

CMP302: Algorithms



Lecture 03: Recurrences and Divide-and-Conquer

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

Agenda

- 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

Solving Recurrences

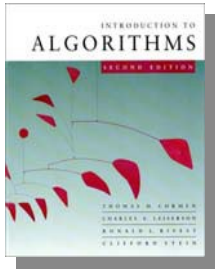
- For Merge Sort, we found that the running time was described by the recurrence

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

which has a solution

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

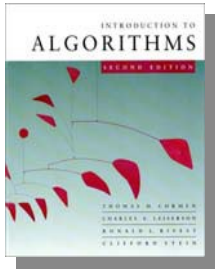
- In general, we need to solve recurrence to analyze Divide-and-Conquer algorithms



Substitution method

The most general method:

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



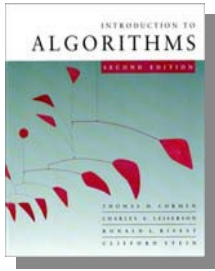
Substitution method

The most general method:

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

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

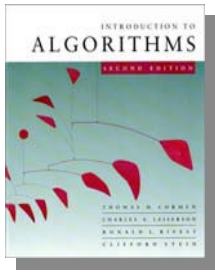
- [Assume that $T(1) = \Theta(1)$.]
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \leq ck^3$ for $k < n$.
- Prove $T(n) \leq cn^3$ by induction.



Example of substitution

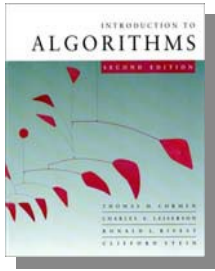
$$\begin{aligned}T(n) &= 4T(n/2) + n \\ &\leq 4c(n/2)^3 + n \\ &= (c/2)n^3 + n \\ &= cn^3 - ((c/2)n^3 - n) \leftarrow \textit{desired} - \textit{residual} \\ &\leq cn^3 \leftarrow \textit{desired}\end{aligned}$$

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



Example (continued)

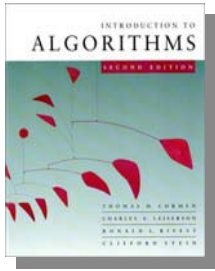
- We must also handle the initial conditions, that is, ground the induction with base cases.
- **Base:** $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
- For $1 \leq n < n_0$, we have “ $\Theta(1)$ ” $\leq cn^3$, if we pick c big enough.



Example (continued)

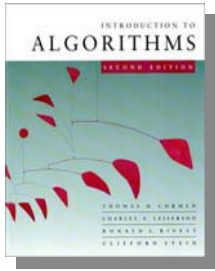
- We must also handle the initial conditions, that is, ground the induction with base cases.
 - **Base:** $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
 - For $1 \leq n < n_0$, we have “ $\Theta(1)$ ” $\leq cn^3$, if we pick c big enough.
-

This bound is not tight!



A tighter upper bound?

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

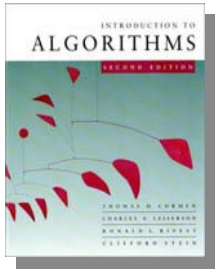


A tighter upper bound?

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

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

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



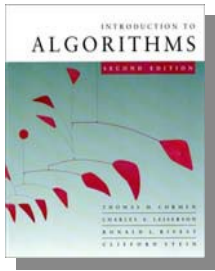
A tighter upper bound?

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

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

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

~~$O(n^2)$~~ **Wrong!** We must prove the I.H.



A tighter upper bound?

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

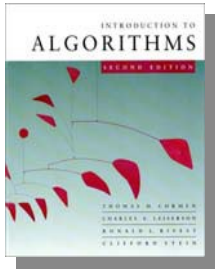
Assume that $T(k) \leq ck^2$ for $k < n$:

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

~~$O(n^2)$~~ **Wrong!** We must prove the I.H.

$$= cn^2 - (-n) \quad [\text{desired} - \text{residual}]$$

$\leq cn^2$ for **no** choice of $c > 0$. Lose!

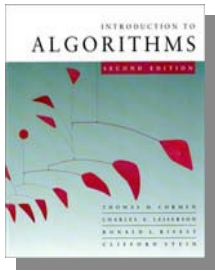


A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

- *Subtract* a low-order term.

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



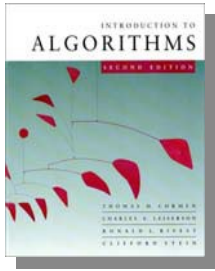
A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

- *Subtract* a low-order term.

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

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



A tighter upper bound!

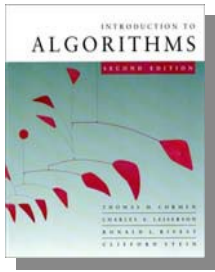
IDEA: Strengthen the inductive hypothesis.

- *Subtract* a low-order term.

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

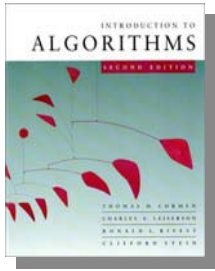
$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4(c_1(n/2)^2 - c_2(n/2)) + n \\ &= c_1 n^2 - 2c_2 n + n \\ &= c_1 n^2 - c_2 n - (c_2 n - n) \\ &\leq c_1 n^2 - c_2 n \quad \text{if } c_2 \geq 1. \end{aligned}$$

Pick c_1 big enough to handle the initial conditions.



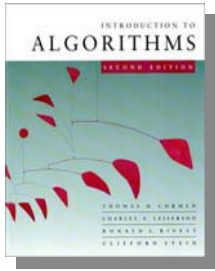
Recursion-tree method

- 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

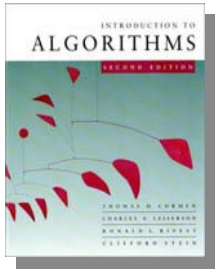
Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

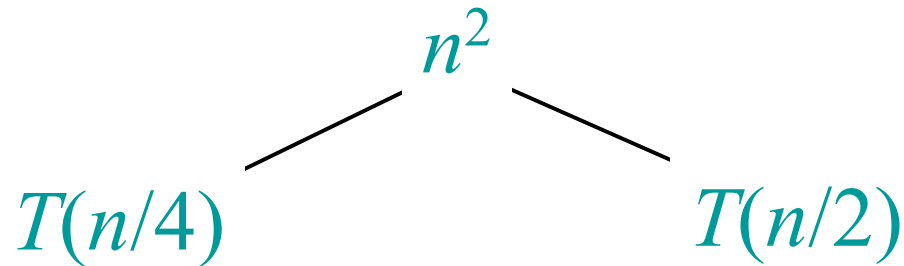
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

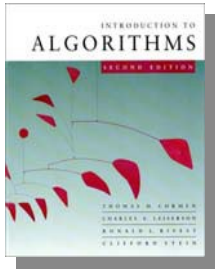
$$T(n)$$



Example of recursion tree

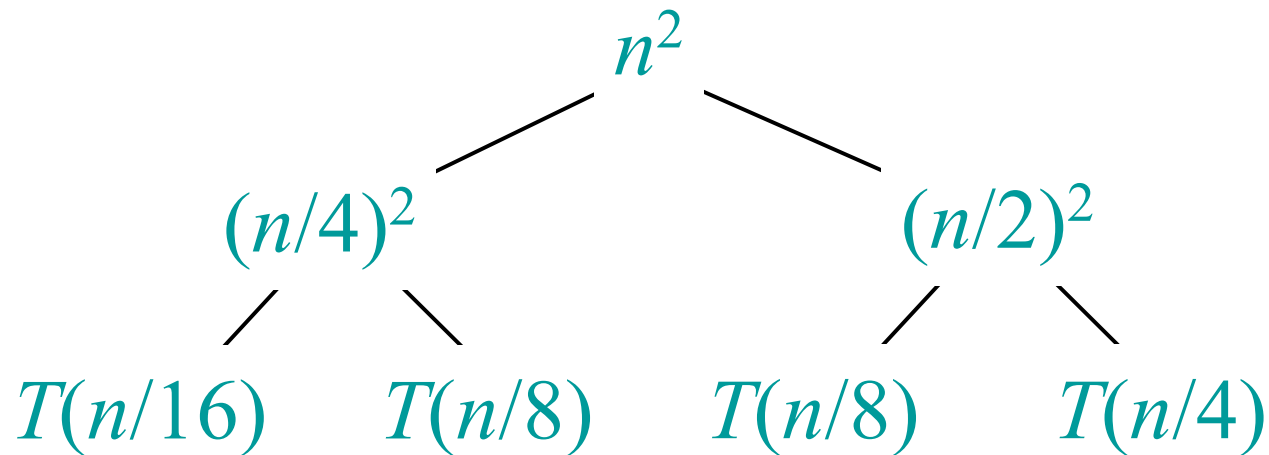
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

