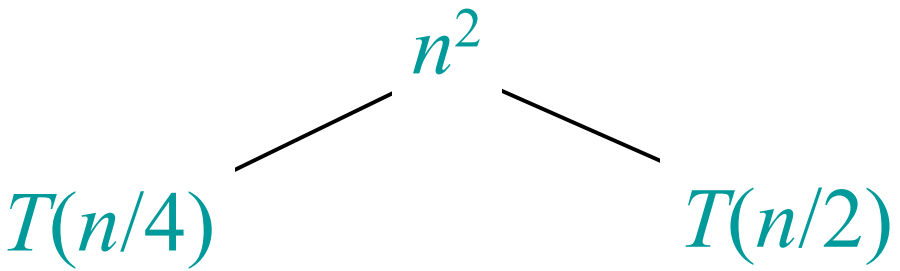
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

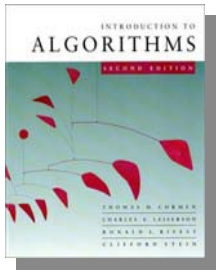
$T(n)$




# Example of recursion tree

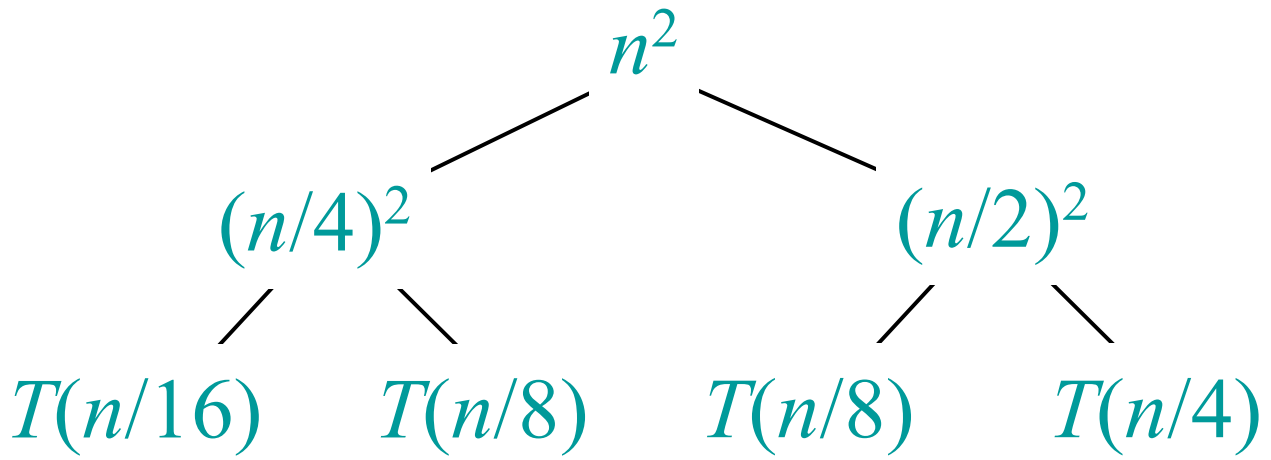
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

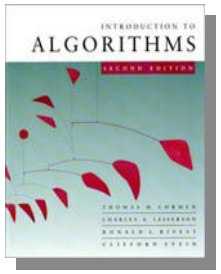





# Example of recursion tree

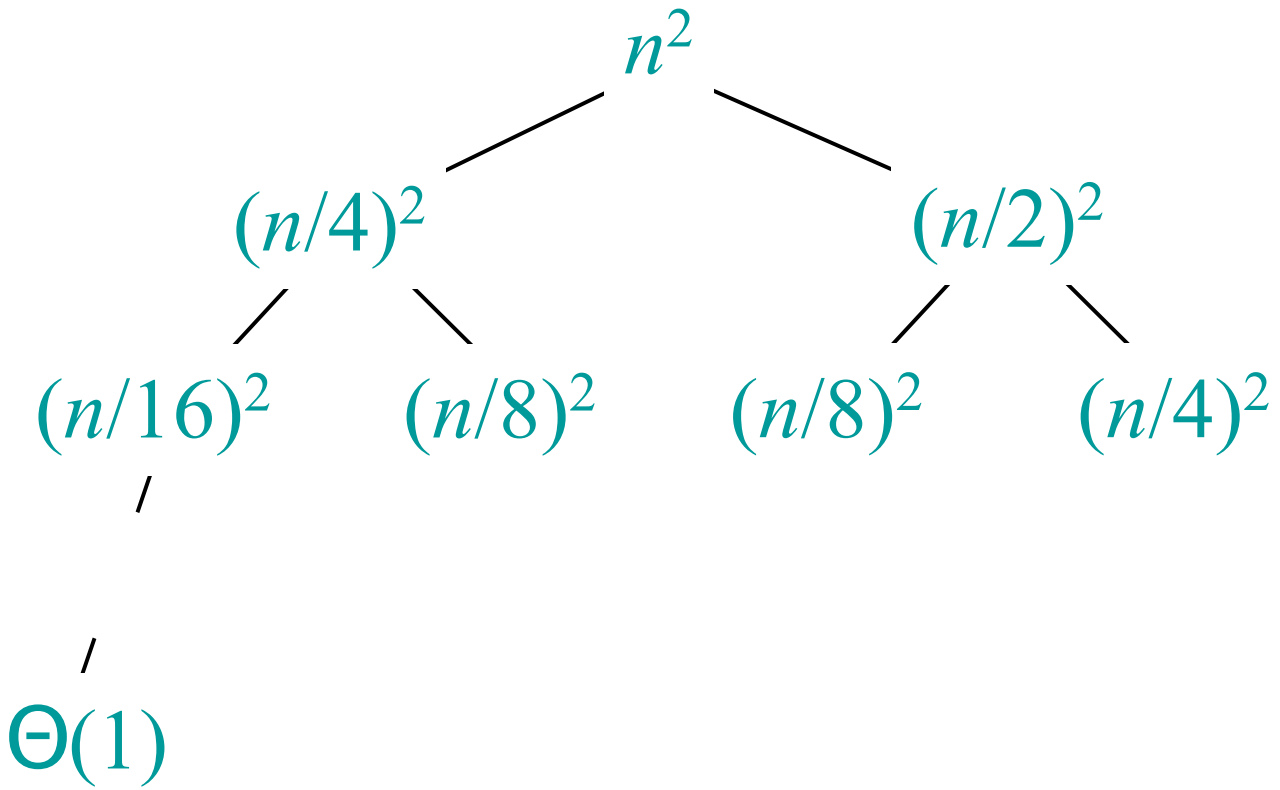
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

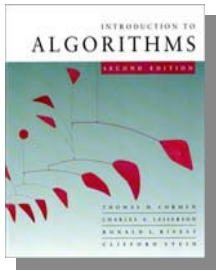





# Example of recursion tree

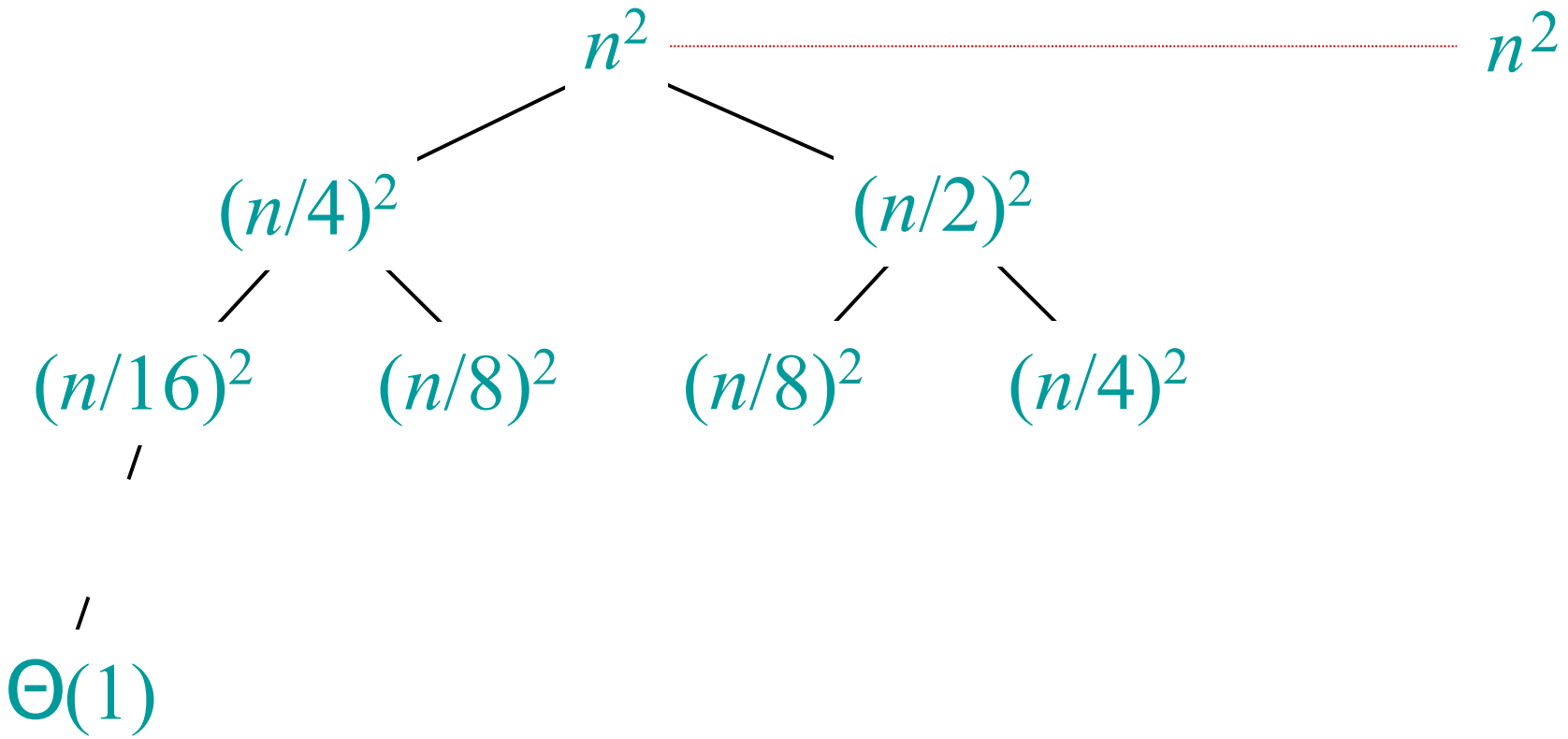
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

