

Computational Algebra

-

Transcript

Fabio Gratl

June 18, 2015

Contents

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

1.2.3	Algorithm 3 (Euclidean algorithm)	16
1.2.4	Theorem 4 (Correctness of Algorithm 3)	16
1.2.5	Theorem 5 (Runtime of Algorithm 3)	17
1.2.6	Algorithm 6 (Extended Euclidean Algorithm)	17
1.3	Primality testing	18
1.3.1	Theorem 1 (Cyclic group)	18
1.3.2	Algorithm 2 (Fermat Test)	19
1.3.3	Algorithm 3 (Fast exponentiation)	19
1.3.4	Definition 4 (Pseudo-prime, witness, Carmichael numbers)	20
1.3.5	Proposition 5 (Number of witnesses)	20
1.3.6	Proposition 6 (Inference from Fermat)	20
1.3.7	Algorithm 7 (Miller-Rabin-test)	21
1.3.8	Definition 8 (strong pseudo-prime / witness)	21
1.3.9	Theorem 9 (Bit-complexity of Algorithm 7)	21
1.3.10	Theorem (Ankeny & Bach)	23
1.3.11	Proposition 10 (Modulo over ideals)	24
1.3.12	Algorithm 11 (Test for perfect power)	24
1.3.13	Algorithm 12 (AKS-test)	25
1.3.14	Lemma 13 (Least common multiple)	25
1.3.15	Lemma 14 (Property of r in Algorithm 12)	26
1.3.16	Theorem 15 (Bit-Complexity of Algorithm 12)	26
1.3.17	Lemma 16 (Rules for ideals)	27
1.3.18	Theorem 17 (Correctness of Algorithm 12)	27
1.3.19	Lemma 18 (Property of binomial coefficients)	28
1.4	Cryptology	29
1.4.1	Algorithm (RSA)	30
1.4.2	Algorithm 1 (Finding a divisor)	31
1.4.3	Proposition 2 (Complexity of Algorithm 1)	31
1.4.4	Diffie-Hellmann Key Exchange	32
1.4.5	Elliptic curve cryptography (ECC)	32
1.5	Factorization	33
1.5.1	Algorithm 1 (Sieve of Eratosthenes)	33
1.5.2	Proposition 2 (length of periods)	33
1.5.3	Algorithm 3 (Pollard's ρ - Algorithm)	34
1.5.4	Theorem 4 (Bit-complexity of Algorithm 3)	35
1.5.5	Algorithm 5 (Pollard's ρ - 1 method)	35
1.5.6	Algorithm 6 (Quadratic sieve, simplified version)	38
2	Systems of equations	41
2.6	Linear Algebra	41
2.6.1	Proposition 1 (Complexity of usual algorithms)	41
2.6.2	Algorithm 2 (Strassen-multiplication)	42
2.6.3	Theorem 3 (Running time of Algorithm 2)	43
2.6.4	Proposition 4 (Complexity of matrix inversion)	43

2.6.5	Algorithm 5 (Transforming a matrix)	44
2.6.6	Theorem 6 (Correctness and running time of Algorithm 5)	45
2.6.7	Theorem 7	46
3	Notes	48
3.1	Notation	48
3.2	Various stuff	48
3.3	Algebraic structures	49
3.4	Invertible elements	50

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 ($a_i \in \{0, \dots, B-1\}$)
- 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:

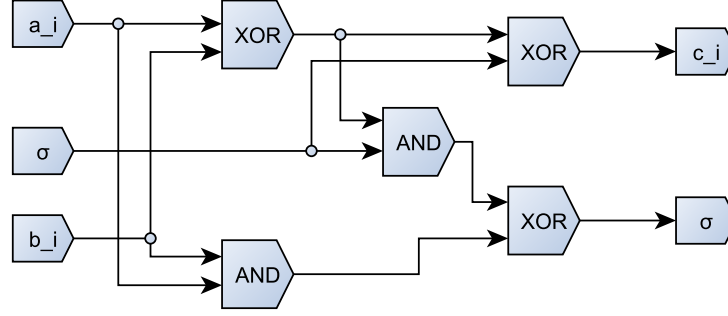


Figure 1: Logic circuit for addition

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, $x \geq y$

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

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

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

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

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

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 $O(n^l)$.

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:

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

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

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) $R = \mathbb{Z}/(8)$, $\mu = \bar{3} \Rightarrow \mu \cdot B$ 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:

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

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

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

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

1.1.14 Proposition 14 (Finding good roots of unity)

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

Assume:

- a) 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
- b) $n = 2^b, \mu^{\frac{n}{2}} = -1$, then $h > 0 \wedge \text{char}(R) \neq 2$
 $\rightarrow \mu$ 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 $\wedge s > 0$

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

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

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

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

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

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

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 $B = 2^{2^k}$, $m = 2^{2^k \cdot 4} + 1$, $R = \mathbb{Z}/(m)$, $\mu = \bar{2}^4 \in R$
 $(\Rightarrow \text{so } \mu \text{ is a good primitive } 2^{k+1}\text{-th root of 1})$
- (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 $x, y < 2^{2^{2k}} = 2^{2^k \cdot 2^k} = B^{2^k}$
- (5) compute: $\hat{x} = DFT_\mu(\bar{x}_0, \dots, \bar{x}_{2^k-1}, \underbrace{0, \dots, 0}_{2^k \text{ zeros}}) \in R^{2^{k+1}}$
 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 $x(t) \sum_{i=0}^{2^k-1} x_i t^i \in \mathbb{Z}[t]$, $y(t)$, $\bar{x}(t) \in R[t], \bar{y}(t), \bar{z}(t)$

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 $\alpha \in R_{>0}$ 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 $DFT_{\mu^{-1}}(\hat{z}) : O(k \cdot 2^{2 \cdot k})$ as (5) 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 $k \in \mathbb{R}_{>1}$

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

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:

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

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

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

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

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

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

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

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

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

$$\text{So } \Theta(k) \in O(n \cdot (\lg(n))^4) \quad \square$$

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

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

(1) $r = a$ $q = 0$

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

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

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

$i = m$; $0 \leq r = a < 2^{m+n} = 2^{m+1} \cdot 2^{n-1} \leq 2^{m-1} \cdot b$ $i < m$ 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 $r \in \mathbb{Z}$ 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,

$R = \mathbb{Z}[i] = \{a + bi \mid a, b \in \mathbb{Z}\} \subseteq \mathbb{C}, \quad i^2 = -1$)

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:
 $r_{i+1} = r_{i-1} - q_i \cdot r_i, \quad \text{with } |r_{i+1}| \leq \frac{1}{2}|r_i|$

(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: $m, x \in \mathbb{N}$ such that m, x co-prime (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 $\frac{\sqrt{n}}{|n|^{1/\sqrt{n}}}$ 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 $G = (\mathbb{Z}/(p))^\times \quad a^{p-1} \equiv 1 \pmod{p} \in \mathbb{P} \quad \forall a \in \mathbb{Z} \quad \text{with } p \nmid a$

In fact $(\mathbb{Z}/(p))^\times \cong Z_{p-1}$ is cyclic

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

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

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

1.3.1 Theorem 1 (Cyclic group)

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

Proof:

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

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

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

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

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

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 $(p+1)^{p^{k-1}} \equiv p^k + 1 \pmod{p^{k+1}}$

$k=1$ ✓

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

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

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

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

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

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

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

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

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

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 $a^{n-1} \not\equiv 1 \pmod{n}$ return " $n \notin \mathbb{P}$ "
else 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 $a^{n-1} = (a^2)^{280} \equiv 1 \pmod{3}$

$a^{n-1} \equiv 1 \pmod{n}$ 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 (Pseudo-prime, witness, Carmichael numbers)

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 composition 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 (Number of witnesses)

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

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

Proof: Consider

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

group homomorphism. By assumption,

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

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

Miller-Rabin Test

1.3.6 Proposition 6 (Inference from Fermat)

Let $p \in \mathbb{P} \text{ odd}$, $a \in \{1, \dots, (p-1)\}$ write $p-1 = 2^k \cdot m$ with $m \text{ odd}$ Then:

$a^m \equiv 1 \pmod{p}$ or $\exists i \in \{0, \dots, k-1\} : a^{2^i \cdot m} \equiv -1 \pmod{p}$

Proof:

Little Fermat: $\bar{a}^{2^k \cdot m} = 1 \in \mathbb{F}_p$

Assume $\bar{a}^m \neq 1$ take i maximal such that:

$\bar{b} = \bar{a}^{2^i \cdot m} \neq 1 \Rightarrow \bar{b}^2 = 1 \Rightarrow \bar{b} \in \mathbb{F}_p$ is a zero of $x^2 - 1 \in \mathbb{F}_p[x] \Rightarrow \bar{b} = -1$

1.3.7 Algorithm 7 (Miller-Rabin-test)

input : $n \in \mathbb{N}_{>1}, \text{odd}$

output: either " $n \notin \mathbb{P}$ " or "probably $n \in \mathbb{P}$ " \rightarrow Monte Carlo Algorithm.

- (1) write $n - 1 = 2^k \cdot m$ with m odd
- (2) Choose $a \in \{2, \dots, n - 1\}$ randomly
- (3) Compute $b := a^m \pmod n$
- (4) if $(b \equiv \pm 1 \pmod n)$
return "probably $n \in \mathbb{P}$ "
- (5) for $(i = 0, \dots, k - 1)$ do steps (6) - (7)
- (6) set $b := b^2 \pmod n$
- (7) if $(b \equiv -1 \pmod n)$
return "probably $n \in \mathbb{P}$ "
- (8) return $n \notin \mathbb{P}$

1.3.8 Definition 8 (strong pseudo-prime / witness)

Let $n \in \mathbb{N}_{>1}, \text{odd}$ $a \in \{1, \dots, n - 1\}$

- (a) n is called a strongly pseudo-prime to base a if Proposition 6 holds for a and p replaced by n .
- (b) Otherwise a is called a strong witness of composition of n .

Example

Let $n \in \mathbb{N}_{>1}, \mathbb{P} \text{ odd}$

$a = 2$ strong witness if $n < 2047$ (including 561)

2 or 3 strong witness if $n < 1373653$

2,3 or 5 strong witness if $n < 25326001$

1.3.9 Theorem 9 (Bit-complexity of Algorithm 7)

- (a) Algorithm 7 requires $O(l(n)^3)$ bit operations. \rightarrow "qubic complexity" \rightarrow fast!
- (b) if $b \in \mathbb{P}$ then Algorithm 7 returns "probably $b \in \mathbb{P}$ " \rightarrow no false positives.
- (c) if $n \notin \mathbb{P}$ then more than half of the numbers in $\{1, \dots, n - 1\}$ are strong witnesses.

Proof:

- (a) Step 1 takes $O(l(n))$ bit operations:
Using Algorithm 3, we need $O(l(n-1))$ multiplications in $\mathbb{Z}/(n)$ each requiring $O(l(n)^2)$ bit operations.
- (b) Proposition 6
- (c) split in three cases:

Case 1: n is not a Carmichael number. $\xRightarrow{\text{Prop 5}}$ more than half of all numbers are.

Fermat witness thus also strong witness.

Case 2: $n = p^r \cdot l$ with $p \in \mathbb{P}$ $r > 1$ $l \in \mathbb{N}_{>0}$ $p \nmid l$

Theorem 1 $\exists x \in \mathbb{Z}$ such that $x^p \equiv 1 \pmod{p^r}$ $x \not\equiv 1 \pmod{p^r}$

Chinese remainder theorem: $\exists a \in \mathbb{Z}$ such that $a \equiv x \pmod{p^r}$ $a \equiv 1 \pmod{l}$

So $\bar{a}^p = 1 \in \mathbb{Z}/(n) \Rightarrow \bar{a}^n = 1 \Rightarrow \bar{a} \in (\mathbb{Z}/(n))^\times$

i.e. $\gcd(n, a) = 1$ if $\bar{a}^{n-1} = 1$ then $\bar{a} = 1$

But $a \equiv x \not\equiv 1 \pmod{p^r}$ so $\bar{a}^{n-1} \neq 1$ hence n is not Carmichael \rightarrow Case 1.

