Divide-and-Conquer

Divide-and-Conquer design paradigm

- Divide : divide a problem into subproblems
- Conquer : solve the subproblems recursively
- Combine : combine the subproblem solutions appropriately

Multiplication

- Problem : multiply two *n*-bit numbers.
- ex) 41×42 (in binary, 101001×101010).
- Add 41 to itself 42 times:
 - $-\Theta(2^n)$ additions.
- Better algorithm?

Multiplication

- Problem: multiply two *n*-bit numbers.
 ex) 41×42 (in binary, 101001 × 101010).
- Add 41 to itself 42 times : $\Theta(2^n)$ additions.
- Better algorithm?

Divide-and-Conquer Multiplication

• Problem : Give a divide-and-conquer algorithm to multiply two *n*-bit numbers.

$$X = 2^{n/2}A + B$$
 A B $Y = 2^{n/2}C + D$ C D

$$XY = 2^n AC + 2^{n/2} BC + 2^{n/2} AD + BD$$

$$T(n) = 4T(n/2) + cn$$

Karatsuba algorithm

$$XY = 2^{n}AC + 2^{n/2}BC + 2^{n/2}AD + BD$$

$$(2^{n} - 2^{n/2})AC + 2^{n/2}(A + B)(C + D) + (1 - 2^{n/2})BD$$

$$T(n) = 3T(n/2) + c'n$$

 $O(n^{\log_2 3}) \approx O(n^{1.585})$

Can we do better?

- Karp used a Fast Fourier Transform $O(n \log^2 n)$
- Schonhage and Strassen improved it to $O(n \log n \log \log n)$ in 1971
- Furer improved log log n term with $2^{O(\log^* n)}$ in 2007
- Open question : Is $O(n \log n)$ possible?

Matrix Multiplication

Input:
$$A = [a_{ij}], B = [b_{ij}].$$

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

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

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

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}
```

Divide-and-Conquer algorithm

IDEA:

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

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

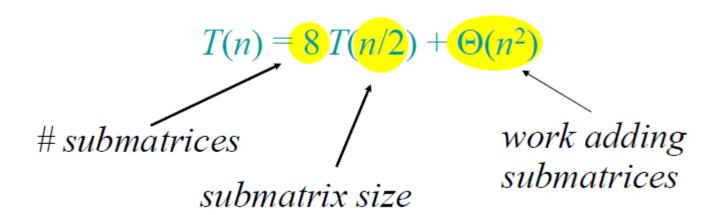
$$C = A \cdot B$$

$$r = ae + bg$$

 $s = af + bh$
 $t = ce + dg$
 $u = cf + dh$

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

<u>Analysis</u>



Strassen's idea

Multiply 2×2 matrices with only 7 recursive mults.

$$P_{1} = a \cdot (f - h)$$
 $r = P_{5} + P_{4} - P_{2} + P_{6}$
 $P_{2} = (a + b) \cdot h$ $s = P_{1} + P_{2}$
 $P_{3} = (c + d) \cdot e$ $t = P_{3} + P_{4}$
 $P_{4} = d \cdot (g - e)$ $u = P_{5} + P_{1} - P_{3} - P_{7}$
 $P_{5} = (a + d) \cdot (e + h)$
 $P_{6} = (b - d) \cdot (g + h)$
 $P_{7} = (a - c) \cdot (e + f)$

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.

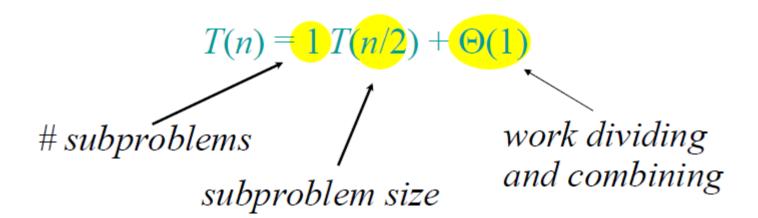
$$T(n) = 7 T(n/2) + \Theta(n^2)$$

Binary Search

Find an element in a sorted array:

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

Analysis of Binary Search



Powering a number

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

Naive algorithm: $\Theta(n)$.

Divide-and-Conquer

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

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

Fibonacci number

Recursive definition:

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

Recursive Squaring

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

Proof

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

Proof of theorem. (Induction on n.)

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

Inductive step $(n \ge 2)$:

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