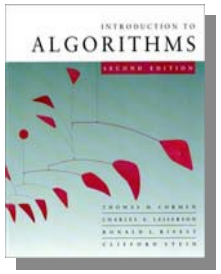





# Example of recursion tree

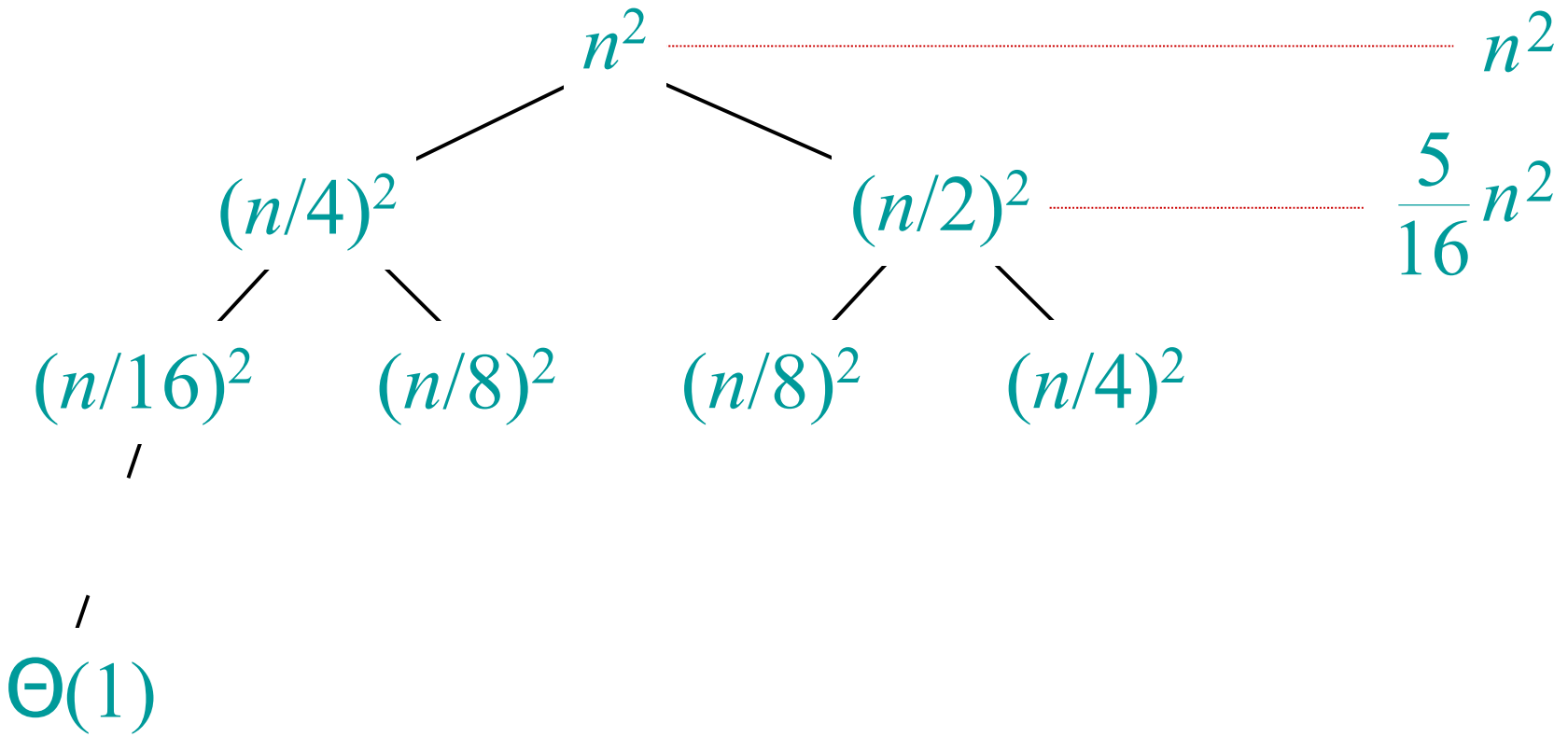
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

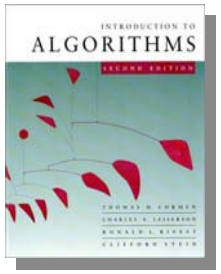





# Example of recursion tree

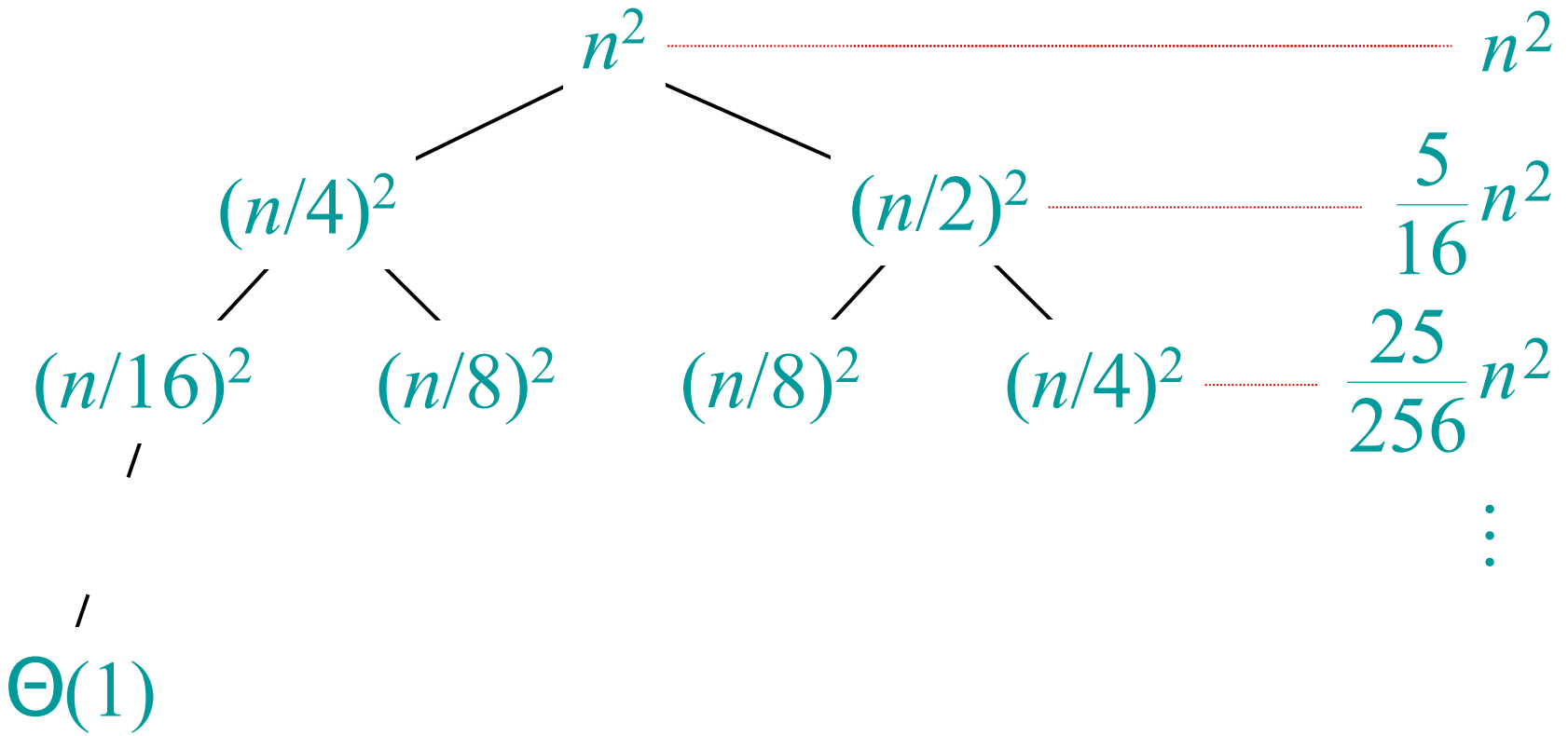
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