Case 3: n is a Carmichael number. By Case 2 have $n = p \cdot l$ with $p \in \mathbb{P}$ $p \nmid l$ $l \geq 3$
 n Carmichael: $\forall a \in \mathbb{Z}$ with $\gcd(a, n) = 1$

have $a^{2^k \cdot m} \equiv 1 \pmod{n}$ (where $n-1 = 2^k \cdot m$)

$a^{2^k \cdot m} \equiv 1 \pmod{p}$ Take j minimal such that

$a^{2^j \cdot m} \equiv 1 \pmod{p}$ $\forall a \in \mathbb{Z}$ such that $\gcd(a, n) = 1$

so $0 \leq j \leq l$ in fact, $j > 0$ since $(-1)^{2^0 \cdot m} = -1$ with m odd.

Consider the subgroup $H := \{\bar{a} \in \mathbb{Z}/(n) \mid \bar{a}^{2^{j-1} \cdot m} \in \{1, -1\} \subseteq (\mathbb{Z}/(n))^\times\}$

Let $a \in \{1, \dots, n-1\}$ $\gcd(n, a) = 1$ a not a strong witness.

Claim 1: $\bar{a} \in H$

Case 3.1: $\bar{a}^{2^{j-1} \cdot m} = 1 \Rightarrow \bar{a} \in H$

Case 3.2: $a^{2^{j-1} \cdot m} \not\equiv 1 \pmod{n}$ $a^m \not\equiv 1 \pmod{n}$

$\xRightarrow{\text{a nonwitness}} \exists i$ such that $\underbrace{a^{2^i \cdot m} \equiv -1 \pmod{n}}_*$

$\Rightarrow a^{2^i \cdot m} \equiv -1 \pmod{p} \xRightarrow{\text{def of } j} i < j$

if $i < j-1$ then $a^{2^{j-1} \cdot m} = (a^{2^i \cdot m})^{2^{j-1-i}} \equiv (-1)^{2^{j-1-i}} = 1 \pmod{n}$

$\xRightarrow{\text{with } *} \text{not in case 3.2}$

Claim 2: $H \subseteq (\mathbb{Z}/(n))^\times$ proper subgroup.

By definition of j $\exists x \in \mathbb{Z}$ such that $x^{2^{j-1} \cdot m} \not\equiv 1 \pmod{p}$

Chinese remainder: $\exists a \in \mathbb{Z}$ such that

$a \equiv x \pmod{p}$ $a \equiv 1 \pmod{l}$

$\Rightarrow a^{2^{j-1} \cdot m} \not\equiv 1 \pmod{p} \equiv 1 \pmod{l} \Rightarrow \bar{a} \notin H$

Claim 2 \checkmark

It follows that $|H| \leq \frac{|(\mathbb{Z}/(n))^\times|}{2} < \frac{n-1}{2}$

so the number of witnesses is $\geq n-1-|H| > \frac{n-1}{2}$ \square

Remarks:

- (a) A more careful analysis shows that $2\frac{3}{4}$ of all candidates are strong witnesses
- (b) Calling Algorithm 7 repeatedly decreases the probability of false positives. Running time for prescribed error probability p is $O(\lg(p^{-1} \cdot l(n)^3))$
(Independence assumptions!)

Connection with Riemann hypothesis

Let $n \in \mathbb{N}_{>0}$ $\bar{X} : (\mathbb{Z}/(n))^\times \rightarrow \mathbb{C}^\times$ group homomorphism

$$X : \mathbb{Z} \rightarrow \mathbb{C}, a \mapsto \begin{cases} \bar{X}(\bar{a}) & \text{if } \gcd(a, n) = 1 \text{ for } (\bar{a} = a + n\mathbb{Z}) \\ 0 & \text{otherwise} \end{cases}$$

"residue class character (mod n)

$$Ex : n = 1 \Rightarrow X(a) = 1 \forall a \in \mathbb{Z}$$

Dirichlet L-series:

$$L_X(s) = \sum_{n=1}^{\infty} \frac{X(n)}{n^s} \text{ converges for } s \in \mathbb{C} \text{ until } Re(s) > 1$$

$L_X(s)$ extends to a meromorphic function on $\mathbb{C} \mapsto$ "Dirichlet L-function".

For $n = 1 : L_X(s) = \zeta(s)$ Riemann Zeta-function.

Euler Product:

$$\text{From } (1 - X(p) \cdot p^{-s})^{-1} = \sum_{i=0}^{\infty} (X(p) \cdot p^{-s})^i = \sum_{i=0}^{\infty} \frac{X(p^i)}{p^{is}} \quad \text{derive } L_X(s) = \prod_{p \in \mathbb{P}} \frac{1}{1 - X(p) \cdot p^{-s}}$$

Generalized Riemann hypothesis (GRH):

For X residue class character, $s \in \mathbb{C}$

with $L_X(s) = 0$, $0 < Re(s) < 1$ ("critical strip")

then $Re(s) = \frac{1}{2}$

For $X = 1 \rightarrow$ ordinary Riemann hypothesis.

1.3.10 Theorem (Ankeny & Bach)

GRH $\Rightarrow \forall X \neq 1$ residue class character

$$\exists p \in \mathbb{P} : X(p) \neq 1, p < 2 \ln(n)^2$$

Let $H \subsetneq (\mathbb{Z}/(n))^\times =: G$ proper subgroup.

Choose $N \subsetneq G$ maximal proper subgroup such that $H \subseteq N \Rightarrow G/N$ cyclic.

$$\bar{X} : G \mapsto \mathbb{C}^\times \text{ with } N = \text{Ker}(\bar{X}) \Rightarrow H \subseteq \text{Ker}(\bar{X})$$

$$\xrightarrow{\text{GRH, Thm1}} \exists p \in \mathbb{P} : p + n\mathbb{Z} \not\subseteq H, p < 2 \cdot \ln(n)^2$$

Corollary: Assume GRH.

Let $n \in \mathbb{N}_{>1}$ \mathbb{P} odd Then there is a strong witness a of compositeness of n with $a < 2 \cdot \ln(n)^2$.

\rightarrow Obtain deterministic primality test with time $O(\ln(n)^5)$ bit operations.

AKS-test

A deterministic polynomial time primality test \rightarrow "holy grail"

Agrawal, Kayal, Saxena: PRIMES is in P, Annals of Mathematics, 2004.

1.3.11 Proposition 10 (Modulo over ideals)

Let $n \in \mathbb{P}$ $a \in \mathbb{Z} \Rightarrow (x + a)^n \equiv x^n + a \pmod{n}$

where x is a indeterminate and for $r \in \mathbb{N}$:

$$(x + a)^n \equiv (x^n + a) \pmod{n, x^r - 1} \quad (1)$$

(i.e. $(x + a)^n - (x^n + a) = n \cdot f + (x^r - 1) \cdot g$ with $f, g \in \mathbb{Z}[x]$)

Proof:

$$(x + a)^n = \sum_{i=0}^n \binom{n}{i} \cdot a^{n-i} \cdot x^i \quad (\text{where } \binom{n}{i} \text{ is a multiple of } n \text{ for } 0 < i < n)$$

$$\equiv x^n + a^n \quad (\leftarrow \text{little Fermat})$$

$$\equiv x^n + a \quad (1) \text{ follows by weakening this.}$$

Cost analysis for checking (1) with $l = \text{length}(n)$.

Using Algorithm 3, need $O(l)$ multiplications in $\mathbb{Z}[x]/(n, x^r - 1) =: R$

Elements of R are represented as polynomials of degree $< r$,
coefficients between 0 and n .

Multiply polynomials: $O(r^2)$ operation in $\mathbb{Z}/(n) : O(r^2 \cdot l^2)$

since $x^{r+k} \equiv x^k \pmod{x^r - 1}$,

add coefficients of x^{r+k} of product polynomial to coefficients $x^k : O(r \cdot l)$

Total for checking (1): $O(r^2 \cdot l^3)$ bit operations.

Reduction $\pmod{x^r - 1}$ is just for keeping the cost under control.

The following is part of AKS-test:

1.3.12 Algorithm 11 (Test for perfect power)

input : $n \in \mathbb{N}_{>1}$

output: $m, e \in \mathbb{N}$ $e > 1$ such that $n = m^e$ or "n is not a perfect power"

(1) for $(e = 2, \dots, \lfloor \lg(n) \rfloor)$ perform (2) - (7) //possible exponents

(2) set $m_1 = 2, m_2 = n$ //initialize interval $[m_1, m_2]$ for searching $\sqrt[e]{n}$

(3) while($m_1 \leq m_2$) do (4) - (7)

(4) set $m = \lfloor \frac{m_1 + m_2}{2} \rfloor$ // bisect interval

(5) if $m^e = n$ return m, e

(6) if $m^e > n$ set $m_2 = m - 1$

(7) if $m^e < n$ set $m_1 = m + 1$

(8) return "not a perfect power"

Cost: (for $l = \text{length}(n)$)

Compute $m^e : O(\lg(l) \cdot l^2)$ (abort computation once the result exceeds n)

Number of passages through inner loops $\leq \lg(n)$

Number of passages through outer loops $\leq \lg(n)$

Total cost of Algorithm 11: $O(l^4 \cdot \lg(l))$

1.3.13 Algorithm 12 (AKS-test)

input : $n \in \mathbb{N}_{>1}$ of length $l = \text{length}(n) = \lfloor \lg(n) \rfloor + 1$

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

- (1) if (n is a perfect power)
return " $n \notin \mathbb{P}$ "
- (2) find $r \in \mathbb{N}_{>1}$ minimal such that $r|n \vee n^i \not\equiv 1 \pmod{r} \quad \forall i = 1, \dots, l^2$
//exhaustive search (we will show that $r \leq l^5$)
- (3) if $r|n$
if ($r = n$) return " $n \in \mathbb{P}$ "
if ($r < n$) return " $n \notin \mathbb{P}$ "
- (4) for $a = 1, 2, \dots, \lfloor \sqrt{r} \cdot l \rfloor$ do (5)
- (5) if $((x+a)^n \not\equiv (x^n + a) \pmod{(n, x^r - 1)})$
return " $n \notin \mathbb{P}$ "
- (6) return " $n \in \mathbb{P}$ "

1.3.14 Lemma 13 (Least common multiple)

For $n \in \mathbb{N}_{>0}$ have $\lambda(n) := \text{lcm}(1, 2, \dots, n) \geq 2^{n-2}$

Proof: For $f = \sum_{i=0}^m a \cdot x^i \in \mathbb{Z}(x) \quad a_i \in \mathbb{Z}$

$$\Rightarrow \int_0^1 f(x) dx = \sum_{i=0}^m \frac{a_i}{i+1} = \frac{k}{\lambda(m+1)}$$

with $k \in \mathbb{Z}$. Consider $f_m = x^m \cdot (1-x)^m$

For $0 < x < 1$:

$$0 < f_m(x) \leq 4^{-m}$$

$$\Rightarrow 0 < \int_0^1 \underbrace{f_m(x)}_{\frac{k_m}{\lambda(2m+1)}} dx \leq 4^{-1}$$

$$\lambda(2 \cdot m + 1) \geq k_m \cdot 4^m \geq 4^m$$

$$\text{For } n \in \mathbb{N}_{>0} \lambda(n) \geq \lambda(2 \cdot \lfloor \frac{n-1}{2} \rfloor + 1) \geq 4^{\lfloor \frac{n-1}{2} \rfloor} \geq 4^{\frac{n-1}{2}} = 2^{n-2} \quad \square$$

