Chapter 00_03

Introducing Foundation

Body of Knowledge Coverage:
Basis Analysis (AL)

- Basis Analysis (AL)
  - Asymptotic Analysis, empirical measurement.
  - Differences among best, average, and worst case behaviors of an algorithm.
  - Complexity classes, such as constant, logarithmic linear, quadratic, and exponential.
  - Recurrence Relations and their solutions.
  - Time and space trade-offs in algorithms.

Modular Arithmetic for Applications to Cryptography

Outlines

- Elementary number-theoretic notions (5)
- Division Theorem, the remainder (7, 8)
- Congruent modulo n, and modular equivalence (9, 10, 15, 21)
- Compute $a^k \; mod \; n$, when k is a power of 2. (31-36, 39-41)

Elementary Number-Theoretic Notions

- An application of number-theoretic algorithms is in *cryptography*
  - the discipline concerned with encrypting a message sent from one party to another, such that someone who intercepts the message will not be able to decode it.

- Let the set Z = { …., -2, -1, 0, 1, 2, 3, ….} of integers.
- Let the set N = {0, 1, 2, 3, ….} of natural numbers (also non-negative integers).
- The notation d | a (read "d *divides* a") means
  - a = k*d for some integer k, (i.e., a is k multiple of d).

The Division Theorem, Remainders, and Modular Equivalence

For any two integers x and y, where $y \neq 0$,

- the *quotient* q of x divided by y is given by

$$q = \left\lfloor \frac{x}{y} \right\rfloor$$

- the *remainder* r of dividing x by y is given by

$$r = x - q*y.$$

Algorithm divide(x, y), $x \geq 0$, $y \geq 1$

```
q = 0; r = x;
while (r ≥ y) {
    r = r – y;
    q ++; };
output q, r;
```

$[1]_2$ = { …, -7, -5, -3, -1, 1, 3, 7, 9, 11, 13, 15, 17, 19, … } is *the equivalence class modulo 2 containing 1.*

The Division Theorem, Remainders, and Modular Equivalence

Theorem 0.1(Division Theorem)

For any integer x and any positive integer y,
there are unique integers q and r such that $0 \le r < y$ and $x = q*y + r$.
Proof: Followed by definition. QED

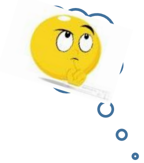
Definition of x mod y
x mod y is the remainder r = x – q*y, where q = $\lfloor x / y \rfloor$.

$x = q*y + r$
$\frac{x}{y}$ = (q, r)
$\frac{-7}{2}$ = (-4, 1)
-7 = -4 *2 +1
$\frac{-5}{2}$ = (-3, 1)
-5 = -3 *2 +1
$\frac{-3}{2}$ = (-2, 1)
-3 = -2 *2 +1
$\frac{-1}{2}$ = (-1, 1)
-1 = -1 *2 +1
$\frac{1}{2}$ = (0, 1)
1 = 0*2 +1
$\frac{3}{2}$ = (1, 1)
3 = 1*2 +1
$\frac{5}{2}$ = (2, 1)
5 = 2*2 +1
$\frac{7}{2}$ = (3, 1)
7 = 3*2 +1

x mod y is the remainder of $\frac{x}{y}$:

- *Definition of* x mod y

  m mod n is the remainder r = x – q*y, where q = $\lfloor \frac{x}{y} \rfloor$.



- Example: -18 mod 7 = 3, 3 = -18 – (-3) * 7.

  -11 mod 7 = 3, 3 = -11 – (-2) * 7.

  - 4 mod 7 = 3, 3 = -4 – (-1) * 7.

  3 mod 7 = 3, 3 = 3 – 0 * 7.

  10 mod 7 = 3, 3 = 10 – 1 * 7.

  17 mod 7 = 3, 3 = 17 – 2 * 7.

  $[3]_7$ = { …,-18, -11. -4, 3, 10, 17, …}

- Denote “ y divides x” as y | x.
- y | x means that x = i * y for some integer i.
- y | x iff x mod y = 0.

Congruence Modulo n

Definition of Congruent Modulo n:

Let m and k be integers and n be a positive integer ($n > 0$).

- **m *is congruent to* k *modulo* n**
  - (or we say, m and k are congruent modulo n)
  - (or we say, *m and k are equivalent* ($\equiv$) *mod n*)

  denoted as

$$m \equiv k \bmod n$$

if, and only if $n \mid (m - k)$ or $n \mid (k - m)$.

- In symbol, $m \equiv k \bmod n \leftrightarrow n \mid (m - k) \text{ or } n \mid (k - m)$.
- Example: $-18 \equiv 3 \bmod 7$, $-11 \equiv 3 \bmod 7$, $-4 \equiv 3 \bmod 7$, $10 \equiv 3 \bmod 7$,

  $17 \equiv 3 \bmod 7$, $24 \equiv 3 \bmod 7$, $24 \equiv 10 \bmod 7$.