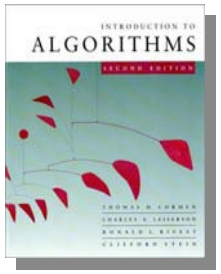





# Example of recursion tree

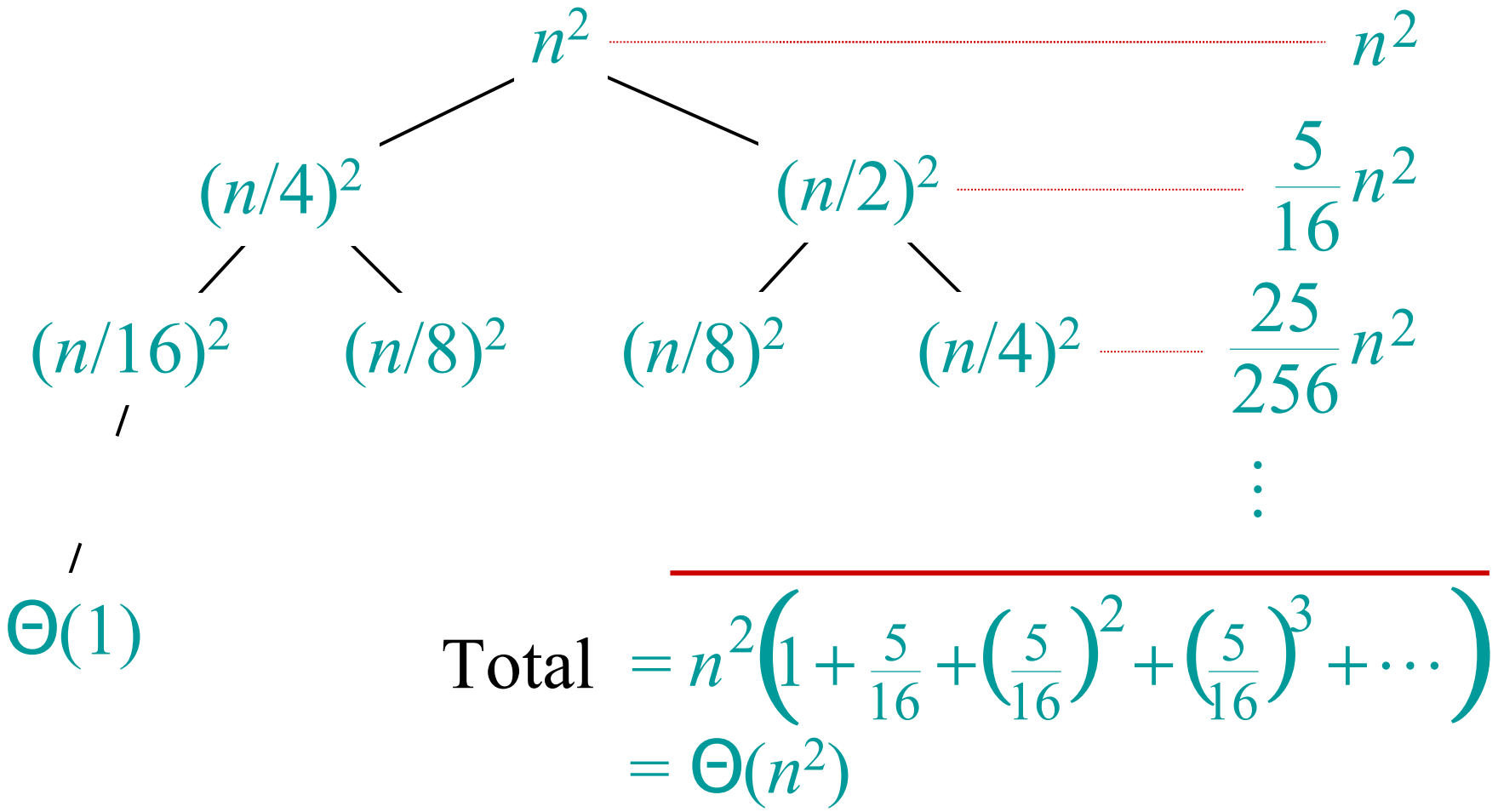
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

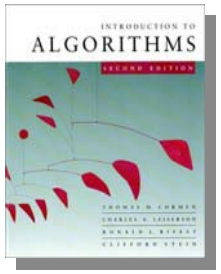





# Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



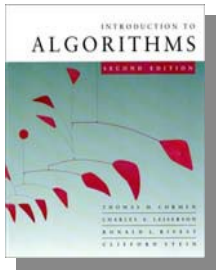



# The master method

The master method applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n)\ ,$$

where $a \geq 1$, $b > 1$, and $f$ is asymptotically positive.

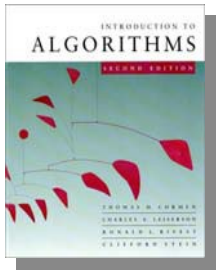



# Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an $n^{\epsilon}$ factor).

   ***Solution:*** $T(n) = \Theta(n^{\log_b a})$ .




# Three common cases

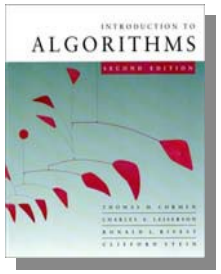
Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an $n^{\epsilon}$ factor).

   ***Solution:*** $T(n) = \Theta(n^{\log_b a})$ .

2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   ***Solution:*** $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .



