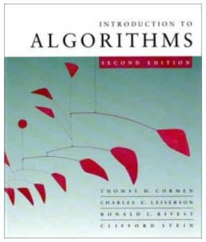
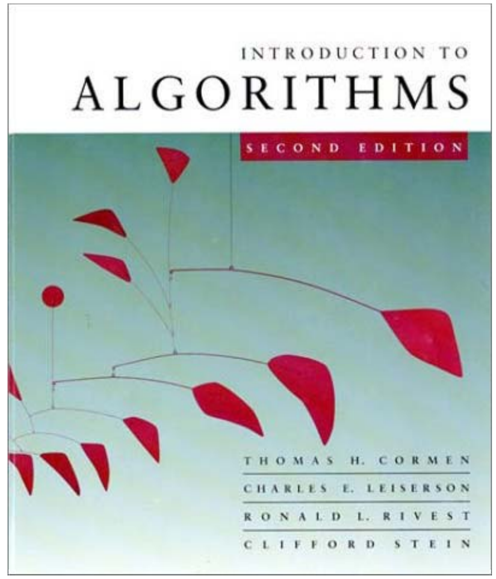


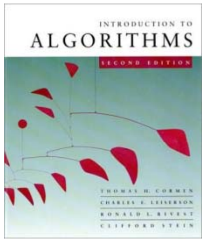
CS 3343 – Fall 2011



Master Theorem

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk



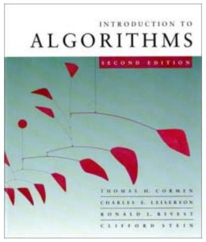
The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.

   a subproblems, **each** of size n/b

2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.

   Runtime for divide and combine is $f(n)$



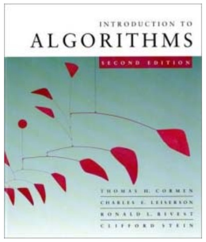
Example: merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort a=2 subarrays of size $n/2=n/b$
3. ***Combine:*** Linear-time merge, runtime $f(n) \in O(n)$

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

 # subproblems　　*subproblem size*　　*work dividing and combining*

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

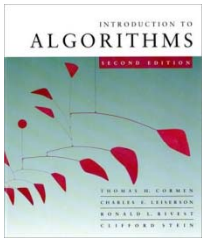


The master method

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.



Master Theorem

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

CASE 1:

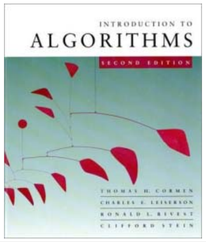
$$f(n) = O(n^{\log_b a - \varepsilon}) \quad \Rightarrow T(n) = \Theta(n^{\log_b a})$$

CASE 2:

$$f(n) = \Theta(n^{\log_b a} \log^k n) \quad \Rightarrow T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

CASE 3:

$$\left.\begin{array}{l} f(n) = \Omega(n^{\log_b a + \varepsilon}) \\ \text{and } a\,f(n/b) \le c\,f(n) \\ \text{for some constant } c < 1 \end{array}\right\} \Rightarrow T(n) = \Theta(f(n))$$



How to apply the theorem

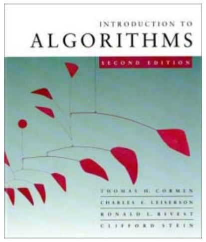
Compare $f(n)$ with $n^{\log_b a}$:

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

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

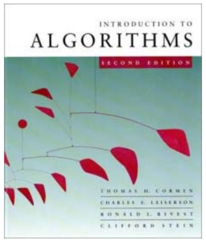


How to apply the theorem

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 $a f(n/b) \leq c f(n)$ for some constant $c < 1$.

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



Example: merge sort

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

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

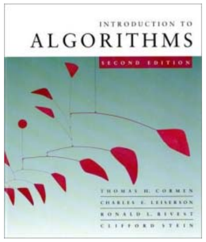
 # subproblems → 2

subproblem size → $n/2$

work dividing and combining → $O(n)$

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

$$\Rightarrow T(n) = \Theta(n \log n)\,.$$



Example: binary search

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

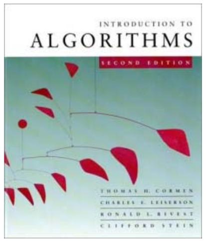
 # subproblems → 1

subproblem size → $n/2$

work dividing and combining → $\Theta(1)$

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

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



Examples

Ex. $T(n) = 4T(n/2) + \mathrm{sqrt}(n)$

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

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1.5$.

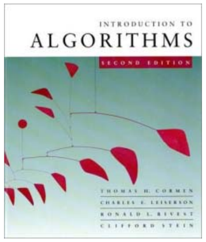
$\therefore\ T(n) = \Theta(n^2)$.

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

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



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

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

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

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

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $\log n \in o(n^\varepsilon)$.