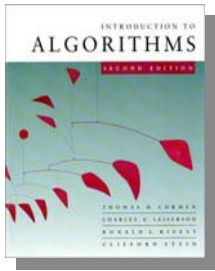




Example of recursion tree

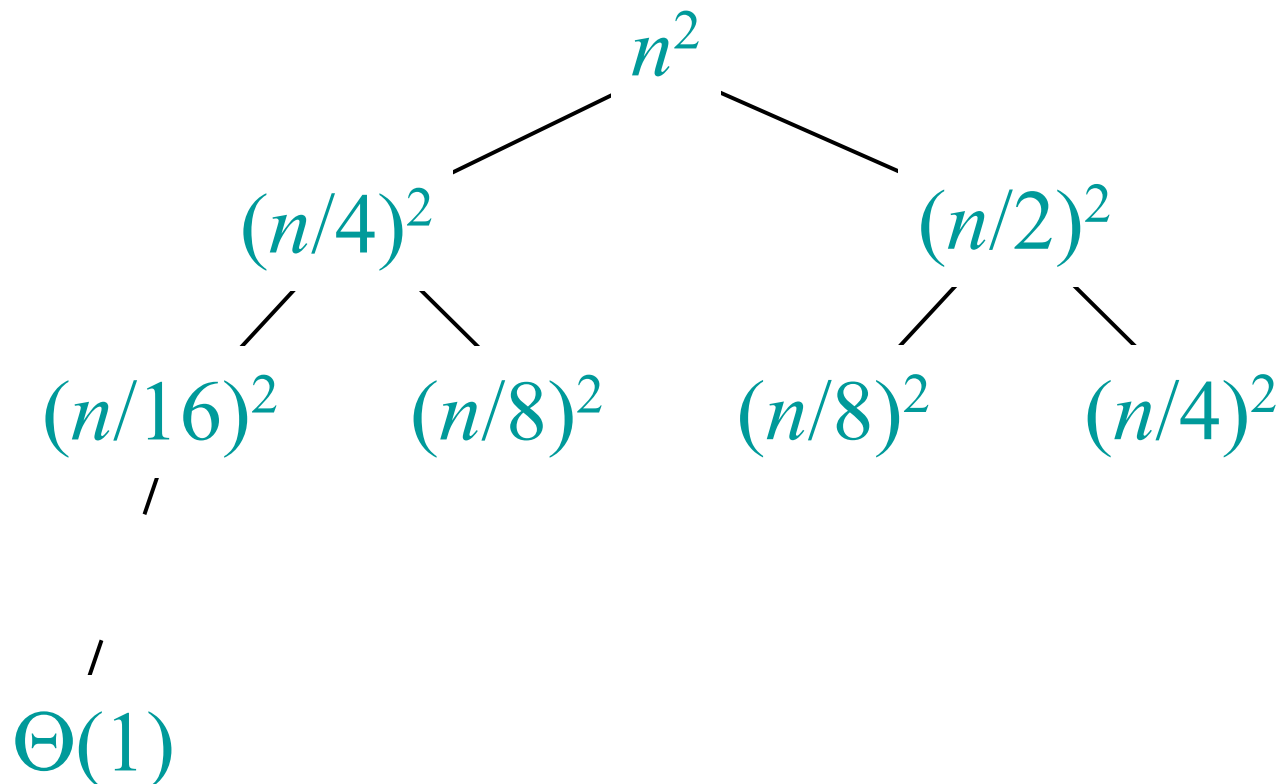
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

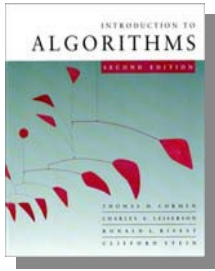




Example of recursion tree

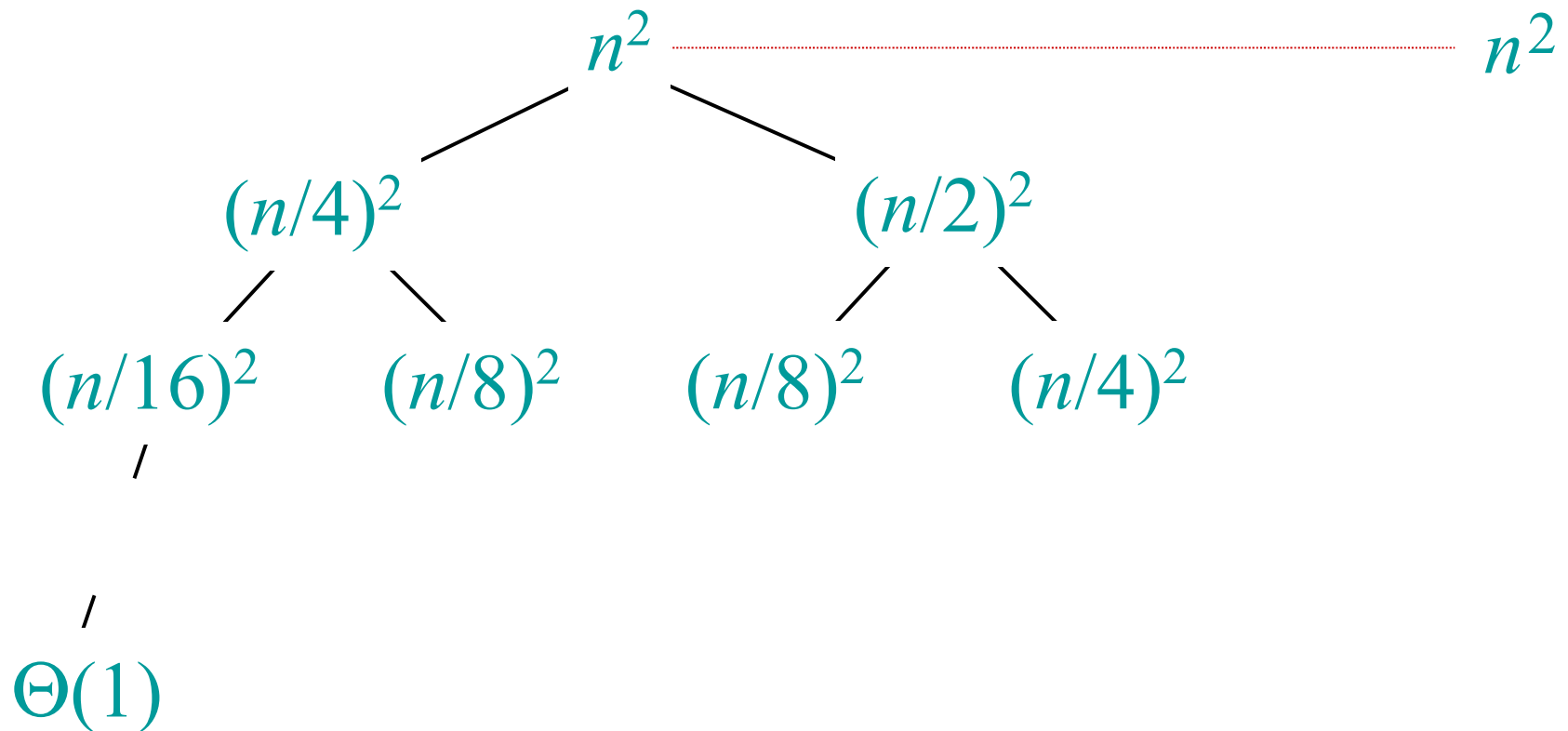
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

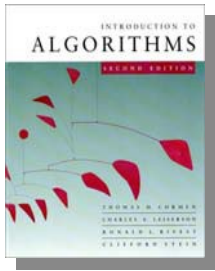




Example of recursion tree

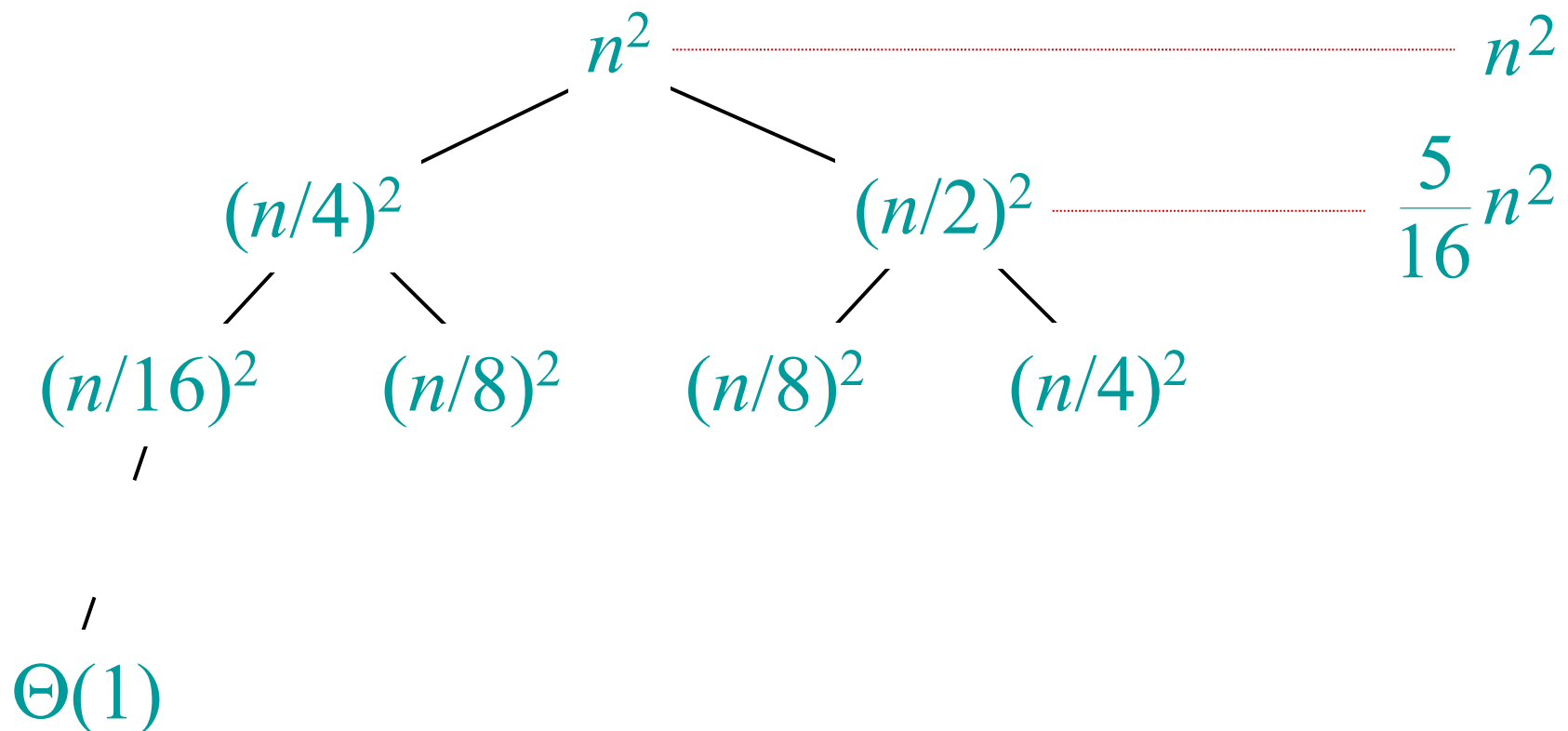
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

