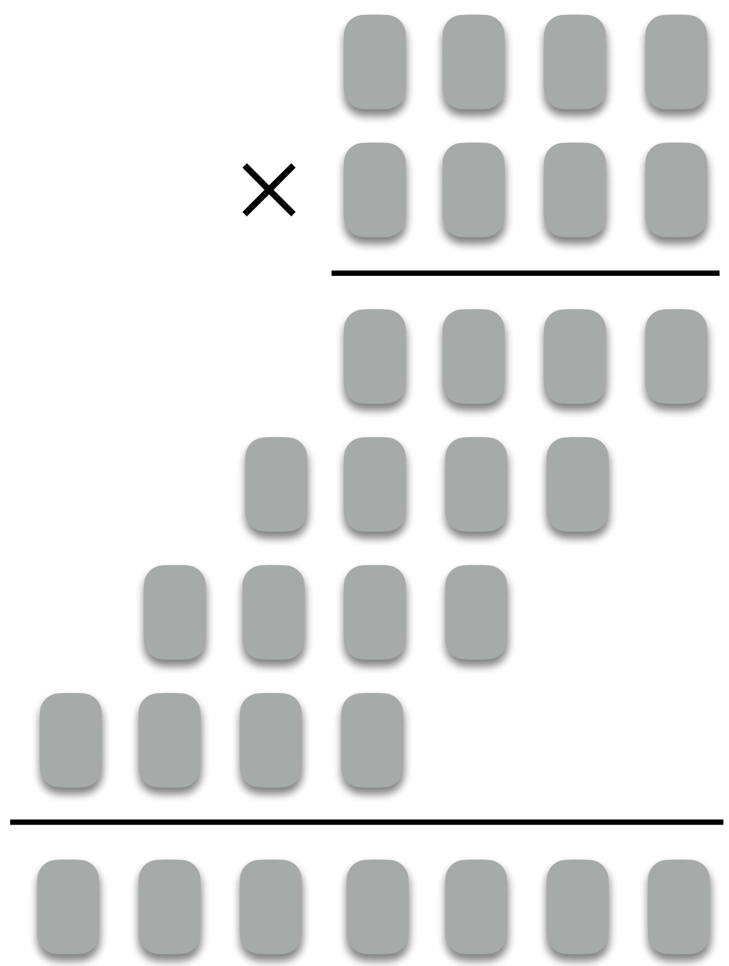
CSE202
Design and Analysis of Algorithms

Week 2 — Divide & Conquer 1: How fast can we multiply?

Naive Multiplication



Input: two n-digit integers

n multiplications $+ O(n)$ carries

⋮

n multiplications $+ O(n)$ carries

$O(n^2)$ additions $+ O(n)$ carries

Output: $\leq 2n$ digits

Total: $O(n^2)$ digit (or bit) operations

For integers $\leq N$ this is $O(\log^2 N)$

Quadratic algorithm: #operations $O(n^2)$ for an input size n

One Second of Computation

With a good polynomial or integer library,
1 sec. is sufficient to

multiply two integers with 30,000,000 digits;