Observation:
$n \mid (m - k)$ or $n \mid (k - m)$. $m \bmod n = k \bmod n$ (i.e., they have the same remainder.
$m = k + i * n$, where $i \in Z = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$

Congruent modulo n:

- *m and k are congruent modulo n*

  if they differ by a multiple of n (i.e., $i * n$).
- Equivalently, we write $|m - k| = i * n$,

  for some $i \in Z = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, the set of integers.
- Example:
  - $[3]_7 = \{\ldots -11, -4, 3, 10, 17, 24, 31, \ldots\}$ are congruent modulo 7.
  - $|-11 - (-4)| = 1 * 7$, $|-11 - 3| = 2 * 7$, $|-11 - 10| = 3 * 7$, …
  - $|-4 - 3| = 1 * 7$ $|-4 - 10| = 2 * 7$, $|-4 - 17| = 3 * 7$
  - $|3 - 10| = 1 * 7$, $|3 - 17| = 2 * 7$, $|3 - 24| = 3 * 7$, …
  - $|10 - 17| = 1 * 7$, $|10 - 24| = 2 * 7$, $|10 - 31| = 3 * 7$, …
  - $|17 - 24| = 1 * 7$, $|17 - 24| = 2 * 7$, …
  - $|24 - 31| = 1 * 7$, …

$m = i * n + k$, $0 \leq k < n$
$m = i * 7 + 3$
When $i \in \{0, 1, 2, 3, \ldots\}$,
$m \in \{3, 10, 17, 24, \ldots\}$,
respectively.
When $i \in \{-1, -2, -3, \ldots\}$,
$m \in \{-4, -11, -18, \ldots\}$,
respectively.

Congruence Modulo n

Example 0.47: n = 5; $m \equiv k \bmod n$ *if, and only if* $n \mid (m - k)$ *or* $n \mid (k - m)$.

What is *the equivalence class modulo 5 containing 3*, $[r]_n = [3]_5$.
That is, *each of the integer m in the class mod 5 has the remainder 3*.

$[3]_5 = \{ \ldots, -7, -2, 3, 8, 13, 18, 23, 28, 33, \ldots \}$ is *the equivalence class modulo 5 containing 3*.

$x = q*y + r$

Since $5 \mid (33 - 33)$, $33 \equiv 33 \bmod 5$. $|33 - 33| = 0 * 5$. $33 = 6*5 + 3$

Since $5 \mid (33 - 28)$, $33 \equiv 28 \bmod 5$. $|33 - 28| = 1 * 5$. $28 = 5*5 + 3$

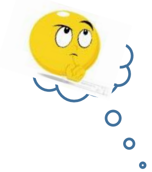
Since $5 \mid (33 - 23)$, $33 \equiv 23 \bmod 5$. $|33 - 23| = 2 * 5$. $23 = 4*5 + 3$

Since $5 \mid (33 - 18)$, $33 \equiv 18 \bmod 5$. $|33 - 18| = 3 * 5$. $18 = 3*5 + 3$

Since $5 \mid (33 - 13)$, $33 \equiv 13 \bmod 5$. $|33 - 13| = 4 * 5$. $13 = 2*5 + 3$

Since $5 \mid (33 - 8)$, $33 \equiv 8 \bmod 5$. $|33 - 8| = 5 * 5$. $8 = 1*5 + 3$

Since $5 \mid (33 - 3)$, $33 \equiv 3 \bmod 5$. $|33 - 3| = 6 * 5$. $3 = 0*5 + 3$



Since $5 \mid (33 - (-2))$, $33 \equiv -2 \bmod 5$. $|33 - -2| = 7 * 5$. $-2 = 1*5 + 3$

Since $5 \mid (33 - (-7))$, $33 \equiv -7 \bmod 5$. $|33 - -7| = 8 * 5$. $-7 = 2*5 + 3$

Congruence Modulo n

$[3]_5 = \{ \ldots, -7, -2, 3, 8, 13, 18, 23, 28, 33, \ldots \}$ is *the equivalence class modulo 5 containing 3.*

Theorem 0.1.4.1 Modular Equivalences

Let a and b and n be any integers and suppose $n > 1$.

The following statements are all equivalent:

1. $n \mid (a - b)$
2. $a \equiv b \pmod{n}$
3. $a = b + i*n$ for some integer i
4. a and b have the same (nonnegative) remainder when divided by n
5. a mod n = b mod n.

Proof: Obvious. Example: $5 \mid (33 - 18)$.

$r = x \bmod y.$

$x = q*y + r \Longrightarrow [\, r \,]_y = \{\, r + q*y \mid q \in Z \,\}$

Let Z be the set of integers {…, -2, -1, 0, 1, 2, … }.

All integers can be partitioned into **n** equivalence classes, according to their remainders modulo **n**.

