Computing with large integers

Jean-Sébastien Coron

University of Luxembourg

Summary

- Basic algorithms for computing with large integers
  - Addition, subtraction, multiplication, division with reminder
  - Modular exponentiation
- Probabilistic primality testing
  - How to generate large primes efficiently for RSA

Computing with large integers

- Limited precision by word size of CPU
  - 32 bits or 64 bits. Computing with values $< 2^{32}$ or $< 2^{64}$
- Computing with large integers :
  - One represents the big integers in base B in an array, with a bit sign.

$$a = \quad \pm \; \boxed{}\boxed{}\boxed{}\boxed{} \; \cdots \; \boxed{}\boxed{}\boxed{}$$

  - One implements addition, multiplication, division on such arrays.
- Existing libraries :
  - GMP: `www.swox.com/gmp`
  - NTL: `www.shoup.net`
  - Some parts written in assembly for better efficiency.

Representation of large integers

- Representing large integers :
  - An integer is represented as an array of digits in base B, with a sign bit.
    $$a = \pm \sum_{i=0}^{k-1} a_i B^i = \pm (a_{k-1} \dots a_0)_B$$
    with $0 \leq a_i < B$.
  - If $a \neq 0$, we must have $a_{k-1} \neq 0$.
- Choice of B
  - One generally takes $B = 2^v$ for some v.

Algorithms for large integers

- Here we describe algorithms for positive integers
  - Can be easily adapted to signed integers
- Low-level arithmetic operations
  - We assume that our programming language can do low-level addition, subtraction, multiplication and integer division
  - with integers of absolute value $< B^2$.

$$a = \pm \sum_{i=0}^{k-1} a_i B^i = \pm (a_{k-1} \ldots a_0)_B$$

- Example: C programming language
  - With type `unsigned long int` on a 64-bit computer, take $B = 2^{32}$
  - More efficient implementations are possible

Algorithms for large integers

- Here we describe algorithms for positive integers
  - Can be easily adapted to signed integers
- Low-level arithmetic operations
  - We assume that our programming language can do low-level addition, subtraction, multiplication and integer division
  - with integers of absolute value $< B^2$.

$$a = \pm \sum_{i=0}^{k-1} a_i B^i = \pm (a_{k-1} \ldots a_0)_B$$

- Example: C programming language
  - With type `unsigned long int` on a 64-bit computer, take $B = 2^{32}$
  - More efficient implementations are possible

Addition

- Computing $c = a + b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$. Let $c = (c_k c_{k-1} \dots c_0)$

    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
    $\quad tmp \leftarrow a_i + b_i + carry$
    $\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    for $i = \ell$ to $k - 1$ do
    $\quad tmp \leftarrow a_i + carry$
    $\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    $c_k \leftarrow carry$
- In every loop iteration
  - $0 \leq tmp \leq 2B - 1$, $carry \in \{0, 1\}$.
- Complexity: $\mathcal{O}(k)$

Addition

- Computing $c = a + b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Let $c = (c_k c_{k-1} \dots c_0)$

    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
      $tmp \leftarrow a_i + b_i + carry$
      $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    for $i = \ell$ to $k - 1$ do
      $tmp \leftarrow a_i + carry$
      $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    $c_k \leftarrow carry$
- In every loop iteration
  - $0 \leq tmp \leq 2B - 1$, $carry \in \{0, 1\}$.
- Complexity: $\mathcal{O}(k)$

Addition: example in base $B = 10$

$\rightarrow$ $carry \leftarrow 0$

for $i = 0$ to $\ell - 1$ do

$tmp \leftarrow a_i + b_i + carry$

$carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$

for $i = \ell$ to $k - 1$ do

$tmp \leftarrow a_i + carry$

$carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$

$c_k \leftarrow carry$

a_i		6	4	7	$k = 3$
b_i			8	5	$\ell = 2$
c_i					

i $\boxed{}$ tmp $\boxed{}$ $carry$ $\boxed{0}$

Addition: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
$\rightarrow$ $\quad tmp \leftarrow a_i + b_i + carry$
$\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
for $i = \ell$ to $k - 1$ do
$\quad tmp \leftarrow a_i + carry$
$\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
$c_k \leftarrow carry$

$\downarrow$

a_i		6	4	7		$k = 3$
b_i			8	5		$\ell = 2$
c_i						

i $\boxed{0}$ $\quad tmp$ $\boxed{12}$ $\quad carry$ $\boxed{0}$

Addition: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
  $tmp \leftarrow a_i + b_i + carry$
$\rightarrow$ $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
for $i = \ell$ to $k - 1$ do
  $tmp \leftarrow a_i + carry$
  $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
$c_k \leftarrow carry$