Corollary: (not related to AKS)

For $n \in \mathbb{N}$

$$\pi(n) := |\{p \in \mathbb{P} | p \leq n\}| \geq \frac{n-2}{\lg(n)}$$

Proof:

$$2^{n-2} \leq \lambda(n) = \prod_{p \in \mathbb{P}, p \leq n} p^{\lfloor \log_p(n) \rfloor} \leq \prod_{p \leq n} p^{\log_p(n)} = n^{\pi(n)} = 2^{\lg(n)\pi(n)} \quad \square$$

Prime number theorem:

$$\lim_{n \rightarrow \infty} \frac{\pi(n)}{n/\ln(n)} = 1$$

Interpretation:

The average distance of two primes around some value $x \in \mathbb{R}_{>1}$ is $\ln(x)$

1.3.15 Lemma 14 (Property of r in Algorithm 12)

In Algorithm 12, have $r \leq l^5$

Proof:

if $r < l^5 \Rightarrow \forall k \in \{2, \dots, l^5\} : \exists i \in \{1, \dots, l^2\}$

$$n^i \equiv 1 \pmod{k}$$

$$\Rightarrow k \mid \prod_{i=1}^{l^2} (n^i - 1)$$

$$\Rightarrow \lambda(l^5) \mid \prod_{i=1}^{l^2} (n^i - 1)$$

$$\xrightarrow[\text{Lemma 13}]{=} 2^{l^5-2} < \prod_{i=1}^{l^2} n^i = n^{\frac{l^2(l^2+1)}{2}}$$

$$\Rightarrow l^5 - l^3 < 4 \quad \text{not true since } l \geq 2 \quad \square$$

1.3.16 Theorem 15 (Bit-Complexity of Algorithm 12)

Algorithm 12 requires $O(l^{16.5})$ bit operations ("polynomial complexity")

Proof:

Step(1): $O(l^4 \cdot \lg(l)) \checkmark$

Step(2): For each r need:

- test $r \mid n : O(l^2)$
- compute all $n^i \pmod{r} : O(l^2 \cdot \lg(r)^2) \xrightarrow[\text{Lemma 14}]{} O(l^2 \cdot \lg(l)^2)$

Step(3): $O(1)$

$$\text{Step(4): } O(\sqrt{r} \cdot l \cdot r^2 \cdot l^3) \xrightarrow[\text{Lemma 14}]{} O(l^{16.5}) \quad \square$$

Reminder: There is a variant of Algorithm 12 with running time $\tilde{O}(l^6)$, i.e., $O(l^6 \cdot \lg(l)^m)$ with $m \in \mathbb{N}$.

Correctness:

For $r \in \mathbb{N}_{>0}$ and $p \in \mathbb{P}$ write $I(r, p) := \{m, f\} \in \mathbb{N} \times \mathbb{F}_p[x] \mid f(x)^m \equiv f(x^m) \pmod{x^r - 1}\}$
 "m is introspective for f and r ".

Example: Proposition 10 says that:

$$(p, x + \bar{a}) \in I(r, p) \text{ for } a \in \mathbb{Z} \quad r \in \mathbb{N}_{>0} \quad p \in \mathbb{P} \quad (1)$$

1.3.17 Lemma 16 (Rules for ideals)

- (a) $(m, f), (m', f) \in I(r, p) \Rightarrow (m \cdot m', f) \in I(r, p)$
- (b) $(m, f), (m, g) \in I(r, p) \Rightarrow (m, f \cdot g) \in I(r, p)$
- (c) $(m \cdot p, f) \in I(r, p), p \nmid r \Rightarrow (m, f) \in I(r, p)$

Proof:

- (a) $f(x)^{m \cdot m'} \equiv f(x^m)^{m'} \pmod{(x^r - 1)}$
 $f(x^m)^{m'} \equiv f(x^{m \cdot m'}) \pmod{(x^{m \cdot r} - 1)}$
 But $(x^r - 1) \mid (x^{m \cdot r} - 1)$
- (b) $(f \cdot g)(x)^m = f(x)^m \cdot g(x)^m \equiv f(x^m) \cdot g(x^m) = (f \cdot g)(x^m) \pmod{(x^r - 1)}$
- (c) $(f(x)^m)^p \equiv f((x^m)^p) \stackrel{\text{Frobenius homomorphism}}{\equiv} (f(x^m))^p \pmod{(x^r - 1)}$
 $\Rightarrow (x^r - 1) \mid ((f(x)^m)^p - f(x^m)^p) \stackrel{\text{Frobenius homomorphism}}{\equiv} (f(x)^m - f(x^m))^p$
 $p \nmid r \Rightarrow x^r - 1$ is square free. So
 $(x^r - 1) \mid (f(x)^m - f(x^m)) \Rightarrow (m, f) \in I(r, p) \quad \square$

1.3.18 Theorem 17 (Correctness of Algorithm 12)

Algorithm 12 is correct.

Proof:

If the algorithm terminates in step(1),(3) or (5), it is correct. To show: If it terminates in step(6) it is correct, i. e. $n \in \mathbb{P}$

Claim 1: $\exists p \in \mathbb{P} : p \mid n \quad p \not\equiv 1 \pmod{r} \quad p > r$

Indeed if all prime divisors of n were $\equiv 1 \pmod{r}$ then $n \equiv 1 \pmod{r}$

Contradiction to step(2). All prime divisors of n are $> r$ by step (2) and (3) ✓

Steps(2) and (3) imply that $\gcd(n, r) = 1 \Rightarrow G := \langle \bar{n}, \underbrace{\bar{p}}_{p \bmod r} \rangle \subseteq (\mathbb{Z}/(r))^\times$

Step(2): $\text{ord}(\bar{n}) > l^2 \Rightarrow l^2 < |G| < r$ (2)

Set $s := \text{ord}(\bar{p} \in G) \Rightarrow r \mid (p^s - 1)$ with $q := p^s \Rightarrow r \mid |\mathbb{F}_q^\times| \Rightarrow \exists \zeta \in \mathbb{F}_q$ r -th root of 1

Set $k := \lfloor \sqrt{r} \cdot l \rfloor \quad m := \left(\frac{n}{p}\right)$

By (1) $(p, x + \bar{a}) \in I(r, p)$ with $\bar{a} \in \mathbb{F}_p$

By step(4), have $(n, x + \bar{a}) \in I(r, p)$

For $\underline{e} = e_0, \dots, e_k \in \mathbb{N}_0$ set $f_{\underline{e}} := \prod_{a=0}^k (x + \bar{a})^{e_a}$

Lemma 16 (b): $(p, f_{\underline{e}}) \in I(r, p)$

$(n, f_{\underline{e}}) \in I(r, p)$

$\xRightarrow{\text{Lemma 16(c)}} (m, f_{\underline{e}}) \in I(r, p)$

$\xRightarrow{\text{Lemma 16(a)}} \forall s, t \in \mathbb{N}_0 : (p^s \cdot m^t, f_{\underline{e}}) \in I(r, p)$

$\Rightarrow f_{\underline{e}}(\zeta^{p^s \cdot m^t}) = f_{\underline{e}}(\zeta)^{p^s \cdot m^t}$ (3)

Set $H := \langle \zeta + \bar{a} | a \in \{0, \dots, k\} \rangle \subseteq \mathbb{F}_q^\times$
 $(\zeta \notin \mathbb{F}_p \text{ since } r \nmid (p-1) \text{ by Claim 1})$

Consider: $T := \{(e_0, \dots, e_k) \in \mathbb{N}_0^{k+1} \mid \sum_{a=0}^k e_a < |G|\}$

$\Phi : T \mapsto H, (e_0, \dots, e_k) \mapsto f_{\underline{e}}(\zeta) = \prod_a (\zeta + \bar{a})^{e_a} \in H$

Claim 2: Φ is injective.

Indeed, take $(\underline{e}), (\underline{\hat{e}}) \in T$ such that $\Phi(\underline{e}) = \Phi(\underline{\hat{e}})$

$$\Rightarrow \forall s, t \in \mathbb{N}_0 : f_{\underline{e}}(\zeta^{p^s \cdot m^t}) \stackrel{(3)}{=} f_{\underline{e}}(\zeta)^{p^s \cdot m^t} = f_{\underline{\hat{e}}}(\zeta)^{p^s \cdot m^t} \stackrel{(3)}{=} f_{\underline{\hat{e}}}(\zeta^{p^s \cdot m^t})$$

$f_{\underline{e}} - f_{\underline{\hat{e}}}$ has roots ζ^e with $e \in G$ since $G = \langle \bar{p}, \bar{m} \rangle$

These are all distinct (since ζ is primitive)

But $\deg(f_{\underline{e}} - f_{\underline{\hat{e}}}) < |G|$ So $f_{\underline{e}} - f_{\underline{\hat{e}}} = 0$

Since $k \leq \sqrt{r} \cdot l < r < p$ the $(x + \bar{a})$ with $a \in \{0 \dots k\}$ are primitive distinct.

So $(\underline{e}) = (\underline{\hat{e}})$ ✓

So is $|H| \geq |T|$?

Let M be the set of all $\{x_0, \dots, x_k\} \subseteq \{1, \dots, |G| + k\}$

with $x_0 < x_1 < \dots < x_k$

For $\{x_0, \dots, x_k\} \in M$ define $(e_0, \dots, e_k) \in \mathbb{N}_0^{k+1}$ by $e_a = x_a - x_{a-1}$ with $x_{-1} := 0$

$$\Rightarrow \sum_{a=0}^k e_a = \sum_{a=0}^k (x_a - x_{a-1} - 1) = x_k - (k+1) < |G|$$

Obtain injection $M \Leftrightarrow T$

$$\text{So } |H| \geq |T| \geq |M| = \binom{|G|+k}{k+1} \stackrel{(2)}{\geq} \binom{\lfloor l\sqrt{|a|} \rfloor + 1 + k}{k+1} = \binom{\lfloor l\sqrt{|a|} \rfloor + 1 + k}{\lfloor l\sqrt{|a|} \rfloor} \stackrel{(2)}{\geq} \binom{2 \cdot \lfloor l\sqrt{|a|} \rfloor + 1}{\lfloor l\sqrt{|a|} \rfloor}$$

1.3.19 Lemma 18 (Property of binomial coefficients)

$$\forall n \in \mathbb{N}_{>1} : \binom{2 \cdot n + 1}{n} > 2^{n+1}$$

Proof:

$n = 2 :$

$$\binom{5}{2} = 10 > 2^3$$

$n - 1 \rightarrow n :$

$$\binom{2 \cdot n + 1}{n} = \binom{2 \cdot n}{n-1} + \binom{2 \cdot n}{n} = \binom{2 \cdot n - 1}{n-2} + \binom{2 \cdot n - 1}{n-1} + \binom{2 \cdot n - 1}{n-1} + \binom{2 \cdot n - 1}{n} \geq 2 \cdot \binom{2 \cdot n - 1}{n-1} \stackrel{ind.}{>} 2 \cdot 2^n = 2^{n+1}$$

Continuation of Proof of Theorem 17

$$|H| > 2^{\lfloor l \cdot \sqrt{|a|} \rfloor + 1} \geq 2^{l \cdot \sqrt{|a|}} \geq 2^{\lg(n) \cdot \sqrt{|a|}} = n \sqrt{|a|} \quad (4)$$

Assume $n \notin \mathbb{P}$ By step (1) m is not a perfect power

\Rightarrow the map $\mathbb{N}_0 \times \mathbb{N}_0 \mapsto \mathbb{N} \quad (s, t) \mapsto p^s m^t$ is injective.