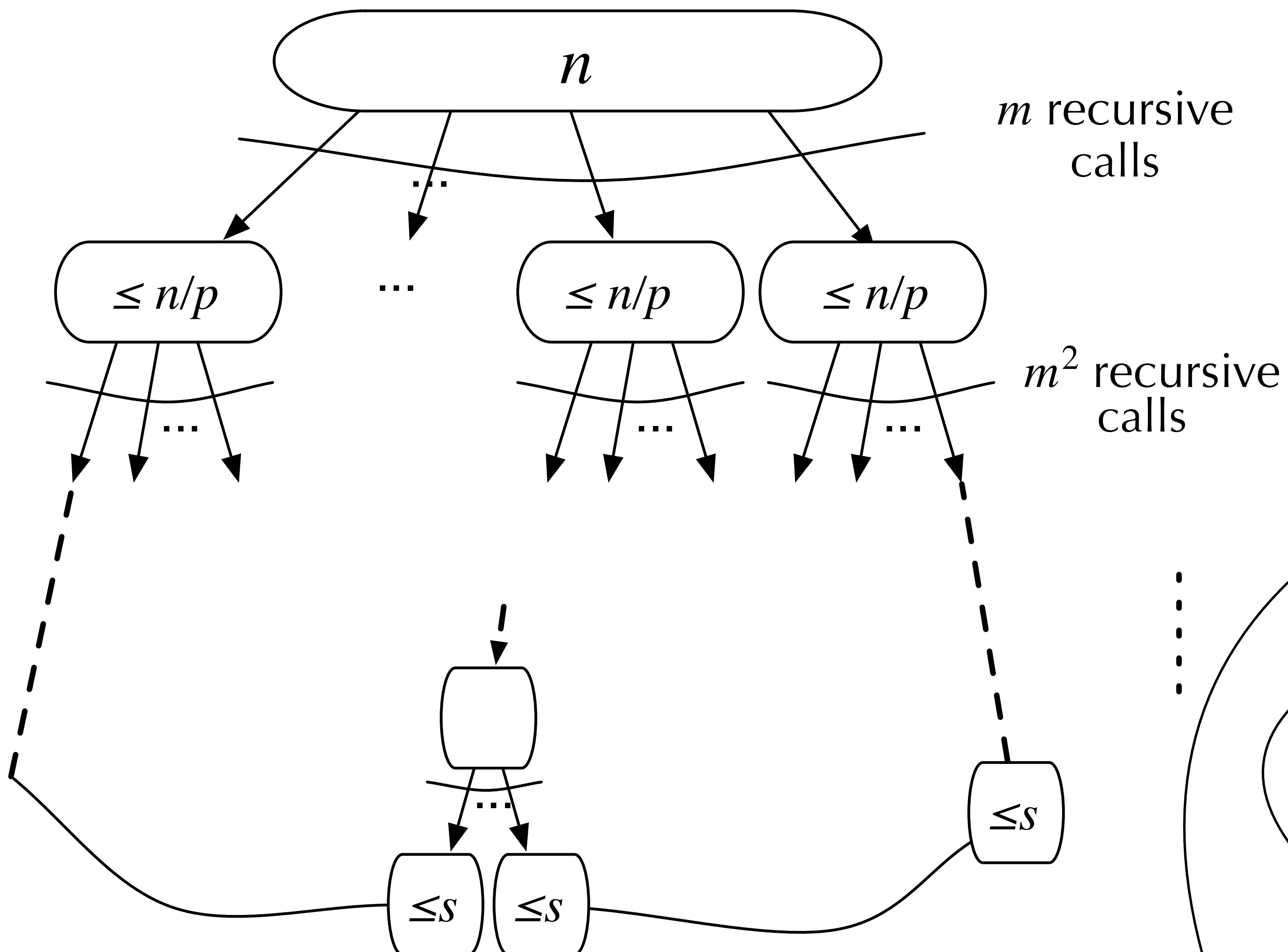
multiply two polynomials of degree 650,000;

multiply two matrices of size 850x850;

(but factor an integer with 42 digits only).

1 sec. is in the asymptotic regime of the algorithms

Divide and Conquer



The total cost may be

- mostly at the top (quickselect, Newton)
- mostly at the leaves (Karatsuba, Strassen)
- balanced along the levels (binary powering, mergesort, FFT)

later this week next week

I. Polynomials

Polynomials and Integers

Polynomials		Integers
$$\begin{aligned} &(7x^2 + 3x + 5) \times (2x + 3) \\ = &\quad (7x^2 + 3x + 5) \times 3 \\ &+ (7x^2 + 3x + 5) \times 2x \\ = &\quad 21x^2 + 9x + 15 \\ &+ 14x^3 + 6x^2 + 10x \\ = &\quad 14x^3 + 27x^2 + 19x + 15 \end{aligned}$$	$\xrightarrow[x \mapsto 10]{}$	$$\begin{array}{r} 735 \\ \times\quad 23 \\ \hline 2205 \\ +\ 1470 \\ \hline 16905 \end{array}$$

degree n_1 $\times$ degree n_2 $\qquad$ n_1 digits $\times$ n_2 digits

$$\longrightarrow \begin{array}{ll} (n_1 + 1)(n_2 + 1) & \text{multiplications} \\ +O(n_1 n_2) & \text{additions} \end{array} \quad \text{of} \begin{cases} \text{coefficients} \\ \text{digits} \end{cases}$$

