CSE202
Design and Analysis of Algorithms

Week 4 — Divide & Conquer 3:
Fast multiplication becomes contagious

Previously on CSE202…

Let $\mathrm{Mul}(n)$ be a bound on the number of coefficient operations needed to multiply two polynomials of degree at most n. Then,

$$\mathrm{Mul}(n) = \begin{cases} O(n^2) & \text{by the naive algorithm;} \\ O(n^{\log_2 3}) & \text{by Karatsuba's algorithm;} \\ O(n^{\log_k (2k-1)}) & \text{by Toom-Cook's algorithm;} \\ O(n \log n) & \text{by FFT (with primitive roots of 1)}. \end{cases}$$

They all satisfy

$$\mathrm{Mul}(n_1) + \mathrm{Mul}(n_2) \le \mathrm{Mul}(n_1 + n_2), \quad \mathrm{Mul}(mn) \le m^2\, \mathrm{Mul}(n).$$

Previously on CSE202…

Let $\mathrm{Mul}_{\mathbb{Z}}(n)$ be a bound on the number of bit operations needed to multiply two integers of at most n bits.
Then,

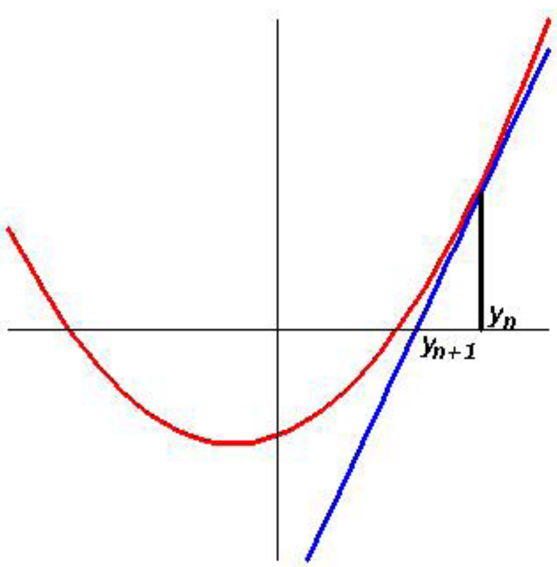
$$\mathrm{Mul}_{\mathbb{Z}}(n) = \begin{cases} O(n^2) & \text{by the naive algorithm;} \\ O(n^{\log_2 3}) & \text{by Karatsuba's algorithm;} \\ O(n^{\log_k (2k-1)}) & \text{by Toom-Cook's algorithm;} \\ O(n \log n) & \text{by FFT (not in the course)}. \end{cases}$$

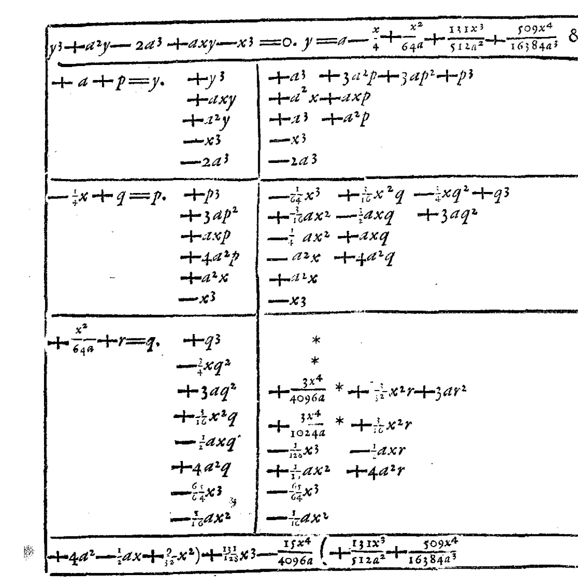
They all satisfy

$$\mathrm{Mul}_{\mathbb{Z}}(n_1) + \mathrm{Mul}_{\mathbb{Z}}(n_2) \leq \mathrm{Mul}_{\mathbb{Z}}(n_1 + n_2), \ \ \mathrm{Mul}_{\mathbb{Z}}(mn) \leq m^2\,\mathrm{Mul}_{\mathbb{Z}}(n).$$

Fast Computation

Today: a common tool for all these operations is another Divide and Conquer algorithm, Newton's iteration.