Set $A := \{p^s m^t \mid s, t \in \{0, \dots, \lfloor \sqrt{a} \rfloor\}\} \subseteq \mathbb{N}$

$$\Rightarrow |A| = (\lfloor \sqrt{|a|} \rfloor + 1)^2 > |G|$$

Since $G = \langle \bar{p}, \bar{m} \rangle \subseteq (\mathbb{Z}/(r))^\times$ this implies that $\exists n, \hat{n} \in A$

such that $n \neq \hat{n}$ but $b \equiv \hat{n} \pmod{r}$.

$$\text{Let } h \in H \Rightarrow h = f_{\underline{e}}(\zeta) \text{ with } (\underline{e}) \in \mathbb{N}_0^{k+1} \Rightarrow h^n \stackrel{(3)}{=} f_{\underline{e}}(\zeta^n) \stackrel{n \equiv \hat{n} \pmod{r}}{=} f_{\underline{e}}(\zeta^{\hat{n}}) \stackrel{(3)}{=} h^{\hat{n}}$$

So the polynomial $Y^n - Y^{\hat{n}} \in \mathbb{F}_q[Y]$ has all elements of H as zeros.
 But $\deg(Y^n - Y^{\hat{n}}) \leq \max\{n, \hat{n}\} \leq (p \cdot m)^{\lfloor \sqrt{|G|} \rfloor} \leq n\sqrt{|G|} < |H|$
 \Rightarrow contradiction since $Y^n - Y^{\hat{n}} \neq 0$ \square

1.4 Cryptology

A ("Alice") wants to send a message to B ("Bob") such that an eavesdropper E ("Eve") can not read the clear message. So A and B encrypt the message.



Figure 2: Scheme of eavesdropping

Symmetric-key cryptography

A and B share secret keys for encryption ($x \mapsto x'$) and decryption ($x' \mapsto x$). Only A and B know the keys.

Example: AES approved by the US government in 2002

Application:

- sending messages
- encrypt files (A=B)

Problem: Key exchange between A and B

Public-key cryptography

Encryption-map $\phi : x \mapsto x'$ is made public by B, but decryption $\phi : x' \mapsto x$ is kept secret.

Advantage: No confidential key exchange.

Disadvantages:

- more costly than symmetric key cryptography
- doubt whether E can reconstruct ϕ^{-1} from ϕ with enough computing power

Applications:

- sending messages
- exchange of symmetric keys
- authentication: Together with x , B sends $\phi^{-1}(x)$ (or $\phi^{-1}|$ Part of x together with date). A verifies by applying ϕ .
 Better: challenge-response-protocol.

Examples: RSA, elliptic curve

1.4.1 Algorithm (RSA)

- (1) B chooses $p, q \in \mathbb{P}$ large (> 100 digits)
with $p \neq q$ $n := p \cdot q$
- (2) B chooses $e, f \in \mathbb{N}$ large such that $e \cdot f \equiv 1 \pmod{\phi(n)}$
with $\phi(n) = (p-1)(q-1)$
- (3) B makes n, e public, keep f secret
- (4) The message is encoded as an element $x \in \mathbb{Z}/(n)$
- (5) A computes $\phi(x) = x^e = y \in \mathbb{Z}/(n)$ and sends y
- (6) B receives y and computes $y^f = x \in \mathbb{Z}/(n)$

Comments on steps of RSA:

- (6) Have $e \cdot f = a \cdot (p-1) \cdot (q-1) + 1$ with $a \in \mathbb{N}_{>0}$
 $y^f = x^{e \cdot f}$

$$1: q \nmid f, q \nmid x \Rightarrow x^{a(p-1)(q-1)} = (x^{\phi(n)})^a \stackrel{\text{LittleFermat}}{\equiv} 1^a = 1 \Rightarrow x^{e \cdot f} = x \quad \checkmark$$

- Case 2: $p|x, q \nmid x \Rightarrow x^{e \cdot f} \equiv 0 \equiv x \pmod{p}$
 $x^{e \cdot f} \equiv x \pmod{q}$ as above.

Case 3: $q|x$ As Case 2

\Rightarrow Correctness of decryption

Cost:

- (1) Finding p, q of length approximately l . Prime-number theorem: Gap between two primes of length $\approx l$ is $O(l)$
Using Miller Rabin with error probability 2^m . Expected cost of (1) is $O(m \cdot l^4)$ bit operations.
- (2) Choose e co-prime to $\phi(n)$ obtain $f = \text{inverse} \pmod{\phi(n)}$ by extended euclidean Algorithm: $O(l^2)$
- (5)(6) Fast exponentiation: $O(l^3)$

Security of RSA: p and q must be so large that factorization of n is "impossible". Assumption that factorization is expensive could not be shown! But could f be obtained without knowing p and q ? The following algorithm gives a negative answer. It shows that the problem of breaking RSA is always basically factorization.

Remember: $\phi(n) | (e \cdot f - 1) \Rightarrow m \leq n^2$

1.4.2 Algorithm 1 (Finding a divisor)

Input : $n \in \mathbb{N}_{>2}$ odd squarefree $\notin \mathbb{P}$ and $m \in \mathbb{N}_{>0}$ such that $\phi(n) \mid m$ $m \leq n^2$

Output: $d \in \mathbb{N}$ with $d \mid n$ $1 < d < n$

- (1) Choose $a \in \{2, \dots, (n-2)\}$ randomly
set $k := m$
- (2) If $d := \gcd(a, n) \neq 1$
return d
- (3) Repeat steps (4) - (8) //while(true)
- (4) compute $d := \gcd(n, a^k - 1)$
- (5) If $d = 1$ go to (1)
- (6) If $d < n$ return d
- (7) if k is odd go to (1)
- (8) set $k := \frac{k}{2}$

Correctness is clear. What about termination and running time?

1.4.3 Proposition 2 (Complexity of Algorithm 1)

Algorithm 1 terminates in expected time $O(l(n)^4)$ bit operations (Las Vegas Algorithm).

Proof:

Set $l := \text{length}(n)$

Have $n = \prod_{i=1}^r p_i$ with $p_i \in \mathbb{P}$ distinct.

$\phi(n) = \prod_{i=1}^r (p_i - 1) \mid m$ So initially all $(p_i - 1)$ divide k .

At some iteration it happens for the first time that $(p_i - 1) \nmid k$

Then $k \equiv \frac{p_1-1}{2} \pmod{(p_1-1)} \Rightarrow a^k \equiv \pm 1 \pmod{p_i}$ -1 occurs for some a

For those j with $(p_j - 1) \mid k$ have $a^k \equiv 1 \pmod{p_j}$

Consider the group homomorphism: $\phi_i(\mathbb{Z}/(n))^{\times} \mapsto (\mathbb{Z}/(p_1))^{\times} \times \dots \times (\mathbb{Z}/(p_r))^{\times}$
 $\bar{a} \mapsto (a^k \pmod{p_1}, \dots, a^k \pmod{p_r})$

The image of ϕ is a product of groups $\{\pm 1\}$ or $\{1\}$ depending whether $(p_i - 1) \nmid k$ or $(p_i - 1) \mid k$

Conclusion:

For at least half of all a 's, $\phi(\bar{a})$ is neither $(1, \dots, 1)$ nor $(-1, \dots, -1)$

If $a^k \equiv 1 \pmod{p_j}$ then $p_j \mid (a^k - 1) \Rightarrow p_j \mid d$

If $a^k \equiv -1 \pmod{p_j}$ then $p_j \nmid (a^k - 1) \Rightarrow p_j \nmid d$

So for these a the algorithm is successful.

This means that the expected number of a 's that need to be tested is ≤ 2

(Since $\sum_{i=1}^{\infty} i \cdot \left(\frac{1}{2}\right)^i = 2$ More generally for $0 < p < 1 : p \cdot \sum_{i=1}^{\infty} i \cdot (1-p)^{i-1} = \frac{1}{p}$)

Analysis of running time (in bit operations) for each a (using gcd is quadratic) leads to the claim. \square

Problems of RSA:

- How difficult is factorization of integers (lower bound?)
- decryption of some or all messages without having f ?

1.4.4 Diffie-Hellmann Key Exchange

Goal: A, B want to exchange a symmetric key via a public channel

- (1) A and B agree on a $p \in \mathbb{P}$ (should be large) and $g \in (\mathbb{Z}/(p))^{\times}$ public
- (2) A chooses $a \in \{2, \dots, (p-2)\}$ randomly and sends $u := g^a$ to B
- (3) B chooses $b \in \{2, \dots, (p-2)\}$ randomly and sends $v := g^b$ to A
- (4) A computes $v^a = (g^b)^a = g^{a \cdot b}$
B computes $u^b = (g^a)^b = g^{a \cdot b}$

\Rightarrow A and B share $g^{a \cdot b}$

Example:

A chooses $a = 7$

$$\bar{3}^7 = \bar{11} \in \mathbb{Z}/(17)$$

$$\bar{13}^7 = \bar{4}$$

B chooses $b = 4$

$$\bar{3}^4 = \bar{13} \in \mathbb{Z}/(17)$$

$$\bar{11}^4 = \bar{4}$$

If Eve reconstructs a, b from g^a and g^b she can compute $g^{a \cdot b}$

The Security of Diffie-Hellmann depends on the difficulty of the discrete logarithm problem (DLP):

Given $g \in G$ element of a group or monoid and given $g^a \in G$, determine a (or determine $a' \in \mathbb{Z}$ such that $g^a = g^{a'}$)

1.4.5 Elliptic curve cryptography (ECC)

ECC uses elliptic curves as groups.

$$y^2 = x^3 + a \cdot x + b \rightsquigarrow y^2 z = x^3 + axz^2 + bz^3$$

ECC uses suitable elliptic curves on \mathbb{F}_a

1.5 Factorization

Let $m \in \mathbb{N}_{>1}$ $n \notin \mathbb{P}$ Find a divisor d with $1 < d < n$. From this we obtain the factorization of n by recursion.

Naive method: Trial division. Cost essentially exponential in $l(n)$

1.5.1 Algorithm 1 (Sieve of Eratosthenes)

Input : $n \in \mathbb{N}_{>1}$

Output: All primes $\leq n$

- (1) Create a list of all numbers $\leq n$
- (2) $p := 2$
- (3) Mark all multiples of p in the List
- (4) if all numbers are marked
return
- (5) Let p be the smallest number that is not marked
- (6) $p \in \mathbb{P}$ Go to (3)

Running time of Algorithm 1 is exponential.

Pollard's rho (ρ) algorithm:

Idea: Choose a function $\mathbb{Z}/(m) \mapsto \mathbb{Z}/(n)$ e.g. $f(x) = x^2 + 1$

Choose $x_0 \in \mathbb{Z}/(n)$ set $x_i := f^i(x_0)$ iterative application.

Let $p \mid n$ be a prime. Since $|\mathbb{Z}/(p)| < \infty$ then $\exists i < j : x_i \equiv x_j \pmod{p}$

Starting at x_i the sequence of x_j will be periodic.

$p \mid x_i - x_j \quad p \mid n \Rightarrow p \mid \gcd(n, x_i - x_j) =: d$

If $x_i \not\equiv x_j \pmod{n}$ (which is not guaranteed) then d is a proper divisor of n .