$\downarrow$

a_i		6	4	7	$k = 3$
b_i			8	5	$\ell = 2$
c_i				2	

i $\boxed{0}$ tmp $\boxed{12}$ $carry$ $\boxed{1}$

Addition: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
$\rightarrow$ $\quad tmp \leftarrow a_i + b_i + carry$
$\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
for $i = \ell$ to $k - 1$ do
$\quad tmp \leftarrow a_i + carry$
$\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
$c_k \leftarrow carry$

$\downarrow$

a_i		6	4	7	$k = 3$
b_i			8	5	$\ell = 2$
c_i				2	

i $\boxed{1}$ tmp $\boxed{13}$ $carry$ $\boxed{1}$

Addition: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i + b_i + carry$
$\rightarrow$ $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
for $i = \ell$ to $k - 1$ do
    $tmp \leftarrow a_i + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
$c_k \leftarrow carry$

			$\downarrow$		
a_i		6	4	7	$k = 3$
b_i			8	5	$\ell = 2$
c_i			3	2	

i $\boxed{1}$ tmp $\boxed{13}$ $carry$ $\boxed{1}$

Addition: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
  $tmp \leftarrow a_i + b_i + carry$
  $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
for $i = \ell$ to $k - 1$ do
$\rightarrow$ $tmp \leftarrow a_i + carry$
  $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
$c_k \leftarrow carry$

$\downarrow$

a_i		6	4	7		$k = 3$
b_i			8	5		$\ell = 2$
c_i			3	2		

i $\boxed{2}$ tmp $\boxed{7}$ $carry$ $\boxed{1}$

Addition: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
  $tmp \leftarrow a_i + b_i + carry$
  $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
for $i = \ell$ to $k - 1$ do
  $tmp \leftarrow a_i + carry$
$\rightarrow$ $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
$c_k \leftarrow carry$

$\downarrow$

a_i		6	4	7	$k = 3$
b_i			8	5	$\ell = 2$
c_i		7	3	2	

i $\boxed{2}$ tmp $\boxed{7}$ $carry$ $\boxed{0}$

Addition: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
  $tmp \leftarrow a_i + b_i + carry$
  $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
for $i = \ell$ to $k - 1$ do
  $tmp \leftarrow a_i + carry$
  $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
$\rightarrow$ $c_k \leftarrow carry$

$\downarrow$

a_i		6	4	7		$k = 3$
b_i			8	5		$\ell = 2$
c_i	0	7	3	2		

i $\boxed{}$ tmp $\boxed{7}$ $carry$ $\boxed{0}$

Subtraction

- Same algorithm as addition, with $a_i + b_i$ replaced by $a_i - b_i$
- Computing $c = a - b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Let $c = (c_k c_{k-1} \dots c_0)$
    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
    $\quad tmp \leftarrow a_i - b_i + carry$
    $\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    for $i = \ell$ to $k - 1$ do
    $\quad tmp \leftarrow a_i + carry$
    $\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    $c_k \leftarrow carry$
- In every loop iteration
  - $-B \leq tmp \leq B - 1$, $carry \in \{-1, 0\}$.
- If $a \geq b$ then $c_k = 0$, otherwise $c_k = -1$.
  - If $c_k = -1$, compute $c' = b - a$ and let $c := -c'$.

Subtraction

- Same algorithm as addition, with $a_i + b_i$ replaced by $a_i - b_i$
- Computing $c = a - b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Let $c = (c_k c_{k-1} \dots c_0)$
    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
      $tmp \leftarrow a_i - b_i + carry$
      $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    for $i = \ell$ to $k - 1$ do
      $tmp \leftarrow a_i + carry$
      $carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    $c_k \leftarrow carry$
- In every loop iteration
  - $-B \leq tmp \leq B - 1$, $carry \in \{-1, 0\}$.
- If $a \geq b$ then $c_k = 0$, otherwise $c_k = -1$.
  - If $c_k = -1$, compute $c' = b - a$ and let $c := -c'$.

Subtraction

- Same algorithm as addition, with $a_i + b_i$ replaced by $a_i - b_i$
- Computing $c = a - b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Let $c = (c_k c_{k-1} \dots c_0)$
    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
    $\quad tmp \leftarrow a_i - b_i + carry$
    $\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    for $i = \ell$ to $k - 1$ do
    $\quad tmp \leftarrow a_i + carry$
    $\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_i \leftarrow tmp \bmod B$
    $c_k \leftarrow carry$
- In every loop iteration
  - $-B \leq tmp \leq B - 1$, $carry \in \{-1, 0\}$.
- If $a \geq b$ then $c_k = 0$, otherwise $c_k = -1$.
  - If $c_k = -1$, compute $c' = b - a$ and
    let $c := -c'$.