Define ***the equivalence class modulo n containing an integer a*** to be

$$[a]_n = \{a + i * n \mid i \in Z\},$$

For example, $[3]_7 = \{ \ldots, -25, -18, -11, -4, 3, 10, 17, 24, 31, 38, \ldots\}$.

i.e., $b \in [a]_n$ iff $b \equiv a \pmod{n}$.

iff $n \mid (b - a)$. i.e., b must be equal to $a + i*n$.

$b \in [a]_n$ iff $b \equiv a \pmod{n}$. i.e., a and b are in the same class $[a]_n$.

*The **set** of all such equivalence classes* Z_n is

$$Z_n = \{[a]_n \mid 0 \le a \le n - 1\}. \quad \text{i.e., } z = i * n + a,\ 0 \le a < n.$$

There are *seven equivalence classes modulo 7*.

$Z_7 = \{[0]_7, [1]_7, [2]_7, \ldots, [6]_7\}$ where

$[0]_7 = \{\ldots, -14, -7, 0, 7, 14, 21, \ldots\}$ $21 \equiv 0 \pmod{7}$, $(21 - 0) / 7 = 0$

$[1]_7 = \{\ldots, -13, -6, 1, 8, 15, 22, \ldots\}$ $22 \equiv 1 \pmod{7}$, $(22 - 1) / 7 = 0$

$[2]_7 = \{\ldots, -12, -5, 2, 9, 16, 23, \ldots\}$ $23 \equiv 2 \pmod{7}$, $(23 - 2) / 7 = 0$

$[3]_7 = \{\ldots, -11, -4, 3, 10, 17, 24, \ldots\}$ $24 \equiv 3 \pmod{7}$, $(24 - 3) / 7 = 0$

…

$[6]_7 = \{\ldots, -8, -1, 6, 13, 20, 27, \ldots\}$ $27 \equiv 6 \pmod{7}$, $(27 - 6) / 7 = 0$

Congruence Modulo n

Congruence modulo n is an equivalence (≡) relation on the set of all integers.

Theorem 0.1.4.2 Congruence Modulo n is an Equivalence Relation

Let $n > 1$ be any integer. The distinct equivalence classes of the relation are the sets $[0]_n$, $[1]_n$, $[2]_n$, …, $[n - 1]_n$, where for each $a = 0, 1, 2, \ldots, n - 1$,

$$[a]_n = \{m \in Z \mid m \equiv a \pmod{n}\},$$

or, equivalently,

$$[a]_n = \{m \in Z \mid m = i*n + a \text{ for some integer } i\}.$$

Proof: Left for exercise.

Example: The equivalence class mod 7 containing 3 is

$$[3]_7 = \{3, 10, 17, 24, \ldots\}.$$

Congruence Modulo n

- m and k are congruent modulo n
- *m and k are equivalent* (≡) *mod n*
- $m \equiv k \pmod{n}$

Theorem 0.1.4.3 Modular Arithmetic

Let a, b, c, d, and n be integers with $n > 1$, and suppose

$$a \equiv c \pmod{n} \quad \text{and } b \equiv d \pmod{n}.$$

Then

1. $(a + b) \equiv (c + d) \pmod{n}$
2. $(a - b) \equiv (c - d) \pmod{n}$
3. $a\, b \equiv c\, d \pmod{n}$
4. $a^m \equiv c^m \pmod{n}$ for all integers m.

Proof: left for exercise.

Congruence Modulo n

Corollary 0.1.4.4

Let a, b, and n be integers with $n > 1$. Then

$$a\,b \equiv [(a \bmod n)\,(b \bmod n)] \pmod{n},$$

or, equivalently,

$$a\,b \bmod n = [(a \bmod n)\,(b \bmod n)] \pmod{n}.$$

If m is a positive integer, then

$$a^m \equiv [(a \bmod n)^m] \pmod{n}.$$

or, equivalently,

$$a^m\, mod\, n = [(a \bmod n)^m] \pmod{n}.$$

Some of the properties: Modular arithmetic (a, b are integers, n is a positive integer)

(a + b) mod n = (a mod n + b mod n) mod n

(a * b) mod n = ((a mod n) * (b mod n)) mod n

Example:

(15 + 21) mod 5 = 36 mod 5 ≡ 1

(15 + 21) mod 5 = (15 mod 5 + 21 mod 5) mod 5

= (0 + 1) mod 5 ≡ 1

(32*32) mod 31 = ((32 mod 31) * (32 mod 31)) mod 31

= 1 mod 31 ≡ 1, where as 1024 mod 31 ≡ 1

Congruence Modulo n

Example: Using Corollary 0.1.4.4

$$
\begin{aligned}
(55 * 26) \bmod 4 &= \{(55 \bmod 4)\ (26 \bmod 4\} \bmod 4 \\
&= (3 * 2) \bmod 4 \\
&= 6 \bmod 4 \\
&\equiv 2
\end{aligned}
$$

That is, $2 \equiv (55 * 26) \bmod 4$

or $(55 * 26) \equiv 2 \bmod 4$

Under such substitution, addition, multiplication remain well-defined.