- Recall that gcd computation is cheap
- Testing all pairs is a lot
- Proposition 2 helps with this

1.5.2 Proposition 2 (length of periods)

Let M be a set of functions $f : M \mapsto M$ and $x_0 \in M$ $x_i := f^i(x_0)$

If $x_{t+l} = x_t$ for $l, t \in \mathbb{N} > 0$ ($\rightarrow t$ "off-period", l "length of period")

$\Rightarrow \exists j \in \mathbb{N}$ with $0 < j \leq t + l$ such that $x_j = x_{2j}$

Proof:

$f^l(x_t) = x_t \Rightarrow \forall a \in \mathbb{N} \quad f^{a \cdot l}(x_t) = x_t$ Assume $j = a \cdot l \geq t$ $a \in \mathbb{N}$

$x_{2j} = x_{t+(j-t)+a \cdot l} = f^{(j-t)}(x_{t+a \cdot l}) = f^{(j-t)}(f^{al}(x_t)) = f^{(j-t)}(x_t) = x_j$

Case 1 $t = 0$ $j = l$ ✓

Case 2 $t > 0$ $j = t + (-t \bmod l) \in 0, \dots, (l-1)$ ✓

1.5.3 Algorithm 3 (Pollard's ρ - Algorithm)

Input : $n \in \mathbb{N}_{>1}, n \notin \mathbb{P}$

Output: a proper divisor of n or "FAIL"

- (1) Choose $x \in \{0, \dots, (n-1)\}$ randomly
set $y := x$
- (2) repeat (3)-(6)
- (3) $x := x^2 + 1 \pmod{n}$ $y := (y^2 + 1)^2 + 1 \pmod{n}$ $// x := x_j y := x_{2j}$
- (4) $d := \gcd(n, x - y)$
- (5) if $(1 < d < n)$
return d
- (6) if $d = n$
return "FAIL"

One "FAIL" includes no conclusion so you might want to repeat the Algorithm with a different x .

Running time? Assume the $x_i := f^i(x_0)$ are randomly distributed.

When can we expect that a match ($x_i \equiv x_j \pmod{p}$) occurs? \rightarrow "Birthday Problem"

Lemma (Birthday Problem):

We iteratively choose numbers in $\{1, \dots, n\}$ at random. The expected numbers of choices (if we keep choosing until a number has been chosen twice) is $< \sqrt{\frac{\pi \cdot n}{2}} + 2$

Proof:

Let $s \geq 2$ be the numbers of choices until a match occurs. For $k \in \mathbb{N}$ with $P()$ as probability

$$P(s > k) = \prod_{i=1}^k \left(1 - \frac{i-1}{n}\right) \leq \prod_{i=1}^k e^{-\frac{i-1}{n}} = e^{\sum_{i=1}^k -\frac{i-1}{n}} = e^{\frac{k(1-k)}{2n}} \leq e^{-\frac{(k-1)^2}{2n}}$$

* since $f(x) = e^x - (1-x) \geq 0$ for $x \geq 0$

$$f(0) = 0$$

$$f'(x) \geq 0 \text{ if } x \geq 0$$

$$\sum_{k=0}^{\infty} P(s > k) = 2 + \sum_{k=2}^{\infty} P(s > k) \leq 2 + \sum_{k=2}^{\infty} e^{-\frac{(k-1)^2}{2n}} \leq 2 + \int_1^{\infty} e^{-\frac{(x-1)^2}{2n}} dx$$

$$\stackrel{x:=x-1}{=} 2 + \int_0^{\infty} e^{-\frac{x^2}{2n}} dx = 2 + \int_0^{\infty} e^{-\left(\frac{x}{\sqrt{2n}}\right)^2} dx$$

$$\stackrel{x:=\frac{x}{\sqrt{2n}}}{=} 2 + \sqrt{2n} \int_0^{\infty} e^{-x^2} dx = 2 + \sqrt{2n} \cdot \frac{\pi}{2} = 2 + \sqrt{\frac{n \cdot \pi}{2}}$$

Example:

People arrive at a party. When can you expect to have two that share their birthday?

\rightarrow when 26 have arrived!

1.5.4 Theorem 4 (Bit-complexity of Algorithm 3)

under suitable assumptions on the distribution $f^i(x)$ for $f(x) = x^2 + 1$ Algorithm 3 has the expected running time of $O(\sqrt[n]{n} \lg(n)^2)$ bit operations

Proof:

By Proposition 2 and the Lemma the expected number of runs through the loop is

$$O(\sqrt[p]{p}) = O(\sqrt[p]{n}) \text{ as } p \leq \sqrt{n}$$

Each run through the loop takes $O(\lg(n)^2)$ bit operations. \square

Pollard's p-1 Algorithm

Motivation: Let $p \mid n$ prime divisor

$$\Rightarrow \forall a \in \mathbb{Z} : a^{p-1} \equiv 1 \pmod{p} \quad \text{whith } \gcd(a, p) = 1$$

$$\Rightarrow \forall m \in \mathbb{Z} \text{ with } (p-1) \mid m : a^m \equiv 1 \pmod{p}$$

$$p \mid \gcd(a^m - 1, n)$$

Let B be an upper-bound for the prime powers dividing $p-1$.

" $p-1$ is B -power-smooth".

$$\text{Then } (p-1) \mid \prod_{(q \leq B) \in \mathbb{P}} q^{\lfloor \log_q(B) \rfloor}$$

Neither p nor B are known! But guess and try B and hope for the best.

1.5.5 Algorithm 5 (Pollard's ρ - 1 method)

Input : $n \in \mathbb{N}_{>1} \setminus \mathbb{P}$

Output: $d \in \mathbb{N}$ with $d \mid n$ $1 < d < n$ or "FAIL"

- (1) Choose a "smoothness bound" B
- (2) Choose $a \in \{2, \dots, (n-2)\}$ randomly
- (3) Use Algorithm 1 to find all $q \in \mathbb{P}$ with $q \leq B$
For every q perform steps (4) - (5)
- (4) $k := q^{\lfloor \log_q(B) \rfloor}$
set $a := a^k \pmod{n}$
compute $d := \gcd(n, a - 1)$
- (5) if $1 < d < n$
return d
- (6) return "FAIL" //or increase B and go to (1)

Consequence: when setting up RSA p, q should be chosen such that $p-1$ and $q-1$ have large prime divisors.

The quadratic sieve (State of the art factorization algorithm)

Observation: if $n = x^2 - y^2$ then $n = (x - y) \cdot (x + y)$

Conversely if $n = a \cdot b$ then $n = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2$

1-st Idea: Find $x, y \in \mathbb{Z}$ such that $x^2 \equiv y^2 \pmod{n} \quad \wedge \quad x \not\equiv \pm y \pmod{n}$

Then $n \mid (x - y) \cdot (x + y)$

\Rightarrow for every $y : \quad e \in \mathbb{P}$ with $p \mid n : p \mid (x - y) \vee p \mid (x + y)$

$\Rightarrow p \mid \gcd(x - y, n) \vee p \mid \gcd(x + y, n)$

Since both gcd are $< a$ receive a non-trivial divisor of n

If $x^2 \equiv y^2 \pmod{n}$ how probable is it that $x \equiv \pm y \pmod{n}$?

Let $n = \prod_{i=1}^r p_i^{k_i}$ odd with $p_i \in \mathbb{P}$ distinct.

Assume $p_i \nmid x \forall i = 1 \dots r$ Since $(\mathbb{Z}/(p_i^{k_i}))^\times$ is cyclic there are 2^r classes $y \pmod{n}$ such that $x^2 \equiv y^2 \pmod{n}$

[Reason: These classes are given by $y \equiv \pm x \pmod{p_i^{k_i}}$ These are the only solutions since $\mathbb{Z}/(p_i^{k_i})^\times$ is cyclic of even order.

$G = \langle \sigma \rangle$ cyclic of order $2m$

$x = \sigma^i$ Find $l \in \mathbb{Z}$ such that $x^2 = (\sigma^l)^2$

$\Leftrightarrow 2j \equiv 2l \pmod{2m} \Leftrightarrow j \equiv l \pmod{m}$

$\Leftrightarrow l \equiv j \pmod{2m}$ or $l \equiv j + m \pmod{2m}$]

But have $x \equiv \pm y$ only for $2y$'s.

Failure probability: 2^{1-r}

Handle case $r = 1$ by Algorithm 11 in 1.3

Example 1

$n = 91$ Search $x, y \in \mathbb{Z} \quad k \in \mathbb{Z}$ such that $x^2 = k \cdot n + y^2$

Good chance if x is slightly bigger than $\sqrt{k \cdot n}$

$k := 1 \Rightarrow \sqrt{91} \approx 9,54 \Rightarrow x := 10 \Rightarrow 10^2 = 100 \equiv 3^2 \pmod{91}$

$n = 10^2 - 3^2 = (10 - 3) \cdot (10 + 3) = 7 \cdot 13 \quad \checkmark$

Another try:

$k := 8 \Rightarrow \sqrt{8 \cdot 91} \approx 26,98 \Rightarrow 27^2 \equiv 1^2 \pmod{91} \quad \gcd(26, 91) = 13$

Example 2

$n = 4633 \quad k := 3$

$\sqrt{3 \cdot n} \approx 117,89 \Rightarrow x^2 = 118^2 \equiv 5^2 \pmod{n}$

$\gcd(118 - 5, n) = 113$

$\gcd(118 + 5, n) = 41 \quad \checkmark$

2-nd Idea: Choose $B \in \mathbb{N}$ "smoothness bound" suitable.

Let $p_2, \dots, p_r \in \mathbb{P}$ be all primes $\leq B$ (Algorithm 1) set $p_1 := -1$

The p_i form a "factor basis".

For $a \in \mathbb{Z}$ write $(a \pmod{n})$

for the $x \in \mathbb{Z}$ with $x \equiv a \pmod{n}$ and $-\frac{n}{2} < x \leq \frac{n}{2}$

Procedure:

Search numbers $a_1, \dots, a_m \in \mathbb{Z}$ such that $(a_i^2 \pmod{n}) = \prod_{j=1}^r p_j^{e_{ij}}$

with $e_{ij} \in \mathbb{Z}$ ("B numbers")

So for $\mu_1, \dots, \mu_m \in \mathbb{N}_0$ have $\left(\prod_{i=1}^m a_i^{\mu_i} \right)^2 \equiv \prod_{i=1}^m \prod_{j=1}^r p_j^{\mu_i \cdot e_{ij}} \pmod{n} = \prod_{j=1}^r p_j^{\sum_{i=1}^m \mu_i \cdot e_{ij}} \pmod{n}$

If the vectors (e_{i1}, \dots, e_{ir}) become linearly dependant mod 2 (guaranteed if $m > r$)

then $\exists \mu_1, \dots, \mu_m \in \{0, 1\}$ not all 0 such that:

$$\sum_{i=1}^m \mu_i \cdot e_{ij} = 2 \cdot k_j \quad k_j \in \mathbb{N}_0$$

with $x := \prod_{i=1}^m a_i^{\mu_i}$ $y := \prod_{j=1}^r p_j^{k_j}$ obtain $x^2 \equiv y^2 \pmod{n}$

Example: $n = 4633$ choose $B = 3 \Rightarrow$ factor basis $-1, 2, 3$

Search $a \in \mathbb{Z}$ such that $|a_i^2 \pmod{n}|$ is small. Idea: $a \approx \sqrt{n} = 68.06\dots$

$$a_1 := 68 : 68^2 = n - 9 \equiv (-1) \cdot 3^2 \pmod{n}$$

$$\rightarrow e_1 = (1, 0, 2) \rightarrow (1, 0, 0) \in \mathbb{F}_2^3$$

$$a_2 := 69 : 69^2 = n + 128 \equiv 2^7 \pmod{n}$$

$$\rightarrow e_2 = (0, 7, 0) \rightarrow (0, 1, 0) \in \mathbb{F}_2^3$$

$$a_3 := 67 : 67^2 = n - 144 \equiv (-1) \cdot 2^4 \cdot 3^2$$

$$\rightarrow e_3 = (1, 4, 2) \rightarrow (1, 0, 0) \in \mathbb{F}_2^3$$

$$e_1 + e_3 \equiv 0 \pmod{2} \quad \text{In fact:}$$

$$e_1 + e_3 = 2 \cdot \underbrace{(1, 2, 2)}_{(k_1, k_2, k_3)} \rightarrow \mu_1 = 1 \quad \mu_2 = 0 \quad \mu_3 = 1$$

$$x := a_1 \cdot a_3 \equiv -77 \pmod{n}$$

$$y := (-1) \cdot 2^2 \cdot 3^2 = -36$$

$$x - y = -41 \quad x + y = -113$$

$$\gcd(n, x - y) = 41 \quad \gcd(n, x + y) = 113 \quad \checkmark$$

3rd Idea: Look for a_i of the form $t + \lfloor \sqrt{n} \rfloor$ with t in a "suitable".

