

Computational Algebra

-

Transcript

Fabio Gratl

April 30, 2015

Contents

1	Integer Arithmetic	3
1.1	Addition and Multiplication	3
1.1.1	Algorithm 1 (Simple addition)	3
1.1.2	Definition 2 (Bit-Operation)	4
1.1.3	Definition 3 (Big O)	4
1.1.4	Theorem 4 (Lower bound for addition)	4
1.1.5	Algorithm 5 (Multiplication by "grid method")	5
1.1.6	Theorem 6 (Runtime of Algorithm 5)	5
1.1.7	Algorithm 7 (Karatsuba)	5
1.1.8	Theorem 8 (Runtime of Algorithm 7)	6
1.1.9	Definition 9 (Root of unity)	6
1.1.10	Algorithm 10 (Fast Fourier transformation FFT)	7
1.1.11	Theorem 11 (Runtime of Algorithm 10)	7
1.1.12	Definition 12 (Good root of unity)	8
1.1.13	Proposition 13 ($DFT_{\mu^{-1}}$)	8
1.1.14	Proposition 14 (Finding good roots of unity)	8
1.1.15	Algorithm 15 (Polynomial multiplication using DFT)	9
1.1.16	Theorem 16 (Runtime of Algorithm 15)	9
1.1.17	Proposition 17 (Add and mul in $O(l)$)	10
1.1.18	Proposition 18 (Sort of summary)	10
1.1.19	Algorithm 19 (Multiplication using FFT)	10
1.1.20	Theorem 20 (Runtime of Algorithm 19)	11
1.1.21	Theorem 21 (Schönhage-Strassen 1971)	12
1.2	Division with remainder, Euclidean algorithm	13
1.2.1	Algorithm 1 (Division with remainder)	13
1.2.2	Proposition 2 (Runtime of Algorithm 1)	13

1.2.3	Algorithm 3 (Euclidean algorithm)	14
1.2.4	Theorem 4 (Correctness of Algorithm 3)	15
1.2.5	Theorem 5 (Runtime of Algorithm 3)	15
1.2.6	Algorithm 6 (Extended Euclidean Algorithm)	15
1.3	Primality testing	16
1.3.1	Theorem 1	16
1.3.2	Algorithm 2 (Fermat Test)	17
1.3.3	Algorithm 3 (fast exponentiation)	17
1.3.4	Definition 4	18
1.3.5	Proposition 5:	18

2 Notes

19

1 Integer Arithmetic

Topics:

- Addition and Multiplication
- GCD computation
- Primality testing
- Factorization

1.1 Addition and Multiplication

Agreement:

- $a, x \in \mathbb{N}$ represented as $x = \sum_{i=0}^{n-1} a_i \cdot B^i$ $B \in \mathbb{N}_{>1}$ fixed Base
($a_i \in \{0, B-1\}$)
- if $x \neq 0$, assume $a_{n-1} \neq 0$ then define:
length of $x := l(x) = n$ = number of digits = $\lfloor \log_B(x) \rfloor + 1$
(mnemonic: $\log_B(B) + 1 = 2$)
- $l(0) = 1$
(Amount of memory required to store $x = 0$)
- $l(x) := l(|x|)$
- for $x \in \mathbb{Z}$ represent if as $x = \text{sgn}(x) * |x|$

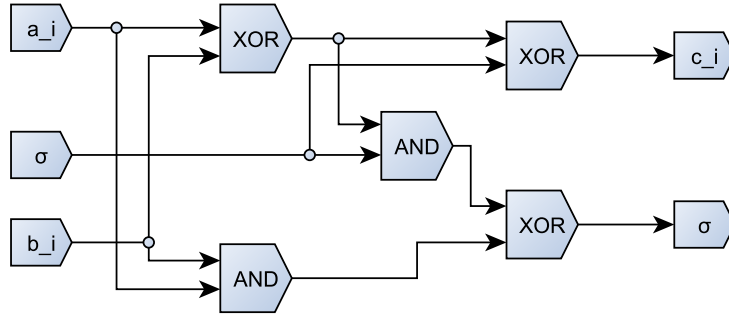
1.1.1 Algorithm 1 (Simple addition)

input : $x = \sum_{i=0}^{n-1} a_i \cdot B^i$, $y = \sum_{i=0}^{n-1} b_i \cdot B^i$, $x, y \in \mathbb{N}$

output: $x + y = \sum_{i=0}^n c_i \cdot B^i$

- (1) $\sigma = 0$
- (2) for $i = 0, \dots, (n-1)$:
- (3) set $c_i := a_i + b_i + \sigma_i$ and $\sigma := 0$
- (4) if $(c_i \geq B)$
- (5) set $c_i = c_i - B$
- (6) set $\sigma = 1$
- (7) set $c_n = \sigma$

If $B = 2$ then (3) - (6) can be realized by logic gates:



1.1.2 Definition 2 (Bit-Operation)

A bit operation is an operation that can be performed by a logic gate or by searching or writing a bit from / into memory.

1.1.3 Definition 3 (Big O)

Let M be a set (usually $M = \mathbb{N}$), $f, g : M \mapsto \mathbb{R}_{>0}$
 we write $f \in O(g)$ if $\exists c \in \mathbb{R} : f(x) \leq c \cdot g(x) \forall x \in M$

1.1.4 Theorem 4 (Lower bound for addition)

Let $f : \mathbb{N} \mapsto \mathbb{R}$, $n \mapsto$ maximal number of bit operations required by Algorithm 1 to add $x_y \in \mathbb{N}$ with $l(x), l(y) \leq n$

Let $g = id_{\mathbb{N}}$ Then $f \in O(g)$

We say Algorithm 1 requires $O(n)$ bit operations for adding two numbers of length $\leq n$.
 \Rightarrow "linear complexity"

Set $M := \{\text{Set of all algorithms for addition in } \mathbb{N}\}$

For $A \in M$ define $f_A : \mathbb{N} \mapsto \mathbb{R}$ as above.

We would like to find $f_{odd} : \mathbb{N} \mapsto \mathbb{R}$, $n \mapsto \inf\{f_A(n) | A \in M\}$

Since one needs to read x, y (and write the result) we can not do any better than linear complexity for addition.

