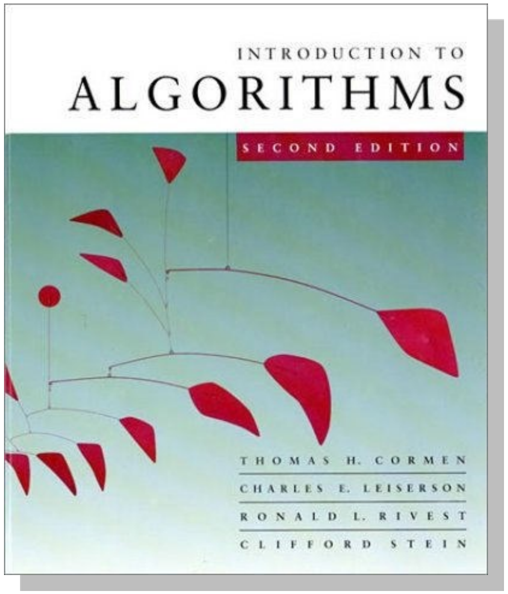
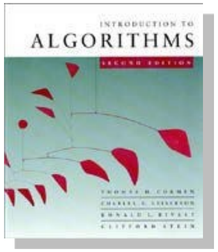
Introduction to Algorithms

LECTURE 2



Asymptotic Notation
- O-, Ω-, and Θ-notation

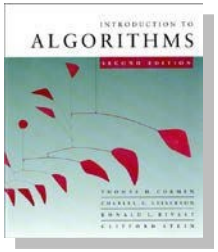
Recurrences
- Substitution method
- Iterating the recurrence
- Recursion tree
- Master method



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 $n \ge n_0$.

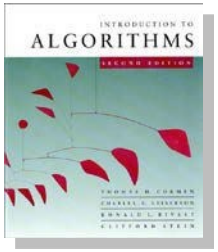


Asymptotic notation

O-notation (upper bounds):

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

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



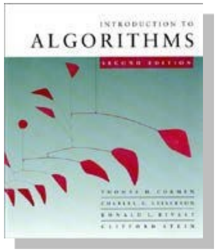
Asymptotic notation

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

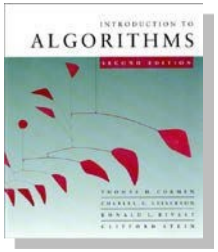
O-notation (upper bounds):

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

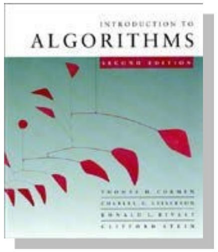
functions, not values

funny, "one-way" equality



Set definition of O-notation

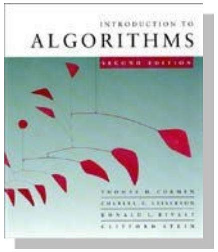
$O(g(n)) = \{ f(n) :$ there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$ $\}$



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

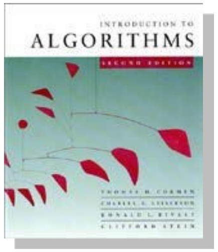


Set definition of O-notation

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

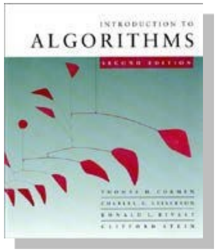
EXAMPLE: $2n^2 \in O(n^3)$

(*Logicians:* $\lambda n.2n^2 \in O(\lambda n.n^3)$, but it's convenient to be sloppy, as long as we understand what's *really* going on.)



Macro substitution

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



Macro substitution

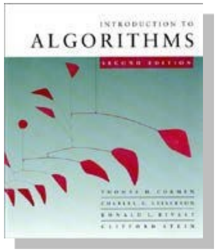
Convention: A set in a formula represents an anonymous function in the set.

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.

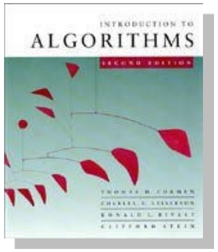
EXAMPLE: $n^2 + O(n) = O(n^2)$

means

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

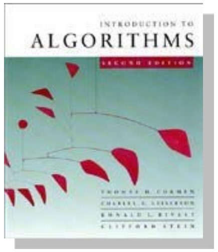
$$n^2 + f(n) = h(n)$$

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



Ω-notation (lower bounds)

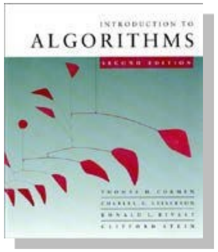
O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.



