# Algorithms for Numbers and Public-key Cryptography

Jean-Sébastien Coron

Université du Luxembourg

April 3, 2015

# Summary

- C programming
  - Functions
- Algorithmic number theory
  - Subtraction
  - Euclidean division
  - Euler function

# Functions

- ```
  double max(double a,double b)
  {
    double m;
    if(a>b)
    {
      m=a;
    }
    else
    {
      m=b;
    }
    return m;
  }
  ```
  ```

# Using functions

- For function
  `double max(double a,double b)`
  - Let `x,y,z` be variables of type `double`.
  - Then instruction

    ```
    z=max(x,y);
    ```

    applies function `max` to variables `x` and `y`.
  - and stores the result in `z`.

# Call by value

- When function is called, the value of variables given as argument are copied in the parameter variables of the function.
  - `double max(double a,double b)`
  - `z=max(x,y);`
  - The content of variables `x` and `y` is copied into `a` and `b`.
- Call by value
  - If the content of variables `a` or `b` is modified inside the function, this does not affect variables `x` and `y`.

# Call by reference

- We would like to modify the value of variables given as argument.
  - We would like a function `swap(u,v)` that swaps the variables.

```
void swap(int a,int b) {
  int m=a;  a=b;  b=m;
}
int main()
{
  int u=1; int v=2;
  swap(u,v);
  printf("u=%d v=%d\n",u,v); // u=1 v=2
}
```

# Analysis

- The previous example does not work.
  - The function `swap` only swap the values of variables `a,b`, not the values of `u,v`.
- Solution: use pointers:
  - We give to `swap` the address of variables `u,v`.
  - The function `swap` will exchange the values at these two adresses.
  - One call `swap(&u,&v);`

# Call by reference

- Address of a variable d'une variable=pointer
  - The function swap takes as input two pointers.

```
void swap(int *a,int *b) {
  int m=*a;
  *a=*b;  *b=m;
}
int main()
{
  int u=1;  int v=2;
  swap(&u,&v);
  printf("u=%d v=%d\n",u,v); // u=2 v=1
}
```

# Conclusion

- When do we use call by reference ?
  - When we want to modify the value of a variable given as argument.
  - Otherwise, it is better to use call by value.