Subtraction

let x, y as Algorithm 1, $x \geq y$

$$\text{For } \bar{y} := \sum_{i=0}^{n-1} (B-1-b_i)B^i \quad (\text{digitwise / bitwise complement})$$

$$\Rightarrow x + \bar{y} = x - y + B^n - 1$$

$$\Rightarrow x - y = x + \bar{y} + 1 - B^n \quad (\text{initially set } \sigma = 1)$$

Conclusion: Addition and Subtraction have cost $O(n)$

1.1.5 Algorithm 5 (Multiplication by "grid method")

input : $x = \sum_{i=0}^{n-1} a_i \cdot 2^i, \quad y = \sum_{i=0}^{m-1} b_i \cdot 2^i$

output: $z = x \cdot y$

- (1) $z := 0$
- (2) for $i = 0, \dots, (n-1)$
- (3) if $(a_i \neq 0)$ set $z := z + \sum_{j=0}^{m-1} b_j 2^{i+j}$

1.1.6 Theorem 6 (Runtime of Algorithm 5)

Algorithm 5 requires $O(n * m)$ bit operations.

As of the total input length $n + m$:

$$n \cdot m \leq \frac{1}{2}(n + m)^2 \rightarrow O((n + m)^2)$$

\Rightarrow Quadratic complexity

Karatsuba-multiplication:

Observation for polynomials:

$$a + bx, c + dx \text{ have } (a + bx)(c + dx) = ac + (ac + db - (a - b)(c - d))x + bdx^2$$

The point: only used 3 multiplications instead of 4.

Specialize $x = B$ "large" such that $x = a + bB$ partition into two blocks. Then multiply the blocks by a recursive call.

1.1.7 Algorithm 7 (Karatsuba)

input : $x, y \in \mathbb{N}$

output: $z = x \cdot y$

- (1) Choose $k \in \mathbb{N}$ minimal such that $l(x), l(y) \leq 2^k$.
Set $B = 2^{2^{k-1}}$
- (2) if $(k = 0)$ return $x \cdot y$ (by bit-operation AND)
- (3) write $x = x_0 + x_1 B, \quad y = y_0 + y_1 B$ with $l(x_i), l(y_i) \leq 2^{k-1}$
- (4) compute $x_0 \cdot y_0, \quad x_1 \cdot y_1, \quad (x_0 - x_1) \cdot (y_0 - y_1)$ by a recursive call
- (5) return $z = x_0 y_0 + (x_0 y_0 + x_1 y_1 - (x_0 - x_1)(y_0 - y_1))B + x_1 y_1 B^2$

1.1.8 Theorem 8 (Runtime of Algorithm 7)

For multiplying two numbers of length $\leq n$ Algorithm 7 requires $O(n^{\log_2 3}) \approx O(n^{1.59})$ bit operations.

Proof:

Set $\Theta(k) :=$ maximal numbers of bit operations for $l(x), l(y) \leq 2^k$

We have for $k > 0$: $\Theta(k) \leq 3 \underbrace{\Theta(k-1)}_{\text{recursive calls}} + c \underbrace{2^k}_{\text{additions}}$ with (c some constant)

Claim: $\Theta(k) \leq 3^k + 2c(3^k - 2^k)$

Proof by Induction on k :

$k = 0$: $\Theta(k) = 1$

$$\begin{aligned} k-1 \rightarrow k : \Theta(k) &= 3\Theta(k-1) + c2^{k-1} \\ &\leq 3(3^{k-1} + 2c(3^{k-1} - 2^{k-2})) + c2^{k-1} \\ &= 3^k + 2c(3^k - 2^k) \end{aligned}$$

So $\Theta(k) \leq (2c+1)3^k$

Now $l(x) \leq n$ hence $2^{k-1} < n$ by minimality of k

So $k-1 < \log_2 n$

$$\begin{aligned} \Rightarrow \Theta(k) &\leq 3(2c+1)3^{\log_2(n)} \\ &= 3(2c+1)2^{\log_2(3) \log_2(n)} \\ &= 3(2c+1)n^{\log_2(3)} \quad \square \end{aligned}$$

One can modify the terminal condition of Karatsuba to switch to Grid-Multiplication, which is faster for small numbers.

Fast-Fourier Transform

Reminder: For a function $f : \mathbb{R} \mapsto \mathbb{C}$ define:

$\hat{f} : \mathbb{R} \mapsto \mathbb{C}$ by

$$\hat{f}(\omega) = \int_{\mathbb{R}} f(t) e^{-i\omega t} dt \quad (\text{if it exists})$$

Think of ω as frequency.

Definition (Convolution)

Let $f, g : \mathbb{R} \mapsto \mathbb{C}$

$$(f * g)(x) = \int_{\mathbb{R}} f(t) g(x-t) dt$$

Convolution is analogous to polynomial multiplication **Formula:** $\underbrace{(f * g)}_{\text{(Cauchy formula)}} = \hat{f} \cdot \hat{g}$

For a function $M \mapsto C$ with $|M| < \infty$ we need the discrete Fourier transform (DFT)

1.1.9 Definition 9 (Root of unity)

Let R be a commutative ring with 1. An element $\mu \in R$ is called an n -th root of unity (= root of 1) if $\mu^n = 1$.

It is called primitive if $\mu^i \neq 1$ for $(0 < i < n)$ i.e. $\text{ord}(\mu) = n$

let μ be a primitive n -th root of 1 (e.g. $e^{2\pi \frac{i}{n}} \in \mathbb{C}$)

Then the map $DFT_{\mu} : R^n \mapsto R^n$

$$(\hat{a}_0, \dots, \hat{a}_n) \mapsto (\hat{a}_0, \dots, \hat{a}_n) \quad \text{with } \hat{a}_i = \sum_{j=0}^{n-1} \mu^{ij} a_j$$

is called discrete Fourier transformation

For polynomials:

$$DFT_\mu : R[x] \mapsto R^n$$

$$f \mapsto (f(\mu^0), \dots, f(\mu^{n-1}))$$

Convolution rule: (from $f(\mu^i)g(\mu^i) = (f * g)(\mu^i)$)

$$DFT_\mu(f * g) = DFT_\mu(f) \cdot DFT_\mu(g) \quad (\text{component wise product})$$

Addition of two polynomials in $R[x]$ of $\deg(n)$ require $O(n)$ ring operations. Multiplication require $O(n^2)$.

