

# Computing with Large Integers

Jean-Sébastien Coron

University of Luxembourg  
<http://www.jscoron.fr>

## 1 Addition of positive integers

### 1.1 Addition algorithm

Implement the big integer addition algorithm, for positive integers. If using the C language you can use the structure:

```
typedef struct {  
    int size;  
    int *tab;  
} bignum;
```

### 1.2 Application: Fibonacci Sequence

We define the Fibonacci sequence  $u_0 = 1, u_1 = 1, u_n = u_{n-1} + u_{n-2}$  for  $n \geq 2$ . Write a program that computes the  $n$  terms of the Fibonacci sequence, for a given  $n$ , using the previous addition algorithm. You can use base  $B = 10$ . Check that  $u_{100} = 573147844013817084101$ . What is the value of  $u_{101}$  ?

## 2 Multiplication of positive integers

### 2.1 Multiplication algorithm

Implement the multiplication algorithm on big integers, for positive integers.

### 2.2 Application: factorial

We define  $n! = n \cdot (n - 1) \dots 2 \cdot 1$ . Write a program computing  $n!$  for a given  $n$ , using the previous multiplication algorithm. Check that  $30! = 26525285981219105863630848000000$ . What is the value of  $40!$  ?

## 3 Modular Exponentiation

Implement the modular exponentiation algorithm from the course.

```
$ expmod 2342 6762 9343  
7147
```

because  $2342^{6762} \equiv 7147 \pmod{9343}$ .

## 4 Optional: big number library and RSA implementation

The goal is to implement a big number library in C or C++, and to implement the RSA algorithm on top of it. A big integer will be represented using an array of digits in base  $B$ . The following struct can be used:

```
typedef struct {
    int sign;
    int size;
    int *tab;
} bignum;
```

where `sign` is the sign bit, and `size` is the size of the dynamic array `tab`.

### 4.1 Functions to be implemented

```
bignum str2bignum(char *str)
```

converts a string to a bignum.

```
bignum add(bignum a, bignum b)
```

adds the integers  $a$  and  $b$ .

```
bignum sub(bignum a, bignum b)
```

return  $a - b$ .

```
bignum mult(bignum a, bignum b)
```

returns the product of  $a$  and  $b$ .

```
bignum remainderbignum(bignum a, bignum n)
```

returns the remainder of the division of  $a$  by  $n$ .

```
bignum addmod(bignum a, bignum b, bignum n)
```

returns  $a + b \pmod n$ .

```
bignum multmod(bignum a, bignum b, bignum n)
```

returns  $a \cdot b \pmod n$ .

```
bignum expmod(bignum a, bignum b, bignum n)
```

returns  $a^b \pmod n$ .

```
bignum inversemod(bignum a, bignum n)
```

return  $a^{-1} \pmod n$  if  $\gcd(a, n) = 1$ .

```
bignum genrandom(int length)
```

generates a random integer of size `length` bits.

```
int fermat(bignum a, int t)
```

performs the Fermat test on integer  $a$  with security parameter  $t$ .

```
bignum genrandomprime(int length)
```

generates a random prime of size `length` bits, using the Fermat primality test.

### 4.2 The RSA algorithm

The goal is to implement the RSA algorithm using the previous library. The following functions must be implemented:

```
void keygen(bignum *n, bignum *e, bignum *d, int length)
```

generates an RSA modulus  $n = p \cdot q$ , where  $p$  and  $q$  are two prime integers of size `length` bit. The function also generates the public/private exponent pair  $(e, d)$ .

`bignum RSAencrypt(bignum m, bignum e, bignum n)`

takes as input a message  $m$ , a public exponent  $e$  and a RSA modulus  $n$  and returns the corresponding ciphertext  $c$ .

`bignum RSAdecrypt(bignum c, bignum d, bignum n)`

takes as input a ciphertext  $c$ , a private exponent  $d$  and a RSA modulus  $n$  and returns the corresponding plaintext  $m$ .

`void testRSA(int length)`

generates an RSA public-key  $(e, n)$  and its corresponding private-key  $(d, n)$ . It asks the user for a message  $m$  to encrypt, and outputs the corresponding ciphertext encrypted with public-key  $(n, e)$ . It then applies the decryption algorithm with private-key  $(d, n)$  and checks that the original message is recovered.

## References

1. V. Shoup, *A Computational Introduction to Number Theory and Algebra*, available at <http://shoup.net/ntb/>.