Multiplication

- Schoolbook method

$$\begin{array}{cccccc} & & & 5 & 3 & 2 \\ \times & & & 8 & 3 & 5 \\ \hline & & 2 & 6 & 6 & 0 \\ & 1 & 5 & 9 & 6 & \\ 4 & 2 & 5 & 6 & & \\ \hline 4 & 4 & 4 & 2 & 2 & 0 \end{array}$$

- Drawback: storage of intermediate results
  - Space complexity $\mathcal{O}(n^2)$ for n digits
- We can do much better by accumulating the intermediate results

Multiplication

- Computing $c = a \cdot b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k, \ell \geq 1$. Let $c = (c_{k+\ell-1} \dots c_0)$

    $carry \leftarrow 0$
    for $i = 0$ to $k + \ell - 1$ do
      $c_i \leftarrow 0$
    for $i = 0$ to $k - 1$ do
      $carry \leftarrow 0$
      for $j = 0$ to $\ell - 1$ do
        $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
        $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
      $c_{i+\ell} \leftarrow carry$
- In every loop iteration
  - $0 \leq tmp \leq B^2 - 1$, $0 \leq carry \leq B - 1$.
- Complexity: $\mathcal{O}(k \cdot \ell)$

Multiplication

- Computing $c = a \cdot b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k, \ell \geq 1$. Let $c = (c_{k+\ell-1} \dots c_0)$

    $carry \leftarrow 0$
    for $i = 0$ to $k + \ell - 1$ do
        $c_i \leftarrow 0$
    for $i = 0$ to $k - 1$ do
        $carry \leftarrow 0$
        for $j = 0$ to $\ell - 1$ do
            $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
            $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
        $c_{i+\ell} \leftarrow carry$
- In every loop iteration
  - $0 \leq tmp \leq B^2 - 1$, $0 \leq carry \leq B - 1$.
- Complexity: $\mathcal{O}(k \cdot \ell)$

Multiplication: example in base $B = 10$

$\rightarrow$ $carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7	$k = 2$
b_i			8	5	$\ell = 2$
c_i					

i [ ] j [ ] tmp [ ] $carry$ [0]

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
$\rightarrow$ for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7	$k = 2$
b_i			8	5	$\ell = 2$
c_i	0	0	0	0	

i $\boxed{}$ j $\boxed{}$ tmp $\boxed{}$ $carry$ $\boxed{0}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
$\rightarrow$ $\quad carry \leftarrow 0$
$\quad$ for $j = 0$ to $\ell - 1$ do
$\quad\quad tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
$\quad\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
$\quad c_{i+\ell} \leftarrow carry$

a_i			3	7 ↓		$k = 2$
b_i			8	5		$\ell = 2$
c_i	0	0	0	0		

i $\boxed{0}$ $\quad j$ $\boxed{}$ $\quad tmp$ $\boxed{}$ $\quad carry$ $\boxed{0}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
$\rightarrow$ $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7 ↓		$k = 2$
b_i			8	5 ↓		$\ell = 2$
c_i	0	0	0	0		

i $\boxed{0}$ j $\boxed{0}$ tmp $\boxed{35}$ $carry$ $\boxed{0}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
$\rightarrow$ $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i $\boxed{3}\boxed{7}$ $k = 2$

b_i $\boxed{8}\boxed{5}$ $\ell = 2$

c_i $\boxed{0}\boxed{0}\boxed{0}\boxed{5}$

i $\boxed{0}$ j $\boxed{0}$ tmp $\boxed{35}$ $carry$ $\boxed{3}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
$\rightarrow$ $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7 $\downarrow$		$k = 2$
b_i			8 $\downarrow$	5		$\ell = 2$
c_i	0	0	0	5		

i $\boxed{0}$ j $\boxed{1}$ tmp $\boxed{59}$ $carry$ $\boxed{3}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
$\rightarrow$ $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7	$k = 2$
b_i			8	5	$\ell = 2$
c_i	0	0	9	5	

i $\boxed{0}$ j $\boxed{1}$ tmp $\boxed{59}$ $carry$ $\boxed{5}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
$\rightarrow$ $c_{i+\ell} \leftarrow carry$

a_i			3	7 ↓		$k = 2$
b_i			8	5		$\ell = 2$
c_i	0	5	9	5		

i $\boxed{0}$ j $\boxed{}$ tmp $\boxed{59}$ $carry$ $\boxed{5}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
$\rightarrow$ $\quad carry \leftarrow 0$
$\quad$ for $j = 0$ to $\ell - 1$ do
$\quad\quad tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
$\quad\quad carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
$\quad c_{i+\ell} \leftarrow carry$

a_i			$\downarrow$ 3	7		$k = 2$
b_i			8	5		$\ell = 2$
c_i	0	5	9	5		

i $\boxed{1}$ $\quad j$ $\boxed{}$ $\quad tmp$ $\boxed{59}$ $\quad carry$ $\boxed{0}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
→ $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7		$k = 2$
b_i			8	5		$\ell = 2$
c_i	0	5	9	5		