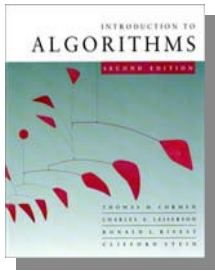




Example of recursion tree

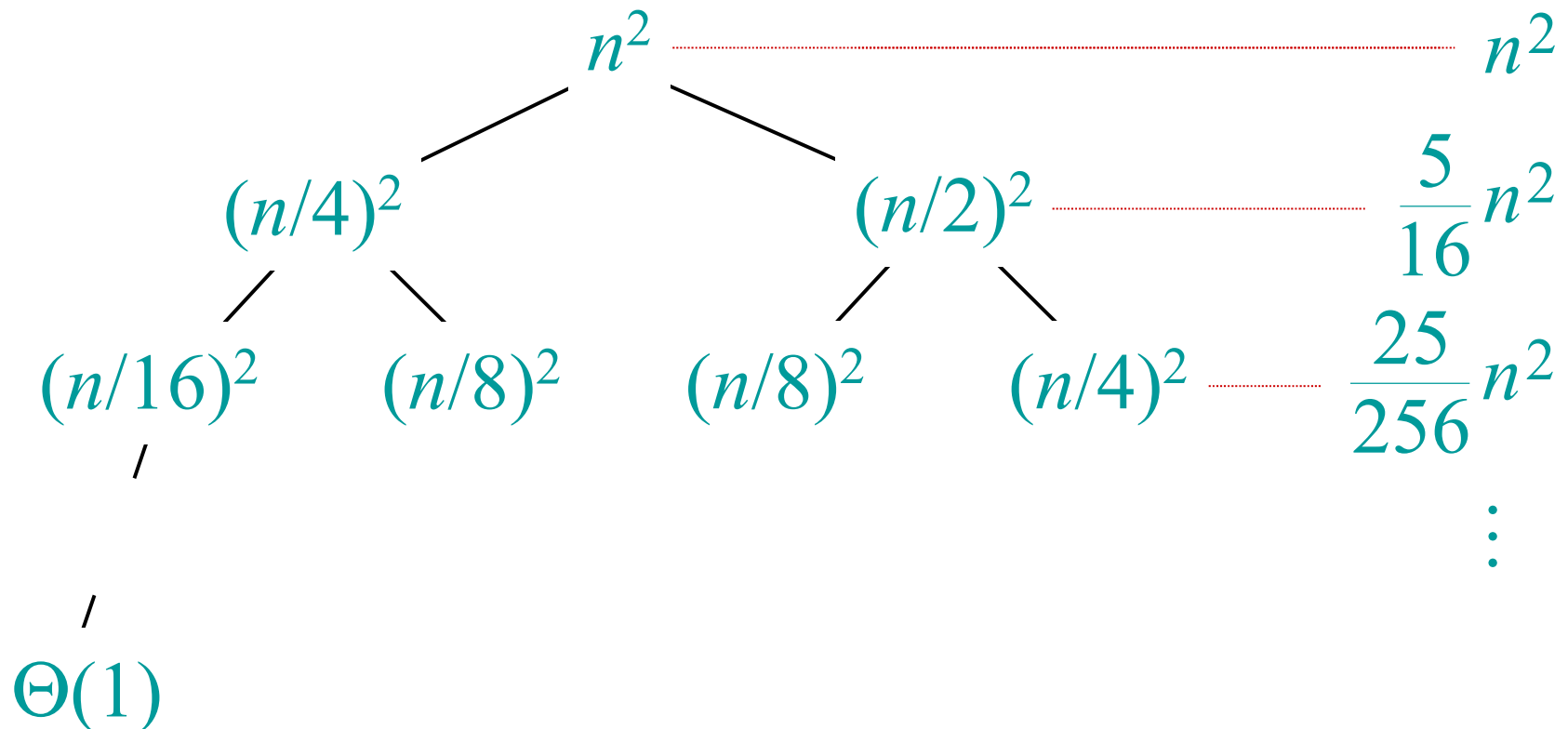
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

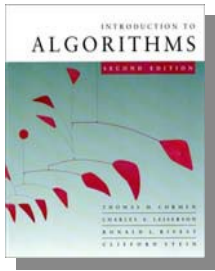




Example of recursion tree

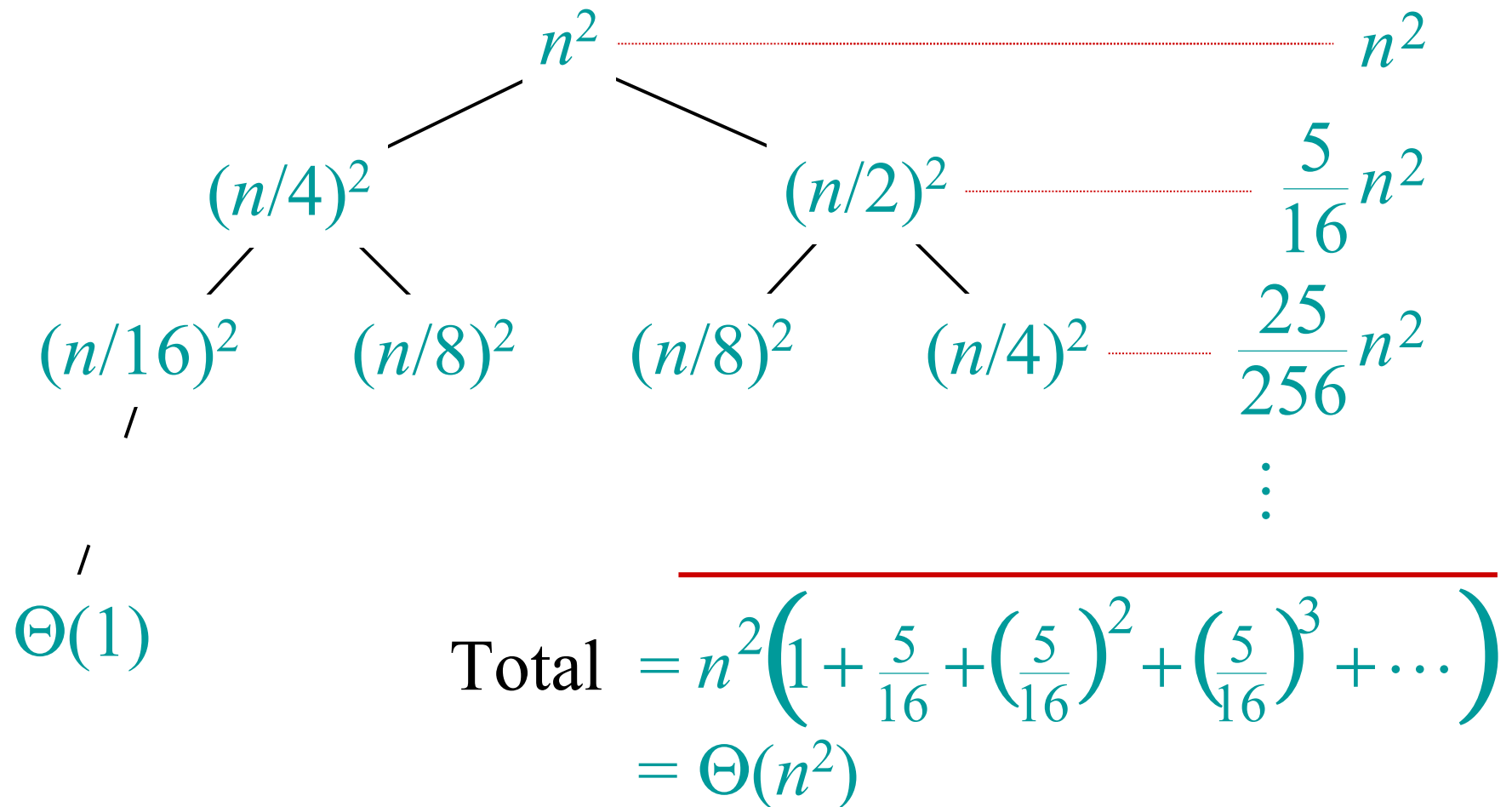
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

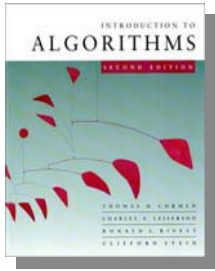




Example of recursion tree

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



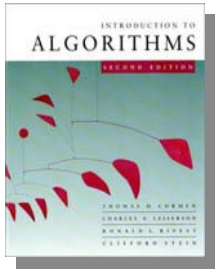


The master method

The master method applies to recurrences of the form

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

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



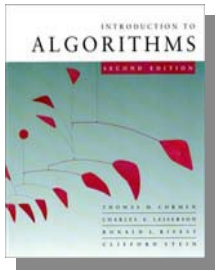
Three common cases

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^ϵ 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 $\epsilon > 0$.

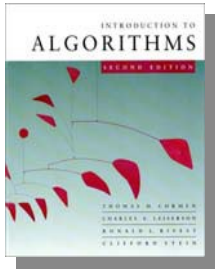
- $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^ϵ factor).

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.)

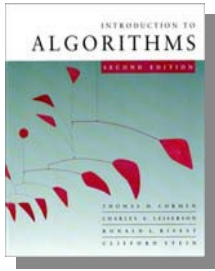
Compare $f(n)$ with $n^{\log_b a}$:

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^ϵ factor),

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

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



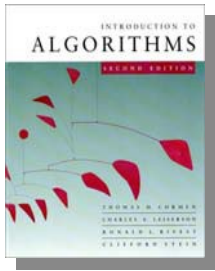
Examples

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 $\epsilon = 1$.