With Karatsuba have $O(n^{\log_2(3)})$ ring operations.

Cost $DFT_\mu(f) \cdot DFT_\mu(g) : O(n)$ ring operations (with μ as $2n$ -th root of 1)

Want: Cheap way of doing DFT and back-transformation.

1.1.10 Algorithm 10 (Fast Fourier transformation FFT)

input : $f \in R[x]$, $\mu \in R$ primitive 2^k -th root of 1, such that $\mu^{2^{k-1}} = -1$

output: $DFT_\mu(f)$

- (1) Write $f(x) = g(x^2) + xh(x^2)$ with $f, g, h \in R[x]$
- (2) if $k = 1$ ($\Rightarrow \mu = 1$) return $DFT_\mu(f) = (g(1) + h(1), g(1) - h(1))$
- (3) Recursive call: compute $DFT_{\mu^2}(g) = \hat{g}, DFT_{\mu^2}(h) = \hat{h} \in R^{2^{k-1}}$
- (4) return $DFT_\mu(f) = (\hat{f}_0, \dots, \hat{f}_{2^k-1})$ with $\hat{f}_i = \hat{g}_i + \mu \hat{h}_i$
where $\hat{g}_i = \hat{g}_{i-2^{k-1}}$ for $i \geq 2^{k-1}$

Note: Components of \hat{g} and \hat{h} are:

$$\hat{g} = g(\mu^{2^i}), \quad \hat{h}_i = h(\mu^{2^i}) \quad \text{so}$$

$$\hat{f}_i := f(\mu^i) = \hat{g}_i(\mu^{2^i}) + \mu h(\mu^{2^i}) = \hat{g}_i + \mu \hat{h}_i$$

Convention: $\lg(x) = \log_2(x)$

1.1.11 Theorem 11 (Runtime of Algorithm 10)

Let $n = 2^k$, $f \in R[x]$ with $\deg(\psi) < n$ Then Algorithm 10 requires $O(n \cdot \lg(n))$ ring operations.

Better than $O(n^{1+\epsilon})$, $\forall \epsilon > 0$!

Proof:

Set $\Theta(k) = \max$ number of ring operations required. By counting obtain for $k > 1$:

$$\Theta(k) \leq 2\Theta(k-1) + \underbrace{(\text{compute } \mu^i (i \leq 2^{k-1}))}_{2^{k-1}} + \underbrace{(\mu^i \hat{k}_i)}_{2^{k-1}} + \underbrace{(\text{sums and differences})}_{2^k}$$

$$= 2\Theta(k-1) + 2^{k+1}$$

Claim: $\Theta(k) \leq (2k-1)2^k$

$$k=1 : f = a_0 + a_1 \cdot x \quad DFT_\mu(f) = (a_0 + a_1 \cdot a_0 - a_1) \Rightarrow \Theta(a) = 2$$

$$k-1 \rightarrow k : \Theta(k) \leq 2 \cdot \Theta(k-1) + 2^{k+1} \leq 2 \cdot (2k-3) \cdot 2^{k-1} + 2^{k+1} = (2k-1) \cdot 2^k$$

since $k = \lg(n)$ obtain $O(k) \leq (2 \cdot \lg(n) - 1) \cdot n \in O(n \cdot \lg(n)) \quad \square$

Back-transformation?

1.1.12 Definition 12 (Good root of unity)

A primitive n -th root of unity is called good (caveat: this is ad-hoc terminology) if:

$$\sum_{j=0}^{n-1} \mu^{ij} = 0 \quad \text{for } (0 < i < n)$$

example:

(1) $\mu = e^{2\pi \frac{i}{n}}$ is a good primitive root of unity

(2) $R = \mathbb{Z}/(8), \quad \mu = \bar{3} \Rightarrow \mu \cdot B$ is primitive 2^{nd} root of unity
But $\bar{B}^0 + \bar{3}^1 = \bar{u} \neq \bar{0}$ so μ is not good.

1.1.13 Proposition 13 ($DFT_{\mu^{-1}}$)

let $\mu \in R$ be a good root of 1

$$(a) = (a_0, \dots, a_{n-1}) \in R^n \Rightarrow DFT_\mu^{-1}(DFT_\mu(a)) = n \cdot (a) \quad \text{where } n = 1 + \dots + 1 \in R$$

Proof:

$$DFT_\mu(a) = (\hat{a}) = (\hat{a}_0, \dots, \hat{a}_{n-1})$$

$$\text{with } \hat{a}_j = \sum_{k=0}^{n-1} \mu^{jk} a_k$$

$$DFT_{\mu^{-1}}(\hat{a}) = (\hat{\hat{a}}_0, \dots, \hat{\hat{a}}_1)$$

$$\text{with } \hat{\hat{a}}_i = \sum_{j=0}^{n-1} \mu^{-ij} \sum_{k=0}^{n-1} \mu^{jk} a_k = \sum_{k=0}^{n-1} (a_k \cdot \underbrace{\sum_{j=0}^{n-1} \mu^{j(k-i)}}_{=0 \text{ if } n \neq k-i \text{ (i.e. } k=i)}) = a_i \cdot n \quad \square$$

1.1.14 Proposition 14 (Finding good roots of unity)

let $\mu \in R, n \in \mathbb{N}$

Assume:

- R is an integral Domain and μ is a primitive or n -th root of 1
(Integral Domain: nonzero commutative ring in which the product of two nonzero elements is nonzero)
 \Rightarrow Granted by FFT
- $n = 2^b, \mu^{\frac{n}{2}} = -1$, then $h > 0 \vee \text{char}(R) \neq 2$
 $\rightarrow \mu$ is a good primitive n -th root of 1 ("root of unity")