Sieve Interval: $[-s, s] \cap \mathbb{Z}$

As it turns out if $s \leq \frac{\sqrt{5}-2}{2} \lfloor \sqrt{n} \rfloor$ then $(t + \lfloor \sqrt{n} \rfloor)^2 \pmod{n} = (t + \lfloor \sqrt{n} \rfloor)^2 - n =: f(t)$

When does $p_j^{e_j}$ divide $f(t)$ (with $j \geq 2$)? Precisely if $(t + \lfloor \sqrt{n} \rfloor)^2 \equiv n \pmod{p_j^{e_j}}$

If this holds for some t then it also holds for all $t + k \cdot p_j^{e_j}$ with $k \in \mathbb{Z}$ Moreover if it holds then $\bar{n} \in \mathbb{F}_{p_j}$ is square. So may remove all p_j such that $\bar{n} \in \mathbb{F}_{p_j}$ is a non-square from the factor basis.

Obtain a sieving procedure:

For $t \in [-s, s] \cap \mathbb{Z}$ with $p_j^{e_j} \mid f(t)$ "mark" all elements $t + k \cdot p_j^{e_j} \in [-s, s]$

1.5.6 Algorithm 6 (Quadratic sieve, simplified version)

Input : $n \in \mathbb{N}_{>1} \setminus \mathbb{P}$ odd

Output: A non trivial divisor of n or "FAIL"

- (1) if $(n = m^e)$ with $m, e \in \mathbb{N}_{>1}$
return m // can be done with Algorithm 11 § 3
- (2) Choose a "smooteness bound" $B \in \mathbb{N}$ and a "sieve bound" $s \in \mathbb{N}$ suitably
- (3) Let $p_1 = -1$ p_2, \dots, p_r be the factor basis given by B . Delete those p_j such that $\bar{n} \in \mathbb{F}_{p_j}$ is a non-square
- (4) for $(t = -s, -s+1, \dots, s-1)$
compute $f_t := |(t + \lfloor \sqrt{n} \rfloor)^2 - n| \in \mathbb{N}_{>0}$
- (5) for $(t = -s, \dots, s)$
set $e_t := (0, \dots, 0) \in \mathbb{N}_0^r$ // initialize exponent vectors
- (6) for $(t = -s, \dots, 0)$
set $e_{t,1} := 1$ // \rightarrow first entry of each e_t is the exponent of $p_1 = -1$ in $f(t)$
- (7) for $(j = 2, \dots, r)$ repeat (8) - (10)
- (8) for $(e = 1, \dots, \lfloor \log_{p_j}(B) \rfloor)$ repeat (9) - (10) // or maybe a bit larger
- (9) solve $(t + \lfloor \sqrt{n} \rfloor)^2 \equiv n \pmod{p_j^e}$
Let $(t_i \pmod{p_j^e}), \dots, (t_m \pmod{p_j^e})$ be the solutions.
// We will see that $m \in \{0, 2, 4\}$ with $m = 2$ most frequent.
- (10) for all $t = t_i + k \cdot p_j^e \in [-s, s]$ with $k \in \mathbb{Z}, i = 1, \dots, m$
set $e_{t,j} := e_{t,j} + 1$
 $f_t := \frac{f_t}{p_j}$
- (11) let t, \dots, t_m be those $t \in [-s, s] \cap \mathbb{Z}$ for which $f_t = 1$
/* So the $a_i = t_i + \lfloor \sqrt{n} \rfloor$ are B -numbers and the factorization
* of $a_i^2 \pmod n = a_i^2 - n = f(t)$ is given by the exponent
* vectors e_t */
- (12) if the $(e_{t_i} \pmod 2) \in \mathbb{F}_2^r (i = 1, \dots, m)$ are not linearly dependent.
return "FAIL"
- (13) compute $\mu_1, \dots, \mu_m \in \{0, 1\}, k_1, \dots, k_r \in \mathbb{N}_0$ such that $\sum_{i=1}^m \mu_i e_{t_i} = 2 \cdot (k_1, \dots, k_r)$
- (14) set $x := \prod_{i=1}^m (t_i + \lfloor \sqrt{n} \rfloor)^{\mu_i} \pmod n$
 $y := \prod_{j=1}^r p_j^{k_j} \pmod n$ // Now $x^2 \equiv y^2 \pmod n$

(15) if $\gcd(n, x - y)$ or $\gcd(n, x + y)$ is a non-trivial divisor
 return the non-trivial divisor
 else
 return "FAIL"

With good heuristics it will almost certainly never return FAIL.

Example: $n = 20437$

Choose $B := 10$ $s := 3$

Factor basis: $p_1 = -1$ $p_2 = 2$ $p_3 = 3$ $p_4 = 7$

(5 omitted as: $n \equiv 2 \pmod{5}$ non-square)

$\lfloor \sqrt{n} \rfloor = 142$

Solve $(t + 142)^2 \equiv n \pmod{p_j^e}$

$p_2 = 2$: Compute modulo 2,4,8. $n \equiv 5 \pmod{8}$

t odd $\Rightarrow (t + 142)^2 \equiv 1 \pmod{8} \Rightarrow (t + 142)^2 \equiv n \pmod{4}$ but not $\pmod{8}$

t even $\Rightarrow (t + 142)^2 \equiv 0 \pmod{2} \not\equiv n \pmod{2}$

$$\Rightarrow e_{t,2} = \begin{cases} 2 & t \text{ odd} \\ 0 & t \text{ even} \end{cases}$$

$p_3 = 2$: $n \equiv 1 \pmod{3}$ $\lfloor \sqrt{n} \rfloor \equiv 1 \pmod{3}$

So $3 \mid f(t) \Leftrightarrow t + 1 \equiv \pm 1 \pmod{3} \Leftrightarrow t \equiv 0 \text{ or } 1 \pmod{3}$

$e = 2$ $n \equiv 7 \equiv (\pm 4)^2 \pmod{9}$ $\lfloor \sqrt{n} \rfloor \equiv 7 \pmod{9}$

So $9 \mid f(t) \Leftrightarrow t + 7 \equiv \pm 4 \pmod{9} \Leftrightarrow t \equiv -3, -2 \pmod{9}$

$p_4 = 7$ $n \equiv 4 \pmod{7}$ $4 = (\pm 2)^2$ $\lfloor \sqrt{n} \rfloor \equiv 2 \pmod{7}$

So $7 \mid f(t) \Leftrightarrow t + 2 \equiv \pm 2 \pmod{7} \Leftrightarrow t \equiv 0 \text{ or } 3 \pmod{7}$

t	-3	-2	-1	0	1	2	3
$f_t = f(t) $	1116	837	556	273	12	295	588
p_1 component of e_t	1	1	1	1	0	0	0
p_2 component	2	0	2	0	2	0	2
f_t divided by 2-power	279	837	139	273	3	299	147
p_3 component	2	2	0	1	1	0	1
f_t	31	93	139	91	1	299	49
p_4 component	0	0	0	1	0	0	2
f_t	31	93	139	13	1	299	1

Obtain $m = 2$: $t_1 = 1$ $t_2 = 3$ $e_1 = (0, 2, 1, 0)$ $e_3 = (0, 2, 1, 2)$

They are lineary dependent $\pmod{2}$

$e_1 + e_3 = 2 \cdot (0, 2, 1, 1)$

$x = (142 + 1) \cdot (142 + 3) \equiv 298 \pmod{n}$

$y = p_2^2 \cdot p_3 \cdot p_4 = 2^2 \cdot 3 \cdot 7 = 84$

$\gcd(n, x - y) = \gcd(n, 214) = 107$

$\gcd(n, x + y) = 191$

Indeed $n = 107 \cdot 191$

Computing square roots (mod p^e)

Case 1: p odd

Find x with $x^2 \equiv n \pmod{p}$ by trying $x \pmod{p}$ (exactly two solutions). Suppose we have found x with $x^2 \equiv n \pmod{p^e}$

So $x^2 - n = p^e \cdot r \quad r \in \mathbb{Z}$

New x should be $x + y \cdot p^e$

Compute modulo p^{e+1} : $(x + y \cdot p^e)^2 - n = x^2 + 2yxp^e + y^2p^{2e} - n \equiv p^e \cdot (r + 2xy) \pmod{p^{e+1}}$

So $(x + y \cdot p^e)^2 \equiv n \pmod{p^{e+1}} \Leftrightarrow 2xy \equiv -r \pmod{p}$ uniquely and easily solvable

\rightarrow Obtain two solutions (mod p^e)

\Rightarrow special case of "Hensel lifting"

Case 2: $p = 2$

Find $x \in \mathbb{Z}$ with $x^2 \equiv n \pmod{8}$ (0 or 4 solutions since n odd)

Assume we have $x^2 \equiv n \pmod{2^e} \quad e \geq 3$

So $x^2 - n = r \cdot 2^e$

$\Rightarrow (x + y \cdot 2^{e-1})^2 - n = x^2 + xy \cdot 2^e + y^2 2^{2e-2} - n \equiv 2^e(r + xy) \pmod{2^{e+1}}$

So $(x + y \cdot 2^{e-1})^2 \equiv n \pmod{2^{e+1}} \Leftrightarrow y \equiv r \pmod{2}$

\rightarrow 0 or 4 solutions

Running time of quadratic sieve

Choose $B \approx \exp\left(\sqrt{\frac{1}{2} \ln(n) \cdot \ln(\ln(n))}\right)$

If $s \approx B$ then running time is: $O\left(\exp\left(\sqrt{\ln(n) \cdot \ln(\ln(n))}\right)\right)$

which is "slightly" sub-exponential

Factorization algorithm with best complexity (known to date):

Number field sieve

This also uses ideas 1 and 2, but an algebraic number field is used for generating B -numbers.

Heuristic Running time (modulo some conjectures): $O\left(\exp\left(\ln(n)^{\frac{1}{3}} \cdot \ln(\ln(n))\right)^{\frac{2}{3}}\right)$

2 Systems of equations

2.6 Linear Algebra

Tasks:

- solving systems of linear equations (= linear systems)
- inversions of matrices
- rank determination
- determinants
- matrix products

K field, $K^{m \times n}$ = set of $n \times n$ matrices

$GL_n(K)$...

Count the cost of algorithms in terms of field operations. If K is a finite field this translates directly to bit operations.

2.6.1 Proposition 1 (Complexity of usual algorithms)

- Solving an $m \times n$ -linear system by Gaussian elimination requires $O(\max\{m, n\}^3)$ field operations
- For $A \in GL_n(K)$ computing A^{-1} by usual method requires $O(n^3)$ field operations.
- Computing $\det(A)$ "as usual" requires $O(n^3)$ bit operations.
- Computing $A \cdot B$ for $A \in K^{m \times n}$ $B \in K^{n \times l}$ requires $O(m \cdot n \cdot l)$ field operations.

→ all cubic!