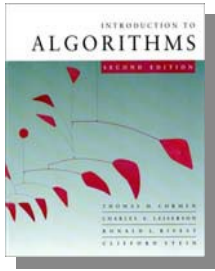
$$\therefore T(n) = \Theta(n^2).$$

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

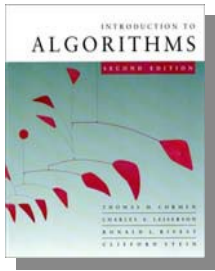
Ex. $T(n) = 4T(n/2) + n^3$

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

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

and $4(n/2)^3 \leq cn^3$ (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 $4(n/2)^3 \leq cn^3$ (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

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

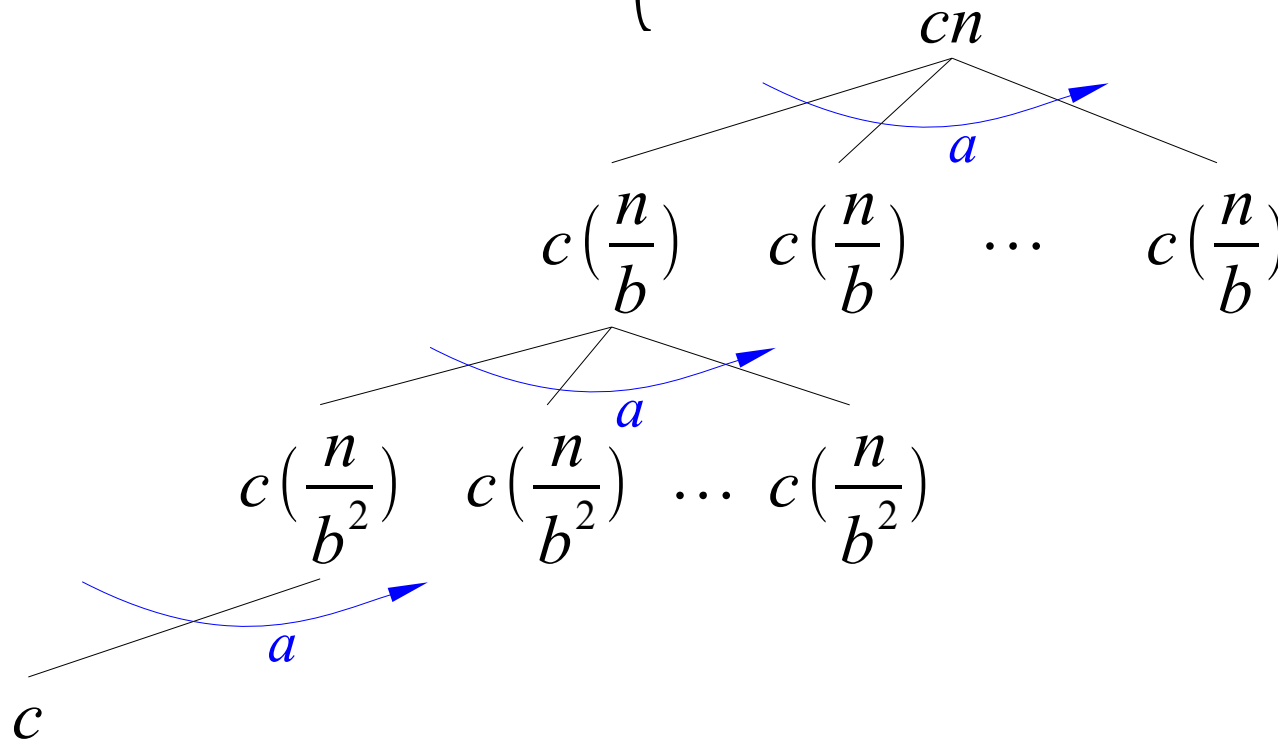
Intuition of Master Theorem

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

The diagram illustrates the recursion tree for the recurrence relation $T(n) = aT\left(\frac{n}{b}\right) + cn$. The root node is labeled cn . It branches into a children, each labeled $T\left(\frac{n}{b}\right)$. A blue arc with an arrow points from the root to the right child, indicating the work done at that level.

