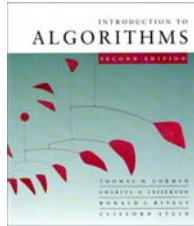




CS 5633 -- Spring 2010



More Divide & Conquer

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

1/21/10

CS 5633 Analysis of Algorithms

1



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 is $f(n)$

1/21/10

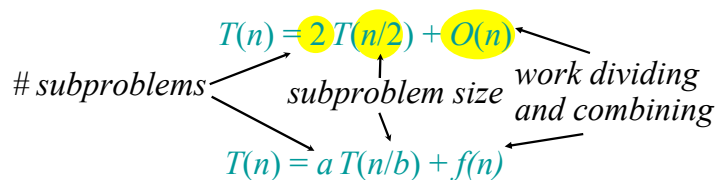
CS 5633 Analysis of Algorithms

2



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 $f(n) \in O(n)$



1/21/10

CS 5633 Analysis of Algorithms

3



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.

1/21/10

CS 5633 Analysis of Algorithms

4



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. $f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$.
 - $f(n)$ and $n^{\log_b a}$ grow at similar rates.
 - Solution:** $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.



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 $af(n/b) \leq cf(n)$ for some constant $c < 1$.
 - Solution:** $T(n) = \Theta(f(n))$.



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, $k = 0$.
 $\therefore T(n) = \Theta(n^2 \log n)$.



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 $\log n \in o(n^\varepsilon)$.



Master theorem (summary)

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

CASE 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$\Rightarrow T(n) = \Theta(n^{\log_b a}) .$$

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: $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af(n/b) \leq cf(n)$
for some constant $c < 1$.

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



Example: merge sort

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

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

subproblems → 2, subproblem size → n/2, work dividing and combining → O(n)

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

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



Recurrence for binary search

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

subproblems → 1, subproblem size → n/2, work dividing and combining → Θ(1)

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

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



Powering a number

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

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm: (recursive squaring)

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

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



Fibonacci numbers

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

0 1 1 2 3 5 8 13 21 34 ...

Naive recursive algorithm: $\Omega(\phi^n)$ (exponential time), where $\phi = (1 + \sqrt{5})/2$ is the *golden ratio*.



Computing Fibonacci numbers

Naive recursive squaring:

$F_n = \phi^n / \sqrt{5}$ rounded to the nearest integer.

- Recursive squaring: $\Theta(\log n)$ time.
- This method is unreliable, since floating-point arithmetic is prone to round-off errors.

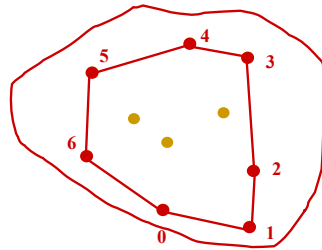
Bottom-up (one-dimensional dynamic programming):

- Compute $F_0, F_1, F_2, \dots, F_n$ in order, forming each number by summing the two previous.
- Running time: $\Theta(n)$.



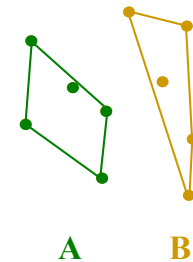
Convex Hull

- Given a set of pins on a pinboard
- And a rubber band around them
- How does the rubber band look when it snaps tight?
- We represent convex hull as the sequence of points on the convex hull polygon, in counter-clockwise order.



Convex Hull: Divide & Conquer

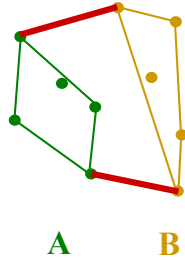
- Preprocessing: sort the points by x-coordinate
- Divide the set of points into two sets **A** and **B**:
 - **A** contains the left $\lfloor n/2 \rfloor$ points,
 - **B** contains the right $\lceil n/2 \rceil$ points
- Recursively compute the convex hull of **A**
- Recursively compute the convex hull of **B**
- Merge the two convex hulls





Merging

- Find upper and lower tangent
- With those tangents the convex hull of $A \cup B$ can be computed from the convex hulls of A and the convex hull of B in $O(n)$ linear time