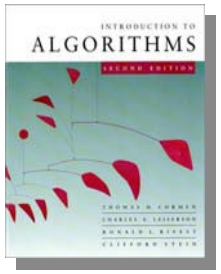

# Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an $n^{\epsilon}$ factor),

   ***and*** $f(n)$ satisfies the ***regularity condition*** that $a f(n/b) \le c f(n)$ for some constant $c < 1$.

   ***Solution:*** $T(n) = \Theta(f(n))$ .



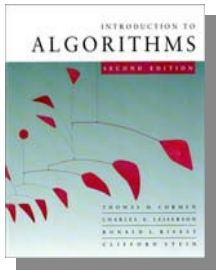

# Examples

**Ex.** $T(n) = 4T(n/2) + n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n.$

**CASE 1**: $f(n) = O(n^{2-\epsilon})$ for $\epsilon = 1$.

$\therefore\ T(n) = \Theta(n^2).$




# Examples

**Ex.** $T(n) = 4T(n/2) + n$

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n.$

**CASE 1**: $f(n) = O(n^{2-\epsilon})$ for $\epsilon = 1$.

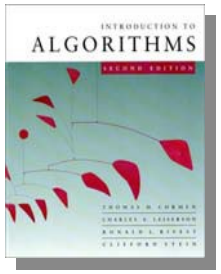
$\therefore\ T(n) = \Theta(n^2).$

**Ex.** $T(n) = 4T(n/2) + n^2$

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2.$

**CASE 2**: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

$\therefore\ T(n) = \Theta(n^2 \lg n).$




# Examples

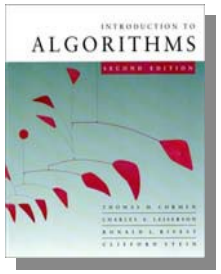
**Ex.** $T(n) = 4T(n/2) + n^3$

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^3.$

**CASE 3**: $f(n) = \Omega(n^{2+\epsilon})$ for $\epsilon = 1$

***and*** $4(n/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore\ T(n) = \Theta(n^3)$.




# Examples

**Ex.** $T(n) = 4T(n/2) + n^3$

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^3.$

**CASE 3**: $f(n) = \Omega(n^{2+\epsilon})$ for $\epsilon = 1$

***and*** $4(n/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore\ T(n) = \Theta(n^3)$.

**Ex.** $T(n) = 4T(n/2) + n^2/\lg n$

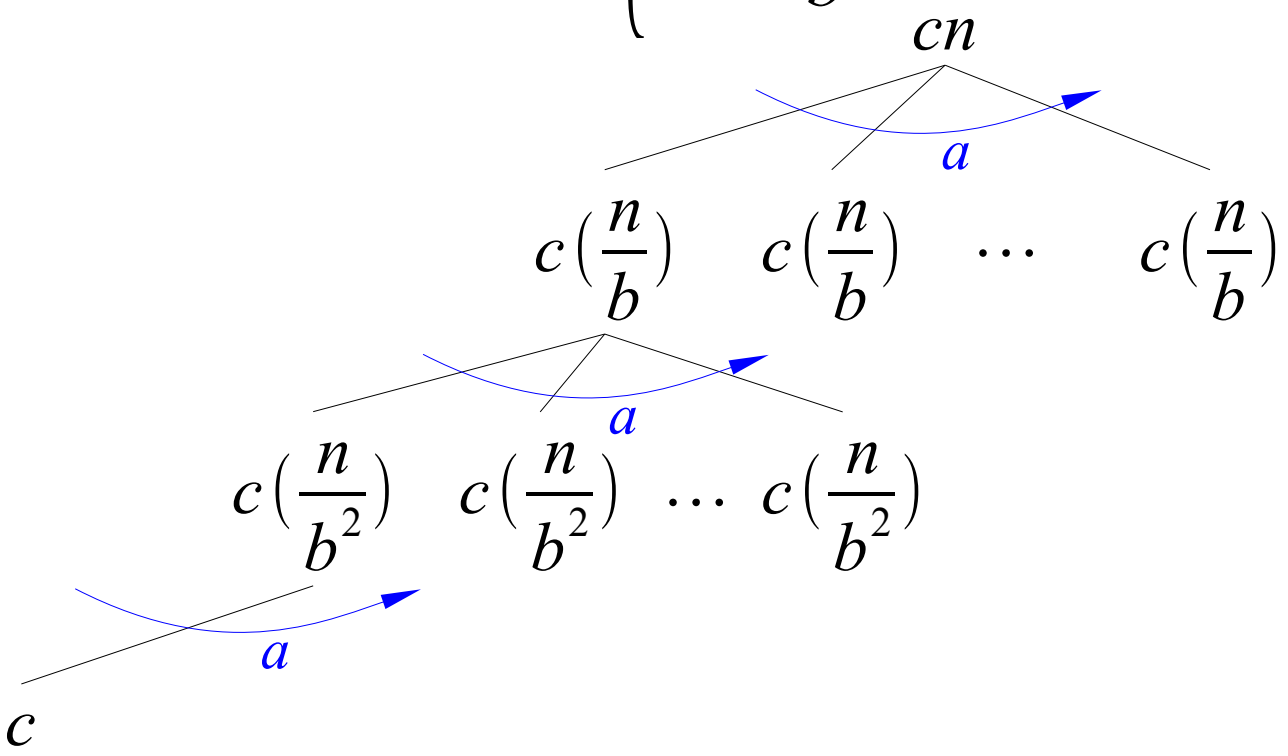
$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2/\lg n.$

Master method does not apply. In particular, for every constant $\epsilon > 0$, we have $n^\epsilon = \omega(\lg n)$.

# Intuition of Master Theorem

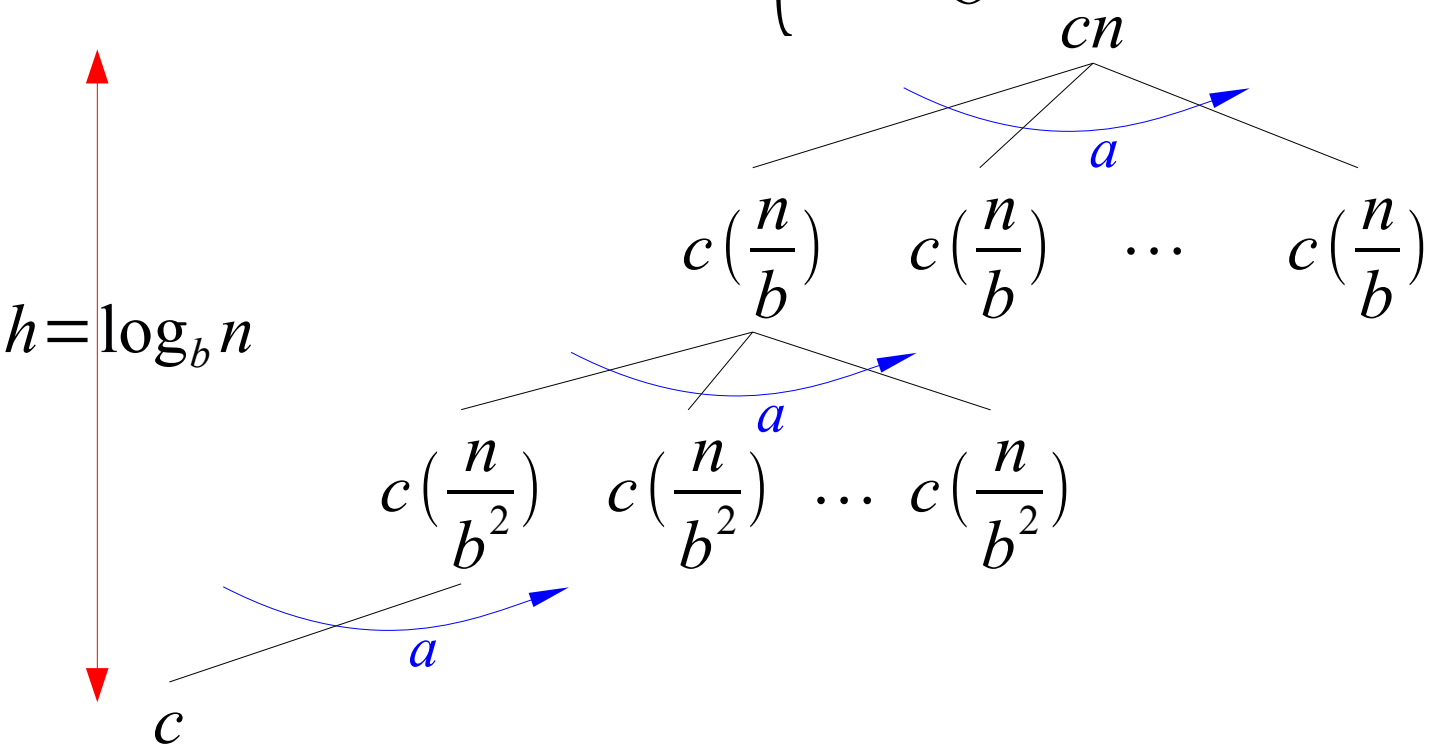
$$f(n) = cn$$

$$T(n) = \begin{cases} c & n = 1 \\ a\,T\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$



# Intuition of Master Theorem

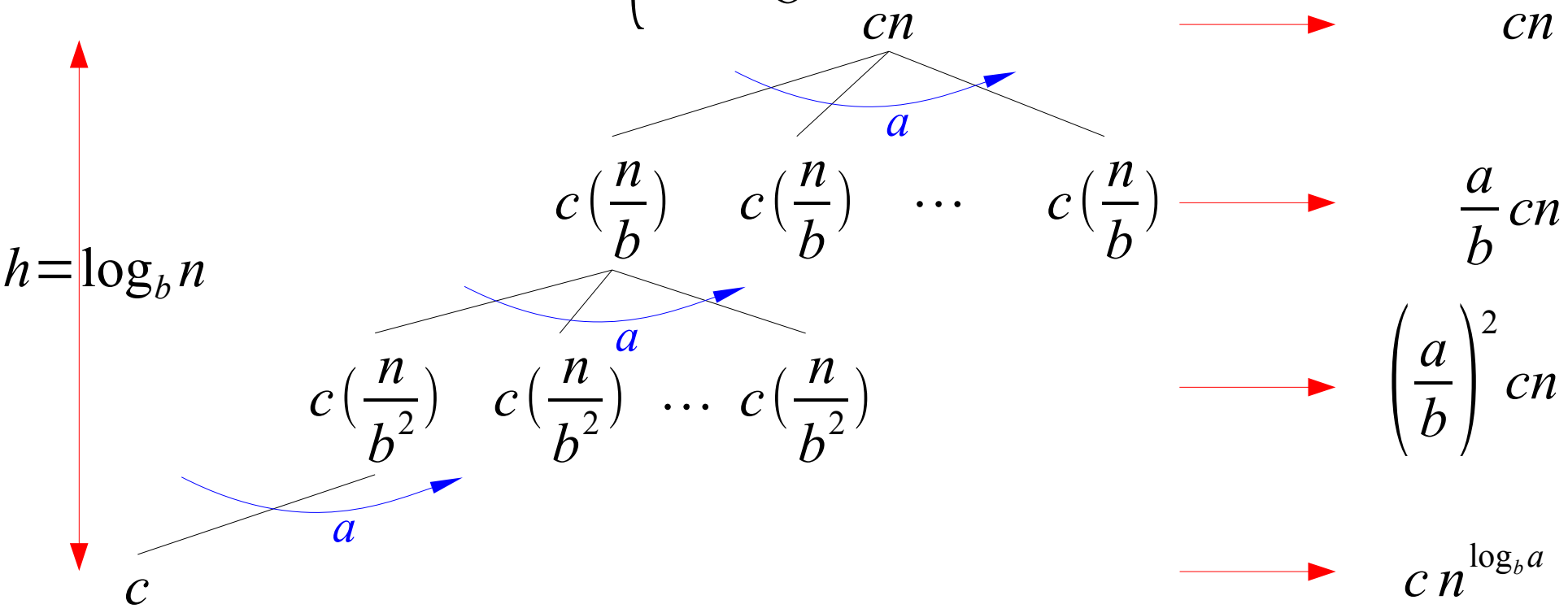
$$T(n)=\begin{cases} c & n=1 \\ a\,T\left(\frac{n}{b}\right)+cn & n>1 \end{cases}$$