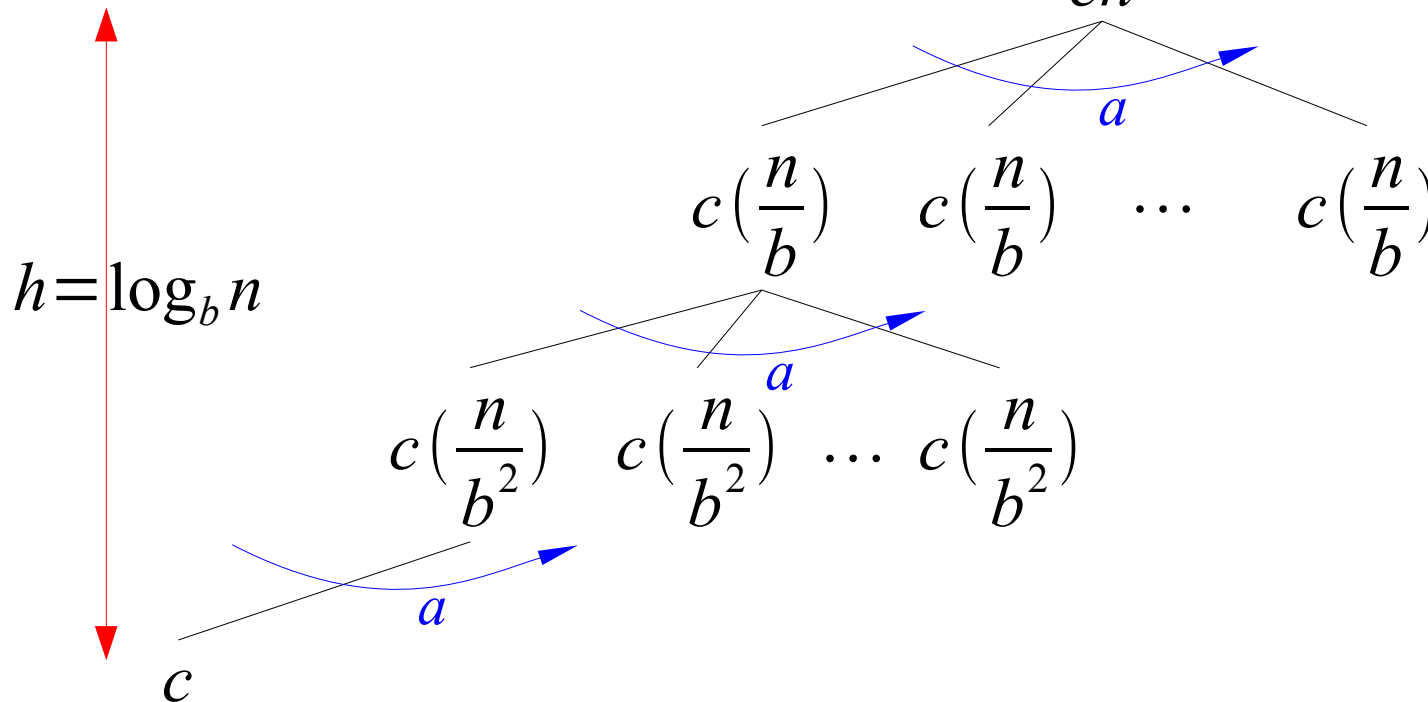
Intuition of Master Theorem

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



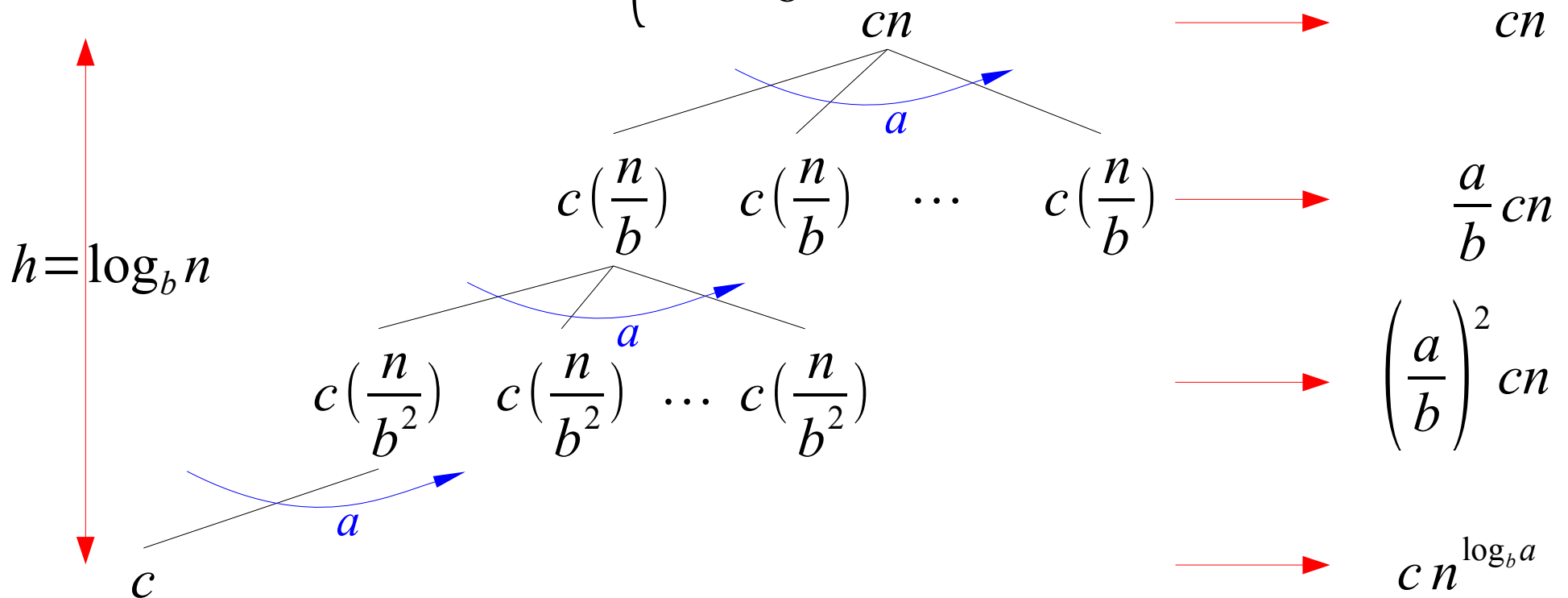
Intuition of Master Theorem

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



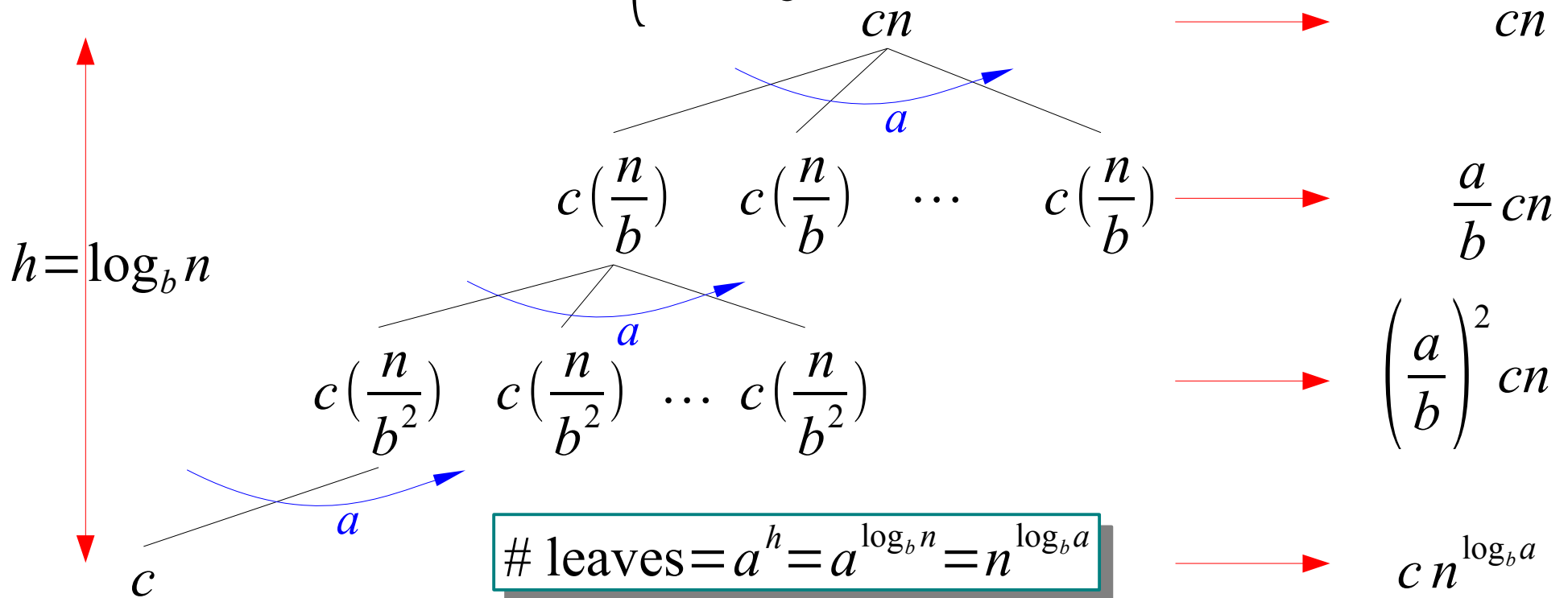
Intuition of Master Theorem

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



Intuition of Master Theorem

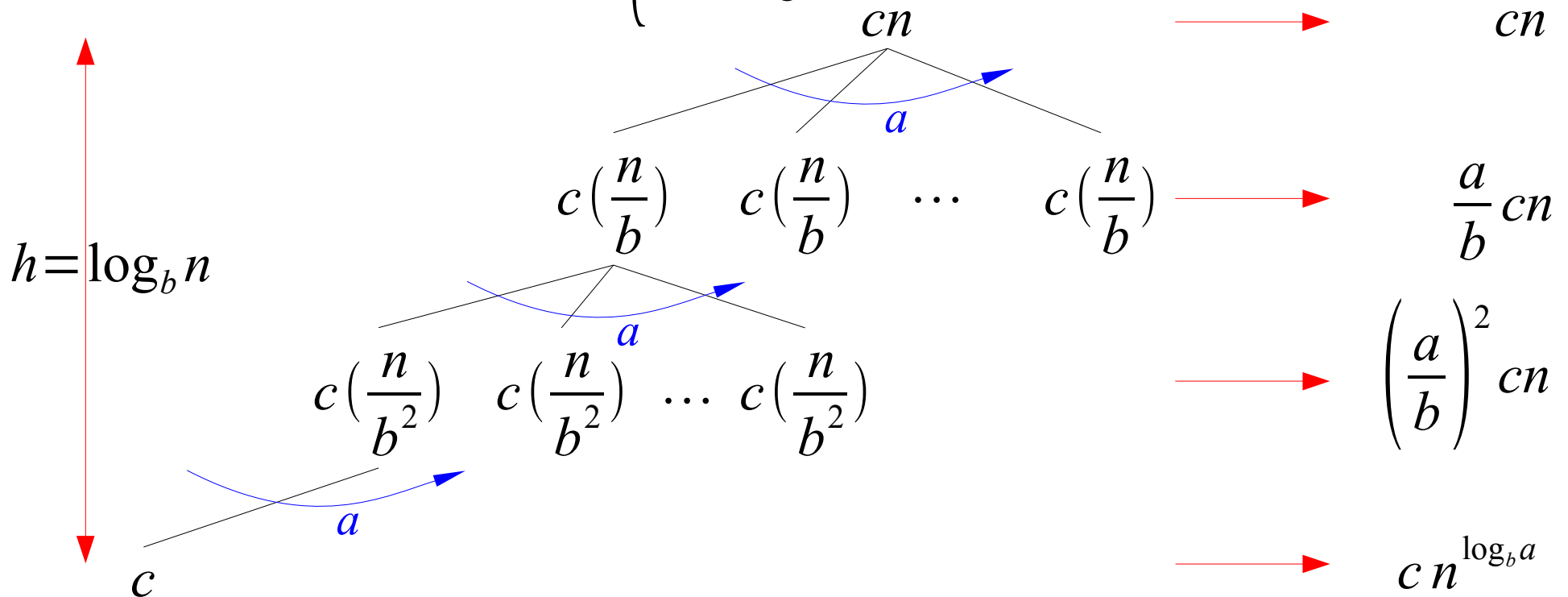
$$T(n) = \begin{cases} cn & n=1 \\ aT\left(\frac{n}{b}\right) + cn & n>1 \end{cases}$$



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

Intuition of Master Theorem

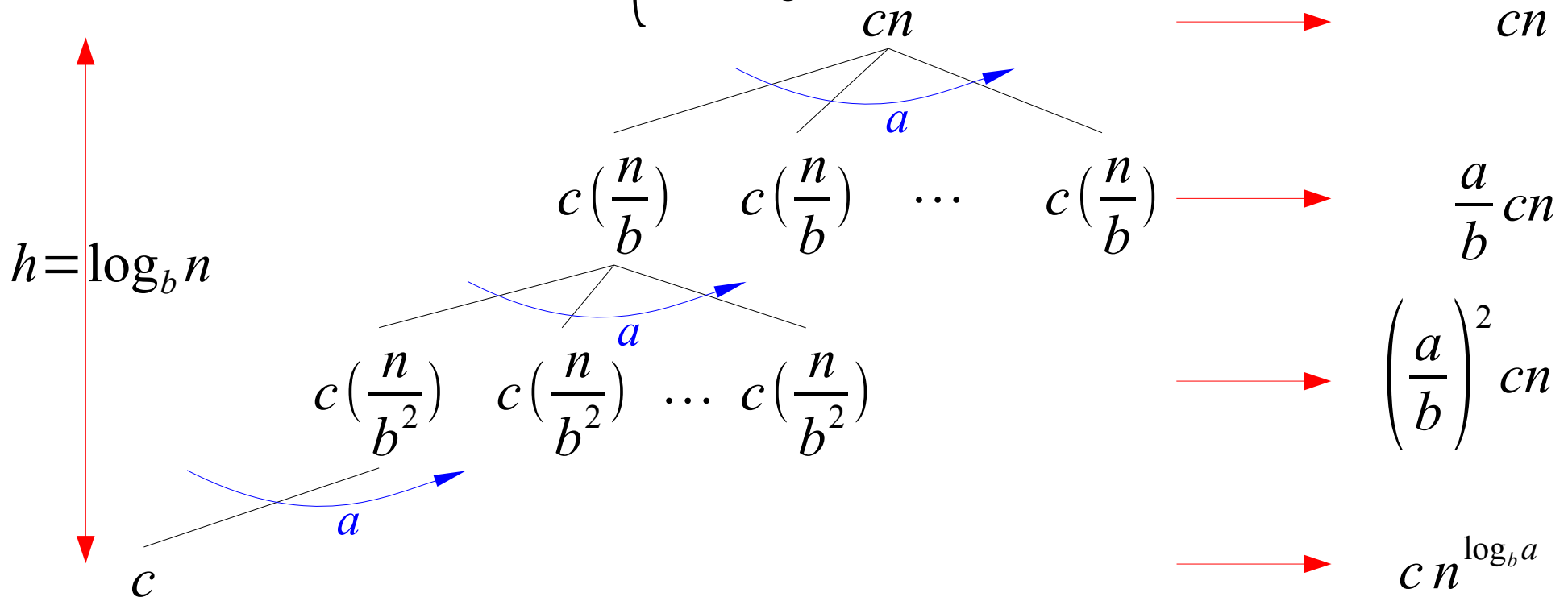
$$T(n) = \begin{cases} f(n) = cn & n=1 \\ aT\left(\frac{n}{b}\right) + cn & n>1 \end{cases}$$