Proof:

a) for $0 < i < n$

$$\underbrace{(\mu^i - 1)}_{\neq 0} \underbrace{\left(\sum_{j=0}^{n-1} \mu^{ij}\right)}_{=0} = \mu^{in} - 1 = 0$$

$\Rightarrow \mu$ is a good root of unity

* Let $0 < i < n$, write $i = 2^{k-s} \cdot r$ with r odd $\vee s > 0$

$$\sum_{j=0}^{2^k-1} \mu^{ij} = \sum_{l=0}^{2^{k-s}-1} \sum_{j=0}^{2^s-1} \mu^{i(l \cdot 2^s + j)}$$

$$\mu^{i \cdot 2^s} = 1$$

$$i \cdot 2^s = 2^{k-s} \sum_{j=0}^{2^s-1} \mu^{ij} = 2^{k-s} \sum_{j=0}^{2^{s-1}-1} (\mu^{ij} + \mu^{i(2^{s-1}+j)})$$

$$\text{But } \mu^{i \cdot 2^{s-1}} = \mu^{2^{k-s} \cdot r \cdot 2^{s-1}} = \mu^{2^{k-1} \cdot r} = (-1)^r = -1$$

$$\text{So } \sum_{j=0}^{n-1} \mu^{ij} = 0 \quad \square$$

b) $\mu^n = 1, n = 2^k \Rightarrow \text{ord}(\mu) | n \Rightarrow \text{ord}(\mu)$ is power of 2

1.1.15 Algorithm 15 (Polynomial multiplication using DFT)

input : $f, g \in R[x]$ with $\deg(f) + \deg(g) < 2^k =: n$
 $\mu \in R$ as a good root of unity; Assume 2 $\in R$ is invertible

output: $h = f \cdot g$

- (1) compute $\hat{f} = DFT_{\mu}(f)$, $\hat{g} = DFT_{\mu}(g)$ with $f, g \in R^n$
- (2) compute $\hat{h} = \hat{f} \cdot \hat{g}$
- (3) compute $(h_0, \dots, h_{n-1}) = DFT_{\mu^{-1}} \hat{h}$ (same as $DFT_{\mu}(\hat{h})$ but with different order)
 = Back-transformation $\cdot 2^k$
 set $h = \frac{1}{2^k} \sum_{i=0}^{n-1} h_i x^i$

1.1.16 Theorem 16 (Runtime of Algorithm 15)

Algorithm 15 uses $O(n \cdot \log(n))$ ring operations for polynomials of $\deg < n$

Proof:

- Choose k minimal so that $\deg(f) \cdot \deg(g) < 2^k$
 $\Rightarrow 2^{k-1} \leq 2n \Rightarrow k \leq \log(n) + 2$
- $\underbrace{O(2k \cdot 2^k)}_{\text{Step 1}} + \underbrace{2^k}_{\text{Step 2}} + \underbrace{O(k \cdot 2^k) + 2^k}_{\text{Step 3}} \in O(2k \cdot 2^k) = O(n \log(n)) \quad \square$

Goal: Multiplication in \mathbb{N} using DFT
 Idea: find roots of 1 in a suitable $\mathbb{Z}/(m)$
 Choose $m = 2^l + 1, \mu = \bar{2} \in R$

1.1.17 Proposition 17 (Add and mul in $O(l)$)

Let $m = 2^l + 1, R = \mathbb{Z}/(m)$

Addition in R and multiplication by $\bar{2}^i \in R$ ($0 \leq i < 2l$) can be done in $O(l)$ bit operations

Proof:

Let $\bar{x} \in R$ with $0 \leq x \leq 2^l$

- Addition: $x + \bar{y}$

(1) compute $x + y \in \mathbb{N}$: $O(l)$

(2) if $x + y > 2^l + 1$ subtract $2^l + 1$: $O(l)$

- Multiplication by $\bar{2}^i$ ($0 \leq i < l$)

(1) Bit-shift i Bits to the left by relocating in memory:

$$\underbrace{O(\text{length}(i))}_{\text{compute addr. of new first bit}} + \underbrace{l}_{\text{copying}} = O(\log(l)) + l \in O(l)$$

- Multiplication by $\bar{2}^i$ ($l \leq i < 2l - 1$)

(1) Multiplication by $\bar{2}^{i-l}$: $O(l)$

(2) take negative $\bar{2}^i \cdot \bar{x} = -\bar{2}^{i-l} \cdot \bar{x}$: $O(l)$

1.1.18 Proposition 18 (Sort of summary)

Let $k, r \in \mathbb{N}, r > 0, m = 2^{2^k \cdot r} + 1, R = \mathbb{Z}/(m), \mu = \bar{2}^r \in R$
 $\Rightarrow 2 \in R$ is invertible, μ is a good primitive 2^{k+1} -th root of 1
 $\Rightarrow \mu^{2^k} = 1$

Proof: \rightarrow from above

1.1.19 Algorithm 19 (Multiplication using FFT)

input : $x, y \in \mathbb{N}$

output: $Z = x \cdot y$

(1) Choose $k \in \mathbb{N}$ minimal such that $l(x), l(y) \leq 2^{2^k}$

(2) if $k \leq 3$, compute $z = x \cdot y$ by Algorithm 5

(3) set $B = 2^{2^k}, m = 2^{2^k \cdot 4} + 1, R = \mathbb{Z}/(m), \mu = \bar{2}^4 \in R$
 (\Rightarrow so μ is a good primitive 2^{k+1} -th root of 1)

- (4) write $x = \sum_{i=0}^{2^k-1} x_i \cdot B^i$, same for y with $(0 \leq x_i, y_i < B)$
possible since $x, y < 2^{2^k} = 2^{2^k \cdot 2^k} = B^{2^k}$
- (5) compute: $\hat{x} = DFT_\mu(\bar{x}_0, \dots, \hat{x}_{2^k-1}, \underbrace{0, \dots, 0}_{2^k \text{ zeros}}) \in R^{2^{k+1}}$
same for y
 \rightarrow use FFT
- (6) compute: $\hat{z} = \hat{x} \cdot \hat{b} \in R^{2^{k+1}}$ (component wise multiplication)
Perform multiplication in R as follows:
Multiply representatives (non negative and $< m$) by recursive call.
Then reduce modulo m by "negative bit shift" (see proof of Proposition 17)
- (7) compute: $(\bar{z}_0, \dots, \bar{z}_{2^{k+1}-1}) = \frac{1}{2^{k+1}} DFT_{\mu^{-1}}(\hat{z}) \in R$ with $0 \leq z < m$
- (8) set $z := \sum_{j=0}^{2^{k+1}-1} z_j \cdot B^j$