i $\boxed{1}$ j $\boxed{0}$ tmp $\boxed{24}$ $carry$ $\boxed{0}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
$\rightarrow$ $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

$$\begin{array}{lcccccl}
 & & & \downarrow & & & \\
a_i & & & \boxed{3} & \boxed{7} & & k = 2 \\
 & & & & \downarrow & & \\
b_i & & & \boxed{8} & \boxed{5} & & \ell = 2 \\
c_i & \boxed{0} & \boxed{5} & \boxed{4} & \boxed{5} & &
\end{array}$$

i $\boxed{1}$ j $\boxed{0}$ tmp $\boxed{24}$ $carry$ $\boxed{2}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
$\rightarrow$ $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7		$k = 2$
b_i			8	5		$\ell = 2$
c_i	0	5	4	5		

i $\boxed{1}$ j $\boxed{1}$ tmp $\boxed{31}$ $carry$ $\boxed{2}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
$\rightarrow$ $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
  $c_{i+\ell} \leftarrow carry$

a_i			3	7	$k = 2$
b_i			8	5	$\ell = 2$
c_i	0	1	4	5	

i $\boxed{1}$ j $\boxed{1}$ tmp $\boxed{31}$ $carry$ $\boxed{3}$

Multiplication: example in base $B = 10$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow \lfloor tmp/B \rfloor$; $c_{i+j} \leftarrow tmp \bmod B$
$\rightarrow$ $c_{i+\ell} \leftarrow carry$

a_i			3	7		$k = 2$
b_i			8	5		$\ell = 2$
c_i	3	1	4	5		

i $\boxed{1}$ j $\boxed{}$ tmp $\boxed{31}$ $carry$ $\boxed{3}$

Division with remainder

- Euclidean division
  - Given $a \geq 0$ and $b > 0$, compute q and r such that

$$a = b \cdot q + r, \quad 0 \leq r < b$$

- Algorithm overview

  Input: $a = (a_{k-1} \ldots a_0)_B$ and $b = (b_{\ell-1} \ldots b_0)_B$ with $b_{\ell-1} \neq 0$.
  Output: $q = (q_{m-1} \ldots q_0)_B$ with $m := k - \ell + 1$, and r
  $r \leftarrow a$
  for $i = m - 1$ downto 0 do
  $\quad q_i \leftarrow \lfloor r/(B^i b) \rfloor$
  $\quad r \leftarrow r - B^i \cdot q_i \cdot b$
  output (q, r)

Division with reminder

- Euclidean division:
  Input: $a = (a_{k-1} \dots a_0)_B$ and $b = (b_{\ell-1} \dots b_0)_B$ with $b_{\ell-1} \neq 0$.
  Output: $q = (q_{m-1} \dots q_0)_B$ with $m := k - \ell + 1$, and r
  $r \leftarrow a$
  for $i = m - 1$ downto 0 do
  $\quad q_i \leftarrow \lfloor r/(B^i b) \rfloor$
  $\quad r \leftarrow r - B^i \cdot q_i \cdot b$
  output (q, r)
- Property
  - One can show inductively that $0 \leq r < B^i \cdot b$ after step i
  - Therefore, $0 \leq r < b$ eventually.

Division with reminder

- Euclidean division:

  Input: $a = (a_{k-1} \dots a_0)_B$ and $b = (b_{\ell-1} \dots b_0)_B$ with $b_{\ell-1} \neq 0$.
  Output: $q = (q_{m-1} \dots q_0)_B$ with $m := k - \ell + 1$, and r

  $r \leftarrow a$
  for $i = m - 1$ downto 0 do
  $\quad q_i \leftarrow \lfloor r/(B^i b) \rfloor$
  $\quad r \leftarrow r - B^i \cdot q_i \cdot b$
  output (q, r)
- How to compute $q_i = \lfloor r/(B^i \cdot b) \rfloor$
  - Test all possible values of $0 \leq q_i < B$
  - Not efficient, except if B is small.
  - Possible to do much better, by predicting q_i from the most significant digits of r and b;
    see Shoup's book.
- *Binary* Euclidean division algorithm
  - We assume $B = 2^v$ and first convert a, b to binary representation ($B = 2$)

Division with reminder

- Euclidean division:

  Input: $a = (a_{k-1} \dots a_0)_B$ and $b = (b_{\ell-1} \dots b_0)_B$ with $b_{\ell-1} \neq 0$.
  Output: $q = (q_{m-1} \dots q_0)_B$ with $m := k - \ell + 1$, and r
  $r \leftarrow a$
  for $i = m - 1$ downto 0 do
  $\quad q_i \leftarrow \lfloor r/(B^i b) \rfloor$
  $\quad r \leftarrow r - B^i \cdot q_i \cdot b$
  output (q, r)
- How to compute $q_i = \lfloor r/(B^i \cdot b) \rfloor$
  - Test all possible values of $0 \leq q_i < B$
  - Not efficient, except if B is small.
  - Possible to do much better, by predicting q_i from the most significant digits of r and b;
    see Shoup's book.
- *Binary* Euclidean division algorithm
  - We assume $B = 2^v$ and first convert
    a, b to binary representation $(B = 2)$