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

Intuition of Master Theorem

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



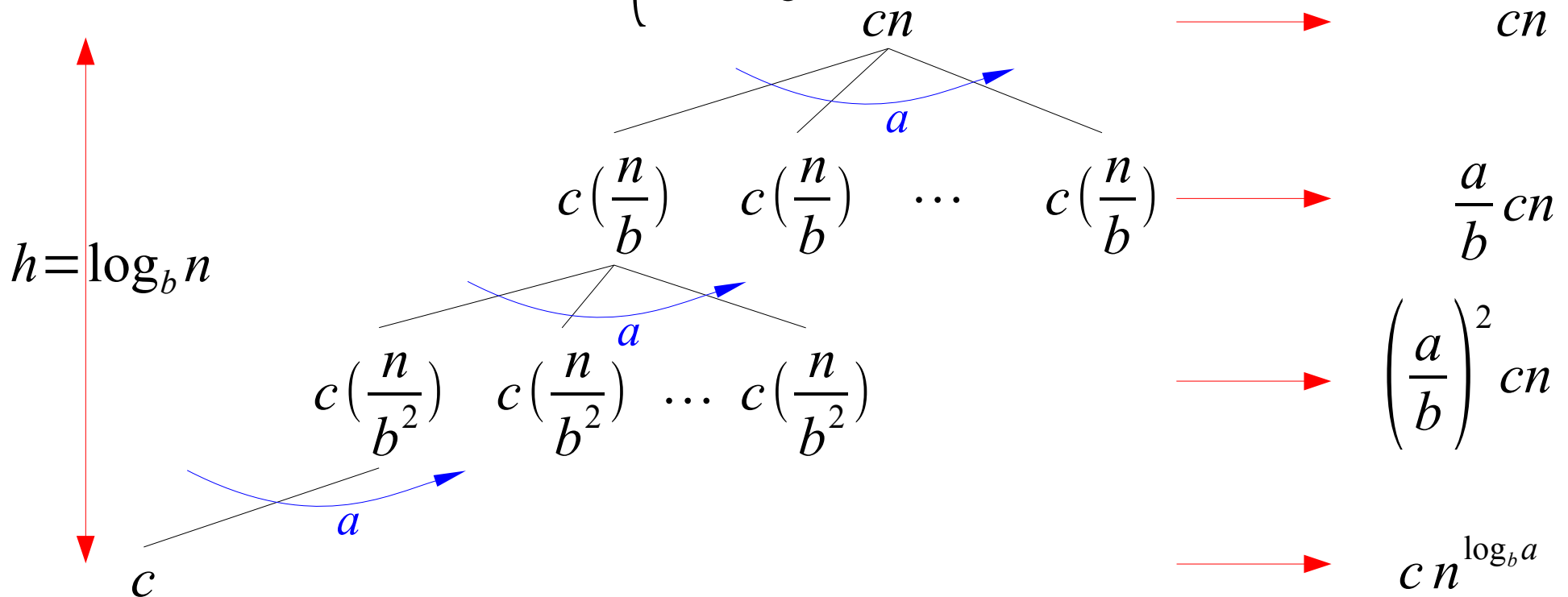
Case 1: $a > b$

Weights are increasing downwards

$$T(n) = cn \frac{1 - (a/b)^{\log_b n + 1}}{1 - a/b} = \Theta(n^{\log_b a})$$

Intuition of Master Theorem

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



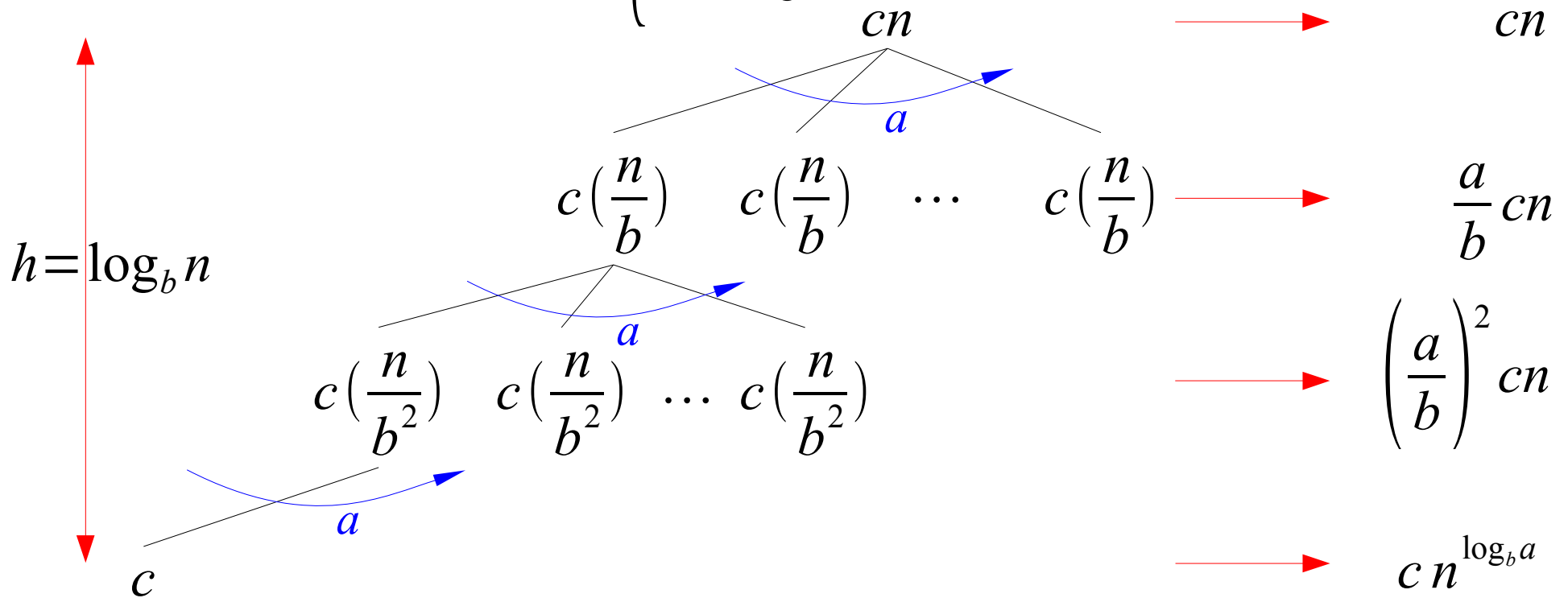
Case 2: $a = b$

Weights are equal across levels

$$T(n) = cn \frac{1 - \left(\frac{a}{b}\right)^{\log_b n + 1}}{1 - a/b} = \Theta(n \log_b n)$$

Intuition of Master Theorem

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



Case 3: $a < b$

Weights are decreasing downwards

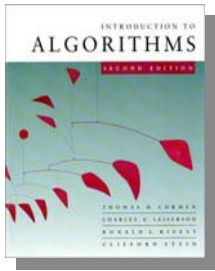
$$T(n) = cn \frac{1 - \left(\frac{a}{b}\right)^{\log_b n + 1}}{1 - a/b} = \Theta(n)$$

Intuition of Master Theorem

Summary

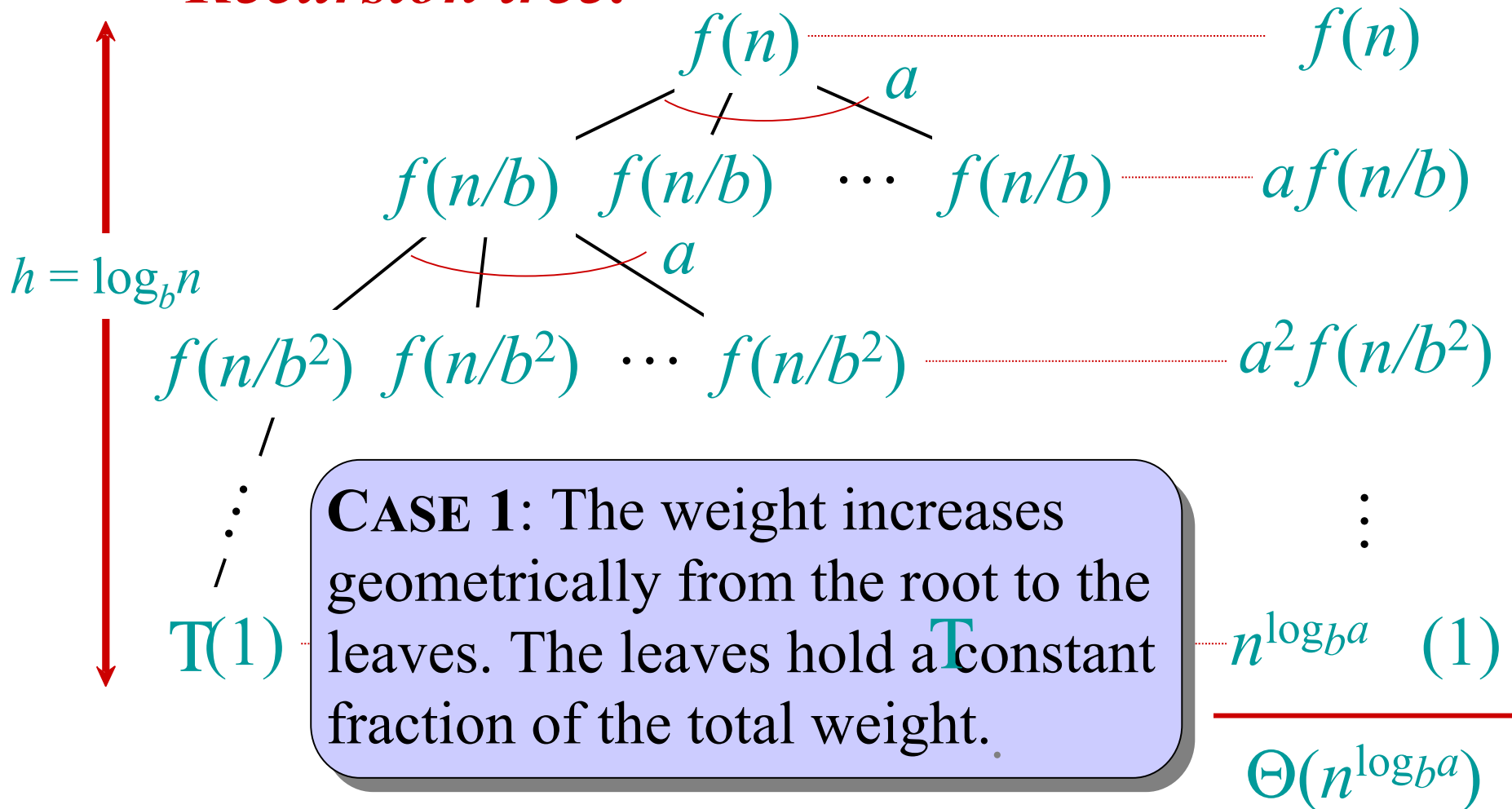
$$T(n) = \begin{cases} f(n) = cn & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