Substitution rule:

If $x \equiv x'$ (mod N) and $y \equiv y'$ (mod N), then

$x + y \equiv x' + y'$ (mod N) and $xy \equiv x'y'$ (mod N)

Example:

For $16 \equiv 7 \bmod 3$, and $2 \equiv 17 \bmod 3$

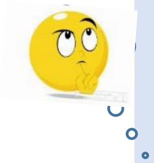
then $(16 +2) \equiv (7 + 17) \bmod 3$.

i.e., $18 \equiv 24 \bmod 3$ iff $3 \mid (24 - 18)$.

Or

$(16 + 17) \equiv (7 + 2) \bmod 3$.

i.e., $33 \equiv 9 \bmod 3$ iff $3 \mid (33 - 9)$.



Example: Suppose we watch an entire season of our favorite television show in one sitting, starting at midnight. There are 25 episodes, each lasting 3 hours.

At what time of day are we done?

The answer is: The hour of completion is (25 * 3) (mod 24), which is 3 am.

A day has 24 hours. Since 3(mod 24) = 3 and 25 (mod 24) = 1

(25 * 3) (mod 24) = (25 (mod 24) * 3 (mod 24))(mod 24)

= 1 * 3 (mod 24)

= 3 (mod 24)

= 3 am. {because the process begins at 0:00 midnight.

What if there are 40 episodes? Then the completion is (40 * 3) mod 24 = 0 (12 midnight).

What if there are 45 episodes? Then the completion is (45 * 3) mod 24 = 15 (3 pm).

What if there are 125 episodes? Then the completion is (125 * 3) mod 24 = (5*3) mod 24

= 15 (3 pm)

Modular arithmetic satisfies associative, commutative and distribute properties of addition and multiplication.

$x + (y + z) \equiv (x + y) + z \pmod N$	Associativity
$xy \equiv yx \pmod N$	Commutativity
$x(y + z) \equiv (xy + yz) \pmod N$	Distributivity

While performing a sequence of arithmetic operations, it is legal to reduce intermediate results to their remainders modulo N *at any stage*. Such simplifications can be a dramatic help in big calculations.

1024(2^{10}),	512,	256,	128(2^7),	64(2^6),	32(2^5),	16,	8,	4,	2,	1(2^0)
1	0	0	0	0	0	0	0	0	0	0

$$(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$$

Example 0.21: Compute 2^{345} mod 31.

$$2^{345} = (2^5)^{69} = (32)^{69} \quad \text{and } 1 \equiv 32 \pmod{31}$$

$$2^{345} \pmod{31} = (2^5)^{69} \pmod{31}$$

$$= (2^5 \pmod{31})^{69} \pmod{31}$$

$$= 1^{69} \pmod{31}$$

$$\equiv 1$$

Compute 2^{345} (mod 15)?

then 2^{345} (mod 15)

$= (2^4)^{86}$ x 2 (mod 15)

$= (2^4)^{86}$ (mod 15) x 2 (mod 15)

$= (2^4 \bmod 15)^{86}$ (mod 15) x 2 (mod 15)

= 1 x 2

= 2

Also compute 2^{345}.

$2^{345} = (2^5)^{69} = (32)^{69} = \ldots$ Then what?

Some Reason: use shift left….

This allows you to use shifting left for your multiplication.

For example: compute $(32)^3$

32	=	1 0 0 0 0 0	
x 32	=	1 0 0 0 0 0	… 2^5
1024	=	1 0 0 0 0 0 0 0 0 0 0	….sL 5 bits
x 32	=	1 0 0 0 0 0	… 2^5
32768	=	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	… sL 5 bits

This allows you to use shifting left for your multiplication.

For example: compute $(32)^4$

32	=	1 0 0 0 0 0	
x 32	=	1 0 0 0 0 0	… 2^5
1024	=	1 0 0 0 0 0 0 0 0 0 0	….sL 5 bits
x 32	=	1 0 0 0 0 0	… 2^5
32768	=	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	… sL 5 bits
x 32	=	1 0 0 0 0 0	. . . 2^5
1048576	=	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	… sL 5 bits

Back to the original problem to compute 2^{345} :

$$2^{345} = (2^5)^{69}$$

$$= (32)^{69}$$

$$= (32)^{68} * (32)^1$$

$$= (32)^{64} * (32)^4 * (32)^1$$

$$= ((32)^4)^{16} * (32)^4 * (32)^1$$

$$= ((((32)^4)^2)^2)^2)^2 * (32)^4 * (32)^1$$

$$= ((((32)^4 * (32)^4)^2)^2)^2 * (32)^4 * (32)^1$$

$$= ((((32)^4*(32)^4)*((32)^4*(32)^4))^2)^2*(32)^4*(32)^1$$

$$= ((((32)^4*(32)^4)*((32)^4*(32)^4) * ((32)^4*(32)^4)*((32)^4*(32)^4)))^2*(32)^4*(32)^1$$

$$= ((((32)^4*(32)^4)*((32)^4*(32)^4) * ((32)^4*(32)^4)*((32)^4*(32)^4)) *$$

$$((((32)^4*(32)^4)*((32)^4*(32)^4) * ((32)^4*(32)^4)*((32)^4*(32)^4)))^2*(32)^4*(32)^1$$

which requires 8 * operations

$2^2 = 2*2$ (one * operation)
$2^5 = 2 * 2^2 * 2^2$ (three * operations)
$(2^5)^2 = 2^5 * 2^5$ (four * operations)
$(2^5)^4 = 2^{5^{2^2}} = (2^5)^2 * (2^5)^2$ (5 * operations)