1.1.20 Theorem 20 (Runtime of Algorithm 19)

Algorithm 19 correctly computes $t = x \cdot y$ and requires $O(n \cdot (\log n)^4)$ bit operations for $l(x), l(y) \leq n$

Proof: Correctness

write $x(t) = \sum_{i=0}^{2^k-1} x_i t^i \in \mathbb{Z}[t]$, $y(t) = \sum_{j=0}^{2^k-1} y_j t^j \in \mathbb{Z}[t]$, $\bar{x}(t) \in R[t], \bar{y}(t) \in R[t], \bar{z}(t) \in R[t]$

by Proposition 18 and Proposition 13 we have $\bar{z}(t) = \bar{x}(t) \cdot \bar{y}(t)$

The l -th coefficient of $x(t) \cdot y(t)$ is $0 \leq \sum_{i+j=l} x_i \cdot y_j < 2^k \cdot B^2 = 2^{k+2 \cdot 2^k} \leq 2^{2^{k+2}} < m$

So $z(t) = x(t) \cdot y(t) \Rightarrow z = z(B) = x(B) \cdot y(B) = x \cdot y$ Cost:

Write $\Theta(k) := \max$ number of bit operations

Analyze Steps:

- (1) compute $\max \{l(x), l(y)\} : O(l(n)) = O(k)$
- (2) $O(1)$
- (3) no bit operations
- (4) compute starting addresses of x_i, y_i in memory: $2 \cdot 2^k$ increments of the address:
 $2 \cdot 2 \cdot 2^k = 2^{k+2}$ bit ops
 $\Rightarrow O(2^k)$
- (5) By Theorem 11 need $O(2 \cdot 2^{k+1} \cdot (k+1))$ operations in R which are additions and multiplications by powers of \bar{z} costing $O(2^{k+2})$ bit operations.
Total for (5): $O(k \cdot 2^{2 \cdot k})$

(6) 2^{k+1} multiplications of numbers $< m$, i.e. of length $\leq 2^{k+2}$.

So $k' \leq \frac{k+3}{2}$ for k' : the "new" k used in the next recursion level.

For $\alpha \in \mathbb{R}_{>0}$ define $\Theta(\alpha) := \Theta(\lfloor \alpha \rfloor)$

Total for (6): $2^{k+1}(\Theta(\frac{k+3}{2}) + \underbrace{O(2^{k+2})}_{\text{reduction mod } m})$

(7) For $DFT_{\mu^{-1}}(\hat{z}) : O(k \cdot 2^{2 \cdot k})$ Since \bar{z} is a n root of 1, multiplication by $\bar{2}^{-k-1}$ is multiplication by a positive power of $\bar{2}$, which costs $O(2^{k+2})$

Total for (7): $O(k \cdot 2^{2 \cdot k})$

(8) For $j \leq 2^{k+1}$ have $\sum_{i=0}^{j-1} z_i \cdot B^i \leq (m-1) \sum_{i=0}^{j-1} B^i = (m-1) \frac{B^j-1}{B-1} < 2(m-1) \frac{B^j}{B} =$

$2^{1+2^{k+2}+(j-1)2^k}$ so the sum has length $(j+3) \cdot 2 + 1$

Adding $z_j \cdot B^j$ to this sum happens at $(j \cdot 2^k)$ -th bit and higher \Rightarrow cost is $O(2^k)$

Total for (8): $O(2^{2 \cdot k})$

Grad total: For $k \geq 4$:

$\Theta(k) \leq 2^{k+1} \cdot \Theta(\frac{k+3}{2}) + c \cdot k \cdot 2^{2 \cdot k}$ with c constant

Also for $k \in \mathbb{R}_{\geq 4}$

Define $\Lambda(k) := \frac{\Theta(k)}{2^{2 \cdot k}} \Rightarrow \Lambda(k) \leq \frac{2^{k+1} \Theta(\frac{k+3}{2})}{2^{2 \cdot k}} + c \cdot k = 16 \cdot \Lambda(\frac{k+3}{2}) + c \cdot k$

Define $\Omega(k) := \Lambda(k+3)$ So for $k \in \mathbb{R}_{\geq 1}$

$\Omega(k) \leq 16 \cdot \Lambda(\frac{k}{2} + 3) + c \cdot (k+3) = 16 \underbrace{\Omega(\frac{k}{2})}_* + c \cdot (k+3)$

Claim: For $i \in \mathbb{N}$ with $2^{i-1} \leq k-3$ have:

$\Lambda(k) \leq 16^i \Omega(\frac{k-3}{2^i}) + c \cdot (k+3)(1+8+\dots+8^{i-1}) + 3 \cdot c \cdot (1+16+\dots+16^{i-1})$

Proof by induction:

$i = 0: \Lambda(k) = \Omega(k-3)$

$i \rightarrow i+1: \Lambda(k) \leq 16^i \Omega(\frac{k-3}{2^i}) + c \cdot (k-3)(1+\dots+8^{i-1}) + 3 \cdot c \cdot (1+\dots+16^{i-1}) \leq 2^i \leq k-3$ *

$\leq 16^i (16 \Omega(\frac{k-3}{2^{i+1}})) + c(\frac{k-1}{2^i} + 3) + c(k-3)\dots = \text{claimed result}$

Take $u \in \mathbb{N}$ minimal with $2^u > k-3 \Rightarrow \Omega(\frac{k-3}{2^u}) \leq \Omega(\lfloor \frac{k-3}{2^u} \rfloor) = \Omega(0) =: D$ (constant)