Polynomials behave like integers, without carries

Divide & Conquer by Itself Doesn't Help

F and G of degree $< n \mapsto H := FG$

1. If $n = 1$ return FG
2. Let $k := \lceil n/2 \rceil$
3. Split $F = F_0 + x^k F_1, G = G_0 + x^k G_1$

   F_0, F_1, G_0, G_1 of degree $< k$
4. Compute recursively

   $H_0 := F_0 G_0,\ H_1 := F_0 G_1,\ H_2 := F_1 G_0,\ H_3 := F_1 G_1$
5. Return $H_0 + x^k(H_1 + H_2) + x^{2k} H_3$

Complexity: $C(n) \leq 4C(\lceil n/2 \rceil) + \lambda n$ coefficient operations

Complexity of the Naive DAC

Notation:

$$\lceil x/2 \rceil_1 = \lceil x/2 \rceil$$

$$\lceil x/2 \rceil_{k+1} = \left\lceil \lceil x/2 \rceil_k / 2 \right\rceil$$

N : power of 2 s.t. $n \le N < 2n$

$$C(n) \le 4C(\lceil n/2 \rceil) + \lambda n$$

iterate once

$$\le \lambda n + 4\lambda \lceil n/2 \rceil + 16C(\lceil n/2 \rceil_2)$$

iterate k-1 times, use N

$$\le \lambda N \left(1 + 2 + \cdots + 2^{k-1}\right) + 4^k C(\lceil n/2 \rceil_k)$$

bound geometric series

$$\le 4^k \left(\lambda \frac{N}{2^k} + C(\lceil n/2 \rceil_k) \right)$$

use $k = \lceil \log_2 n \rceil$

$$= O(n^2)$$

An extra idea is needed to beat the naive algorithm

Polynomials of Degree 1

$$F = f_0 + f_1 T, \quad G = g_0 + g_1 T \quad \mapsto \quad H := FG = h_0 + h_1 T + h_2 T^2$$

Naive algorithm:

$$H = (\underbrace{f_0 g_0}) + (\underbrace{f_0 g_1} + \underbrace{f_1 g_0})T + \underbrace{f_1 g_1} T^2$$

4 multiplications & 1 addition

Interpolation from 3 values:

$$h_0 = F(0)G(0) = f_0 g_0$$ 1 mult.

$$h_2 = \text{"}F(\infty)G(\infty)\text{"} = f_1 g_1$$ 1 mult.

$$\tilde{h}_1 = h_0 + h_1 + h_2 = F(1)G(1) = (f_0 + f_1)(g_0 + g_1)$$ 1 mult.

$$FG = h_0 + (\tilde{h}_1 - h_0 - h_2)T + h_2 T^2$$

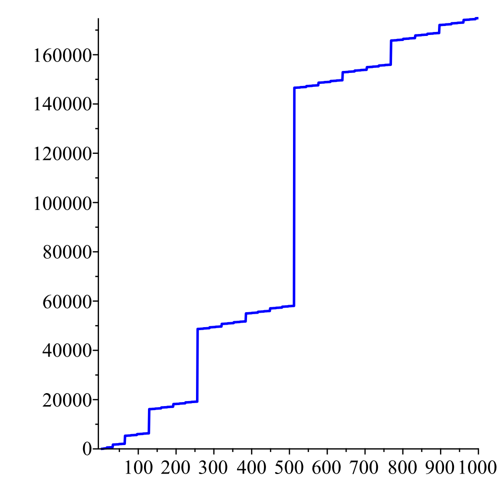
3 multiplications, 2 additions, 2 subtractions

Karatsuba's Algorithm

F and G of degree $< n \mapsto H := FG$

Idea: Evaluate $FG = h_0 + (\tilde{h}_1 - h_0 - h_2)T + h_2T^2$ at $T = x^k$.

1. If n is small, use naive multiplication
2. Let $k := \lceil n/2 \rceil$
3. Split $F = F_0 + x^kF_1, G = G_0 + x^kG_1$
   F_0, F_1, G_0, G_1 of degree $< k$
4. Compute recursively
   $H_0 := F_0G_0,\ H_2 := F_1G_1,\ \tilde{H}_1 := (F_0 + F_1)(G_0 + G_1)$
5. Return $H_0 + x^k(\tilde{H}_1 - H_0 - H_2) + x^{2k}H_2$