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 $0 \le cg(n) \le f(n)$ for all $n \ge n_0$ $\}$

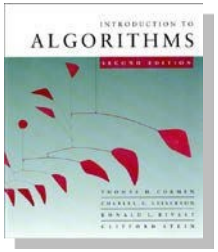


Ω-notation (lower bounds)

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

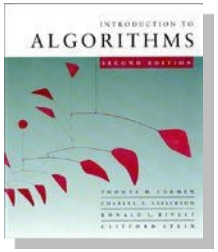
$\Omega(g(n)) = \{ f(n)$: there exist constants $c > 0,\ n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$ $\}$

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



Θ-notation (tight bounds)

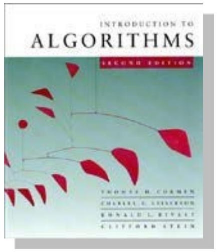
$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$



Θ-notation (tight bounds)

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

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

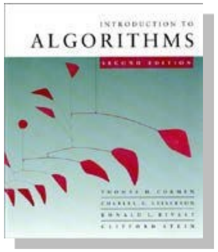


o-notation and ω-notation

O-notation and Ω-notation are like $\leq$ and $\geq$.
o-notation and ω-notation are like $<$ and $>$.

$o(g(n)) = \{ f(n)$: for any constant $c > 0$,
there is a constant $n_0 > 0$
such that $0 \leq f(n) < cg(n)$
for all $n \geq n_0$ $\}$

Example: $2n^2 = o(n^3)$ $\quad (n_0 = 2/c)$

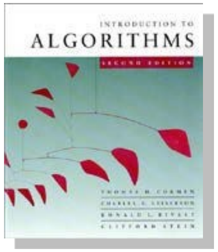


o-notation and ω-notation

O-notation and Ω-notation are like $\leq$ and $\geq$.
o-notation and ω-notation are like $<$ and $>$.

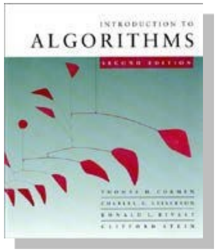
$\omega(g(n)) = \{ f(n)$: for any constant $c > 0$,
there is a constant $n_0 > 0$
such that $0 \leq cg(n) < f(n)$
for all $n \geq n_0$ $\}$

Example: $\sqrt{n} = \omega(\lg n)$ $\quad (n_0 = 1 + 1/c)$



Solving recurrences

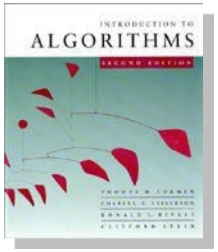
- The analysis of merge sort from ***Lecture 1*** required us to solve a recurrence.
- Recurrences are like solving integrals, differential equations, etc.
  - Learn a few tricks.
- ***Lecture 3***: Applications of recurrences to divide-and-conquer algorithms.



Substitution method

The most general method:

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



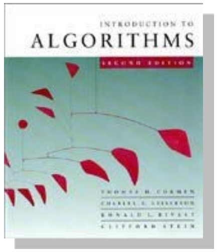
Substitution method

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.

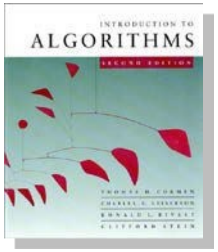


Example of substitution

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

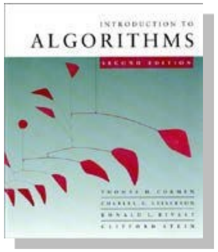
whenever $(c/2)n^3 - n \ge 0$, for example, if $c \ge 2$ and $n \ge 1$.

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.



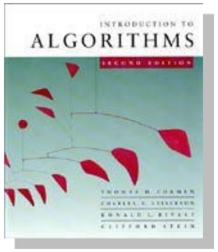
Example (continued)

- 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 $T(n) = O(n^2)$.

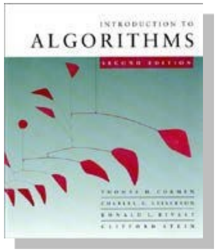


A tighter upper bound?

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

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

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



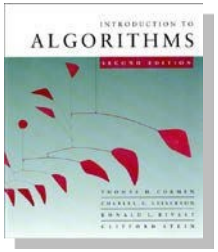
A tighter upper bound?