$(2^5)^8 = (2^5)^4 * (2^5)^4$ (6 * operations)

$(2^5)^{16} = (2^5)^8 * (2^5)^8$ (7 * operations)

$(32)^4 = (32*32)* (32*32)$
2 * operations.

Modular addition and multiplication

Compute $(x + y) \bmod N$… An effective method is :

$(x + y) \bmod N = (x \bmod N + y \bmod N) \bmod N$

Its running time is linear in the sizes (of n bits) of these numbers, i.e., $O(n)$, where $n = \lceil \log_2 \max\{x, y\} \rceil$ is the size of $\max\{x, y\}$.

Assume that $x \geq y$.

If $\lceil \log_2 x \rceil < \lceil \log_2 N \rceil$, then $x \bmod N = x$; and the nos. times of division is 0.

If $\lceil \log_2 x \rceil > \lceil \log_2 N \rceil$, then $x \bmod N$ requires $\lceil \log_2 q \rceil \leq ((\lceil \log_2 x \rceil - \lceil \log_2 N \rceil) + 1)$ nos. times of division, since $x \bmod N = x - q * N = r$, where $0 \leq r < N$. Each time of division has to perform $\lceil \log_2 N \rceil$ additions. Its running time is $\lceil \log_2 q \rceil * \lceil \log_2 N \rceil$ which is linear, $O(n)$, where n is at most the size of $\lceil \log_2 x \rceil$ bits of x. Likewise, If $\lceil \log_2 y \rceil > \lceil \log_2 N \rceil$, then $y \bmod N$ requires $\lceil \log_2 q' \rceil \leq ((\lceil \log_2 y \rceil - \lceil \log_2 N \rceil) + 1)$ nos. times of division, since $y \bmod N = y - q' * N = r'$, where $0 \leq r' < N$. Its running time is $\lceil \log_2 q' \rceil * \lceil \log_2 N \rceil$ which is linear, $O(n')$, where n' is at most the size of $\lceil \log_2 y \rceil$ bits of y.

Finally, $(r + r') \bmod N$, where $(r + r') \leq 2(N-1)$ requires $\lceil \log_2 q'' \rceil \leq (\lceil \log_2 (r+r') \rceil - \lceil \log_2 2*N \rceil) + 1)$ nos. times of division. Its running time $\lceil \log_2 q'' \rceil * \lceil \log_2 N \rceil$ which is linear, $O(n'')$ where n'' is at most the size of $\lceil \log_2 (r+r') \rceil$ bits of $(r + r')$.

Thus, $(x+y) \bmod N$ requires $(\lceil \log_2 q \rceil + \lceil \log_2 q' \rceil + \lceil \log_2 q'' \rceil) \lceil \log_2 N \rceil \leq$ which is linear $O(n)$, $n = \lceil \log_2 N \rceil$ the input size of N.

11 mod 4
= 7 mod 4
= 3 mod 4
= 3

$$\begin{array}{r|l} & 10 \\ \hline 100 & 1011 \\ & 100 \\ \hline & 11 \end{array}$$

```
      31
11 | 345
     33
     ---
      15
      11
     ---
       4
```

```
                   1 1 1 1 1
1011 | 1 0 1 0 1 1 0 0 1
         1 0 1 1
       -------------------
         1 0 1 0 1
           1 0 1 1
       -------------------
           1 0 1 0 0
             1 0 1 1
       -------------------
             1 0 0 1 0
               1 0 1 1
       -------------------
                 1 1 1 1
                 1 0 1 1
       -------------------
                   1 0 0
```

[$\log_2 x$] = 9 bits
[$\log_2 y$] = 4 bits
The nos. of
division needed is
9-4+1 = 6 additions

```
                 0 1 1 1 1 1
1 0 1 1 | 1 0 1 0 1 1 0 0 1
          0 0 0 0
        -------------------
          1 0 1 0 1
            1 0 1 1
        -------------------
            1 0 1 0 1
              1 0 1 1
        -------------------
              1 0 1 0 0
                1 0 1 1
        -------------------
                1 0 0 1 0
                  1 0 1 1
        -------------------
                    1 1 1 1
                    1 0 1 1
        -------------------
                      1 0 0
```