$u_n = n + 3u_{\lceil n/2 \rceil}, u_1 = 1$

Complexity: $C(n) \leq 3C(\lceil n/2 \rceil) + \lambda n$ coefficient operations

Complexity of Karatsuba's Algorithm

Notation:

$\lceil x/2 \rceil_1 = \lceil x/2 \rceil$

$\lceil x/2 \rceil_{k+1} = \left\lceil \lceil x/2 \rceil_k / 2 \right\rceil$

N : power of 2 s.t. $n \leq N < 2n$

$$C(n) \leq 3C(\lceil n/2 \rceil) + \lambda n$$

iterate once

$$\leq \lambda n + 3\lambda \lceil n/2 \rceil + 9C(\lceil n/2 \rceil_2)$$

iterate k-1 times, use N

$$\leq \lambda N \left(1 + \frac{3}{2} + \cdots + \left(\frac{3}{2}\right)^{k-1} \right) + 3^k C(\lceil n/2 \rceil_k)$$

reorder sum

$$\leq \lambda N \left(\frac{3}{2}\right)^{k-1} \left(1 + 2/3 + \cdots + (2/3)^{k-1}\right) + 3^k C(\lceil n/2 \rceil_k)$$

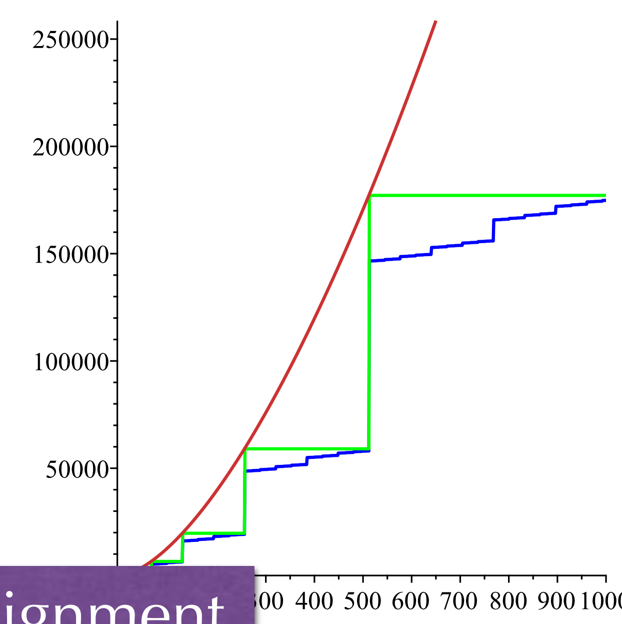
bound geometric series

$$\leq 3^k \left(2\lambda \frac{N}{2^k} + C(\lceil n/2 \rceil_k) \right)$$

use $k = \lceil \log_2 n \rceil$

$$\leq (2\lambda + 1) 3^{\lceil \log_2 n \rceil} = O(n^{\log_2 3})$$

≈ 1.58



Not the final word. See assignment.

II. Integers

From Polynomials to Integers

Recall: Polynomials behave like integers, without carries

No theorem of complexity equivalence exists, but the algorithms over polynomials can often be adapted to integers, with the same complexity.

Karatsuba's Algorithm for Integers

F and G integers $< 2^n \mapsto H := FG$

1. If n is small, use naive multiplication
2. Let $k := \lceil n/2 \rceil$
3. Split $F = F_0 + 2^k F_1, G = G_0 + 2^k G_1$

   $F_0, F_1, G_0, G_1 < 2^k$
4. Compute recursively

   $H_0 := F_0 G_0, \quad H_2 := F_2 G_2, \quad \tilde{H}_1 := (F_0 + F_1)(G_0 + G_1)$
5. Return $H_0 + 2^k(\tilde{H}_1 - H_0 - H_2) + 2^{2k} H_2$

Obtained by changing x into 2 in the polynomial version.

Same algorithm as for polynomials,
similar (not exactly the same) complexity analysis.