Note: u roughly is recursion depth

Have $2^{u-1} \leq k-3 \xrightarrow{\text{claim}} \Lambda(k) \leq 16^u \cdot D + c \cdot \underbrace{(k-3)}_{< 2^u} \cdot \frac{8^u-1}{7} + 3c \cdot \frac{16^u-1}{15} \in O(16^u)$

Have $2^{u-1} \leq k-3 \Rightarrow u \leq \lg(k-3) + 1$

$\Rightarrow \Lambda(k) \in O(16^{\lg(k-3)}) = O((k-3)^4)$

$\Rightarrow \Theta(k) = 2^{2 \cdot k} \cdot \Lambda(k) \in O(2^{2k} \cdot (k-3)^4)$

Have $2^{2(k-1)} < \underbrace{n}_{\max\{l(x) \cdot l(y)\}} \Rightarrow k \leq \frac{\lg(n)}{2} + 1$

So $\Theta(k) \in O(n \cdot (\lg(n))^4)$ \square

1.1.21 Theorem 21 (Schönhage-Strassen 1971)

Multiplication of integers of length $\leq n$ can be done in $O(n \cdot \lg(n) \cdot \lg(\lg(n)))$ bit operations. Schönhage-Strassen is used for integers of length ≥ 100.000 .

Asymptotically faster: Fürer's algorithm.

Comments on Bit complexity

1. Memory requirement may explode!
 \Rightarrow No Problem as bit complexity is upper bound for memory requirements, since memory access is included in bit operations
(\rightarrow only store what is calculated)
2. Computation of addresses in memory take time
 \Rightarrow length of addresses $\approx \lg(\text{memory space})$ computations of addresses $\approx \lg(\text{memory space})^2$
3. As memory requirement gets larger access times will get longer.
 \Rightarrow transportation time for data $\geq \frac{\text{diameter of physical storage}}{2 \cdot \text{speed of light}}$

1.2 Division with remainder, Euclidean algorithm

1.2.1 Algorithm 1 (Division with remainder)

input : $b = \sum_{i=0}^{n-1} b_i 2^i \quad \sum_{i=0}^{n+m-1} a_i 2^i$ with $a_i, b_i \in 0, 1, \quad b_{n-1} = 1$

output: $r, q \in \mathbb{N}$ such that $a = q \cdot b + r, \quad 0 \leq r < b$

(1) $r = a; \quad q = 0$

(2) for $i = m, m-1, \dots, 0$ do

(3) if $r \leq 2^i \cdot b$ then set $r := r - 2^i \cdot b, \quad q = q + 2^i$

1.2.2 Proposition 2 (Runtime of Algorithm 1)

Algorithm 1 is correct and requires $O(n \cdot (m+1))$ bit operations.

Proof:

Always have $a = q \cdot b + r$

Claim:

before step (3), have $0 \leq 2^{i+1} \cdot b$

$i = m; \quad 0 \leq r = a < 2^{m+n} = 2^{m+1} \cdot 2^{n-1} \leq 2^{m-1} \cdot b \quad i < m$ By step (3)

So after last passage through the loop $0 \leq r < b$

Running Time: In step(3), have comparison and (possibly) subtraction. Only n bits involved $\Rightarrow O(n)$

Total: $O(b \cdot (m+1))$

Remarks:

- (1) Division with remainder can be reduced to multiplication.

Precisely: given an algorithm for multiplication that requires $M(n)$ bit operations, there exists an algorithm for division with remainder that requires $O(M(n))$ bit operations.

- (2) Practically relevant:
Jebelean's algorithm (1997): $O(n^{\lg 3})$
- (3) Alternatively, may choose $r\mathbb{Z}$ such that $\lfloor \frac{-b}{2} \rfloor < r \leq \lfloor \frac{b}{2} \rfloor$
- (4) Algorithm 1 extends to \mathbb{Z} .
- (5) All Euclidean rings have division with remainder (by definition).
(e.g., $R = K[x] \rightarrow$ polynomial ring over field,
 $R = \mathbb{Z}[i] = \{a + bi | a, b \in \mathbb{Z}\} \subseteq \mathbb{C}, \quad i^2 = -1$)

1.2.3 Algorithm 3 (Euclidean algorithm)

input : $a, b \in \mathbb{N}$

output: $\gcd(a, b)$ "greatest common divisor"

- (1) set $r_0 := a, \quad r_i := b$
- (2) for $i = 1, 2, 3, \dots$ perform steps (3) and (4)
- (3) if $r_i = 0$ then $\gcd(a, b) = |r_{i-1}|$
- (4) Division with remainder: $r_{i-1} = q \cdot r_i + r_{i+1} \quad r_{i+1} \in \mathbb{Z}$
 $|r_{i+1}| \leq \frac{1}{2}|r_i|$

Example:

$$a = 287, \quad b = 126$$

$$287 = 2 \cdot 126 + 35 \tag{1}$$

$$126 = 4 \cdot 35 - 14 \tag{2}$$

$$35 = (-2) \cdot (-14) + 7 \tag{3}$$

$$-14 = (-2) \cdot 7 + 0 \tag{4}$$

$$\begin{aligned} \text{So: } 7 | (-14) &\xRightarrow{(3)} 7 | 35 \\ &\xRightarrow{(2)} 7 | 126 \\ &\xRightarrow{(1)} 7 | 287 \end{aligned}$$

On the other hand take a common divisor d ; $d | 287$; $d | 126$
 $\xRightarrow{(1)} d | d \xRightarrow{(2)} d | 14 \xRightarrow{(3)} d | 7$

1.2.4 Theorem 4 (Correctness of Algorithm 3)

Algorithm 3 is correct.

