DS 320 Algorithms for Data Science	**Lecture #12**
Spring 2022	Prof. Kira Goldner

Divide & Conquer III: Integer and Matrix Multiplication

Theorem 1 (The Master Theorem)**.** *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by the recurrence*

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

where we interpret n/b as $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for a constant } \varepsilon > 0 \\ \Theta(n^{\log_b a} \log_2 n) & \text{if } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for a constant } \varepsilon > 0 \text{ and} \\ & \quad af(n/b) \leq cf(n) \text{ for a constant } c < 1 \text{ and for all sufficiently large } n. \end{cases}$$

The Problem: Integer Multiplication

Your input for the *integer multiplication* problem is two n-digit numbers x and y. The goal is to output their product, $x \cdot y$.

What's the **naïve algorithm** and what's its **running time**?

Grade-school multiplication! You multiply together the digits of each of the numbers (plus the shifts and additions), which is $\Theta(n^2)$.

Step 1: Define your recursive subproblem.

One idea is to split each number into two parts: $x = 10^{n/2}a + b$ and $y = 10^{n/2}c + d$. Then

$$xy = 10^n ac + 10^{n/2}(ad + bc) + bd.$$

Additions and multiplications by powers of 10 (just shifts) are linear-time, so this reduces the problem to smaller multiplication problems:

$$T(n) = 4T(n/2) + O(n).$$

Which gives what running time?

$\Theta(n^2)$. This is not better.

The Speed Up:

We actually only need to make *three* recursive calls: ac, bd, and $(a+b)(c+d)$.

Step 2: Combine the solutions to your subproblems.

Show why this is enough.

$$(a+b)(c+d) = (ad+bc) + (ac+bd)$$

Then:

$$\begin{aligned} T(n) &= 3T(n/2) + O(n) \\ &= n^{\log_2 3} \approx n^{1.59}. \end{aligned}$$

Matrix Multiplication: Strassen's Algorithm

The Problem: Matrix Multiplication

Your input for the *matrix multiplication* problem is two $n \times n$ matrices A and B. The goal is to output their product, $C = AB$. Recall that the ik^{th} entry of C is given by $c_{ik} = \sum_{j=1}^{n} a_{ij}b_{jk}$.

What's the **running time** of the **naïve algorithm** here and why?

Each entry takes linear time and there are n^2 entries, hence $\Theta(n^3)$.

Step 1: Define your recursive subproblem.

We divide each matrix into four submatrices.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

What running time does this give us?

$$\begin{aligned} T(n) &= 8T(n/2) + \Theta(n^2) \\ &= \Theta(n^3). \end{aligned}$$

The Speed Up:

We compute only the following products.

- $P1 = A(F - H)$
- $P2 = (A + B)H$
- $P3 = (C + D)E$
- $P4 = D(G - E)$
- $P5 = (A + D)(E + H)$
- $P6 = (B - D)(G + H)$
- $P7 = (A - C)(E + F)$

Step 2: Combine the solutions to your subproblems.

Show why this is enough.

$$\begin{aligned} AE + BG &= P5 + P4 - P2 + P6 \\ AF + BH &= P1 + P2 \\ CE + DG &= P3 + P4 \\ CF + DH &= P5 + P1 - P3 - P7 \end{aligned}$$

Then the running time:

$$\begin{aligned} T(n) &= 7T(n/2) + \Theta(n^2) \\ &= n^{\log_2 7} \approx n^{2.8}. \end{aligned}$$