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

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

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

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 $k < n$:

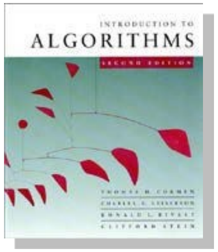
$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^2 + n \\
&= cn^2 + n \\
&\cancel{= O(n^2)}
\end{aligned}
$$

Wrong! We must prove the I.H.

$$
\begin{aligned}
&= cn^2 - (-n) \\
&\le cn^2
\end{aligned}
$$

[desired – residual]

for ***no*** choice of $c > 0$. Lose!



A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

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

Inductive hypothesis: $T(k) \leq c_1k^2 - c_2k$ for $k < n$.

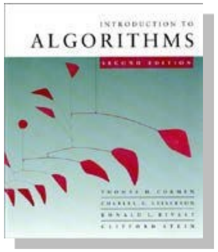
A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

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

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

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



A tighter upper bound!

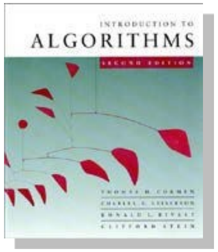
IDEA: Strengthen the inductive hypothesis.

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

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

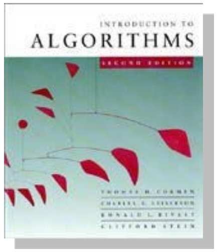
$$\begin{aligned}
T(n) &= 4T(n/2) + n \\
&= 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\le c_1 n^2 - c_2 n \quad \text{if } c_2 \ge 1.
\end{aligned}$$

Pick c_1 big enough to handle the initial conditions.



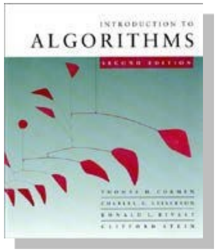
Recursion-tree method

- 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

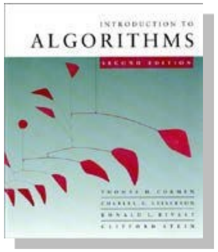
Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

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

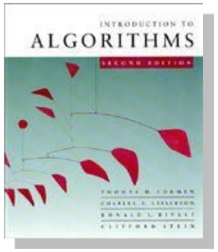
$$T(n)$$



Example of recursion tree

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

$$
\begin{array}{ccc}
 & n^2 & \\
\swarrow & & \searrow \\
T(n/4) & & T(n/2)
\end{array}
$$