# Intuition of Master Theorem

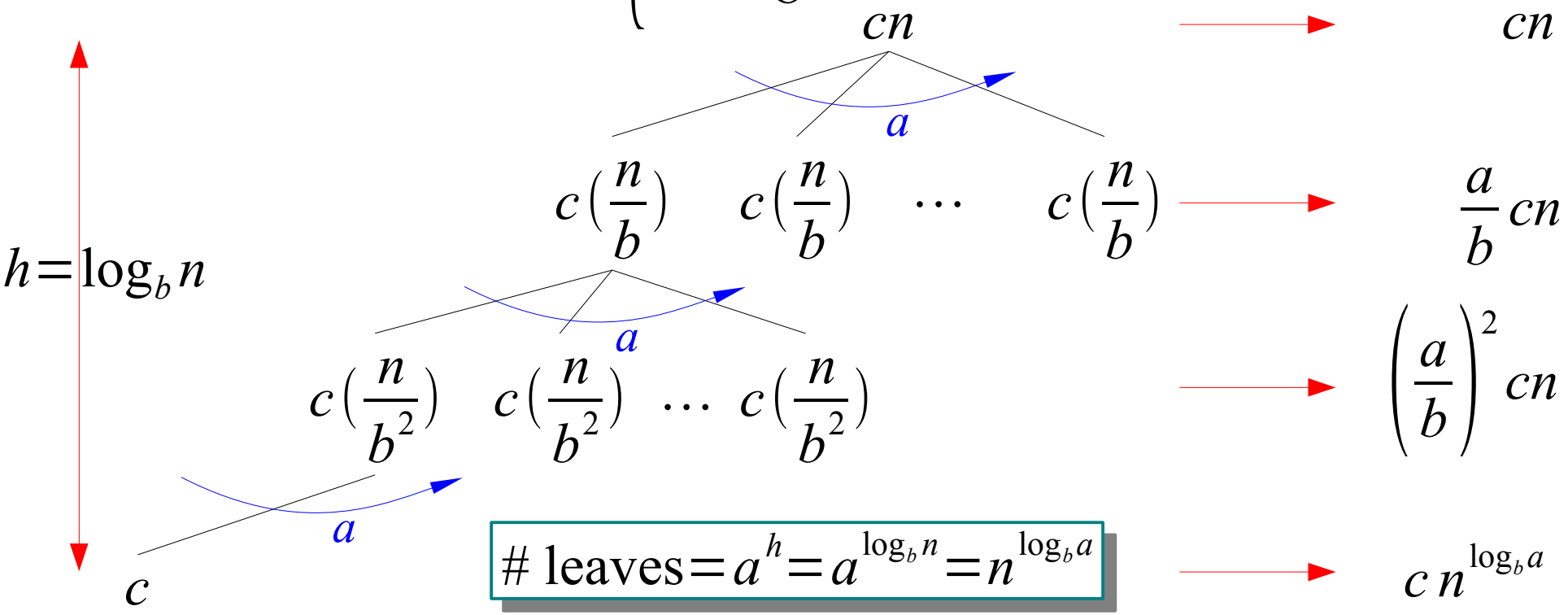
$$f(n) = cn$$

$$T(n) = \begin{cases} c & n = 1 \\ a\,T\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$



# Intuition of Master Theorem

$$T(n)=\begin{cases} f(n)=cn & \\ c & n=1 \\ a\,T\left(\frac{n}{b}\right)+cn & n>1 \end{cases}$$

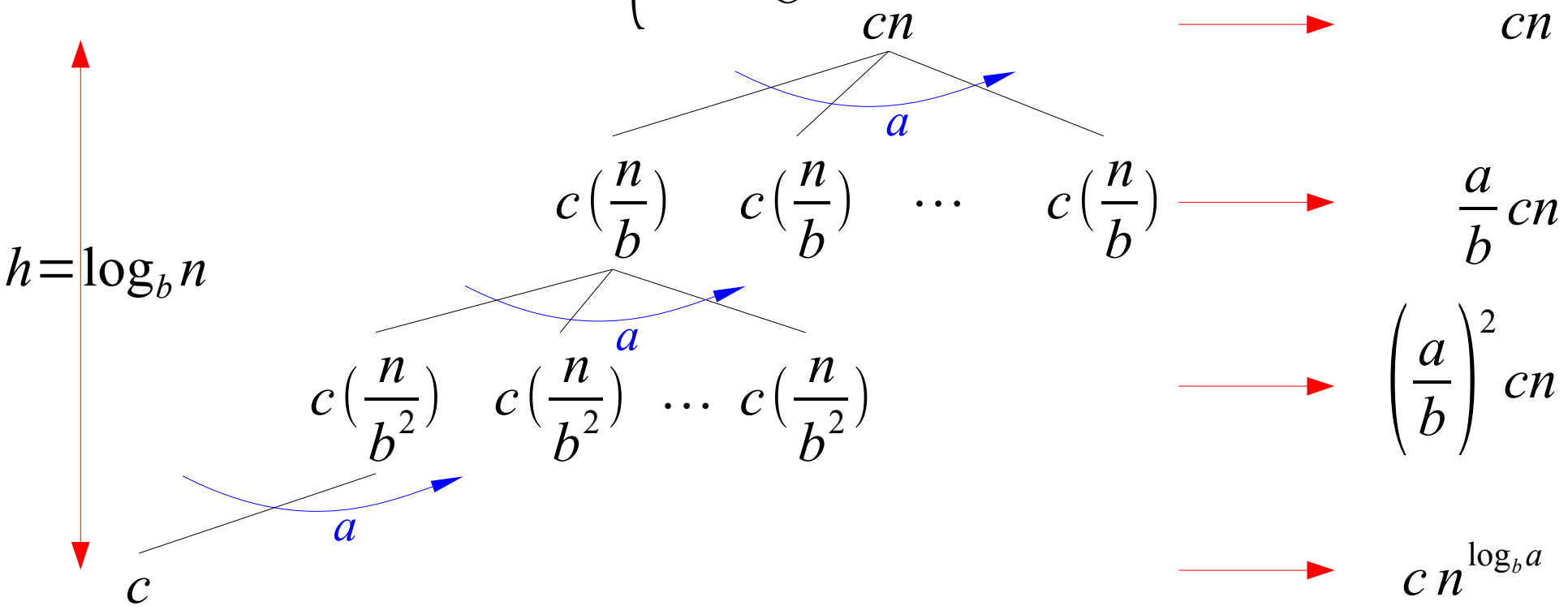



How? $a^{\log_b n}=b^{\log_b\left(a^{\log_b n}\right)}=b^{\log_b n \log_b a}=\left(b^{\log_b n}\right)^{\log_b a}=n^{\log_b a}$

# Intuition of Master Theorem

$$f(n)=cn$$

$$T(n)=\begin{cases} c & n=1 \\ a\,T\left(\frac{n}{b}\right)+cn & n>1 \end{cases}$$

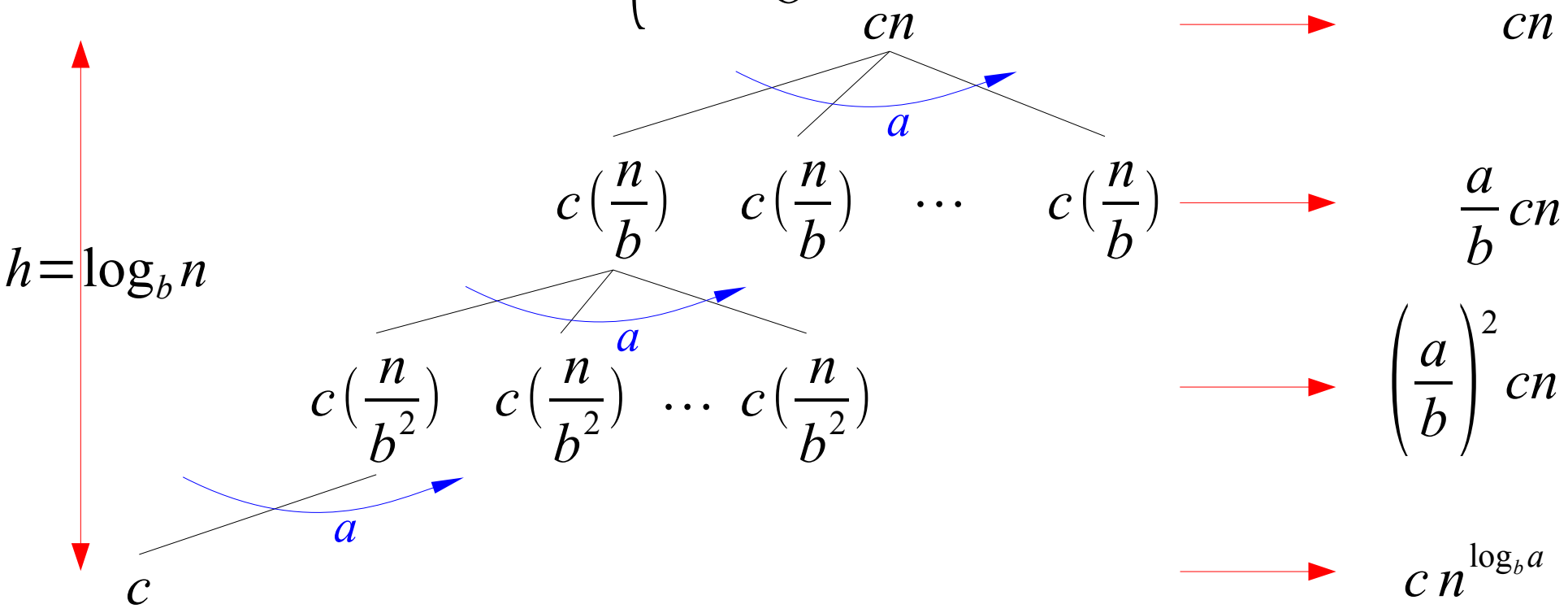



$$T(n)=cn\sum_{k=0}^{\log_b n}\left(\frac{a}{b}\right)^k=c\,n\,\frac{1-(a/b)^{\log_b n+1}}{1-a/b}$$

