

Binary search

Find an element in a sorted array:

- 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

Find an element in a sorted array:

- 1. Divide:* Check middle element.
- 2. Conquer:* Recursively search **1** subarray.
- 3. Combine:* Trivial.

Example: Find **9**



Binary search

Find an element in a sorted array:

- 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

Find an element in a sorted array:

- 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

Find an element in a sorted array:

- 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

Find an element in a sorted array:

- 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

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

subproblems *subproblem size* *work dividing and combining*

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

Powering a number

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

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

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

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 ...

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

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

Matrix multiplication

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

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

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

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:

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

$$C = A \cdot B$$

$$r = ae + bg$$

$$s = af + bh$$

$$t = ce + dh$$

$$u = cf + dh$$

8 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*

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

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

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

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

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 $n \geq 30$ or so.

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

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