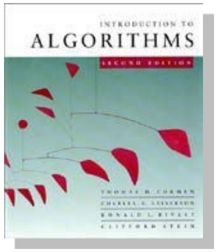
Example of recursion tree

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

$$n^2$$

$$(n/4)^2 \qquad\qquad (n/2)^2$$

$$T(n/16) \qquad T(n/8) \qquad T(n/8) \qquad T(n/4)$$



Example of recursion tree

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

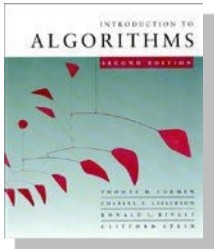
$$n^2$$

$$(n/4)^2 \qquad\qquad (n/2)^2$$

$$(n/16)^2 \qquad (n/8)^2 \qquad (n/8)^2 \qquad (n/4)^2$$

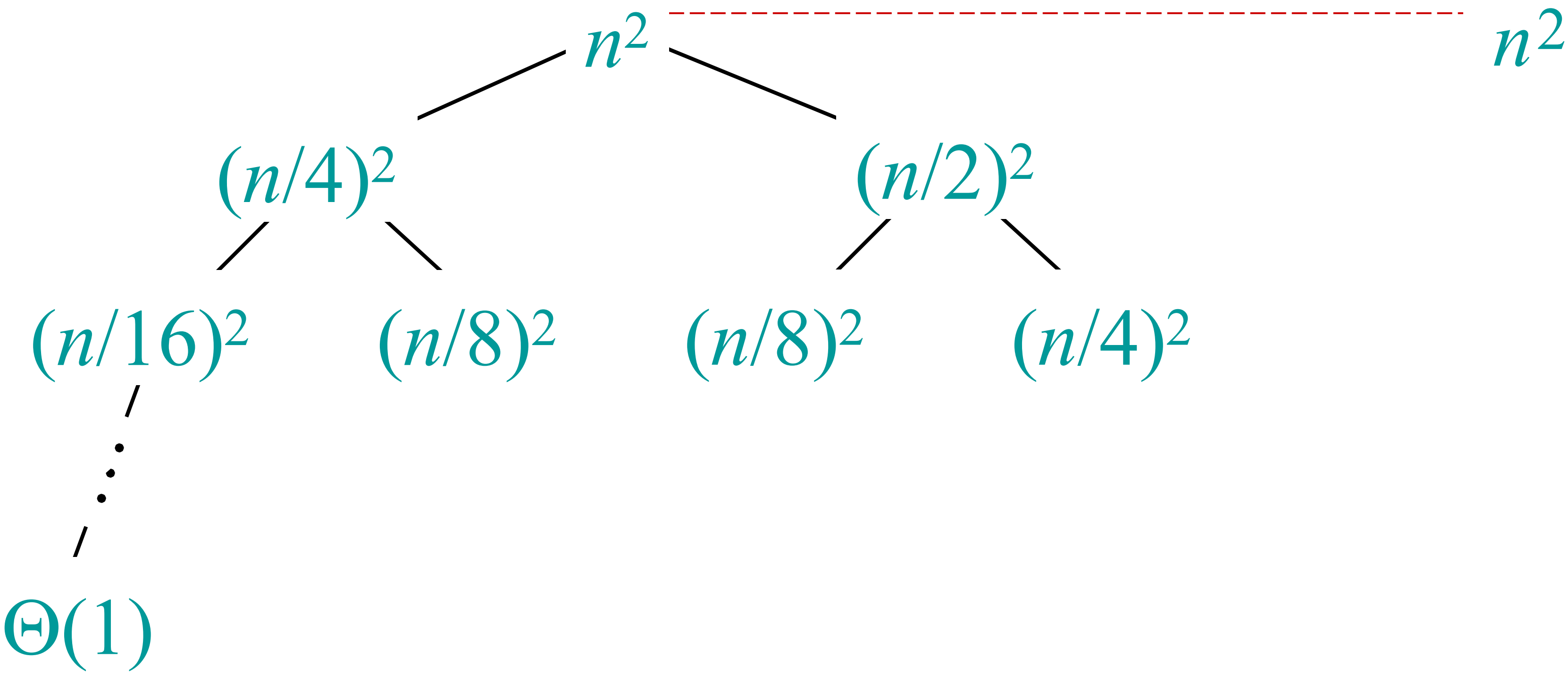
$$\vdots$$

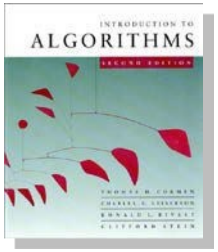
$$\Theta(1)$$



Example of recursion tree

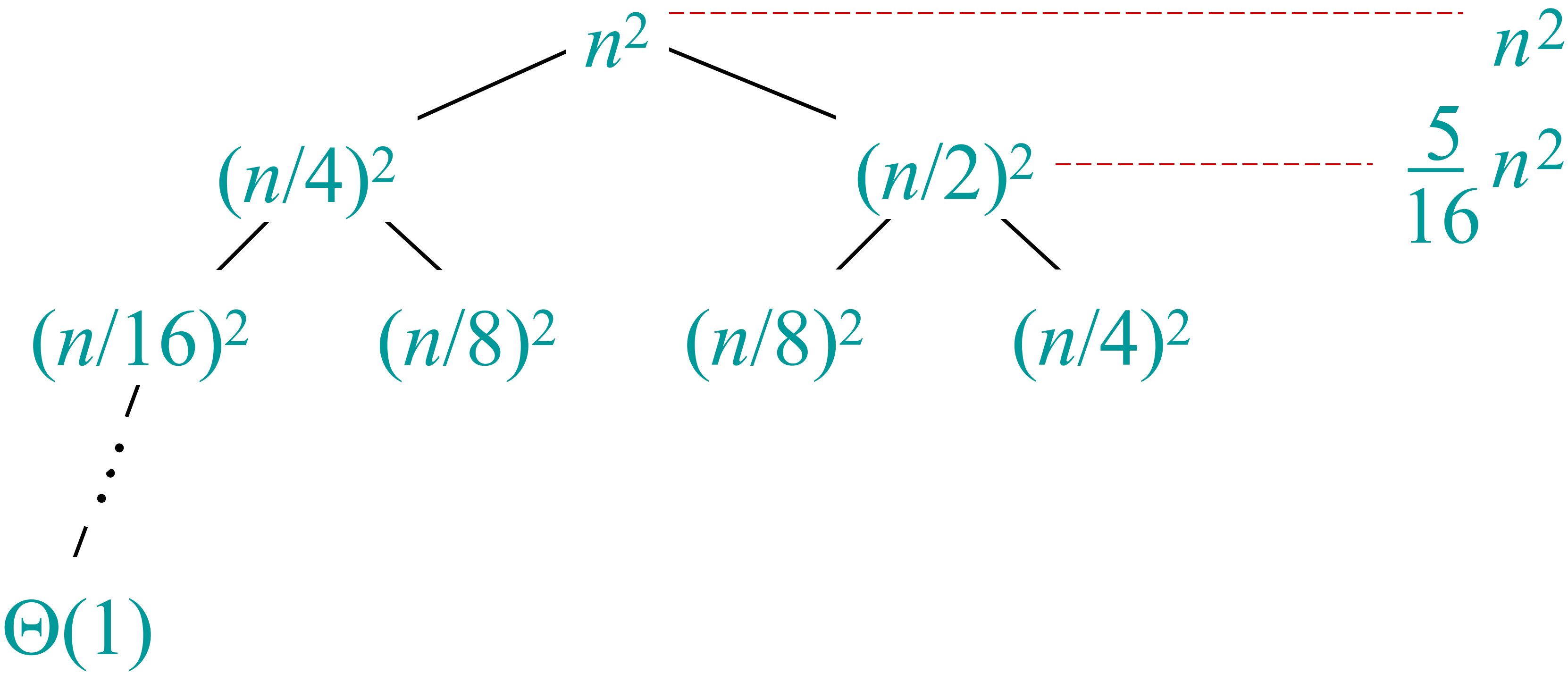
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