```
void addition(int a,int b,int *c) {
  *c=a+b;
}
int main()
{
  int u=1;  int v=2; int w;
  addition(u,v,&w);
  printf("w=%d\n",w); // w=3
}
```

# Algorithmic number theory

- Goal: modular computation with large integers.
  - Addition, multiplication, inversion modulo $n$.
- Euclidean division:
  - Given $a, b$, find $q, r$ such that

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

    where $a, b$ are big integers.

# Division with remainder

- Let $a = (a_{k-1} \dots a_0)_B$ and $b = (b_{\ell-1} \dots b_0)_B$ with $a > b > 0$ and $b_{\ell-1} \neq 0$.
  - Compute $q$ and $r$ such that $a = b \cdot q + r$ and $0 \leq r < b$.
  - $q = (q_{m-1} \dots q_0)_B$, with $m := k - \ell + 1$.
- Algorithm overview:

$r \leftarrow a$
for $i = m - 1$ downto 0 do
  $q_i \leftarrow r/(B^i b)$
  $r \leftarrow r - B^i \cdot q_i \cdot b$
output $r$

# Division with remainder

- For all $i$, $0 \leq r < B^i \cdot b$ after step $i$
  - Therefore, $0 \leq r < b$ eventually.
- How to compute $q_i = r/(B^i \cdot b)$
  - Test all possible values of $0 \leq q_i < B$
  - Not efficient, except if $B$ is small (*e.g.* $B = 10$).
  - Possible to do much better

# Division with remainder

- Complete algorithm (for small $B$)

$r \leftarrow a$
for $i = m - 1$ downto 0 do
  $q_i \leftarrow 0$
  while $r >= 0$
    $r \leftarrow r - B^i \cdot b$
    $q_i \leftarrow q_i + 1$
  $q_i \leftarrow q_i - 1$
  $r \leftarrow r + B^i \cdot b$
output $r$

# 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 $a \neq 0$
  - $\text{len}(0) = 1$
- 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 $\mathcal{O}(\text{len}(a)\,\text{len}(b))$
  - If $b \neq 0$, we can compute the quotient $q$ and the remainder $r$ in $a = b \cdot q + r$ in time $\mathcal{O}(\text{len}(b)\,\text{len}(q))$

- Computing $c = a + b$ in $\mathbb{Z}_n$
  - Let $c \leftarrow a + b$ in $\mathbb{Z}$
  - Let $c \leftarrow c \mod n$.
  - Complexity: $\mathcal{O}(\log n)$
- Computing $c = a \cdot b$ in $\mathbb{Z}_n$
  - Let $c \leftarrow a \cdot b$ in $\mathbb{Z}$
  - Let $c \leftarrow c \mod n$.
  - Complexity: $\mathcal{O}(\log^2 n)$.

# Euler function

- Definition:
  - $\phi(n)$ for $n > 0$ is defined as the number of integers $a$ comprised between 0 and $n-1$ such that $\gcd(a,n) = 1$.
  - $\phi(1) = 1$, $\phi(2) = 1$, $\phi(3) = 2$, $\phi(4) = 2$.
- Equivalently:
  - Let $\mathbb{Z}_n^*$ be the set of integers $a$ comprised between 0 and $n-1$ such that $\gcd(a,n) = 1$.
  - Then $\phi(n) = |\mathbb{Z}_n^*|$.

# Properties

- If $p \geq 2$ is prime, then

$$\phi(p) = p - 1$$

- More generally, for any $e \geq 1$,

$$\phi(p^e) = p^{e-1} \cdot (p - 1)$$

- For $n, m > 0$ such that $\gcd(n, m) = 1$, we have:

$$\phi(n \cdot m) = \phi(n) \cdot \phi(m)$$

# $\phi(p^e) = p^{e-1} \cdot (p-1)$

- If $p$ is prime
  - Then for any integer $1 \leq a < p$, $\gcd(a, p) = 1$
  - Therefore $\phi(p) = p - 1$
- For $n = p^e$, the integers between 0 and $n$ not co-prime with $n$ are
  - $0, p, 2 \cdot p, \ldots, (p^{e-1} - 1) \cdot p$
  - There are $p^{e-1}$ of them.
  - Therefore, $\phi(p^e) = p^e - p^{e-1} = p^{e-1} \cdot (p-1)$

# $\phi(n \cdot m) = \phi(n) \cdot \phi(m)$

- Consider the map:

$$\begin{aligned} f : \mathbb{Z}_{nm}^* &\rightarrow \mathbb{Z}_n^* \times \mathbb{Z}_m^* \\ a &\rightarrow (a \mod n, a \mod m) \end{aligned}$$

  - From the Chinese remainder theorem, the map is a bijection.
  - Moreover, $\gcd(a, n \cdot m) = 1$ if and only if $\gcd(a, n) = 1$ and $\gcd(a, m) = 1$.
  - Therefore, $|\mathbb{Z}_{nm}^*| = |\mathbb{Z}_n^*| \cdot |\mathbb{Z}_m^*|$
  - This implies $\phi(n \cdot m) = \phi(n) \cdot \phi(m)$.

# Theorem

- If $n = p_1^{e_1} \dots p_r^{e_r}$ is the factorization of $n$ into primes, then :

$$\phi(n) = \prod_{i=1}^{r} p_i^{e_i - 1} \cdot (p_i - 1) = n \prod_{i=1}^{r} (1 - 1/p_i)$$

  - Proof: immediate consequence of the two previous properties.

# Multiplicative order

- The multiplicative order of an integer $a$ modulo $n$ is defined as the smallest integer $k > 0$ such that

$$a^k \equiv 1 \mod n$$

- Example

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $1^i \mod 5$ | 1 | 1 | 1 | 1 |
| $2^i \mod 5$ | 2 | 4 | 3 | 1 |
| $3^i \mod 5$ | 3 | 4 | 2 | 1 |
| $4^i \mod 5$ | 4 | 1 | 4 | 1 |

  - Modulo 5, 1 has order 1, 2 and 3 have order 4, and 4 has order 2.

# Euler's theorem

- Theorem
  - For any integer $n > 1$ and any integer $a$ such that $\gcd(a, n) = 1$, we have $a^{\phi(n)} \equiv 1 \mod n$.
- Proof
  - Consider the map $f : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$, such that $f(b) = a \cdot b$ for any $b \in \mathbb{Z}^*$.
  - $f$ is a permutation, therefore :

$$\prod_{b \in \mathbb{Z}_n^*} b = \prod_{b \in \mathbb{Z}_n^*} (a \cdot b) = a^{\phi(n)} \cdot \left( \prod_{b \in \mathbb{Z}_n^*} b \right)$$

  - Therefore, we obtain $a^{\phi(n)} \equiv 1 \mod n$.

# Fermat's little theorem

- Theorem
  - For any prime $p$ and any integer $a \neq 0 \mod p$, we have $a^{p-1} \equiv 1 \mod p$. Moreover, for any integer $a$, we have $a^p \equiv a \mod p$.
- Proof
  - Follows from Euler's theorem and $\phi(p) = p - 1$.