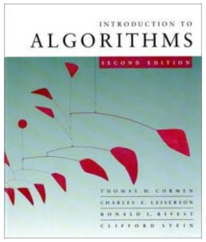

CMPS 2200 – Fall 2012

Divide-and-Conquer III

Carola Wenk

Slides courtesy of Charles Leiserson
with changes and additions 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 $f(n)$

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.

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

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

Master Theorem

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

CASE 1:

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

CASE 2:

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

CASE 3:

$f(n) = \Omega(n^{\log_b a + \varepsilon})$
and $a\,f(n/b) \leq c\,f(n)$
for some constant $c < 1$

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

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

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 $a f(n/b) \leq c f(n)$ 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 — *subproblem size* — *work dividing and combining*

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

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

Example: binary search

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

 # subproblems

subproblem size

work dividing and combining

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

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

Matrix multiplication: Divide-and-conquer algorithm

IDEA:

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

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

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= a \cdot e + b \cdot g \\ s &= a \cdot f + b \cdot h \\ t &= c \cdot e + d \cdot g \\ u &= c \cdot f + d \cdot h \end{aligned}\right\}$$

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

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

Matrix multiplication: Analysis of D&C algorithm

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

 # 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 matrix multiplication algorithm.

Strassen's algorithm

1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form P-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.

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

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

Master theorem: Examples

Ex. $T(n) = 4T(n/2) + \text{sqrt}(n)$
$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = \text{sqrt}(n).$
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, $k = 0$.
$\therefore\ T(n) = \Theta(n^2 \log n).$

Master theorem: 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 $\log n \in o(n^{\varepsilon})$.

Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method .
- Can lead to more efficient algorithms