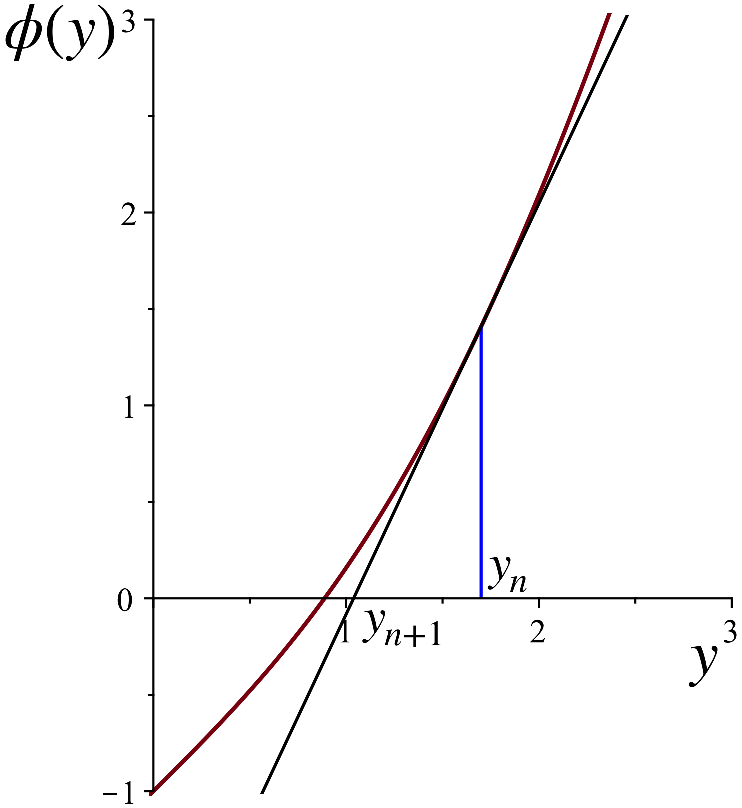




1671

I. Newton’s Iteration

General Principle



To solve $\phi(y) = 0$,

start from a "good" y_0 and iterate:

1. take the tangent: $z = \phi(y_n) + \phi'(y_n)(y - y_n)$
2. y_{n+1} := intersection with the horizontal axis

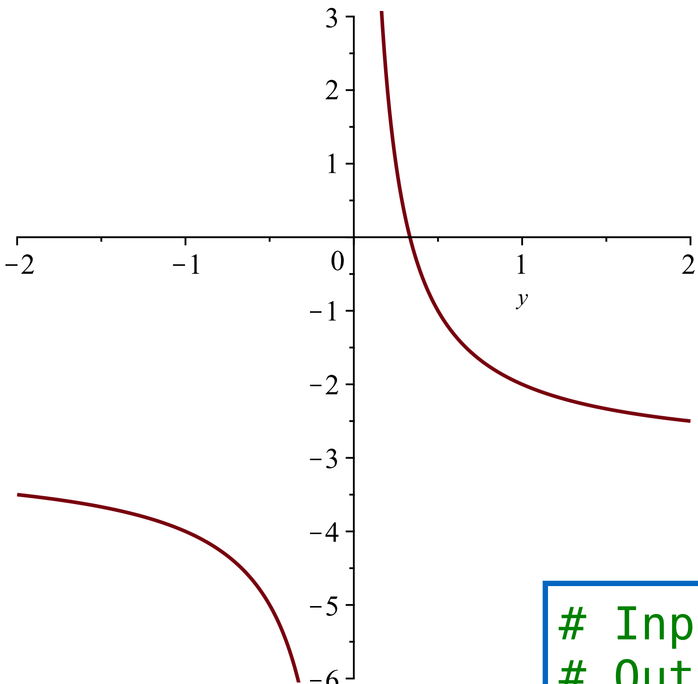
$$\boxed{y_{n+1} = \mathcal{N}(y_n) := y_n - \frac{\phi(y_n)}{\phi'(y_n)}.}$$