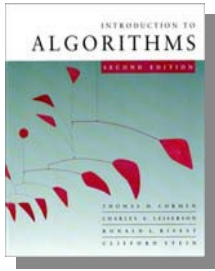
$$T(n) = \begin{cases} \Theta(n) & , a < b \\ \Theta(n \log_b n) & , a = b \\ \Theta(n^{\log_b a}) & , a > b \end{cases}$$



Idea of master theorem

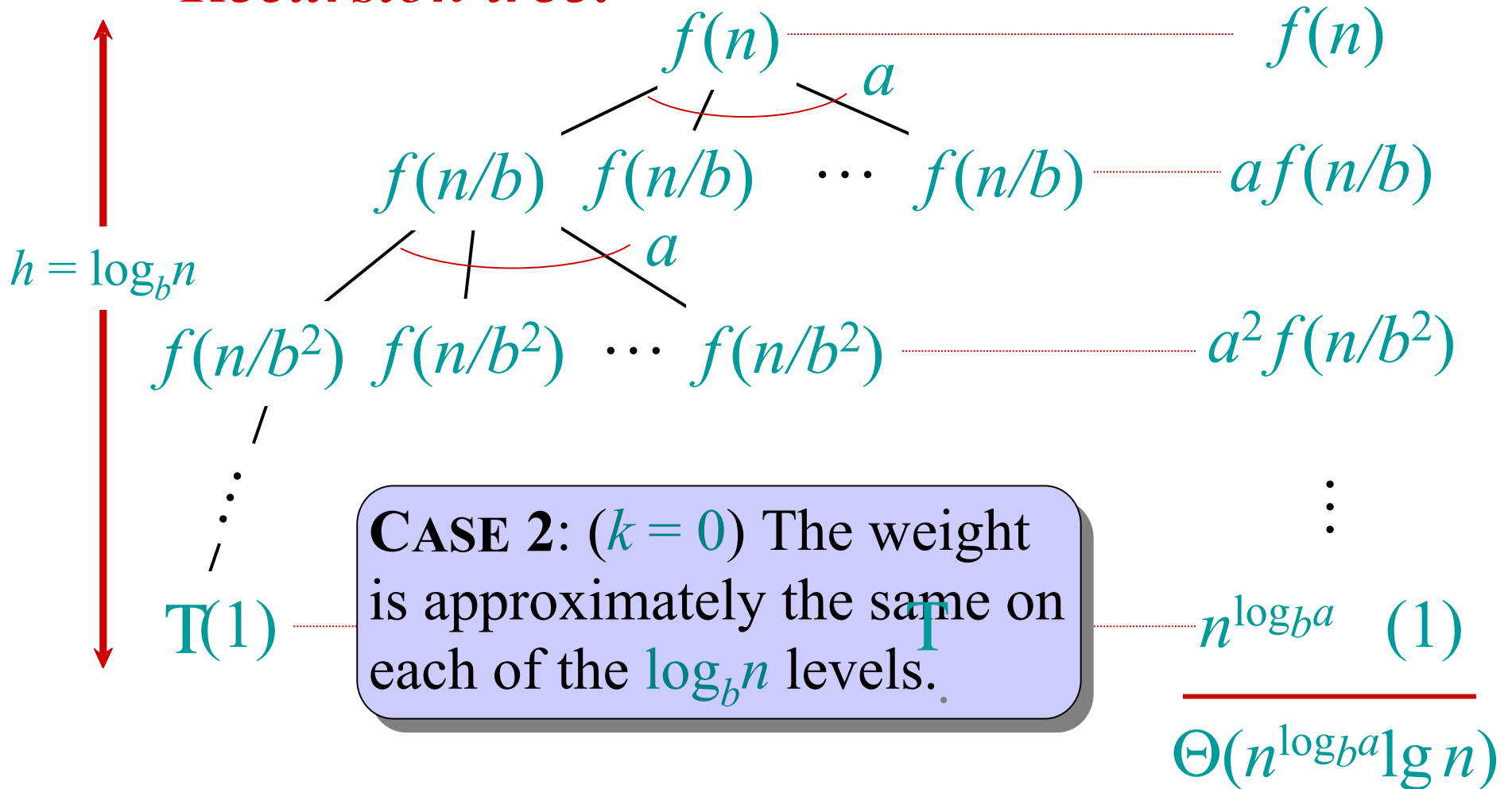
Recursion tree:

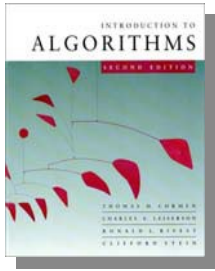




Idea of master theorem

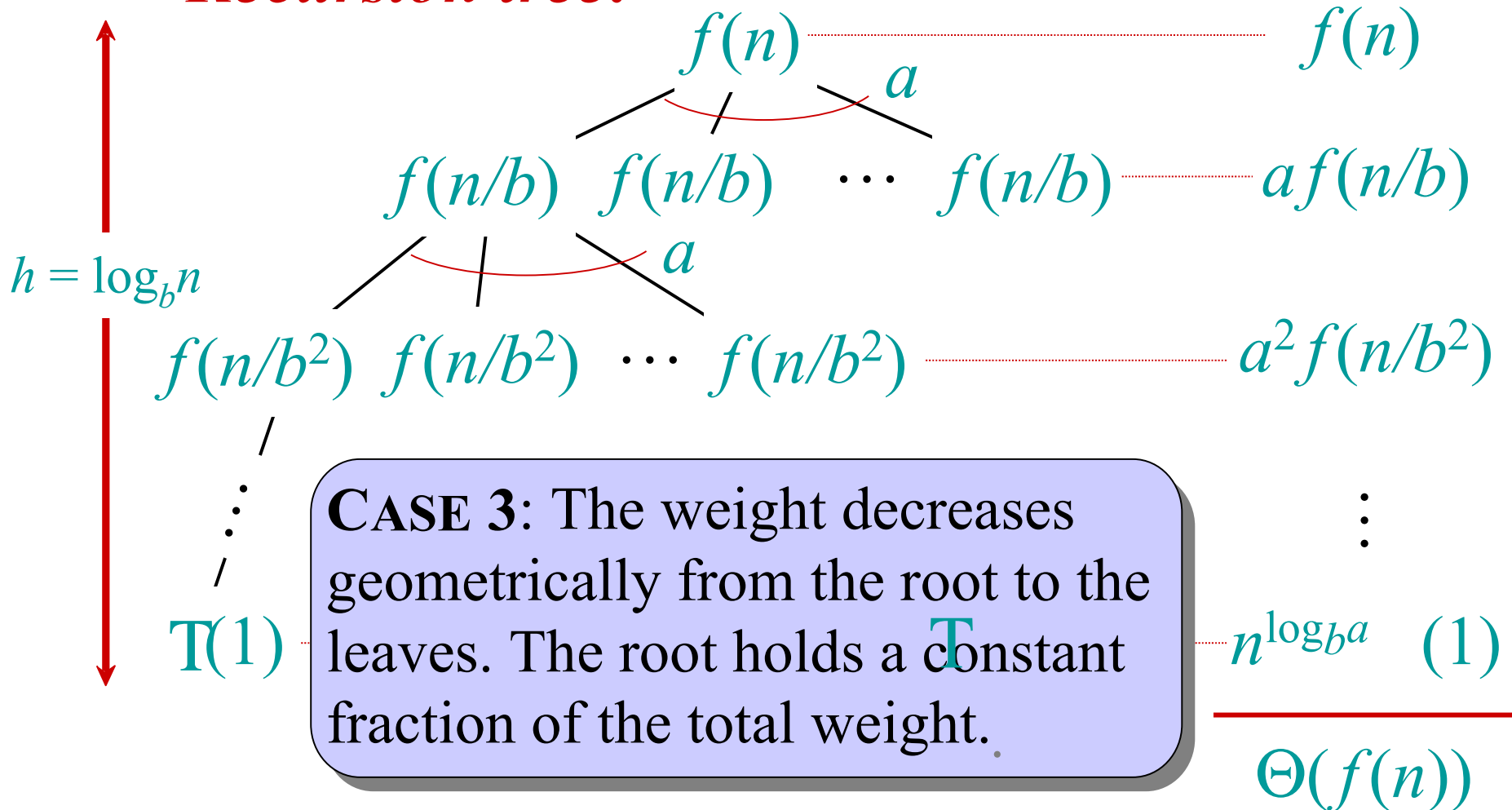
Recursion tree:

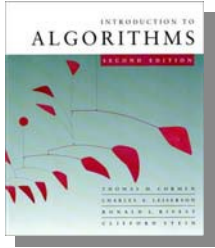




Idea of master theorem

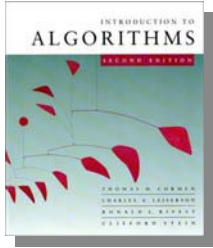
Recursion tree:





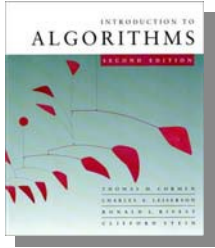
The divide-and-conquer design paradigm

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

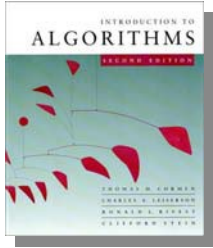
- 1. Divide:** Trivial.
- 2. Conquer:** Recursively sort 2 subarrays.
- 3. Combine:** Linear-time merge.

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

subproblems

subproblem size

work dividing and combining



Master theorem (reprise)

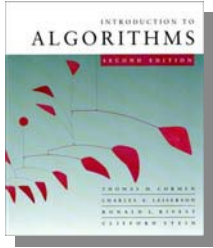
$$T(n) = a T(n/b) + f(n)$$

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

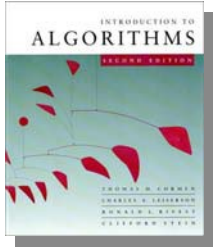
Merge sort: $a = 2, b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$
 \Rightarrow **CASE 2** ($k = 0$) $\Rightarrow T(n) = \Theta(n \lg n)$.