$\rightarrow O(n^{\log_2 3})$ bit operations ≈ 1.58

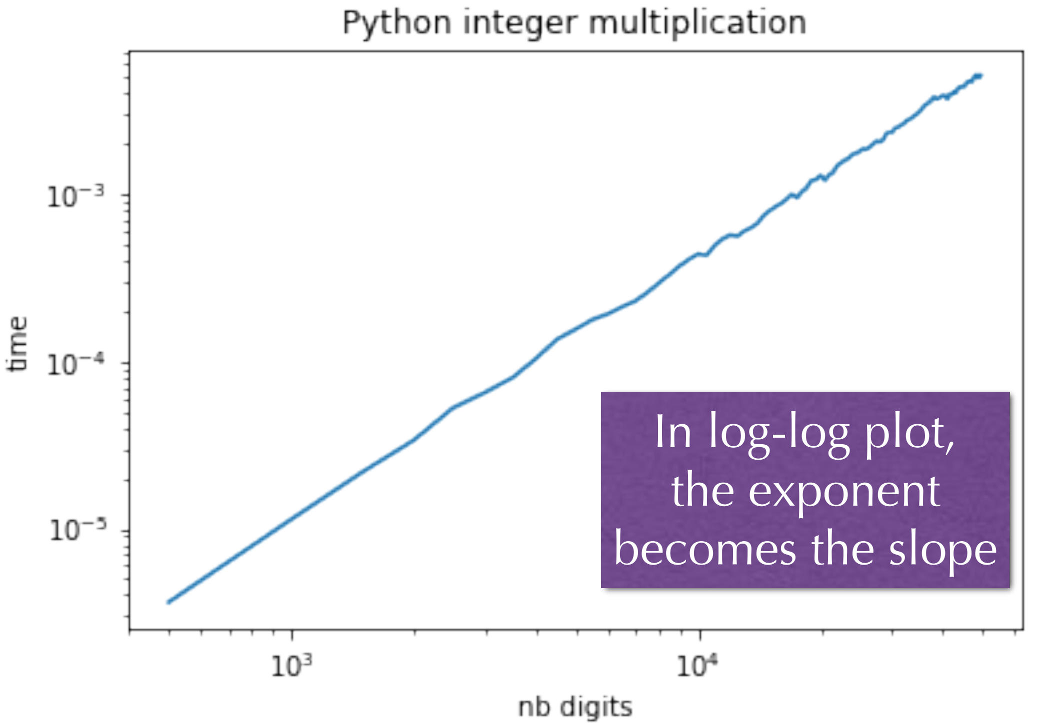
Experiments in Python

A Python long integer is stored using a list of 30-bit digits (CSE 101)

```python
import timeit, math
import matplotlib.pyplot as plt

def testmul(nbdigits,nbtimes):
    # produce two integers with nbdigits decimal digits
    a = 7**math.trunc(nbdigits*math.log(10,7))
    b = 9**math.trunc(nbdigits*math.log(10,9))
    def doit():
        return a*b
    return min(timeit.repeat(doit,number=nbtimes,repeat=3))/nbtimes

t = [500*i for i in range(1,100)]
L = [testmul(i,20) for i in t]
plt.loglog(t,L)
```




```python
import numpy
numpy.polyfit([math.log(i) for i in t[15:]],[math.log(i) for i in L[15:]],1)
```

```
Out[2]: array([ 1.58775052, -22.40117315])
```

clearly Karatsuba's algorithm

Application: Fast Evaluation of Linear Recurrences with Constant Coefficients

Ex.: F_{100} where $F_{n+2} = F_{n+1} + F_n$, $F_0 = F_1 = 1$ (Fibonacci)

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}}_{M} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = M^{n-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

$\longrightarrow$ $O(\log n)$ coefficient operations by binary powering

$$M^{99} = ((M^2M)^{16}M)^2M$$

Exercise (more difficult):
estimate the number of **bit** operations

Works for all linear recurrences with constant coefficients

Which of these Algorithms is Best?

None of them!

GMP (the Gnu Multiprecision Library) uses:

# 64-bit words	approx # digits	Algorithm
0	**0**	Naive
26	**500**	Karatsuba
73	**1.400**	Toom - 3
208	**4.000**	Toom - 4
4736	**90.000**	FFT

Assignment this week

Next week

III. Matrices

Matrix Multiplication: Strassen's Algorithm

Input: two $n \times n$ matrices A, X with $n = 2^k$

Output: AX

1. If $n = 1$, return AX
2. Split $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, X = \begin{pmatrix} x & y \\ z & t \end{pmatrix}$, with $(n/2) \times (n/2)$ blocks
3. Compute recursively the 7 products