Proof:

- Cost of treating the k -th row with Gauss algorithm:
 ≤ 1 inversion, $\leq (n - k)$ multiplications
 $\leq (m - k)(n - k)$ multiplications and additions
 (clearing column below pivot element)
 Back substitution (i.e. clearing columns above pivot element):
 Let $r = rk(A) \leq (k - 1)(n - r)$ multiplications and additions
 Total cost $\leq \sum_{k=1}^r (1 + n - k + 2(m - k)(n - k) + 2(k - 1)(n - r))$
 $= 2mnr - mr^2 - \frac{1}{3}r^3 - nr + \frac{3}{2}r^2 + \frac{5}{6}r - mr$
 $\in O(\max\{m, n\}^3)$
- Inversion is Gaussian elimination of $n \times 2n$ -matrix of rank n
 cost $\leq \frac{8}{3}n^3 - \frac{3}{2}n^2 + \frac{5}{6}n \in O(n^3)$
- reduced to (a)

(d) obvious

Strassen-multiplication

let $A, B \in K^{2n \times 2n}$ Write: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$ with $A_{ij} B_{ij} \in K^{n \times n}$

Then $A \cdot B = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$ with $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} \rightarrow 8$ multiplications.

Set:

$$M_1 := (A_{12} - A_{22})(B_{21} - B_{22})$$

$$M_2 := (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_3 := (A_{11} - A_{21})(B_{11} + B_{12})$$

$$M_4 := (A_{11} + A_{12})B_{22}$$

$$M_5 := A_{11}(B_{12} - B_{22})$$

$$M_6 := (A_{22}(B_{21} - B_{11}))$$

$$M_7 := (A_{21} - A_{22})B_{11}$$

Then:

$$C_{11} = M_1 + M_2 - M_4 + M_6, C_{12} = M_4 + M_5$$

$$C_{21} = M_6 + M_7$$

$$C_{22} = M_2 - M_3 + M_5 - M_7$$

$\rightarrow 7$ Multiplications!

2.6.2 Algorithm 2 (Strassen-multiplication)

Input : $A \in K^{m \times n}, B \in K^{n \times l}$

Output: $A, B \in K^{m \times l}$

- (1) Let k be minimal such that $m, n, l \leq 2^k$
- (2) if $(k = 0)$ $//(\Leftrightarrow A, B \in K^{1 \times 1})$
return $A \cdot B$
- (3) Enlarge A, B by adding zeros such that $A, B \in K^{2^k \times 2^k}$
- (4) write $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$ with $A_{ij}B_{ij} \in K^{2^{k-1} \times 2^{k-1}}$
- (5) compute $M_1 \dots M_7$ as above, do multiplications by recursive call
- (6) compute $A \cdot B = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$ by above formulas
- (7) Output: the upper left $m \times l$ - part of $A \cdot B$

2.6.3 Theorem 3 (Running time of Algorithm 2)

If $m, n, l \leq r$ Algorithm 3 requires $O(r^{\lg(7)})$ field operations

Proof:

Set $\Theta(k)$ = number of field operations.

Step 5: $7 \cdot \Theta(k-1) + 10 \cdot (2^{k-1})^2$

Step 6: $8 \cdot (2^{k-1})^2$

Obtain:

$$\Theta(k) = 7\Theta(k-1) + 18 \cdot 4^{k-1} \quad (*)$$

Claim: $\Theta(k) = 7^{k+1} - 6 \cdot 4^k$

Induction on k

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

$k-1 \rightarrow k : \Theta(k) = 7\Theta(k-1) + 18 \cdot 4^{k-1}$

$= 7(7^k - 6 \cdot 4^{k-1}) + 18 \cdot 4^{k-1}$

induction

$= 7^{k+1} - 4 \cdot 6 \cdot 4^{k-1} \quad \checkmark$

Have $2^{k-1} < r \Rightarrow k < \lg(r) + 1 \Rightarrow \Theta(k) < 7^{\lg(r)+2} = 49 \cdot 2^{\lg(7) \cdot \lg(r)} = 49^{\lg(17)} \quad \checkmark$

Remarks:

(a) $\lg(7) = 2.8074\dots$

(b) Coppersmith-Winograd: $O(r^{2.3754\dots})$

Improved by Stothes (2010), Williams(2011), LeGall(2014): $O(r^{2.3729\dots})$

(c) The cost of the best possible algorithm is unknown, even for $r = 3$

Let $M : \mathbb{N}_{>0} \mapsto \mathbb{R}_{>0}$ be a function such that two matrices in $K^{n \times n}$ can be multiplied in $\leq M(n)$ field operations. Assume $\exists \epsilon > 0 : \forall n :$

$$2^{2+\epsilon} M(n) \leq M(2n) \leq 8 \cdot M(n) \quad (1)$$

Example: $M(n) = 49 \cdot n^{\lg(7)}$

Recall: $A = (a_{ij})$ is upper (lower) triangular $\Leftrightarrow a_{ij} = 0$ for $i > j$ ($i < j$)

2.6.4 Proposition 4 (Complexity of matrix inversion)

An upper of lower triangular matrix $A \in GL_n(K)$ can be inverted in $O(M(n))$ field operations.

Proof:

Let $k \in \mathbb{N}$ be minimal such that $n \leq 2^k$

write $B = \begin{pmatrix} A & 0 \\ 0 & I_{2^k-n} \end{pmatrix} \in GL_{2^k}(K) \Rightarrow B^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_{2^k-n} \end{pmatrix}$

Assume B upper triangular:

$$B = \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix} \quad B_{11}, B_{22} \in GL_{2^{k-1}}(K), B_{12} \in K^{2^{k-1} \times 2^{k-1}}$$

$$B^{-1} = \begin{pmatrix} B_{11}^{-1} & -B_{11}^{-1} \cdot B_{12} \cdot B_{22}^{-1} \\ 0 & B_{22}^{-1} \end{pmatrix}$$

Let $\Theta(k)$ = computation cost depending on k .

$$\Theta(k) \leq 2 \cdot \Theta(k-1) + 2M(2^{k-1}) \stackrel{(1)}{\leq} 2 \cdot \Theta(k-1) + \frac{1}{2} \cdot M(2^k) \quad (**)$$

Claim: $\Theta(k) \leq 2^k + M(2^k)$

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

$$k-1 \rightarrow k : \Theta(k) \stackrel{**}{\leq} 2 \cdot \Theta(k-1) + \frac{1}{2}M(2^k) \stackrel{\text{induction}}{\leq} 2(2^{k-1} + M(2^{k-1})) + \frac{1}{2}M(2^k) \stackrel{(1)}{\leq}$$

$$2^k + \frac{1}{2}M(2^k) + \frac{1}{2}M(2^k) \quad \checkmark$$

$$\text{Have } n > 2^{k-1} \Rightarrow k < \lg(n) + 1 \Rightarrow \Theta(k) < 2 \cdot n + M(2n) \stackrel{(1)}{\leq} 2 \cdot n + 8 \cdot M(n) \quad \square$$

Project: Reduce (most) tasks of linear algebra to multiplication.

The following algorithm transforms a matrix such that all tasks become easy.

2.6.5 Algorithm 5 (Transforming a matrix)

Input : $A \in K^{m \times n}$

Output: Matrices L, Q, P, U such that: $LQAP = \begin{bmatrix} U \\ 0 \end{bmatrix} \begin{matrix} r \leftarrow \text{in row echelon form} \\ \text{m-r} \end{matrix} \in K^{m \times n}$

such that:

- $L \in K^{m \times m}$ lower triangular with 1's on the diagonal
- $Q \in K^{m \times m} \quad P \in K^{n \times n}$ permutation matrices
- $U \in K^{m \times m}$ upper triangular with non-zero diagonal entries ($r = 0$ if $A = 0$)
- If $r = m$ then $Q = I_m$

(1) if ($A = (...)$)
return $L = Q = (1) \quad P = I_n \quad r = 0$

(2) if ($A = a_1, \dots, a_n$)
let i be minimal with $a_i \neq 0$
 $P :=$ matrix exchanging 1st and i -th position in A
return $L = Q = (1) \quad P \quad U = A \cdot P$

(3) let $m_1 = \lfloor \frac{m}{2} \rfloor \quad m_2 = \lceil \frac{m}{2} \rceil$
write $A = \begin{bmatrix} B \\ C \end{bmatrix} \begin{matrix} m_1 \\ m_2 \end{matrix} \quad B \in K^{m_1 \times n} \quad C \in K^{m_2 \times n}$

(4) Applying the algorithm recursively on B

$$\text{obtain } L, Q, B, P = \begin{bmatrix} U_1 \\ Q \end{bmatrix} \begin{matrix} r_1 \\ m_1 - r_1 \end{matrix} \quad \text{with } U_1 \in K^{r_1 \times n}$$

$$(5) \text{ write } L_1 = \begin{bmatrix} L_t & 0 \\ L_l & L_r \end{bmatrix} \begin{matrix} r_1 \\ m_1 - r_1 \end{matrix} \quad Q_1 = \begin{bmatrix} Q_t \\ Q_b \end{bmatrix} \begin{matrix} r_1 \\ m_1 - r_1 \end{matrix} \quad U_1 = \begin{bmatrix} E & U'_1 \end{bmatrix} \begin{matrix} r_1 \\ m_1 - r_1 \end{matrix}$$

$$\text{Form } D := C \cdot P_1 = \begin{bmatrix} F & D' \end{bmatrix} \begin{matrix} m_2 \\ m_1 - r_1 \end{matrix} \quad \text{and } G := D' - FE^{-1}U' \in K^{m_2 \times (n-r_1)}$$

$$(6) \text{ Apply algorithm recursively to } G := L_2 \cdot Q_2 \cdot G \cdot P_2 = \begin{bmatrix} U_2 \\ 0 \end{bmatrix} \begin{matrix} r_2 \\ m_2 - r_2 \end{matrix}$$

(7) return

$$L := \begin{bmatrix} L_t & 0 & 0 \\ -L_2Q_2FE^{-1}L_t & L_2 & 0 \\ L_l & 0 & L_r \end{bmatrix} \begin{matrix} r_1 \\ m_2 \\ m_1 - r_1 \end{matrix}$$

$$Q := \begin{bmatrix} Q_t & 0 \\ 0 & Q_2 \\ Q_b & 0 \end{bmatrix} \begin{matrix} r_1 \\ m_2 \\ m_1 - r_1 \end{matrix}$$

$$P := P_1 \cdot \begin{bmatrix} I_{r_1} & 0 \\ 0 & P_2 \end{bmatrix} \begin{matrix} r_1 \\ n - r_1 \end{matrix}$$

$$U := \begin{bmatrix} E & U'_1P_2 \\ 0 & U_2 \end{bmatrix} \begin{matrix} r - 1 \\ r_2 \end{matrix}$$

2.6.6 Theorem 6 (Correctness and running time of Algorithm 5)

Algorithm 5 is correct and requires $O((\frac{n}{m} + 1) \cdot M(m))$ field operations

Proof:

Correctness by induction on m

$m = 1$ ✓

$m > 1$:

$m_1, m_2 < m$ so recursive calls are correct by induction.

By step (7) L, Q, P, U have desired form.