$\lceil \log_2 x \rceil$ = 13 bits
$\lceil \log_2 y \rceil$ = 4 bits
The nos. of division needed is 13-4+1 = 10 additions

```
       7 4 4
11 | 8 1 9 1
     7 7
       4 9
       4 4
         5 1
         4 4
           7
```

```
              1 0 1 1 1 0 1 0 0 0
1 0 1 1 | 1 1 1 1 1 1 1 1 1 1 1 1 1
          1 0 1 1
            1 0 0 1
            0 0 0 0
            1 0 0 1 1
              1 0 1 1
              1 0 0 0 1
                1 0 1 1
                1 1 0 1
                1 0 1 1
                    1 0 1
                        0
                    1 0 1 1
                    1 0 1 1
                          0 1
                            0
                          0 1 1
                              0
                          0 1 1 1
                                0
                          0 1 1 1
```

Compute xy (mod N) ….. multiplying x mod N and y mod N

$$(x * y) \bmod N = ((x \bmod N) * (y \bmod N)) \bmod N$$

- To multiply two mod N numbers x and y,
  - start with regular multiplication, x*y and
  - then reduce the answer modulo N.
  - The product can be as large as $(N - 1)^2$.
- This is still at most 2n bits long since $\log(N - 1)^2 = 2\log(N - 1) \le 2n$.

[Rationalize: let $n = [\log_2 (N-1)]$, where $2^{n-1} \le N-1 < 2^n$. We have $(N-1)^2 < 2^{2n}$.

Then $\log_2(N-1)^2 < \log_2 2^{2n}$. We obtain $2\log_2(N-1) < 2n\log_2 2$.

That is, $[\log_2 (N-1)^2] \le 2n$.]

Example:
Let N be 31.
Let x and y be 30
$(x * y) \bmod N$
$= 30*30 \bmod 31$
$= 30 * 30$
$= (N - 1)^2$
Let x and y be n bits long.
x*y would be 2n bits long.
The size of $(N - 1)^2$ is $\log_2 (N - 1)^2$
$\le \lfloor (N - 1)^2 \rfloor + 1 = 2n$

- To reduce the answer modulo N, we compute the remainder upon dividing it by N, using a quadratic-time division algorithm. Thus, multiplication remains a quadratic operation.
- If $N \ne 0$, ***division*** can be done in cubic time, $O(n^3)$.

For 20 bits lthe ong, 19th position in bit representation x or y can be {0, 2^{19}}. That is 1000....0000; The largest value would be 2^{19} - 1. That is 0111 ... 1111.

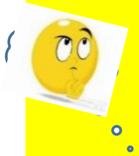
Modular exponentiation

Any cryptography system needs a fast way to compute x^y mod N, where values x, y, and N of several hundred bits long each.

- The result is some number modulo N and is, a few hundred bits long.
- The raw value of x^y could be much longer than this.
- When both x and y are 20 bits numbers [bit positions are from 0 to 19], x^y is at least $(2^{19})^{(2^{19})} = 2^{(19)(524288)}$, about 10 million bits long!
- Image what happens if y is a 500-bit number!

- Compute x^y mod N by repeatedly multiplying by x modulo N to yield
  - x mod N → x^2 mod N → x^3 mod N → x^4 mod N → ... → x^y mod N.
  - A problem arises: We need to perform all intermediate computations modulo N. We need to perform y – 1 (i.e., ≈ 2^{500}) multiplications if y is 500 bits long.
  - This approach is clearly exponential in the size of y.

Can we do better to compute x^y mod N ?



- Starting with x and squaring repeatedly modulo N, compute

  $x \bmod N$

  $x^2 \bmod N = (x \bmod N)\,(x \bmod N) \bmod N$

  $x^4 \bmod N = (x^2 \bmod N)\,(x^2 \bmod N) \bmod N$

  $x^8 \bmod N = (x^4 \bmod N)\,(x^4 \bmod N) \bmod N$

  ...

  $x^y \bmod N = (x^{\frac{y}{2}} \bmod N)\,(x^{\frac{y}{2}} \bmod N) \bmod N.$

- $x \bmod N \rightarrow x^2 \bmod N \rightarrow x^4 \bmod N \rightarrow x^8 \bmod N \rightarrow \ldots \rightarrow x^{2^{\lceil \log_2 y \rceil}} \bmod N$, where $2^k = y$ which implies $k = \lceil \log_2 y \rceil$. That is $2^k = 2^{\lceil \log_2 y \rceil}$.

- Each takes just $O(\log_2^2 N)$ time to compute, and in this case, there are only $\log_2 y$ multiplications. Thus, $O(\log_2 y * \log_2^2 N)$, a polynomial time.
  - Takes $O(\log_2^2 N)$ time to compute $x^2 \bmod N = (x \bmod N * x \bmod N) \bmod N$, since it takes $O(\log_2 N)$ to compute $x \bmod N$.

To ***determine x^y mod N,*** multiply simply together an appropriate subset of these powers, those corresponding to 1’s in the binary representation of y.