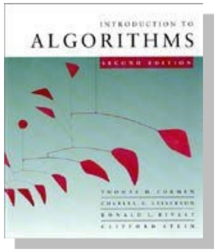




Example of recursion tree

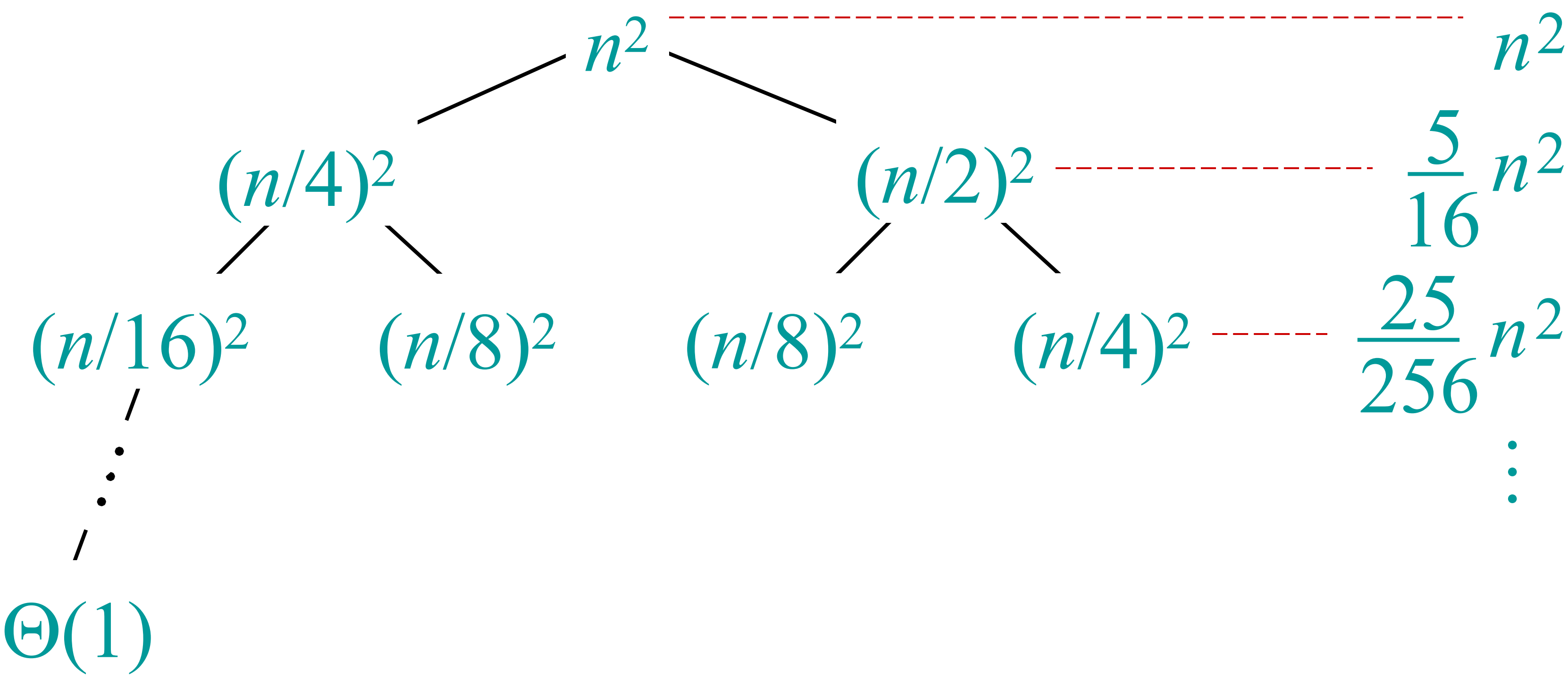
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