Ex. $\phi(y) = 2y - \sin(y) - 1$

$$y_{n+1} = y_n - \frac{2y_n - \sin(y_n) - 1}{2 - \cos(y_n)}$$

$y_0 = 1.7$
$y_1 = 1.0384508857639334498$
$y_2 = 0.8942024477681162624$
$y_3 = 0.8878735991813458814 2$
$y_4 = 0.8878622116076079473 7$
$y_5 = 0.8878622115708660240 3$

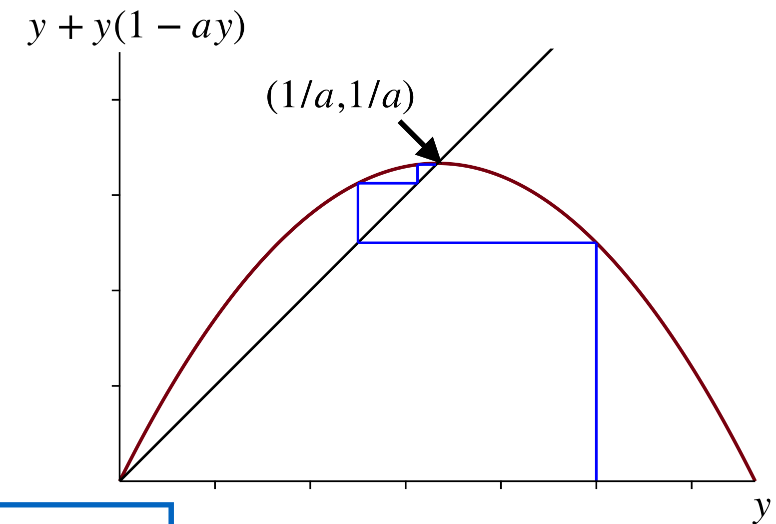
Reciprocal & Division



$$\phi(y) = 1/y - a$$

$$\boxed{y_{n+1} = y_n + y_n(1 - a y_n)}$$

No division needed!




```
# Input: a,y0 (floats)
# Output: 1/a via Newton's iteration starting from y0
def Inverse(a,y0):
    yprev,ynew = 0,y0
    while (ynew!=yprev):
        yprev,ynew = ynew,ynew+ynew*(1-a*ynew)
    return ynew
```

Termination beyond our scope (do not write your code like this).

Quadratic convergence:

$$\frac{1}{a} - y_{n+1} = a\left(\frac{1}{a} - y_n\right)^2$$

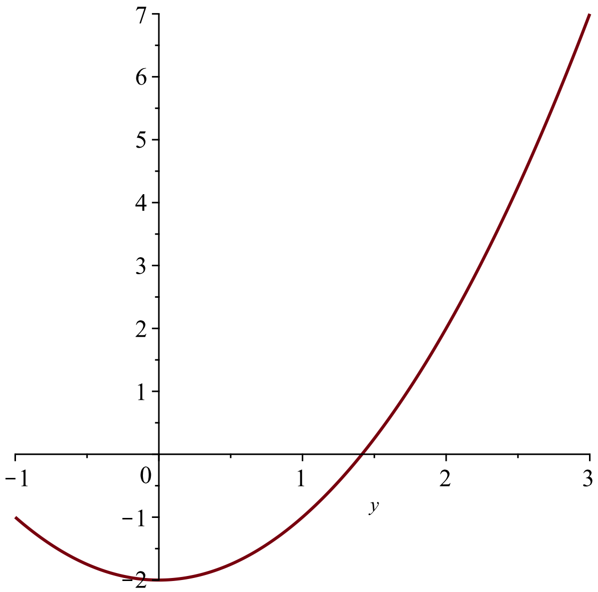
Pentium bug (1994): cost ≈ 475 M$ from a wrong y0.