For example,

$$x^{25} = x^{11001_2} = x^{10000_2} * x^{1000_2} * x^{1_2} = x^{16} * x^8 * x^1$$

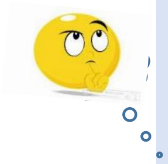
$$x^1$$

$$x^2 = x^1 * x^1$$

$$x^4 = x^2 * x^2$$

$$x^8 = x^4 * x^4$$

$$x^{16} = x^8 * x^8$$



Then, compute $x^{16} * x^8 * x^1$

X^2, X^4, X^8, X^{16},
2* constitute
6 multiplications
$\lfloor (\log_2 25 \rfloor + 1$

A polynomial-time algorithm is finally within reach!

Figure 1.4 Modular exponentiation

```
function modexp(x, y, N)
    //Compute x^y mod N
    Input:  Two n-bits integers x and N, an integer exponent y.
    Output:  x^y mod N.
    if (y = 0) then return 1;
    z = modexp(x, ⌊y/2⌋, N);   // z = x^⌊y/2⌋ mod N
    if (y is even) then return z^2 mod N;
                   else return x * z^2 mod N;
```

Figure 1.4 Modular exponentiation

```
function modexp(x, y, N)
    //Compute x^y mod N
    Input:  Two n-bits integers x and N, an integer exponen
    Output:  x^y mod N.

    if (y = 0) then return 1;
    z = modexp(x, ⌊y/2⌋, N);    // z = x^⌊y/2⌋ mod N
    if (y is even) then return z² mod N;
                   else return x * z² mod N;
```

The self-evidence rule:

$$x^{y} = \begin{cases} (x^{\lfloor y/2 \rfloor})^2 & \text{if y is even} \\ x * (x^{\lfloor y/2 \rfloor})^2 & \text{if y is odd.} \end{cases}$$

Compute x^{11} mod N

me(x, 11, N)

$$\left.\begin{array}{l} z = me(x, \lfloor 11/2 \rfloor, N); \\ z = me(x, \lfloor 5/2 \rfloor, N); \\ z = me(x, \lfloor 2/2 \rfloor, N); \\ z = me(x, \lfloor 1/2 \rfloor, N); \\ z = me(x, \lfloor 0/2 \rfloor, N); \end{array}\right\}$$

if (y = 0), return 1.

z = me(x, ⌊0/2⌋, N) = 1

z = me(x, ⌊1/2⌋, N);

if (y is even) ... else $x * 1^2$ mod N

z = me(x, ⌊1/2⌋, N)= x mod N

z = me(x, ⌊2/2⌋, N);

if (y is even) then $(x \bmod N)^2 \bmod N$

z = me(x, ⌊2/2⌋, N)= $(x^2 \bmod N) \bmod N$

z = me(x, ⌊5/2⌋, N);

if (y is even) then x * $((x^2 \bmod N) \bmod N))^2 \bmod N$

z = me(x, ⌊11/2⌋, N);

if (y is even) then x* (x * $((x^2 \bmod N) \bmod N))^2 \bmod N)^2 \bmod N$

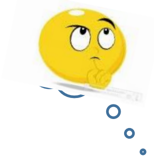
For computing x^y mod n,

- let n be the size in bit-representation max{x, y, N}.
- The algorithm will halt after at most n (in fact, $\log_2 n$, since $\lfloor y/2 \rfloor$) recursive calls.

- During each call it multiplies n-bit numbers (doing computation modulo N save us here).
- ***The total running time is $\log_2 n * O(n^2) \leq O(n^3)$.***

- This recursive algorithm of Figure 1.4, which works by executing, modulo N, the self-evident rule

$$x^y = \begin{cases} (x^{\lfloor y/2 \rfloor})^2 & \text{if y is even} \\ x * (x^{\lfloor y/2 \rfloor})^2 & \text{if y is odd} \end{cases}$$



Example 0.25: Compute x^{25}

$$
\begin{aligned}
x^{25} &= x * x^{24} \\
&= x * (x^{12})^2 \\
&= x * ((x^6)^2)^2 \\
&= x * (((x^3)^2)^2)^2 \\
&= x * (((x * (x^1)^2)^2)^2)^2 \\
&= x * (((x * (x * x^0)^2)^2)^2)^2
\end{aligned}
$$

```
function modexp(x, y, N) {
if (y = 0) then return 1;
z = modexp(x, ⌊y/2⌋, N);
if (y is even) then return z^2 mod N
          else return x* z^2 mod N;
}
```

This closely parallels our recursive multiplication algorithm (Figure 1.1 Multiplication à la Français).