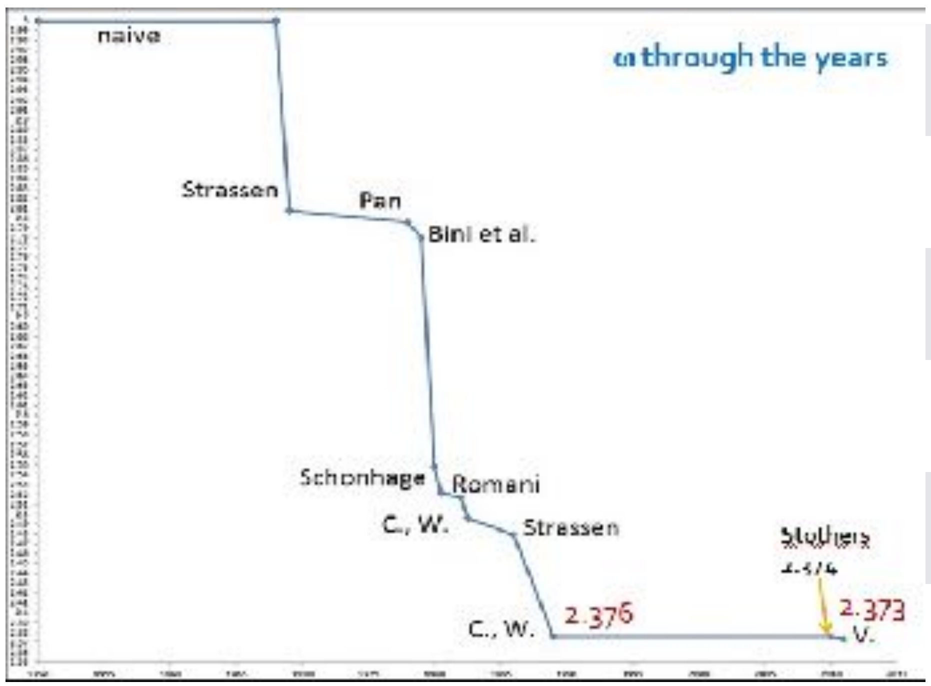
$$q_1 = a(x+z),\ q_2 = d(y+t),\ q_3 = (d-a)(z-y),$$
$$q_4 = (b-d)(z+t),\ q_5 = (b-a)z,$$
$$q_6 = (c-a)(x+y),\ q_7 = (c-d)y$$

4. Return $\begin{pmatrix} q_1 + q_5 & q_2 + q_3 + q_4 - q_5 \\ q_1 + q_3 + q_6 - q_7 & q_2 + q_7 \end{pmatrix}$

Exercise:
prove the complexity
in $O(n^{\log_2 7})$ operations.

World Records

Best algorithm for matrix multiplication



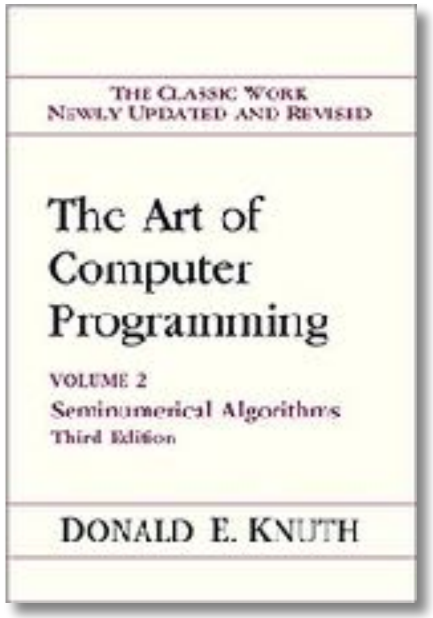
Year	Algorithm	Exponent
	Classical	3
1969	Strassen	2.808
1978	Pan	2.796
1981	Schönhage	2.522
1982	Romani	2.517
1981	Coppersmith-Winograd	2.496
1986	Strassen	2.479
1989	Coppersmith-Winograd	2.376
2012	Vassilevska Williams	2.3729
2014	Le Gall	2.3728

Big Open Question:
is 2 the limit?

References for this lecture

The slides are designed to be self-contained.

They were prepared using the following
books that I recommend if you want to learn more:



Next

Assignment this week: generalisation of Karatsuba's algorithm

Next tutorial: DAC for sequences, sums and polynomials

Next week: fast Fourier transform

Feedback

Moodle

Questions or comments: Bruno.Salvy@inria.fr