# Intuition of Master Theorem

$$f(n)=cn$$

$$T(n)=\begin{cases} c & n=1 \\ a\,T\left(\frac{n}{b}\right)+cn & n>1 \end{cases}$$

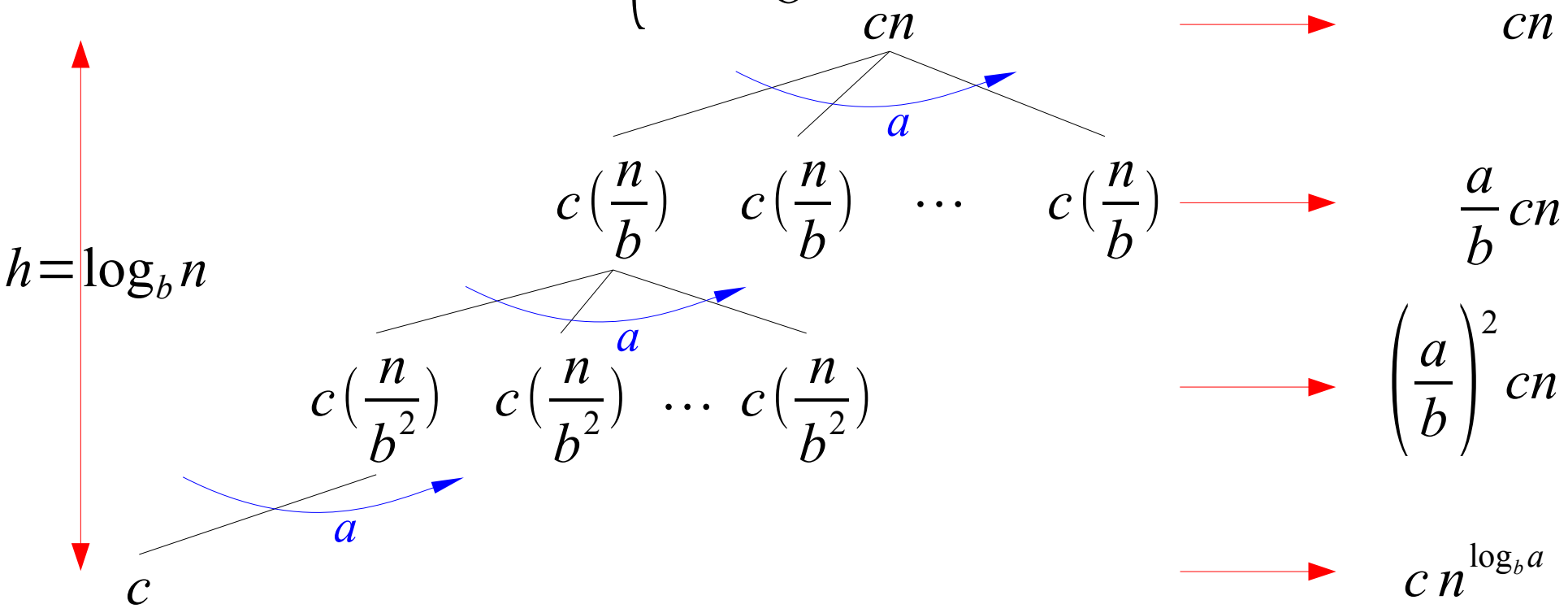



Case 1: a > b

Weights are increasing downwards $T(n)=c\,n\frac{1-(a/b)^{\log_b n+1}}{1-a/b}=\Theta\left(n^{\log_b a}\right)$

# Intuition of Master Theorem

$$T(n)=\begin{cases} c & n=1 \\ a\,T\left(\frac{n}{b}\right)+cn & n>1 \end{cases} \qquad f(n)=cn$$

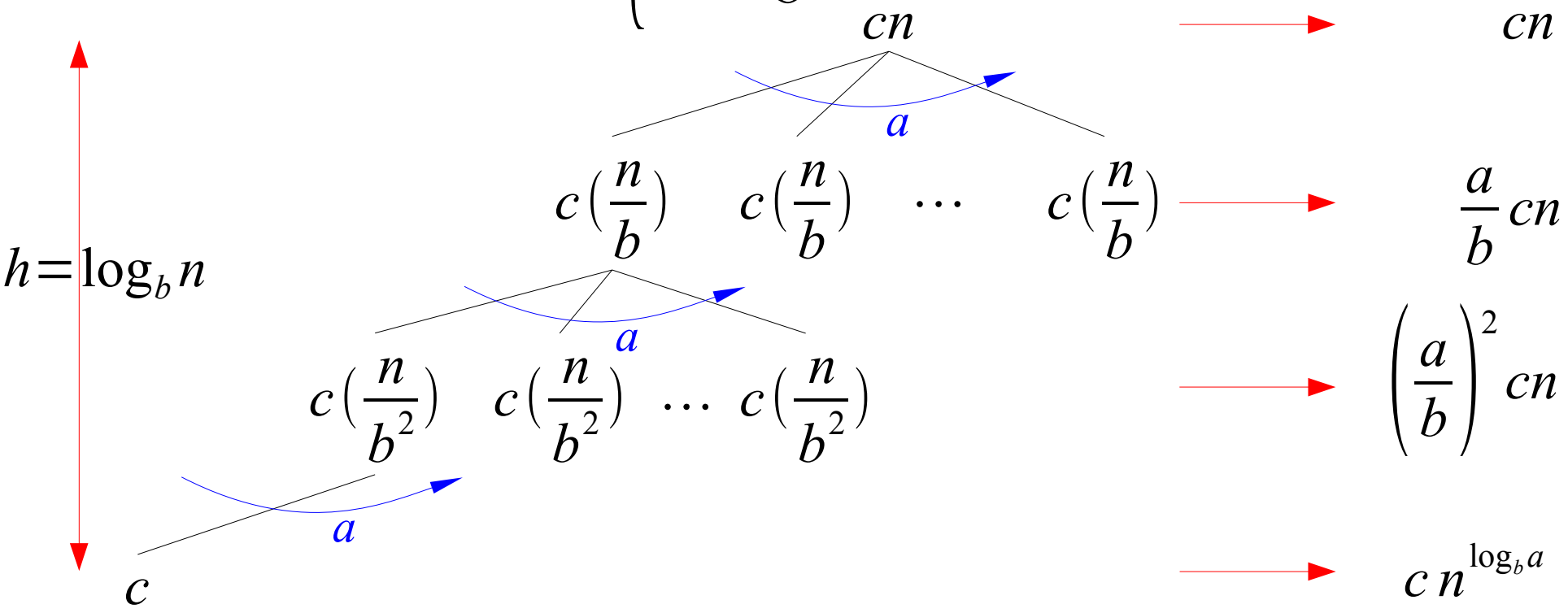



Case 2: a = b

Weights are equal across levels $\quad T(n)=c\,n\,\frac{1-(a/b)^{\log_b n+1}}{1-a/b}=\Theta(n\log_b n)$

# Intuition of Master Theorem

$$T(n)=\begin{cases} c & n=1 \\ a\,T\left(\frac{n}{b}\right)+cn & n>1 \end{cases} \qquad f(n)=cn$$




Case 3: a < b

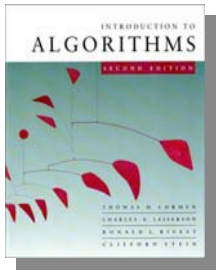
Weights are decreasing downwards $\quad T(n)=c\,n\,\frac{1-(a/b)^{\log_b n+1}}{1-a/b}=\Theta(n)$

