

Data Structures and Algorithm

Xiaoqing Zheng
zhengxq@fudan.edu.cn



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.

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

The diagram illustrates the recurrence relation $T(n) = 2T(n/2) + \Theta(n)$. The terms 2 , $T(n/2)$, and $\Theta(n)$ are highlighted in yellow circles. Arrows point from descriptive text to these terms: 'subproblem number' points to the '2', 'subproblem size' points to ' $T(n/2)$ ', and 'work dividing and combining' points to ' $\Theta(n)$ '.

subproblem number

subproblem size

work dividing and combining

Binary search

- ❑ 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

- ❑ 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

- ❑ 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

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

*subproblem
number*

*subproblem
size*

*work dividing
and combining*

Recurrence for binary search

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

*subproblem
number*

*subproblem
size*

*work dividing
and combining*

$$a = 1, b = 2 \Rightarrow n^{\log_b a} = n^0 = 1$$

$$\text{CASE 2: } f(n) = \Theta(1)$$

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

Powering a number

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

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

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

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 ...

Computing Fibonacci numbers

Naive recursive algorithm: $\Omega(\phi^n)$

(exponential time), where $\phi = (1 + \sqrt{5}) / 2$
is the *golden ratio*.

Bottom-up:

Compute $F_0, F_1, F_2, \dots, F_n$ in order, forming each number by summing the two previous.

Running time: $\Theta(n)$.

Recursive squaring

Theorem:
$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n .$$

Algorithm: Recursive squaring.

$$\text{Time} = \Theta(\lg n) .$$

Proof of theorem. (Induction on n .)

Recursive squaring

Theorem:
$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n .$$

Algorithm: Recursive squaring.

$$\text{Time} = \Theta(\lg n) .$$

Proof of theorem. (Induction on n .)

Base ($n = 1$):
$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$$

Recursive squaring (continued)

Inductive step ($n \geq 2$):

$$\begin{aligned} \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} &= \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 \\ &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 \\ &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \end{aligned}$$

Matrix multiplication

Input: $A = [a_{ij}]$, $B = [b_{ij}]$.
Output: $C = [c_{ij}] = A \cdot B$.
 $\left. \vphantom{\begin{matrix} \text{Input} \\ \text{Output} \end{matrix}} \right\} i, j = 1, 2, \dots, n.$

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

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

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  $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$ 
```

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

Divide-and-conquer algorithm

IDEA:

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

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

$$C = A \cdot B$$

$$\left. \begin{array}{l} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{array} \right\} \begin{array}{l} \text{recursive} \\ 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$

Analysis of D&C algorithm

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

*submatrices
number*

*submatrices
size*

*work dividing
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.

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

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

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

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

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

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

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

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

$$s = P_1 + P_2$$

$$t = P_3 + P_4$$

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

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.

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

Analysis of Strassen

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

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

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.

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).
- ❑ The divide-and-conquer strategy often leads to efficient algorithms.

Any questions?



Xiaoqing Zheng
Fudan University