Proof:

Since $r_{i-1} = q \cdot r_i + r_{i+1}$ every integer $x \in \mathbb{Z}$ satisfies the equivalence $x|r_{i-1}$ and $x|r_i \Leftrightarrow x|r_{i+1}$ and $x|r_i$ so $\gcd(r_{i-1}, r_i) = \gcd(r_i, r_{i+1}) = \gcd(a, b)$ when terminating have $\gcd(a, b) = \gcd(r_{i-1}, 0) = |r_{i-1}| \quad \square$

1.2.5 Theorem 5 (Runtime of Algorithm 3)

Algorithm 3 requires $O(m \cdot n)$ bit operations for $n = l(a), m = l(b)$

Proof:

If $a < b$ then the first passage yields $r_2 = a, r_1 = b$. Cost: $O(n)$

May assume: $a \geq b$. Write $n_i = l(r_i)$

By Proposition 2 $\exists c$ constant such that the total time is $\leq c \cdot \underbrace{\sum_{i=1}^k n_i \cdot (n_{i-1} - n_i + 1)}_{=: \sigma(n_0, \dots, n_k)}$

For $i > 2$: $n_i = n_{i-1} - 1$

Special Case: $n_i = n_{i-1} - 1$ for $i \geq 2$

$\Rightarrow n_i = n_i - i + 1, \quad n_i = m, \quad k = m + 1$

Obtain $\sigma(n_0, \dots, n_k) = m \cdot (n - m + 1) + \sum_{i=2}^{m+1} (m - i + 1) \cdot 2 = m \cdot n - m^2 + m + m(m - 1) = m \cdot n$.

Claim: The special case is the worst (most expensive)!

From any sequence $n_1 > n_2 > \dots > n_k$ get to the special case by iteratively inserting numbers in the gaps. Insert s with $n_{j-1} > s > n_j$.

$\sigma(n_0, \dots, n_{j-1}, s, n_j, \dots, n_k) - \sigma(n_0, \dots, n_k) = \dots = s + (n_{j-1} - s) \cdot (s - n_j)$

$sp\sigma(n_0, \dots, n_k) \leq \sigma(n, m, m - 1, \dots, 2, 1, 0) = n \cdot m \quad \square$

Complexity is quadratic \rightarrow cheap

1.2.6 Algorithm 6 (Extended Euclidean Algorithm)

input : $a, b \in \mathbb{N}$

output: $d = \gcd(a, b)$ and $s, t \in \mathbb{Z}$ such that $d = s \cdot a + t \cdot b$

(1) $r_0 := a, r_1 := b, s_0 := 1, t_0 := 0, s_1 := 0, t_1 := 1$

(2) for $i = 1, 2, \dots$ perform steps (3) - (5)

(3) if $r_i = 0$ set $d = |r_{i-1}|$
 $s := \text{sgn}(r_{i-1}) \cdot s_{i-1}, t := \text{sgn}(r_{i-1}) \cdot t_{i-1}$

(4) division with remainder: $r_{i-1} = q_i \cdot r_i + r_{i+1}$, with $|r_{i+1}| \leq \frac{1}{2}|r_i|$

(5) set $s_{i+1} := s_{i-1} - q_i \cdot s_i$,
 $t_{i+1} := t_{i-1} - q_i \cdot t_i$

Justification : $r_i = s_i \cdot a + t_i \cdot b$ throughout

Application: $m \in \mathbb{N}$, $x \in \mathbb{N}$ such that m, x coprime (i.e. $\gcd(x, m) = 1$)

Algorithm 6 yields: $1 = s \cdot x + t \cdot m \Rightarrow s \cdot x \equiv 1 \pmod{m}$. So obtain inverse of $\bar{x} \in \mathbb{Z}/(m)$

1.3 Primality testing

Let $\mathbb{P} \subseteq \mathbb{N}$ be the set of prime numbers.

Challenge: Given $n \in \mathbb{N}$ decide if $n \in \mathbb{P}$

Naive Method: Trivial division by $m \leq \lfloor \sqrt{n} \rfloor$.

Running time is exponential in $l(n)$. Even when restricted to division by prime numbers,

need approximately $\frac{\sqrt{n}}{|\mathbb{P}| \sqrt{n}}$ trivial divisions (prime number theorem)

\rightarrow hardly any better!

Reminder: (arithmetic modulo m)

G finite group $\Rightarrow \forall a \in G \quad a^{|G|} = 1$ Fermat's little theorem

For $G = (\mathbb{Z}/(p))^x \quad a^{p-1} \equiv 1 \pmod{p} \in \mathbb{P} \quad \forall a \in \mathbb{Z} \quad$ with pfa ???

Infact $(\mathbb{Z}/(p))^x \cong Z_{p-1}$ is cyclic

For $m = p_1^{e_1} \dots p_r^{e_r}$ with $p_i \in \mathbb{P}, e_i \in \mathbb{N}_{>0}$:

$\mathbb{Z}_{(m)} \cong \mathbb{Z}_{(p_1^{e_1})} \oplus \dots \oplus \mathbb{Z}_{(p_r^{e_r})} \Rightarrow \mathbb{Z}_{(m)}^x \cong \mathbb{Z}_{(p_1^{e_1})}^x \times \dots \times \mathbb{Z}_{(p_r^{e_r})}^x$

what is $\mathbb{Z}_{(p^e)}$ for $p \in \mathbb{P}, e \in \mathbb{N}_{>0}$?

1.3.1 Theorem 1

Let $p \in \mathbb{P}$ of $f \quad e \in \mathbb{N}_{>0} \Rightarrow (\mathbb{Z}_{(p^e)})^x = Z_{(p-1) \cdot p^{e-1}}$ cyclic

Proof:

$(\mathbb{Z}_{(p^e)})^x \cong Z_{p-1} \Rightarrow \exists z \in \mathbb{Z} : \text{order}(z + p\mathbb{Z}) = p - 1$

Set $a = \bar{z}^{p^{e-1}} \in (\mathbb{Z}_{(p^e)})^x =: G$

$$a^{p-1} = \bar{z}^{(p-1) \cdot p^{e-1}} = \bar{z}^{|a|} = 1$$

On the other hand, take $i \in \mathbb{Z}$ such that

$$a^i = 1 \Rightarrow \bar{z}^{i \cdot p^{e-1}} \equiv 1 \pmod{p} \Rightarrow (p-1) | (i - p^{e-1}) \Rightarrow (p-1) | i.$$

So $\text{ord}(a) = p - 1$.