# Intuition of Master Theorem

## Summary

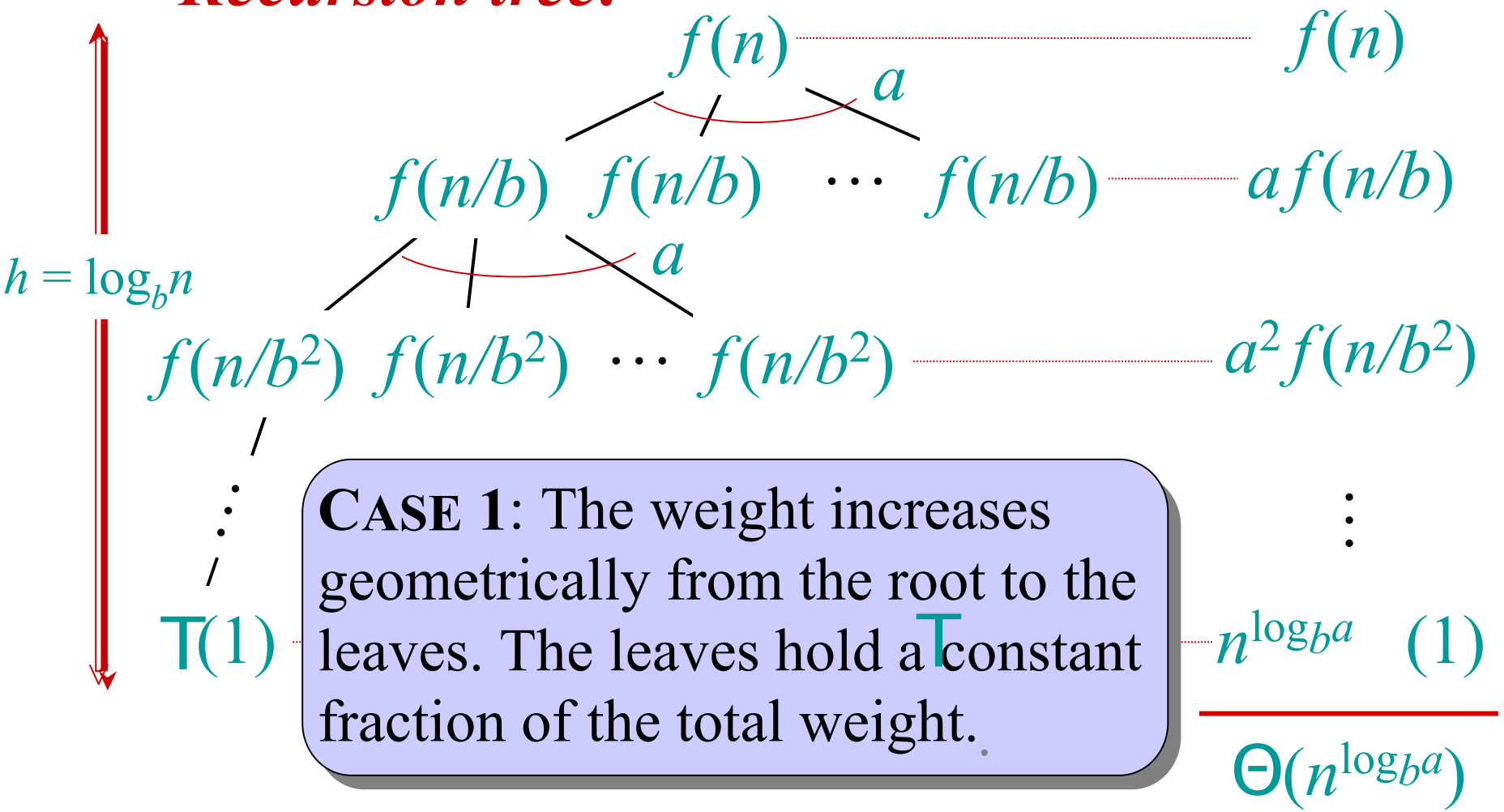
$$T(n)=\begin{cases} f(n)=cn & \\ c & n=1 \\ a\,T\left(\frac{n}{b}\right)+cn & n>1 \end{cases}$$

$$T(n)=\begin{cases} \Theta(n) & ,a<b \\ \Theta(n\log_b n) & ,a=b \\ \Theta\left(n^{\log_b a}\right) & ,a>b \end{cases}$$