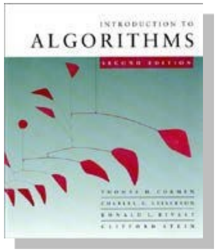




Example of recursion tree

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





Example of recursion tree

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

$$n^2 \quad \cdots\cdots \quad n^2$$

$$(n/4)^2 \qquad (n/2)^2 \quad \cdots\cdots \quad \frac{5}{16}n^2$$

$$(n/16)^2 \qquad (n/8)^2 \qquad (n/8)^2 \qquad (n/4)^2 \quad \cdots\cdots \quad \frac{25}{256}n^2$$

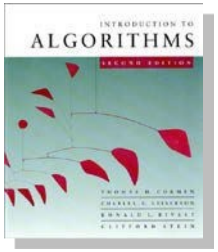
$$\vdots \qquad\qquad \vdots$$

$$\Theta(1)$$

$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$

$$= \Theta(n^2)$$

geometric series

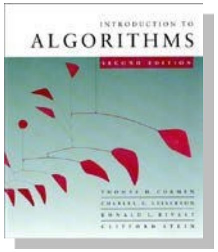


The master method

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.



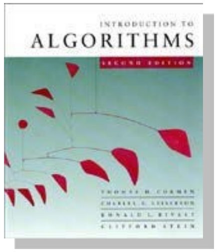
Three common cases

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

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

- $f(n)$ 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. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

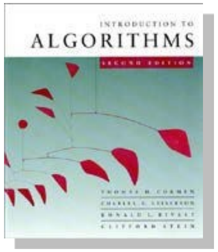
- $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

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



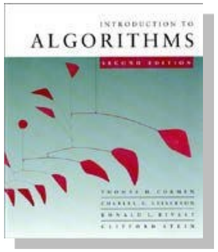
Three common cases (cont.)

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

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

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

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



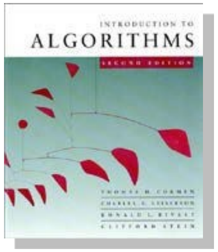
Examples

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

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

Case 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

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



Examples

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

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

Case 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

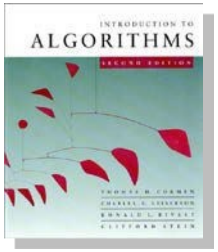
$\therefore\ T(n) = \Theta(n^2).$

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

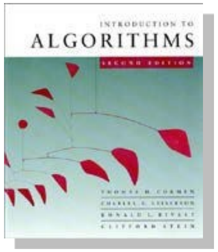
Ex. $T(n) = 4T(n/2) + n^3$

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

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

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

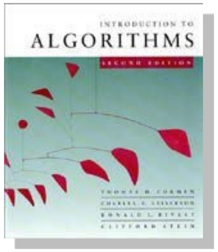
and $4(n/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