Application: Euclidean division

$$(A, B) \mapsto (Q, R), \quad A = BQ + R,$$
$$0 \leq R < B$$

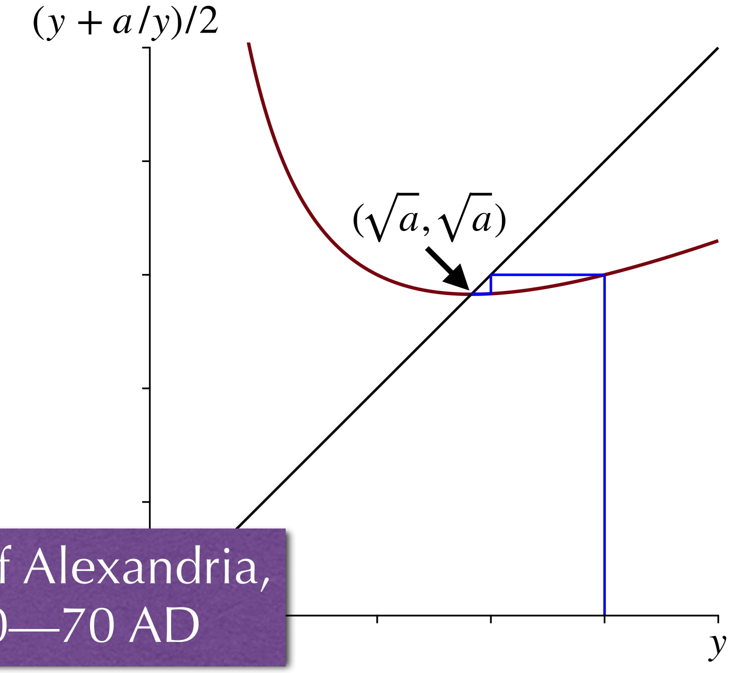
1. Compute $S := 1/B$ to sufficient precision
2. $Q := \lfloor A \times S \rfloor; \; R := A - B \times Q$.

Heron's Iteration for Square Roots



$$\phi(y) = y^2 - a$$

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{a}{y_n}\right)$$



Heron of Alexandria, ca 10—70 AD

```
def Heron(a): # float a -> sqrt(a)
    s = a # initial point > sqrt(a)
    while True:
        y,s = s,(s+a/s)/2
        if y==s: return s
```

$a = 2$

$$y_0 = 2.$$
$$y_1 = 1.50000000000000000000$$
$$y_2 = 1.41666666666666666667$$
$$y_3 = 1.41421568627450980392$$
$$y_4 = 1.41421356237468991063$$
$$y_5 = 1.41421356237309504880$$

Quadratic convergence:

$$y_{n+1} - \sqrt{a} = \frac{(y_n - \sqrt{a})^2}{2y_n}$$

Blackboard proof

II. High Precision in Low Complexity

Big Floats

Python integers stored using lists of 30-bit digits,
but Python floats limited to hardware (53 bits) precision.

Faster multiprecision available in gmpy2.

```
>>> import gmpy2
>>> from gmpy2 import mpz,mpq,mpfr
>>> T = mpz(3204932049820349)*mpz(23049382043289420); T
mpz(73871703239091905050452012407580)
>>> T // 394230480932840
mpz(187382018417993613)
>>> gmpy2.isqrt(T)
mpz(8594864934313505)
>>> mpq(T,23049823048093280348205)
mpq(4924780215939460336696800827172,1536654869872885356547)
>>> gmpy2.sin(gmpy2.const_pi())
mpfr('1.2246467991473532e-16')
>>> gmpy2.get_context().precision=100
>>> gmpy2.const_pi()
mpfr('3.1415926535897932384626433832793',100)
>>> gmpy2.sin(_)
mpfr('1.6956855320737799287917402938778e-31',100)
>>> _.as_mantissa_exp()
(mpz(1089944637035562098257346117665), mpz(-202))
```

Floats:

m : mantissa
e : exponent
encode $m2^e$.

Big integer in gmpy2.

Many of the underlying algorithms rely on Newton's iteration

Truncated Power Series

These are objects of the form

Convergence is not an issue

$$A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1} + O(X^n)$$

They can be added, multiplied,… as expected.
Fast multiplication inherited from polynomials.

Truncated power series are to floating point numbers as polynomials are to integers: similar, but without carries.

Elementary operations:

If $A(X)B(X) = 1 + O(X^n)$, then $B(X) = A(X)^{-1}$.

If $B(X)^2 = A(X) + O(X^n)$, then $B(X) = \sqrt{A(X)}$.