Binary Euclidean division

- Input: $a = (a_{k-1} \dots a_0)_2$ and $b = (b_{\ell-1} \dots b_0)_2$ with $a \geq b > 0$ and $b_{\ell-1} = 1$.
  Output: (q, r)

  $q \leftarrow 0$, $r \leftarrow a$, $c \leftarrow 2^{\max(0,k-\ell)} \cdot b$
  for $i = 0$ to $\max(0, k-\ell)$ do
      $q \leftarrow 2 \cdot q$
      if $r \geq c$ then
          $r \leftarrow r - c$
          $q \leftarrow q + 1$
      $c \leftarrow c/2$
  Return (q, r)
- Complexity: $\mathcal{O}(\ell \cdot (k - \ell + 1))$

Binary Euclidean division

- Input: $a = (a_{k-1} \dots a_0)_2$ and $b = (b_{\ell-1} \dots b_0)_2$ with $a \geq b > 0$ and $b_{\ell-1} = 1$.
  Output: (q, r)

  $q \leftarrow 0$, $r \leftarrow a$, $c \leftarrow 2^{\max(0,k-\ell)} \cdot b$
  for $i = 0$ to $\max(0, k - \ell)$ do
      $q \leftarrow 2 \cdot q$
      if $r \geq c$ then
          $r \leftarrow r - c$
          $q \leftarrow q + 1$
      $c \leftarrow c/2$
  Return (q, r)

- Complexity: $\mathcal{O}(\ell \cdot (k - \ell + 1))$

Summary

- For $a \in \mathbb{Z}$, let $\mathrm{len}(a)$ be the number of bits in the binary representation of $|a|$:
  - $\mathrm{len}(a) = \lfloor \log_2 |a| \rfloor + 1$ if $a \neq 0$
  - $\mathrm{len}(0) = 1$

$$2^{\mathrm{len}(a)-1} \leq a < 2^{\mathrm{len}(a)} \text{ for } a > 0$$

- Let a and b be two arbitrary integers
  - We can compute $a \pm b$ in time $\mathcal{O}(\mathrm{len}(a) + \mathrm{len}(b))$
  - We can compute $a \cdot b$ in time $\mathcal{O}(\mathrm{len}(a)\,\mathrm{len}(b))$
  - We can compute the quotient q and the remainder r in $a = b \cdot q + r$ in time $\mathcal{O}(\mathrm{len}(b)\,\mathrm{len}(q))$

Summary

- For $a \in \mathbb{Z}$, let $\text{len}(a)$ be the number of bits in the binary representation of $|a|$:
  - $\text{len}(a) = \lfloor \log_2 |a| \rfloor + 1$ if $a \neq 0$
  - $\text{len}(0) = 1$

$$2^{\text{len}(a)-1} \leq a < 2^{\text{len}(a)} \text{ for } a > 0$$

- Let a and b be two arbitrary integers
  - We can compute $a \pm b$ in time $\mathcal{O}(\text{len}(a) + \text{len}(b))$
  - We can compute $a \cdot b$ in time $\mathcal{O}(\text{len}(a)\,\text{len}(b))$
  - We can compute the quotient q and the remainder r in $a = b \cdot q + r$ in time $\mathcal{O}(\text{len}(b)\,\text{len}(q))$

Modular exponentiation

- We want to compute $c = a^b \pmod{n}$.
  - Example: RSA
    - $c = m^e \pmod{n}$ where m is the message, e the public exponent, and n the modulus.
- Naive method:
  - Multiplying a in total b times by itself modulo n
  - Very slow: if b is 100 bits, roughly 2^{100} multiplications !
- Example: compute $b = a^{16} \pmod{n}$
  - $b = a \cdot a \cdot \ldots \cdot a \cdot a \pmod{n}$: 15 multiplications
  - $b = (((a^2)^2)^2)^2 \pmod{n}$: 4 multiplications

Square and multiply algorithm

- Let $b = (b_{\ell-1} \dots b_0)_2$ the binary representation of b

$$b = \sum_{i=0}^{\ell-1} b_i \cdot 2^i$$

- Square and multiply algorithm :
  - Input : a, b and n
  - Output : $a^b \pmod n$
  - $c \leftarrow 1$

    for $i = \ell - 1$ down to 0 do

        $c \leftarrow c^2 \pmod n$

        if $b_i = 1$ then $c \leftarrow c \cdot a \pmod n$

    Output c
- Complexity: $\mathcal{O}(\mathrm{len}(n)^3)$

Analysis

- Let B_i be the integer with binary representation $(b_{\ell-1}\ldots b_i)_2$, and let

$$c_i = a^{B_i} \pmod{n}$$

- Initialization

$$\left\{\begin{array}{lcl} B_\ell & = & 0 \\ c_\ell & = & 1 \end{array}\right.$$