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

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

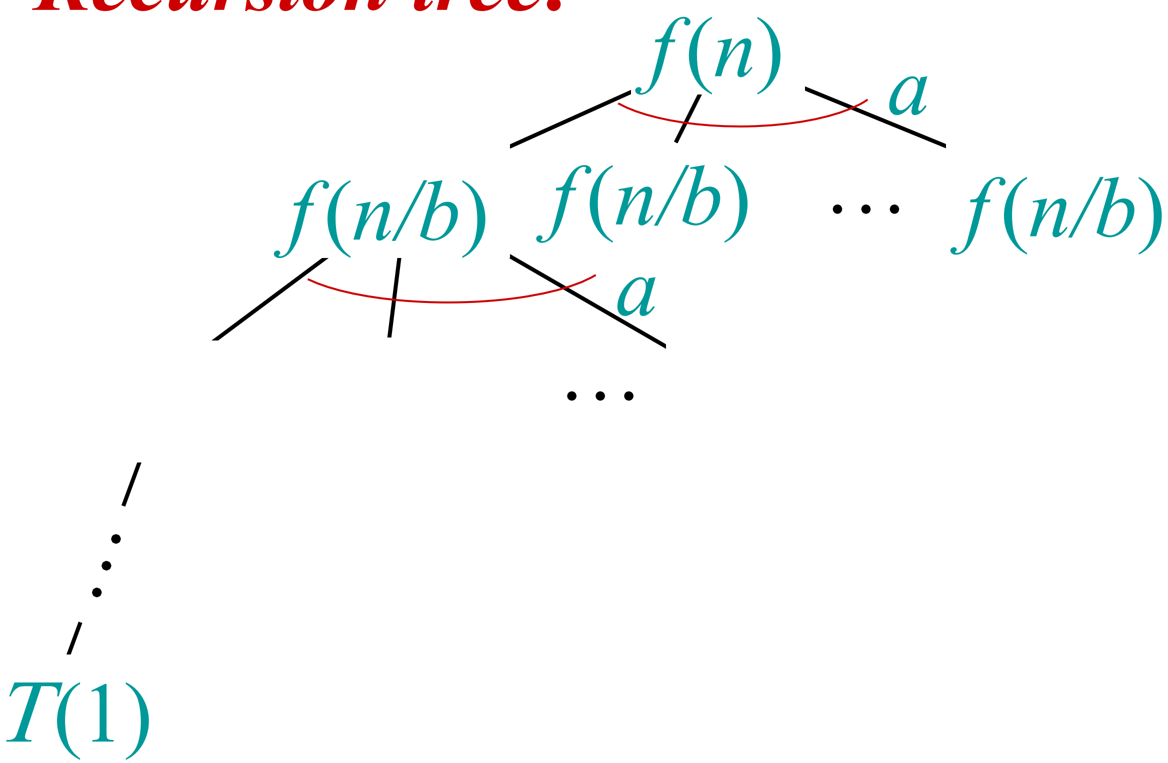
$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2/\lg n.$

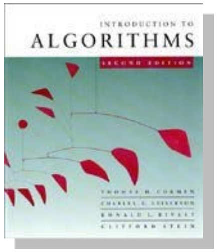
Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^\varepsilon = \omega(\lg n)$.



Idea of master theorem

Recursion tree:





Idea of master theorem

Recursion tree:

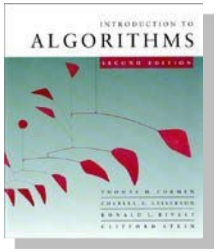
$$f(n) \quad\cdots\cdots\quad f(n)$$

$$\underbrace{f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b)}_{a} \quad\cdots\cdots\quad af(n/b)$$

$$\underbrace{f(n/b^2) \quad f(n/b^2) \quad \cdots \quad f(n/b^2)}_{a} \quad\cdots\cdots\quad a^2 f(n/b^2)$$