From Quadratic Convergence to DAC Algorithm for the Reciprocal

With $\mathcal{N}(y) = y + y(1 - ay)$, $\quad a^{-1} - \mathcal{N}(y) = a(a^{-1} - y)^2$ implies:

for any y, $\quad y = a^{-1} + O(X^{k}) \Longrightarrow \mathcal{N}(y) = a^{-1} + O(X^{2k})$.

$$a = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + O\left(x^7\right)$$

$y_0 = 1 + O(x)$,

$y_1 = 1 - x + O(x^2)$,

$y_2 = 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + O(x^4)$,

$y_3 = 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 - \frac{1}{120}x^5 + \frac{1}{720}x^6 + O\left(x^7\right)$

← avoid the computation of useless coefficients

The number of correct terms is doubled at each iteration.

1 recursive call in size $\lceil n/2 \rceil$

```python
def InvertSeries(ma):
    n = len(ma)
    if n==1: return [-1/ma[0]]
    k = -(-n//2)      # ceil(n/2)
    s = InvertSeries(ma[:k])+[0]*(n-k)
    t = Mul(ma,s,n)   # -a*s
    t[0] += 1         # 1-a*s
    return Add(s,Mul(s,t,n))# s+s(1-a*s)

def NewtonInvert(a):
    return InvertSeries(ScalarMul(-1,a))
```

Complexity Analysis — n a Power of 2

Hyp. $\mathrm{Mul}(n_1) + \mathrm{Mul}(n_2) \leq \mathrm{Mul}(n_1 + n_2)$

Why $3\,\mathrm{Mul}(n/2)$?

$$C(n) \leq 1 \times C(n/2) + 3\,\mathrm{Mul}(n/2) + \lambda n$$

iterate once

$$\leq 3\,\mathrm{Mul}(n/2) + 3\,\mathrm{Mul}(n/4) + \frac{3}{2}\lambda n + C(n/4)$$

use hyp on Mul

$$\leq \frac{3}{2}\mathrm{Mul}(n)\left(1 + \frac{1}{2}\right) + \frac{3}{2}\lambda n + C(n/4)$$

iterate k-1 times

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

bound geometric series

$$\leq 3\,\mathrm{Mul}(n) + 2\lambda n + C(n/2^k)$$

use $k = \log_2 n$

$$= O(\mathrm{Mul}(n))\,.$$

Division is not harder than multiplication!

Complexity Analysis — General n

Hyp. $\begin{cases} \mathrm{Mul}(n_1) + \mathrm{Mul}(n_2) \leq \mathrm{Mul}(n_1 + n_2), \\ \mathrm{Mul}(mn) \leq m^2 \,\mathrm{Mul}(n)\,. \end{cases}$

implies Mul increasing

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.

$n \leq N < 2n$

$$C(n) \leq 1 \times C(\lceil n/2 \rceil) + 3\,\mathrm{Mul}(\lceil n/2 \rceil) + \lambda n$$

use N and the hyp on Mul

$$\leq \frac{3}{2}\mathrm{Mul}(N) + \lambda n + C(\lceil n/2 \rceil)$$

$n/2 \leq N/2 \Rightarrow \lceil n/2 \rceil \leq N/2$
$\Rightarrow 2\,\mathrm{Mul}(\lceil n/2 \rceil) \leq \mathrm{Mul}(N)$

iterate once

$$\leq \frac{3}{2}\mathrm{Mul}(N) + \frac{3}{2}\mathrm{Mul}(N/2) + \frac{3}{2}\lambda n + C(\lceil n/2 \rceil_2)$$

iterate k-1 times

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

bound geometric series

$$\leq 3\,\mathrm{Mul}(N) + 2\lambda n + C(\lceil n/2 \rceil_k)$$

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

$$= O(\mathrm{Mul(N)}) = O(\mathrm{Mul}(n))\,.$$

use 2nd hyp on Mul

Division is not harder than multiplication!