- Recursive step

$$\left\{\begin{array}{lcl} B_i & = & 2 \cdot B_{i+1} + b_i \\ c_i & = & (c_{i+1})^2 \cdot a^{b_i} \pmod{n} \end{array}\right.$$

- Final step

$$\left\{\begin{array}{lcl} B_0 & = & b \\ c_0 & = & a^b \pmod{n} \end{array}\right.$$

Computing in $\mathbb{Z}_n$

- Computing $a + b \bmod n$
  - First compute $a + b$ in $\mathbb{Z}$, then reduce modulo n
  - Complexity: $\mathcal{O}(\mathrm{len}(n))$
- Computing $a \cdot b \bmod n$
  - First compute $a \cdot b$ in $\mathbb{Z}$, then reduce modulo n
  - Complexity: $\mathcal{O}(\mathrm{len}(n)^2)$
- Computing $a^b \bmod n$
  - Complexity: $\mathcal{O}(\mathrm{len}(n)^3)$

Primality Testing

- Motivation for prime generation:
  - Generate the primes p and q in RSA.
  - p and q must be large: at least 512 bits.
- Goal of primality testing:
  - Given an integer n, determine whether n is prime or composite.
- Simplest algorithm: trial division.
  - Test if n is divisible by 2, 3, 4, 5,... We can stop at $\sqrt{n}$.
  - Algorithm determines if n is prime or composite, and outputs the factors of n if n is composite.
  - Very inefficient algorithm
    - Requires $\simeq \sqrt{n}$ arithmetic operations.
    - If n has 256 bits, then 2^{128} arithmetic operations. If 2^{30} operations/s, this takes 10^{22} years !

Primality Testing

- Motivation for prime generation:
  - Generate the primes p and q in RSA.
  - p and q must be large: at least 512 bits.
- Goal of primality testing:
  - Given an integer n, determine whether n is prime or composite.
- Simplest algorithm: trial division.
  - Test if n is divisible by 2, 3, 4, 5,... We can stop at $\sqrt{n}$.
  - Algorithm determines if n is prime or composite, and outputs the factors of n if n is composite.
  - Very inefficient algorithm
    - Requires $\simeq \sqrt{n}$ arithmetic operations.
    - If n has 256 bits, then 2^{128} arithmetic operations. If 2^{30} operations/s, this takes 10^{22} years !

Probabilistic primality testing

- Goal: describe an efficient probabilistic primality test.
  - Can test primality for a 512-bit integer n in less than a second.
- Probabilistic primality testing.
  - The algorithm does not find the prime factors of n when n is composite.
  - The algorithm may make a mistake (pretend that an integer n is prime whereas it is composite).
  - But the mistake can be made arbitrarily small (*e.g.* $< 2^{-100}$), so this makes no difference in practice.

Distribution of prime numbers

- Let $\pi(x)$ be the number of primes in the interval $[2, x]$.
- Theorem (Prime number theorem)
  - $\pi(x) \sim x/\log x$.
- Consequence:
  - A random integer between 2 and x is prime with probability $\simeq 1/\log x$
  - A random n-bit integer is prime with probability

$$\frac{1}{\log 2} \cdot \frac{1}{n}$$

  - Prime numbers are relatively frequent

The Fermat test

- Fermat's little theorem
  - If n is prime and a is an integer between 1 and $n-1$, then $a^{n-1} \equiv 1 \pmod{n}$.
  - Therefore, if the primality of n is unknown, finding $a \in [1, n-1]$ such that $a^{n-1} \not\equiv 1 \pmod{n}$ proves that n is composite.
- Fermat primality test with security parameter t.

```
For i = 1 to t do
    Choose a random a ∈ [2, n − 2]
    Compute r = a^(n−1) mod n
    If r ≠ 1 then return "composite"
Return "prime'
```

- Complexity: $\mathcal{O}(t \cdot \log^3 n)$

The Fermat test

- Fermat's little theorem
  - If n is prime and a is an integer between 1 and $n-1$, then $a^{n-1} \equiv 1 \pmod{n}$.
  - Therefore, if the primality of n is unknown, finding $a \in [1, n-1]$ such that $a^{n-1} \not\equiv 1 \pmod{n}$ proves that n is composite.
- Fermat primality test with security parameter t.

```
For i = 1 to t do
    Choose a random a ∈ [2, n − 2]
    Compute r = a^(n−1) mod n
    If r ≠ 1 then return "composite"
Return "prime'
```