1/21/10

CS 5633 Analysis of Algorithms

17



Finding the lower tangent

$a =$ rightmost point of A

$b =$ leftmost point of B

while $T=ab$ not lower tangent to both convex hulls of A and B do {

while T not lower tangent to convex hull of A do {

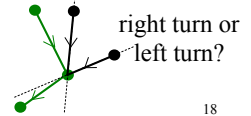
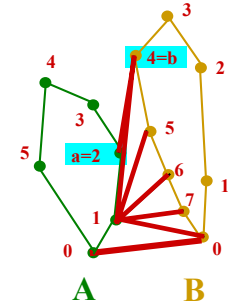
$a=a-1$

} while T not lower tangent to convex hull of B do {

$b=b+1$

}

can be checked in constant time



1/21/10

CS 5633 Analysis of Algorithms

18



Convex Hull: Runtime

- Preprocessing: sort the points by x-coordinate $O(n \log n)$ just once
- Divide the set of points into two sets A and B: $O(1)$
 - A contains the left $\lfloor n/2 \rfloor$ points,
 - B contains the right $\lceil n/2 \rceil$ points
- Recursively compute the convex hull of A $T(n/2)$
- Recursively compute the convex hull of B $T(n/2)$
- Merge the two convex hulls $O(n)$

1/21/10

CS 5633 Analysis of Algorithms

19



Convex Hull: Runtime

- Runtime Recurrence: $T(n) = 2 T(n/2) + cn$
- Solves to $T(n) = \Theta(n \log n)$

1/21/10

CS 5633 Analysis of Algorithms

20



Matrix multiplication

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

$$\begin{bmatrix} c_{11} & c_{12} & \Lambda & c_{1n} \\ c_{21} & c_{22} & \Lambda & c_{2n} \\ \text{M} & \text{M} & \text{O} & \text{M} \\ c_{n1} & c_{n2} & \Lambda & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \Lambda & a_{1n} \\ a_{21} & a_{22} & \Lambda & a_{2n} \\ \text{M} & \text{M} & \text{O} & \text{M} \\ a_{n1} & a_{n2} & \Lambda & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \Lambda & b_{1n} \\ b_{21} & b_{22} & \Lambda & b_{2n} \\ \text{M} & \text{M} & \text{O} & \text{M} \\ b_{n1} & b_{n2} & \Lambda & b_{nn} \end{bmatrix}$$

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



Standard algorithm

```
for i ← 1 to n
  do for j ← 1 to n
    do cij ← 0
      for k ← 1 to n
        do cij ← cij + aik · bkj
```

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



Divide-and-conquer algorithm

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

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

$$C = A \cdot B$$

$r = a \cdot e + b \cdot g$
 $s = a \cdot f + b \cdot h$
 $t = c \cdot e + d \cdot g$
 $u = c \cdot f + d \cdot h$

8 recursive 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

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

No better than the ordinary algorithm.



Strassen's idea

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

$$\begin{aligned}
 P_1 &= a \cdot (f - h) & r &= P_5 + P_4 - P_2 + P_6 \\
 P_2 &= (a + b) \cdot h & s &= P_1 + P_2 \\
 P_3 &= (c + d) \cdot e & t &= P_3 + P_4 \\
 P_4 &= d \cdot (g - e) & u &= P_5 + P_1 - P_3 - P_7 \\
 P_5 &= (a + d) \cdot (e + h) \\
 P_6 &= (b - d) \cdot (g + h) \\
 P_7 &= (a - c) \cdot (e + f)
 \end{aligned}$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!



Strassen's idea

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

$$\begin{aligned}
 P_1 &= a \cdot (f - h) & r &= P_5 + P_4 - P_2 + P_6 \\
 P_2 &= (a + b) \cdot h & &= (a + d)(e + h) \\
 P_3 &= (c + d) \cdot e & &+ d(g - e) - (a + b)h \\
 P_4 &= d \cdot (g - e) & &+ (b - d)(g + h) \\
 P_5 &= (a + d) \cdot (e + h) & &= ae + ah + de + dh \\
 P_6 &= (b - d) \cdot (g + h) & &+ dg - de - ah - bh \\
 P_7 &= (a - c) \cdot (e + f) & &+ bg + bh - dg - dh \\
 & & &= ae + bg
 \end{aligned}$$



Strassen's algorithm

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

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



Analysis of Strassen

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

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

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 $n \geq 30$ or so.

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



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 (so practice this math).
- Can lead to more efficient algorithms