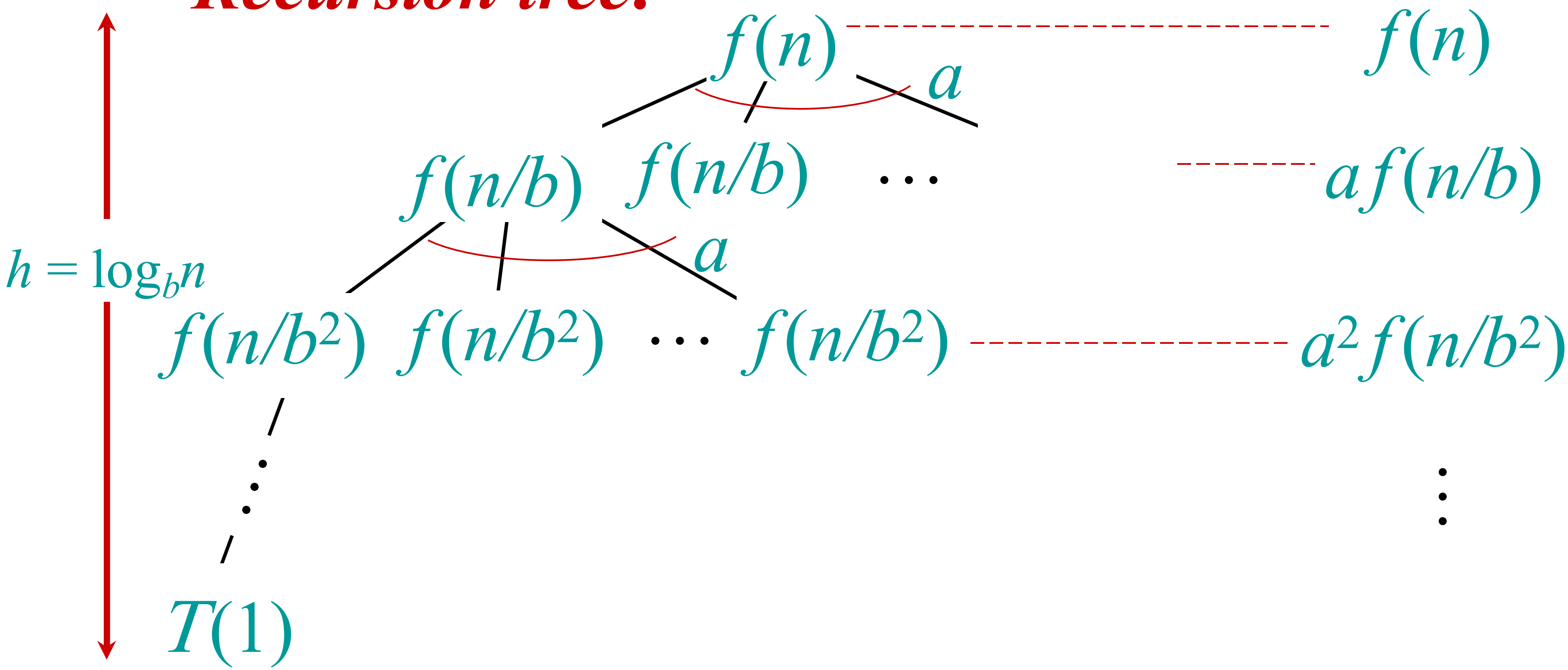
$$\vdots \qquad\qquad\qquad\qquad \vdots$$

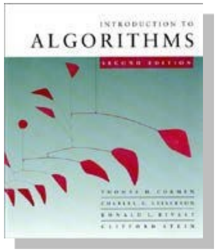
$$T(1)$$



Idea of master theorem

Recursion tree:





Idea of master theorem

Recursion tree:

$f(n)$ — a — $f(n)$

$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$ — a — $af(n/b)$

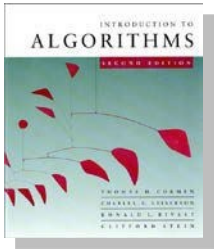
$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$ — $a^2f(n/b^2)$

$h = \log_b n$

$\vdots$

$T(1)$ — $n^{\log_b a}\,T(1)$

$$\begin{aligned}\#\text{leaves} &= a^h \\ &= a^{\log_b n} \\ &= n^{\log_b a}\end{aligned}$$



Idea of master theorem

Recursion tree:

$$f(n) \quad\cdots\cdots\quad f(n)$$

a

$$f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b) \quad\cdots\cdots\quad a\,f(n/b)$$

a

$$h = \log_b n$$

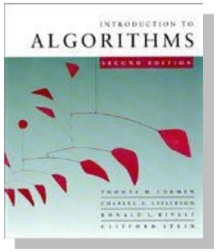
$$f(n/b^2) \quad f(n/b^2) \quad \cdots \quad f(n/b^2) \quad\cdots\cdots\quad a^2 f(n/b^2)$$

$$\vdots$$

$$T(1) \quad\cdots\cdots\quad n^{\log_b a}\,T(1)$$

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

$$\Theta(n^{\log_b a})$$



Idea of master theorem

Recursion tree:

$h = \log_b n$

$f(n)$ ------ $f(n)$

a

$f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b)$ ------ $a f(n/b)$

a

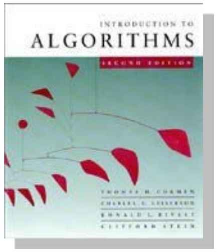
$f(n/b^2) \quad f(n/b^2) \quad \cdots \quad f(n/b^2)$ ------ $a^2 f(n/b^2)$

$\vdots$

$T(1)$ ------ $n^{\log_b a} T(1)$

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

$\Theta(n^{\log_b a} \lg n)$



Idea of master theorem

Recursion tree:

$f(n)$ ---------- $f(n)$

a

$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$ ---------- $a f(n/b)$

a

$h = \log_b n$

$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$ ---------- $a^2 f(n/b^2)$

$\vdots$

$T(1)$ ---------- $n^{\log_b a} T(1)$

$\Theta(f(n))$

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