Now consider $b = (p+1) \in G$

Claim: $\text{ord}(b) = p^{e-1}$

Proof by induction on $k \in \mathbb{N}_{>0}$ that $(p+1)^{p^{k-1}} \equiv p^k + 1 \pmod{p^{k+1}}$

$k = 1 \quad \checkmark$

$k \rightarrow k+1$: By induction have $(p+1)^{p^{k-1}} = 1 + p^k + x \cdot p^{k+1}, \quad x \in \mathbb{Z}$ Compute:

$$(p+1)^{p^k} = ((1+p^k) + x \cdot p^{k+1})^p = \sum_{i=0}^p \binom{p}{i} (p \text{ over } i) (i+p^k)^{p-i} \cdot x^i \cdot p^{i \cdot (k+1)}$$

$$\underbrace{\equiv}_{\text{Only 0-th summand}} (i+p^k) = \sum_{i=0}^p \binom{p}{i} (p \text{ over } i) p^{i \cdot k} \underbrace{\equiv}_{p \text{ odd}} 1 + p^{k+1} \pmod{p^{k+2}} \quad \checkmark$$

For $k = e : (p+1)^{p^{e-1}} \equiv 1 \pmod{p^e} \Rightarrow b^{p^e} = 1 \Rightarrow \text{ord}(b) | p^{e-1}$

But $(p+1)^{p^{e-2}} \equiv p^{e-1} + 1 \pmod{p^e} \Rightarrow b^{p^{e-2}} \neq 1 \in G$

So $\text{ord}(b) = p^{e-1}$

Claim: $\text{ord}(a \cdot b) = (p-1)p^{e-1}$ (\Rightarrow Theorem)

Let $(a \cdot b)^i = 1 \in G$ with $i \in \mathbb{Z}$

Then $1 = (a \cdot b)^{i \cdot (p-1)} = (a^{p-1})^i \cdot b^{i \cdot (p-1)} = b^{i \cdot (p-1)} \Rightarrow p^{e-1} | i \cdot i(p-1) \Rightarrow p^{e-1} | i$

Also $1 = (a \cdot b)^{p^{e-1} \cdot i} = a^{p^{e-1}} \Rightarrow (p-1) | p^{e-1} \cdot i \Rightarrow (p-1) | i \rightarrow (p-1) \cdot p^{e-1} | i$ \square

Reminder: $(\mathbb{Z}/(2^e))^x \cong Z_2 \times Z_2^{e-2}$ ($e \geq 2$)

1.3.2 Algorithm 2 (Fermat Test)

input : $n \in \mathbb{N}_{>0} \text{ odd}$

output: " $n \notin \mathbb{P}$ " or "probably $n \in \mathbb{P}$ "

- (1) Choose $a \in 2, \dots, n-1$ randomly
- (2) Compute $a^{n-1} \bmod n$
- (3) If $a^{n-1} \neq 1 \bmod n$ then return " $n \notin \mathbb{P}$ "
otherwise return "probably $n \in \mathbb{P}$ "

Not very satisfying. Is this fast

1.3.3 Algorithm 3 (fast exponentiation)

input : $a \in G$ G is a monoid, $e \in \mathbb{N}$, $e = \sum_{i=0}^{n-1} e_i 2^i$, $e_i \in \{0, 1\}$

output: $a^e \in G$

- (1) Set $b := a$, $y := 1$
- (2) For $i = 0, \dots, n-1$ perform (3) - (4)
- (3) if $e_i = 1$ set $y := y \cdot b$
- (4) set $b := b^2$
- (5) return y

this requires $O(l(e))$ operations in G

For $G = (\mathbb{Z}/(n))_i$, each multiplication requires $O(l(n)^2)$ bit operations

\Rightarrow Fermat test requires $O(l(n)^3)$ bit operations \rightarrow cubic complexity \rightarrow "fast"!

Example:

$n = 561 = 3 \cdot 11 \cdot 17$ For $a \in \mathbb{Z}$ with $\gcd(a, n) = 1 \Rightarrow$ have $a^{n-1} = (a^2)^{280} \equiv 1 \bmod 3$

$a^{n-1} \equiv 1 \bmod n$ Fermat's test says "probably $n \in \mathbb{P}$ " in 57% of cases.

$n = 2207 \cdot 6619 \cdot 15443$: output "probably $n \in \mathbb{P}$ " in 99,93% of cases.

1.3.4 Definition 4

Let $n \in N_{>1} \text{ odd}$, $a \in 1, \dots, n-1$

- (a) n is pseudo-prime to base a if $a^{n-1} \equiv 1 \pmod{n}$
- (b) otherwise a is called a witness of composite of n
- (c) If $n \notin \mathbb{P}$ but $a^{n-1} \equiv 1 \pmod{n} \quad \forall a$ with $\gcd(n, a) = 1$
then n is called a Carmichael number

There are ∞ Carmichael numbers

1.3.5 Proposition 5:

Let $n \in N_{>1}$, $n \notin \mathbb{P}$ not Carmichael

$\Rightarrow |\{a \in \mathbb{Z} | 0 < a < n \text{ a witness of composite of } n\}| > \frac{n-1}{2}$

Proof: Consider

$\phi : (\mathbb{Z}/(n))^{\times} \rightarrow G, \quad \bar{a} \mapsto \bar{a}^{n-1}$

group homomorphism. By assumption,

$|\text{im}(\phi)| > 1 \Rightarrow |\text{Ker}(\phi)| \leq \frac{|\bar{a}|}{2} < \frac{n-1}{2}$

$\Rightarrow |\{a \in \mathbb{Z} | 0 < a < n \text{ a witness of composite of } n\}| > \frac{n-1}{2} \quad \square$

polynomial rings measure not abs value but max power

2 Notes

- $a|b$
 a is divisible by b
- $\text{ord}(a)$
- $\text{char}(A)$ the smallest positive n such that
 $\underbrace{1 + \dots + 1}_n = 0$ with 1 as the multiplicative identity element
 n summands
- $\mathbb{Z}/(m)$
Ring modulo m
- $\lg(x) = \log_2(x)$
- Average number of bit operations for an increment:
one operation for the last bit + 50% chance for one on the next bit + 25% on the following etc. \Rightarrow Geometrical row
 \Rightarrow on average two bit operations