Binary search

Find an element in a sorted array:

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



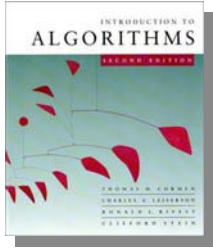
Binary search

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



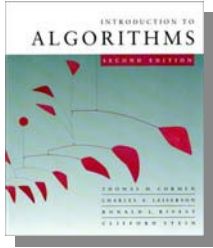
Binary search

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



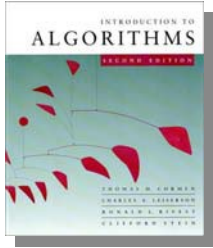
Binary search

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



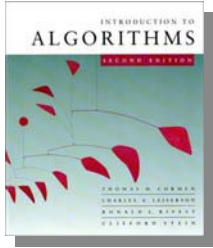
Binary search

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



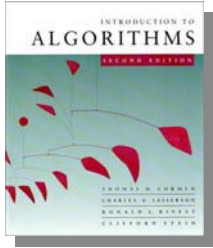
Binary search

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



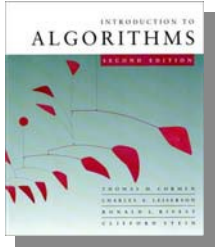
Binary search

Find an element in a sorted array:

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

Example: Find 9

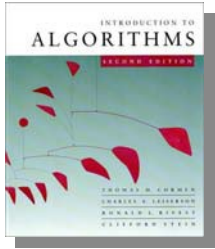
3 5 7 8 **9** 12 15



Recurrence for binary search

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

subproblems *subproblem size* *work dividing and combining*

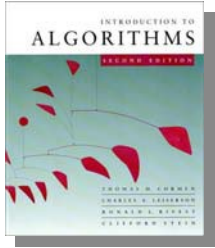


Recurrence for binary search

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

subproblems *subproblem size* *work dividing and combining*

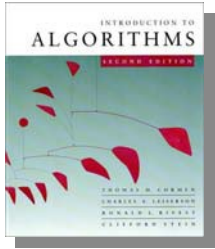
$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$
$$\Rightarrow T(n) = \Theta(\lg n) .$$



Powering a number

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

Naive algorithm: $\Theta(n)$.



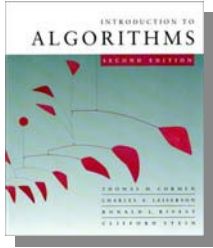
Powering a number

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

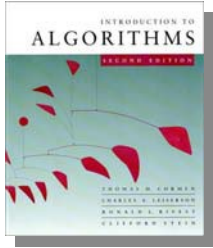
Problem: Compute a^n , where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

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

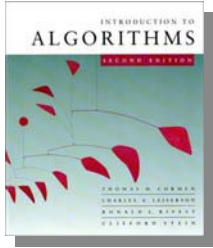


Matrix multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$ } $i, j = 1, 2, \dots, n.$
Output: $C = [c_{ij}] = A B.$

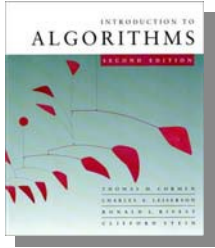
$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$



Standard algorithm

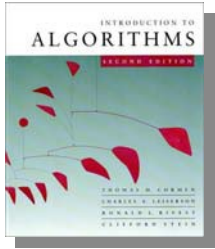
```
for  $i \leftarrow 1$  to  $n$ 
  do for  $j \leftarrow 1$  to  $n$ 
    do  $c_{ij} \leftarrow 0$ 
      for  $k \leftarrow 1$  to  $n$ 
        do  $c_{ij} \leftarrow c_{ij} + a_{ik} b_{kj}$ 
```



Standard algorithm

```
for  $i \leftarrow 1$  to  $n$ 
  do for  $j \leftarrow 1$  to  $n$ 
    do  $c_{ij} \leftarrow 0$ 
      for  $k \leftarrow 1$  to  $n$ 
        do  $c_{ij} \leftarrow c_{ij} + a_{ik} b_{kj}$ 
```

Running time = $\Theta(n^3)$



Divide-and-conquer algorithm

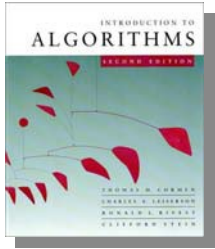
IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & | & s \\ \hline t & | & u \end{bmatrix} = \begin{bmatrix} a & | & b \\ \hline c & | & d \end{bmatrix} \cdot \begin{bmatrix} e & | & f \\ \hline g & | & h \end{bmatrix}$$
$$C = A \cdot B$$

$$\left. \begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned} \right\}$$

8 mults of $(n/2) \times (n/2)$ submatrices
4 adds of $(n/2) \times (n/2)$ submatrices



Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

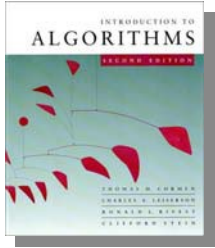
$$\begin{bmatrix} r & | & s \\ \hline t & | & u \end{bmatrix} = \begin{bmatrix} a & | & b \\ \hline c & | & d \end{bmatrix} \cdot \begin{bmatrix} e & | & f \\ \hline g & | & h \end{bmatrix}$$
$$C = A \cdot B$$

$$\left. \begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned} \right\}$$

recursive

8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices



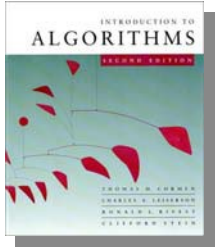
Analysis of D&C algorithm

$$T(n) = 8T(n/2) + \Theta(n^2)$$

submatrices

submatrix size

*work adding
submatrices*

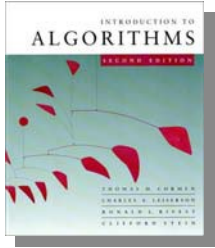


Analysis of D&C algorithm

$$T(n) = 8T(n/2) + \Theta(n^2)$$

submatrices *submatrix size* *work adding submatrices*

$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^3).$$



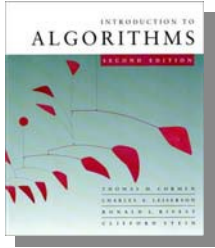
Analysis of D&C algorithm

$$T(n) = 8T(n/2) + \Theta(n^2)$$

submatrices submatrix size work adding submatrices

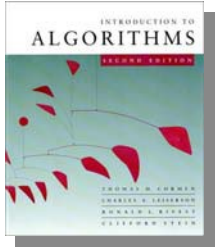
$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

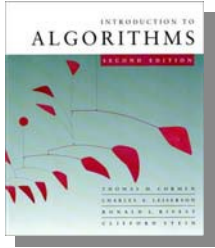
$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

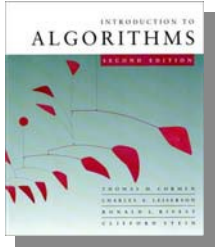
$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$s = P_1 + P_2$$

$$t = P_3 + P_4$$

$$u = P_5 + P_1 - P_3 - P_7$$



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

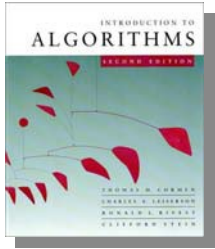
$$s = P_1 + P_2$$

$$t = P_3 + P_4$$

$$u = P_5 + P_1 - P_3 - P_7$$

7 mults, 18 adds/subs.

Note: No reliance on commutativity of mult!



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$= (a + d)(e + h)$$

$$+ d(g - e) - (a + b)h$$

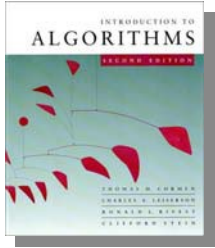
$$+ (b - d)(g + h)$$

$$= ae + ah + de + dh$$

$$+ dg - de - ah - bh$$

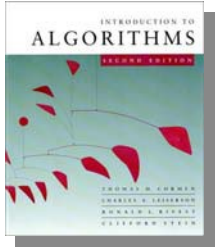
$$+ bg + bh - dg - dh$$

$$= ae + bg$$



Strassen's algorithm

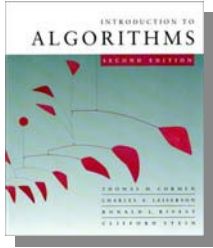
- 1. *Divide:*** Partition A and B into $(n/2) \times (n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
- 2. *Conquer:*** Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.
- 3. *Combine:*** Form C using $+$ and $-$ on $(n/2) \times (n/2)$ submatrices.



Strassen's algorithm

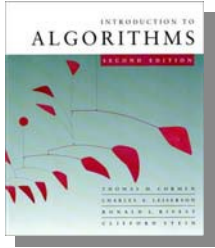
- 1. Divide:** Partition A and B into $(n/2) \times (n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
- 2. Conquer:** Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.
- 3. Combine:** Form C using $+$ and $-$ on $(n/2) \times (n/2)$ submatrices.

$$T(n) = 7 T(n/2) + \Theta(n^2)$$



Analysis of Strassen

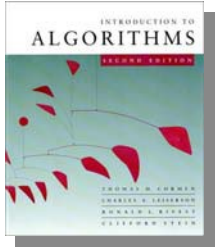
$$T(n) = 7 T(n/2) + \Theta(n^2)$$



Analysis of Strassen

$$T(n) = 7 T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^{\lg 7}).$$

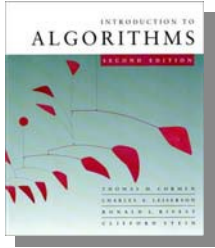


Analysis of Strassen

$$T(n) = 7 T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^{\lg 7}).$$

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.



Analysis of Strassen

$$T(n) = 7 T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^{\lg 7}).$$

The number **2.81** may not seem much smaller than **3**, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

Best to date (of theoretical interest only): $\Theta(n^{2.376\dots})$.

Recap

- Recurrences
 - Substitution Method
 - Recursion Tree
 - Master Theorem
- Divide-and-Conquer Examples
- Next:
 - Heapsort and Quicksort