Compute:

$$LQAP = \begin{bmatrix} L_tQ_t & 0 \\ -L_2Q_2FE^{-1}L_tQ_t & L_2Q_2 \\ L_lQ_t + L_rQ_b & 0 \end{bmatrix} \begin{matrix} m_1 \\ m_2 \\ m_1 - r_1 \end{matrix} \cdot \begin{bmatrix} B \cdot P_1 \\ D \end{bmatrix} \begin{matrix} m_1 \\ m_2 \end{matrix} \cdot \begin{bmatrix} I_{r_1} & 0 \\ 0 & P_2 \end{bmatrix} \begin{matrix} r_1 \\ n - r_2 \end{matrix}$$

$$\stackrel{(4)}{=} \begin{bmatrix} U_1 \\ L_2Q_2(-FE^{-1}U_1 + D) \\ 0 \end{bmatrix} \begin{bmatrix} I_{r_1} & 0 \\ 0 & P_2 \end{bmatrix} \begin{matrix} r_1 \\ n - r_2 \end{matrix}$$

$$= \begin{array}{c|c} r_1 & n-r_1 \\ \hline E & U'_1 \\ 0 & L_2 Q_2 G \\ 0 & 0 \end{array} \begin{array}{c} r_1 \\ m_2 \\ m_1 - r_1 \end{array} \cdot \begin{array}{c|c} I_{r_1} & 0 \\ \hline 0 & P_2 \end{array} \stackrel{(6)}{=} \begin{array}{c|c} r_1 & n-r_1 \\ \hline E & U'_1 P_2 \\ 0 & U_2 \\ 0 & 0 \\ 0 & 0 \end{array} \begin{array}{c} r_1 \\ r_2 \\ m_2 \cdot r_2 \\ m_2 \cdot r \end{array} = \begin{array}{c|c} U \\ \hline 0 \end{array}$$

Suppose $r = m \Rightarrow r_1 = m_1 \quad r_2 = m_2 \Rightarrow Q_1 = I_{m_2} \quad Q_2 = I_{m_2} \Rightarrow Q = I_m$

Cost:

Fix $n \in \mathbb{N}$ and set $\Theta(k) :=$ maximal cost for a matrix $A \in K^{m \times n'}$ with $m \leq 2^k \quad n' \leq n$

Choose $A \in K^{m \times n'}$ with cost = $\Theta(k)$

Step(4) and (6): $\leq \Theta(k-1)$ each

Step(5): E^{-1} : by Proposition 4: $O(M(r_1)) \leq O(M(2^{k-1}))$

$F \cdot E^{-1} \leq M(2^{k-1})$

$F \cdot E^{-1} \cdot U'$: at most the cost of multiplying a $2^{k-1} \times 2^{k-1}$ -matrix by a $2^{k-1} \times n$ matrix.

Split right matrix into square parts.

\Rightarrow cost $\leq \lceil \frac{n}{2^{k-1}} \rceil \cdot M(2^{k-1}) \leq (2^{1-k} \cdot n + 1) \cdot M(2^{k-1})$

G : subtraction: $m_2 \cdot (n - r_1) \leq 2^{k-1} \cdot n \leq 2^{1-k} \cdot n \cdot M(2^{k-1})$

Step (7): $F \cdot E^{-1}$ already computed, $L_2 Q_2$: permuting rows. Cost: $\leq 2 \cdot M(2^{k-1})$

Obtain: $\Theta(k) \leq 2 \cdot \Theta(k-1) + (2^{-k} \cdot n + c) \cdot M(2^k) \quad c$ constant

From this obtain by induction:

$$\Theta(k) \leq \left(2^{-k} \cdot n \cdot \frac{1-2^{-k\epsilon}}{1-2^{-\epsilon}} + 2 \cdot c \cdot (1 - 2^{-k}) \right) \cdot M(2^k) \leq \left(\frac{1}{1-2^{-\epsilon}} \cdot \frac{n}{2^k} + 2c \right) \cdot M(2^l)$$

Finally obtain: Cost $\leq 8 \cdot \max \left\{ \frac{1}{1-2^{-\epsilon}} \cdot c \right\} \cdot \left(\frac{n}{m} + 1 \right) \cdot M(m) \quad \square$

$\begin{array}{c|c} U \\ \hline 0 \end{array}$ is in row echelon form. It's convenient to write $U = \begin{array}{c|c} E & U' \end{array} r \quad U' \in K^{r \times (n-r)}$

Also write $L = \begin{array}{c|c} m \\ \hline L_1 \\ \hline L_2 \end{array} r \quad m-r$

2.6.7 Theorem 7

Let $A \in K^{m \times n}$, $LQAP = \begin{array}{c|c} E & U' \\ \hline 0 \end{array} \begin{array}{c} r \\ m-r \end{array}$ as in Algorithm 5 then

(a) $rk(A) = r$

(b) The columns of $P \cdot \begin{array}{c|c} E^{-1} \cdot U' \\ \hline -I_{n-r} \end{array}$ form a basis of $\ker(A)$

(c) A linear system $Ax = b \quad b \in K^m$ is solvable iff $L_2 Q \cdot b = 0$

(d) if $Ax = b$ is solvable then $x = P \cdot \begin{array}{c|c} E^{-1} L_1 \\ \hline 0 \end{array} \begin{array}{c} r \\ n-r \end{array} \cdot Q \cdot P$ is a solution

$$\begin{aligned}
\text{(e) if } A \in GL_n(K) \text{ then } \det(A) &= \det(P) \cdot \underbrace{\det(E)}_{=\text{prod of diags}} \\
\text{and } A^{-1} &= P \cdot E^{-1} \cdot L
\end{aligned}$$

3 Notes

3.1 Notation

- $\mathbb{N} := \mathbb{N}_0$
- $\lg(x) := \log_2(x)$
- $a \mid b$ a is divisible by b \Leftrightarrow $b \bmod a = 0$
 $a \nmid b$ a is not divisible by b \Leftrightarrow $b \bmod a \neq 0$
- $ord(a)$ order of a group element
 $n > 0$ minimal such that $a^n = e$ with neutral element e
if no such n can be found, $ord(a) = \infty$
- $char(A)$ Characteristic: 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
polynomial rings measure for " $<$ " relations not the absolute value but max power.
- $lcm(a_1, \dots, a_n)$ "least common multiple of all a_i "
- \underline{e} = vector of e's
- $\phi(n) := |\{x \in \mathbb{N} : x < n \wedge \gcd(x, n) = 1\}| = |(\mathbb{Z}/(n))^x|$
Euler's totient function
- $rk(A)$ Rank of matrix A
- $\left(\frac{n}{p}\right) := \begin{cases} 1 & \text{if } p \mid n \\ -1 & \text{if } n \text{ is a square (mod } p) \\ 0 & \text{otherwise} \end{cases}$
Legendre symbol (this is not a fraction)
- $\left(\frac{n}{p}\right) = 1 \Leftrightarrow n^{\frac{p-1}{2}} = \left(\frac{n}{p}\right) \equiv 1 \pmod{p}$
Eulers criterion

3.2 Various stuff

- Lagrange's theorem
Every element in a finite group has finite order
- 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
- "Monte Carlo Algorithm"
Always terminates in reasonable time but might yield false result.

- "Las Vegas Algorithm"
If it terminates the result is correct. No deterministic running time.
- Chinese remainder theorem
Given a system of congruences $x \equiv a_i \pmod{m_i}$ with $i = 1, \dots, r$
 m_i pairwise co-prime. Then the unique solution is:
$$x \equiv a_1 \cdot b \cdot \frac{N}{m_i} + \dots + a_r \cdot b_r \cdot \frac{N}{m_r} \pmod{N} \quad \text{with } b_i \cdot \frac{N}{m_i} \equiv 1 \pmod{m_i}$$
- distance between two square numbers:
 $(n+1)^2 - n^2 = 2n + 1$
 \Rightarrow Squares are much more scarce than primes!
- $ax + by = c$ has solutions in \mathbb{Z} iff $\Leftrightarrow \gcd(a, b) \mid c$ with $a, x, b, y \in \mathbb{Z}$

3.3 Algebraic structures

- Group $(G, *)$
 - one inner operation $(*)$: $G \times G \mapsto G$
 - associativity: $(a * b) * c = a * (b * c) \quad \forall a, b, c \in G$
 - neutral element $e \in G$: $a * e = e * a = a \quad \forall a \in G$
 - inverse element $a^{-1} \in G$: $a * a^{-1} = a^{-1} * a = e \quad \forall a \in G$
- Abelian group $(G, *)$
 - $(G, *)$ is a group
 - commutativity: $a * b = b * a \quad \forall a, b \in G$
- Finite group $(G, *)$
 - associativity: $(a * b) * c = a * (b * c)$
 - unambiguity of reduction: $(a * x = a * x') \wedge (x * a = x' * a) \Rightarrow x = x'$
 $\Rightarrow x \mapsto x * a$ and $x \mapsto a * x$ is bijective
 $\Rightarrow \exists x : a * x = a \Rightarrow$ neutral element
 $\exists x : a * x = x \Rightarrow$ inverse element
- Cyclic group $(G, *)$
 - G is a group
 - G is generated by one Element: $G = \langle g \rangle = \{g^n \mid n \in \mathbb{Z}\}$
 - not necessarily finite.
- Semi group $(S, *)$
 - one inner operation $(*)$: $S \times S \mapsto S$
 - associativity: $(a * b) * c = a * (b * c) \quad \forall a, b, c \in S$
- Field $(K, +, \cdot)$
 - two inner operations $(+, \cdot)$ such that:
 - $(K, +)$ is an abelian group with neutral element 0
 - $(K \setminus \{0\}, \cdot)$ is an abelian group with neutral element 1
 - distributivity: $a \cdot (b + c) = a \cdot b + a \cdot c$
 $(a + b) \cdot c = a \cdot c + b \cdot c \quad \forall a, b, c \in K$
- General linear group $GL_n(K)$
 - K is a field
 - $GL_n(K)$ is the set of $n \times n$ invertible matrices with ordinary matrix multiplication

- Ring $(R, +, \cdot)$
 - $(R, +)$ is an abelian group
 - (R, \cdot) is a semi group
 - distributivity: $a \cdot (b + c) = a \cdot b + a \cdot c$
 $(a + b) \cdot c = a \cdot c + b \cdot c \quad \forall a, b, c \in R$
- Commutative ring $(R, +, \cdot)$
 - $(R, +, \cdot)$ is a ring
 - commutativity for (\cdot) $a \cdot b = b \cdot a \quad \forall a, b \in R$
- Unitary ring (ring with 1) $(R, +, \cdot)$
 - (R, \cdot) is a semi group
 - (R, \cdot) has a neutral element "1"
- Euclidean ring R
 - $\exists F : R \mapsto \mathbb{N}_0 \cup \{0\}$
such that if $\exists q, r \in R \quad a = b \cdot q + r$ and $r = 0$ or $a, b \in R \quad F(r) < F(b)$
- Polynomial ring $R[X]$
 - R is a commutative unitary ring
 - set of all polynomials with coefficients $\in R$

3.4 Invertible elements

- Let $(\mathbb{Z}/(n), +)$ be a group or $(\mathbb{Z}/(n))^\times$ be a group with multiplication.
- $|(\mathbb{Z}/(n))^\times| = \phi(n)$
- $n \in \mathbb{P}$
 $\Rightarrow (\mathbb{Z}/(n))^\times = \{\bar{0}, \dots, \bar{p-1}\} \cong (\mathbb{Z}/(p-1), +) = Z_{p-1}$ (cyclic Group Z)
- n is a power of 2
 $\Rightarrow (\mathbb{Z}/(2^e))^\times \cong \mathbb{Z}/(2) \times \mathbb{Z}/(2^{e-2})$
- n is a power of an odd Prime
 $\Rightarrow (\mathbb{Z}/(p^k))^\times \cong \mathbb{Z}/(p^{k-1} \cdot (p-1)) \cong Z_{(p^{k-1} \cdot (p-1))}$
- $n = p_1^{k_1} \cdot \dots \cdot p_r^{k_r}$
 $\Rightarrow (\mathbb{Z}/(n))^\times \cong (\mathbb{Z}/(p_1^{k_1}))^\times \times \dots \times (\mathbb{Z}/(p_r^{k_r}))^\times$