Example 0.25: Compute x^{25} %N

modexp(x, 25, N); z = ME(x, $\lfloor 25/2 \rfloor$, N);

$$
\begin{aligned}
x^{25} &= x * x^{24} \\
&= x * (x^{12})^2 \\
&= x * ((x^6)^2)^2 \\
&= x * (((x^3)^2)^2)^2 \\
&= x * (((x * (x^1)^2)^2)^2)^2 \\
&= x * (((x * (x * x^0)^2)^2)^2)^2
\end{aligned}
$$

This closely parallels our recursive multiplication algorithm
(Figure 1.1 Multiplication à la Français).

```
function modexp(x, y, N) {
if (y = 0) then return 1;
z = modexp(x, ⌊y/2⌋, N);
if (y is even) then return z^2 mod N
        else return x* z^2 mod N; }
```

ME(x, 25, N)

y =25 $\neq$ 0; z = x * $(((x * ((x * 1^2) \%N)^2 \%N)^2 \%N)^2 \%N)^2 \%N$

z = ME(x, $\lfloor 25/2 \rfloor$, N);

y =12 $\neq$ 0; z = $((x * ((x * 1^2) \%N)^2 \%N)^2 \%N)^2 \%N$

z = ME(x, $\lfloor 12/2 \rfloor$, N);

y =6 $\neq$ 0; z = $(x * ((x * 1^2) \%N)^2 \%N)^2 \%N$

z = ME(x, $\lfloor 6/2 \rfloor$, N);

y =3 $\neq$ 0; z = x * $((x * 1^2) \%N)^2 \%N$

z = ME(x, $\lfloor 3/2 \rfloor$, N);

y =1 $\neq$ 0; z = $(x * 1^2) \%N$

z = ME(x, $\lfloor 1/2 \rfloor$, N);

y = 0; z = x^0 = 1

For RSA cryptography, computations are facilitated by using two properties of exponents:

$x^{2n} = (x^n)^2$ for all real numbers x and a with $x \geq 0$. ……..(e.0.1.4.1)

$x^{a+b} = x^a x^b$ for all real numbers x, a and b with $x \geq 0$. ……..(e.0.1.4.2)

Example:

$x^4 \bmod n = (x^2)^2 \bmod n$

$= (x^2 \bmod n)^2 \bmod n$, using Corollary 0.1.4.4

Example:

$x^7 \bmod n = (x^{4+2+1}) \bmod n$

$= (x^4 x^2\, x^1) \bmod n$

$= \{(x^4 \bmod n)\,(x^2 \bmod n)(x^1 \bmod n)\} \bmod n$

Example: Compute $a^k \; mod \; n$, when k is a power of 2.

Find 144^4 mod 713.

Solution:

Use the property of $x^{2a} = (x^a)^2$ for all real numbers x and a with $x \geq 0$.

144 mod 713 = 144

144^2 mod 713 = 20736 mod 713 = 59

144^4 mod 713 $= (144^2)^2$ mod 713

$= (144^2 \; mod \; 713)^2$ mod 713

$\cancel{= (20736 \; mod \; 713)^2 \text{ mod } 713}$

$= \; 59^2$ mod 713

$=$ 3481 mod 713

$=$ 629

Example: Compute $a^k \; mod \; n$, when k is a power of 2.

Find 12^{43} mod 713.

Solution:

First write the exponent as a sum of powers of 2:

$$43 = 2^5 + 2^{\,3} + 2 + 1 = 32 + 8 + 2 + 1.$$

$$43 = 1\;0\;1\;0\;1\;1$$

Next compute 12^{2^k} $for \; k = 1, 2, 3, 4, 5$

$12 \quad$ mod 713 = 12

$12^2 \quad$ mod 713 = 144

$12^4 \quad$ mod 713 = 144^2 mod 713 = 59

$12^8 \quad$ mod 713 = 59^2 mod 713 = 629

$12^{16} \quad$ mod 713 = 629^2 mod 713 = 639

$12^{32} \quad$ mod 713 = 639^2 mod 713 = 485

floor($\log_2$ 43) = 5 + 1 bits
For n bits long, it needs floor($\log_2$ n) bits.
Each has to do at most 2 mod and 1 multiple.
These can be done by 3*floor($\log_2$ n) of divide/multiple operations.

Use the property of $x^{a+b} = x^a \, x^b$ for all real numbers x and a, and b with x $\geq$ 0.

$$12^{43} = 12^{32+8+2+1} = 12^{32}\; 12^8 \; 12^2 \; 12^1$$

$$12^{43} \text{ mod } 713 = ((12^{32} \text{ mod } 713)\;(12^8 \text{ mod } 713)\;(12^2 mod\; 713)\;(12^1 \text{mod } 713)\;) \text{ mod } 713$$

$$= (485 \;*\; 629 \;*\; 144 \;*\; 12) \text{mod } 713$$

$$= \; 527152320 \text{ mod } 713 \; = 48$$

These can be done by at most floor($\log_2$ n) - 1 of multiple/divide operations.

In summary,

Adding x and y takes $O(n)$;

Multiplying x and y takes $O(n^2)$;

Computing $x^y \bmod N$ takes $O(\log^2 N)$ time; and

Dividing x by y takes $O(n^2)$