- Complexity: $\mathcal{O}(t \cdot \log^3 n)$

Analysis of Fermat's test

- Let $L_n = \{a \in [1, n-1] : a^{n-1} \equiv 1 \pmod{n}\}$
- Theorem:
  - If n is prime, then $L_n = \mathbb{Z}_n^*$. If n is composite and $L_n \subsetneq \mathbb{Z}_n^*$, then $|L_n| \leq (n-1)/2$.
- Proof:
  - If n is prime, $L_n = \mathbb{Z}_n^*$ from Fermat.
  - If n is composite, since L_n is a sub-group of $\mathbb{Z}_n^*$ and the order of a subgroup divides the order of the group, $|\mathbb{Z}_n^*| = m \cdot |L_n|$ for some integer m, with $m > 1$ since by assumption $L_n \subsetneq \mathbb{Z}_n^*$

$$|L_n| = \frac{1}{m}|\mathbb{Z}_n^*| \leq \frac{1}{2}|\mathbb{Z}_n^*| \leq \frac{n-1}{2}$$

Analysis of Fermat's test

- If n is composite and $L_n \subsetneq \mathbb{Z}_n^*$
  - then $a^{n-1} = 1 \pmod n$ with probability at most $1/2$ for a random $a \in [2, n-2]$.
  - The algorithm outputs "prime" wih probability at most 2^{-t}.
- Unfortunately, there are odd composite numbers n such that $L_n = \mathbb{Z}_n^*$.
  - Such numbers are called Carmichael numbers. The smallest Carmichael number is 561.
  - Carmichael numbers are rare, but there are an infinite number of them, so we cannot ignore them.

Analysis of Fermat's test

- If n is composite and $L_n \subsetneq \mathbb{Z}_n^*$
  - then $a^{n-1} = 1 \pmod{n}$ with probability at most $1/2$ for a random $a \in [2, n-2]$.
  - The algorithm outputs "prime" wih probability at most 2^{-t}.
- Unfortunately, there are odd composite numbers n such that $L_n = \mathbb{Z}_n^*$.
  - Such numbers are called Carmichael numbers. The smallest Carmichael number is 561.
  - Carmichael numbers are rare, but there are an infinite number of them, so we cannot ignore them.

- The Miller-Rabin test is a variant of Fermat test with a different L_n. Write $n-1=m2^h$ for odd m.

$$L'_n = \{a \in \mathbb{Z}_n^* : a^{m2^h} = 1 \text{ and } \text{for } j=0,\ldots,h-1,\ a^{m2^{j+1}} = 1 \text{ implies } a^{m2^j} = \pm 1\}$$

- Illustration for $a \in L'_n$

j	0	1	2	$\cdots$	$h-2$	$h-1$	h
	1	1	1	$\cdots$	1	1	1
a^{m2^j}	-1	1	1	$\cdots$	1	1	1
	X	-1	1	$\cdots$	1	1	1
	X	X	X	$\cdots$	X	-1	1

- Equivalently

$$L'_n = \{a \in \mathbb{Z}_n^* : a^m = 1 \text{ or } a^{m2^j} = -1 \text{ for some } 0 \le j \le h-1\}$$

The Miller-Rabin test

- The Miller-Rabin test is a variant of Fermat test with a different L_n. Write $n-1 = m2^h$ for odd m.

$$L'_n = \{a \in \mathbb{Z}_n^* : a^{m2^h} = 1 \text{ and } \text{for } j = 0, \dots, h-1,\ a^{m2^{j+1}} = 1 \text{ implies } a^{m2^j} = \pm 1\}$$

- Illustration for $a \in L'_n$

j	0	1	2	$\cdots$	$h-2$	$h-1$	h
	1	1	1	$\cdots$	1	1	1
a^{m2^j}	-1	1	1	$\cdots$	1	1	1
	X	-1	1	$\cdots$	1	1	1
	X	X	X	$\cdots$	X	-1	1

- Equivalently

$$L'_n = \{a \in \mathbb{Z}_n^* : a^m = 1 \text{ or } a^{m2^j} = -1 \text{ for some } 0 \le j \le h-1\}$$

$$L'_n = \{a \in \mathbb{Z}_n^* : a^{m2^h} = 1 \text{ and } \text{for } j = 0, \ldots, h-1,\ a^{m2^{j+1}} = 1 \text{ implies } a^{m2^j} = \pm 1\}$$

where $n - 1 = m2^h$ for odd m.

- Theorem
  - If n is prime, then $L'_n = \mathbb{Z}_n^*$
  - If n is composite, then $|L'_n| \leq (n-1)/4$
- Proof for n prime
  - Let $a \in \mathbb{Z}_n^*$. By Fermat, $a^{m \cdot 2^h} = a^{n-1} = 1 \pmod n$
  - If $a^{m2^{j+1}} = 1$ for some $0 \leq j \leq h-1$, let $\beta = a^{m2^j}$. Since $\beta^2 = a^{m2^{j+1}} = 1$, then $\beta = \pm 1$.
    - because a polynomial of degree d has at most d roots modulo a prime.
  - Therefore $a \in L'_n$.