Euclidean Division of Polynomials

$$(A(X), B(X)) \mapsto (Q(X), R(X)) \text{ s.t. } \begin{cases} A(X) = B(X)Q(X) + R(X), \\ \deg R(X) < \deg B(X)\,. \end{cases}$$

$$\frac{A(X)}{B(X)} = Q(X) + \frac{R(X)}{B(X)}$$

$X \mapsto 1/T$
and
$\times\, T^{\deg A - \deg B}$

$$\frac{T^{\deg A} A(1/T)}{T^{\deg B} B(1/T)} = T^{\deg A - \deg B} Q(1/T) + \overbrace{\frac{T^{\deg A} R(1/T)}{T^{\deg B} B(1/T)}}^{O(T^{\deg A - \deg B + 1})}$$

The coefficients of Q come first

1. Compute $\tilde{A} = T^{\deg A} A(1/T), \tilde{B} = T^{\deg B} B(1/T)$ (for free)
2. Compute $\tilde{Q} = \tilde{A} \times \mathrm{Inverse}(\tilde{B} + O(T^{\deg A - \deg B + 1}))$
3. Recover $Q = T^{\deg A - \deg B} \tilde{Q}(1/T)$ (for free)
4. Deduce $R = A - BQ\,.$

A similar method is used for integers.

Complexity: If deg A=cn & deg B=n, division in O(Mul(n)).

Square Root: Heron in High Precision

$$a = 1 + a_1 X + \cdots + O(X^m) \mapsto \sqrt{a} = 1 + \frac{a_1}{2} X + \cdots + O(X^m)$$

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{a}{y_n}\right) \quad \text{satisfies} \quad y_{n+1} - \sqrt{a} = \frac{(y_n - \sqrt{a})^2}{2y_n}$$

Algorithm Heron(a,m):

1. If $m = 1$, return 1
2. $k := \lceil m/2 \rceil$
3. $y :=$ Heron(a, k)
4. Return $(y + a \times \mathrm{Invert}(y, m))/2$

Complexity:

$$C(m) \leq C(\lceil m/2 \rceil) + \lambda\,\mathrm{Mul}(m) + \mu m$$

$$\Rightarrow C(m) = O(\mathrm{Mul}(m))$$

Same reasoning as before.

Square root is not harder than multiplication!

A similar method is used for integers.

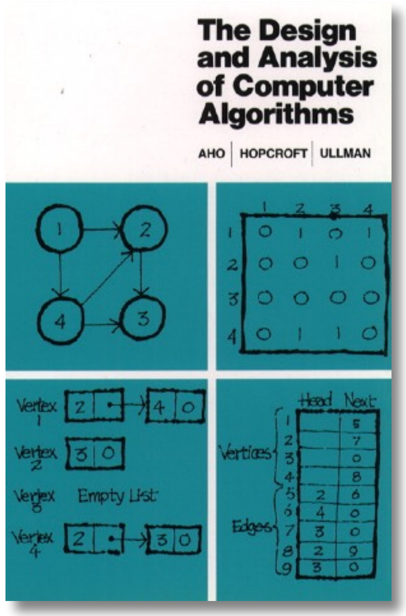
Optimisation: the inverse need not be recomputed at each iteration.

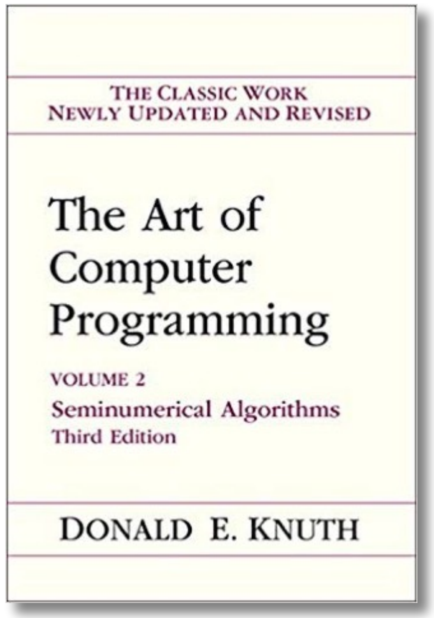
Many other efficient operations by the same method

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: Fast exponential and logarithm

Next tutorial: DAC for sequences and sums

Next week: The end of divide-and-conquer
"Master theorem"; advanced examples

Feedback

Moodle for the slides, TDs and exercises.

Questions or comments: Bruno.Salvy@inria.fr