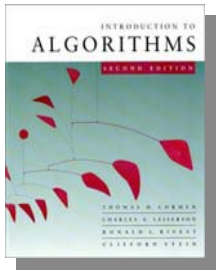



# Idea of master theorem

***Recursion tree:***

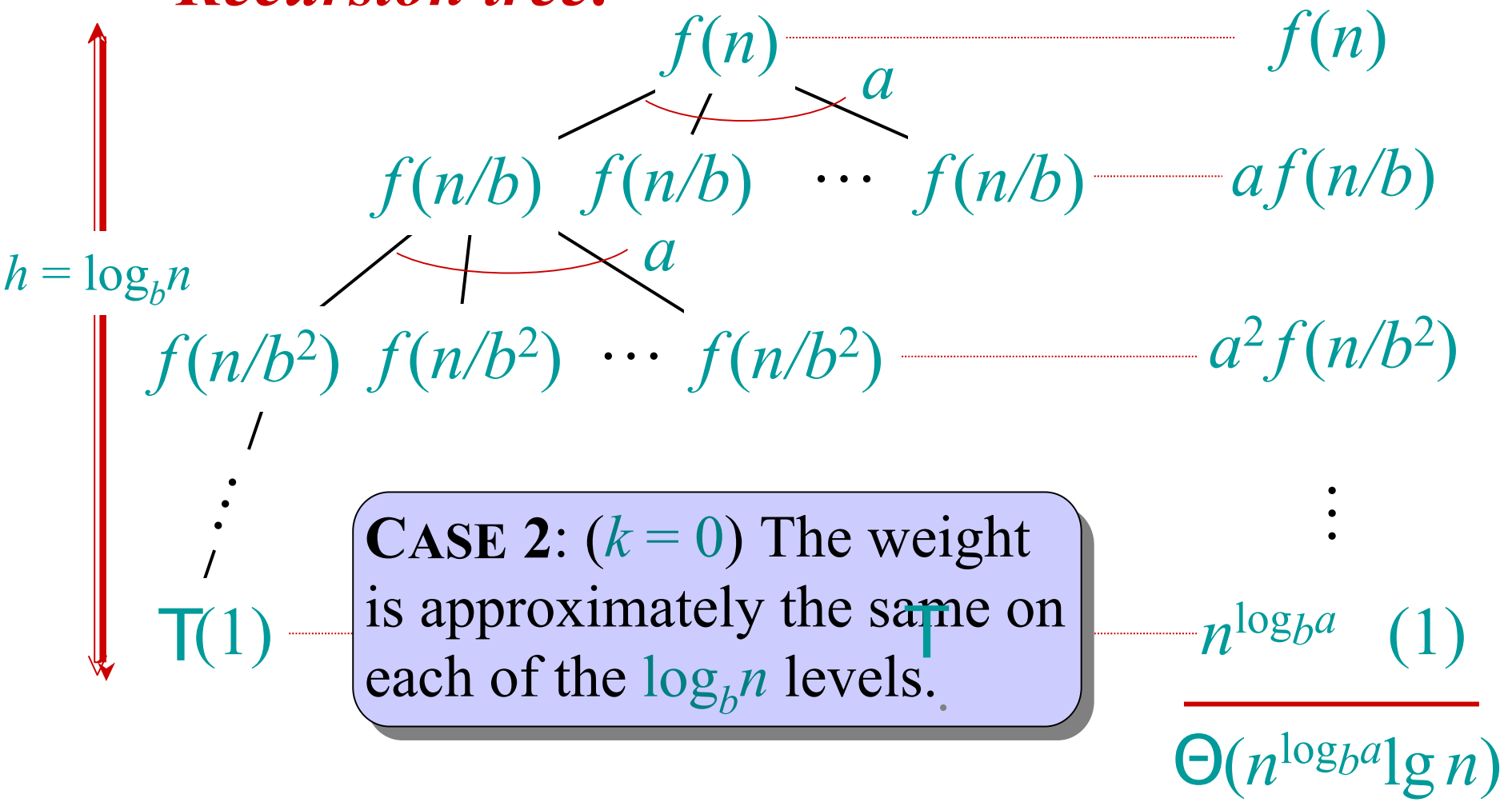



**CASE 1**: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

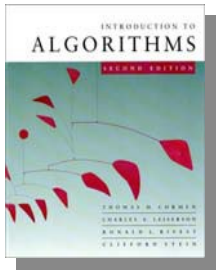



# Idea of master theorem

***Recursion tree:***

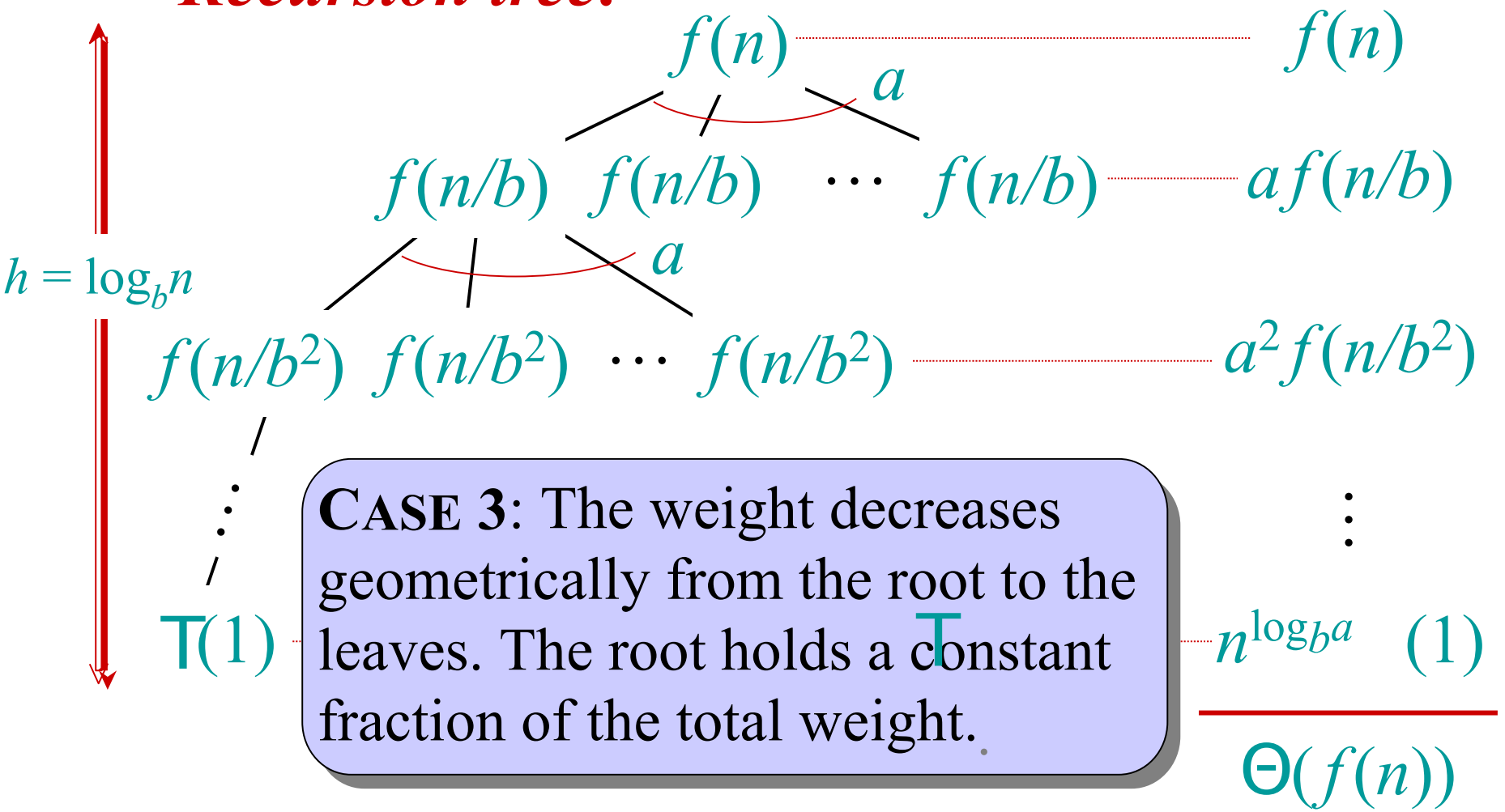



**CASE 2**: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

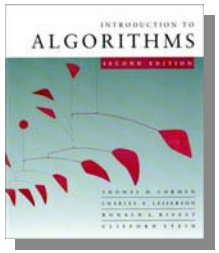



# Idea of master theorem

***Recursion tree:***

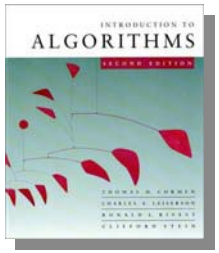



**CASE 3**: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.




# The divide-and-conquer design paradigm

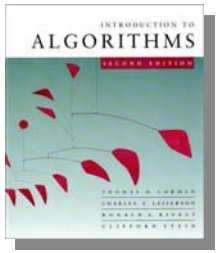
1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.




# Merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.




# Merge sort

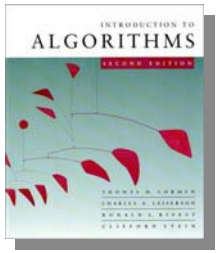
1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = 2\,T(n/2) + \Theta(n)$$

*# subproblems*

*subproblem size*

*work dividing and combining*




# Master theorem (reprise)

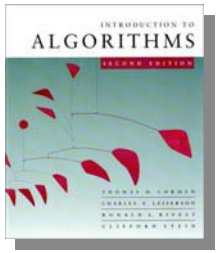
$$T(n) = a\,T(n/b) + f(n)$$

**CASE 1**: $f(n) = O(n^{\log_b a - \epsilon})$, constant $\epsilon > 0$
$\Rightarrow T(n) = \Theta(n^{\log_b a})$ .

**CASE 2**: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$
$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .

**CASE 3**: $f(n) = \Omega(n^{\log_b a + \epsilon})$, constant $\epsilon > 0$, and regularity condition
$\Rightarrow T(n) = \Theta(f(n))$ .

***Merge sort:*** $a = 2$, $b = 2$ $\Rightarrow$ $n^{\log_b a} = n^{\log_2 2} = n$
$\Rightarrow$ CASE 2 ($k = 0$) $\Rightarrow$ $T(n) = \Theta(n \lg n)$ .

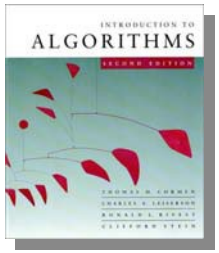



# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.



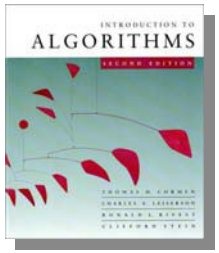

# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

| 3 | 5 | 7 | 8 | 9 | 12 | 15 |
|---|---|---|---|---|---|---|



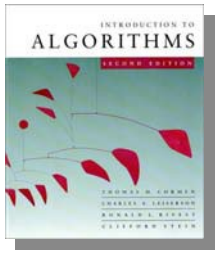

# Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

| 3 | 5 | 7 | 8 | 9 | 12 | 15 |
|---|---|---|---|---|---|---|




# Binary search

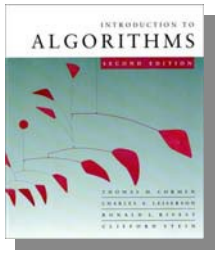
Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 9 12 15




# Binary search

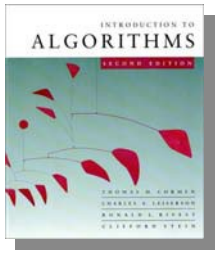
Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 9 12 15




# Binary search

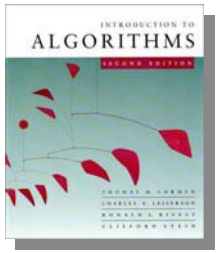
Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 **9** 12 15




# Binary search

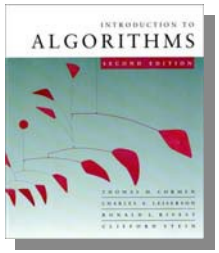
Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

***Example:*** Find 9

3 5 7 8 **9** 12 15




# Recurrence for binary search

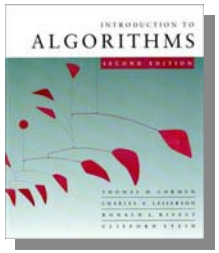
$$T(n) = 1\,T(n/2) + \Theta(1)$$

*# subproblems*

*subproblem size*

*work dividing and combining*




# Recurrence for binary search

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

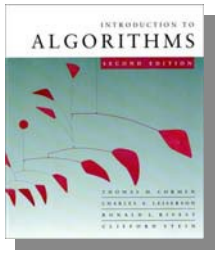
*# subproblems*

*subproblem size*

*work dividing and combining*

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$

$$\Rightarrow T(n) = \Theta(\lg n) .$$

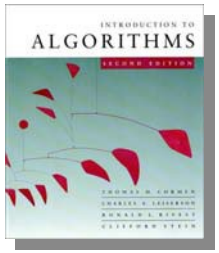



# Powering a number

**Problem:** Compute $a^n$, where $n \in \mathbb{N}$.

**Naive algorithm:** $\Theta(n)$.




# Powering a number

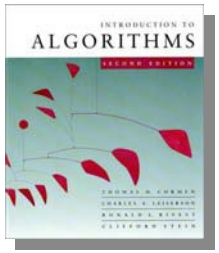
**Problem:** Compute $a^n$, where $n \in \mathbb{N}$.

**Naive algorithm:** $\Theta(n)$.

**Divide-and-conquer algorithm:**

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$




# Powering a number

**Problem:** Compute $a^n$, where $n \in \mathbb{N}$.

**Naive algorithm:** $\Theta(n)$.

**Divide-and-conquer algorithm:**

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n) .$$

# Recap

- Asymptotic Notation
  - O-, Ω-, and Θ-notation
- Recurrences
  - Substitution Method
  - Recursion Tree
  - Master Theorem
- Divide-and-Conquer Examples
- Next:
  - Heapsort and Quicksort