Miller-Rabin test

$$L'_n = \{a \in \mathbb{Z}_n^* : a^{m2^h} = 1 \text{ and } \text{for } j = 0, \ldots, h-1,\ a^{m2^{j+1}} = 1 \text{ implies } a^{m2^j} = \pm 1\}$$

where $n - 1 = m2^h$ for odd m.

- Theorem
  - If n is prime, then $L'_n = \mathbb{Z}_n^*$
  - If n is composite, then $|L'_n| \leq (n-1)/4$
- Proof for n prime
  - Let $a \in \mathbb{Z}_n^*$. By Fermat, $a^{m \cdot 2^h} = a^{n-1} = 1 \pmod{n}$
  - If $a^{m2^{j+1}} = 1$ for some $0 \leq j \leq h-1$, let $\beta = a^{m2^j}$. Since $\beta^2 = a^{m2^{j+1}} = 1$, then $\beta = \pm 1$.
    - because a polynomial of degree d has at most d roots modulo a prime.
  - Therefore $a \in L'_n$.

The Miller-Rabin test

Algorithm 1 Testing whether $\alpha \in L'_n$

1. Write $n - 1 = 2^h \cdot m$ for odd m.
2. $\beta \leftarrow \alpha^m$
3. **if** $\beta = 1$ then **return** true
4. **for** $j = 1$ to $h - 1$ **do**
5.   **if** $\beta = -1$ then **return** true
6.   **if** $\beta = +1$ then **return** false
7. $\beta \leftarrow \beta^2$
8. **end for**
9. **return** false

Algorithm 2 Miller-Rabin test of primality

Input: An odd integer n, and $t \in \mathbb{Z}$.

1. **repeat** t times
2.   Generate a random $\alpha \in \mathbb{Z}_n$
3.   **if** $\alpha \notin L'_n$ **return** false
4. **return** true

The Miller-Rabin test

Algorithm 3 Testing whether $\alpha \in L'_n$

Write $n - 1 = 2^h \cdot m$ for odd m.
$\beta \leftarrow \alpha^m$
if $\beta = 1$ then **return** true
for $j = 1$ to $h - 1$ **do**
    if $\beta = -1$ then **return** true
    if $\beta = +1$ then **return** false
    $\beta \leftarrow \beta^2$
end for
return false

Algorithm 4 Miller-Rabin test of primality

Input: An odd integer n, and $t \in \mathbb{Z}$.
repeat t times
    Generate a random $\alpha \in \mathbb{Z}_n$
    if $\alpha \notin L'_n$ **return** false
return true

The Miller-Rabin test

- Property
  - If n is prime, then the Miller-Rabin test always declares n as prime.
  - If $n \geq 3$ is composite, then the probability that the Miller-Rabin test outputs "prime" is less than $\left(\frac{1}{4}\right)^t$
- Most widely used test in practice.
  - With $t = 40$, error probabitility less than 2^{-80}. Much less than the probability of a hardware failure.
  - Can test the primality of a 512-bit integer in less than a second.
  - Complexity: $\mathcal{O}(t \cdot \log^3 n)$

- To generate a random prime integer of size ℓ bits
  - Generate a random integer n of size ℓ bits
  - Test its primality with Miller-Rabin.
  - If n is declared prime, output n, otherwise generate another n again.
- Complexity
  - A ℓ-bit integer is prime with probability $\Omega(1/\ell)$
  - therefore $\mathcal{O}(\ell)$ trials are necessary.
  - Each primality test takes $\mathcal{O}(t \cdot \ell^3)$ time, so complexity $\mathcal{O}(t \cdot \ell^4)$
  - If a number is composite, only a constant number of Miller-Rabin tests will be required to discard it on average.
  - complexity $\mathcal{O}(\ell^4 + t \cdot \ell^3)$.

Random prime number generation

- To generate a random prime integer of size ℓ bits
  - Generate a random integer n of size ℓ bits
  - Test its primality with Miller-Rabin.
  - If n is declared prime, output n, otherwise generate another n again.
- Complexity
  - A ℓ-bit integer is prime with probability $\Omega(1/\ell)$
  - therefore $\mathcal{O}(\ell)$ trials are necessary.
  - Each primality test takes $\mathcal{O}(t \cdot \ell^3)$ time, so complexity $\mathcal{O}(t \cdot \ell^4)$
  - If a number is composite, only a constant number of Miller-Rabin tests will be required to discard it on average.
  - complexity $\mathcal{O}(\ell^4 + t \cdot \ell^3)$.