Number Theory

Ms. Swati Mali
swatimali@somaiya.edu
Assistant Professor, Department of Computer Engineering
K. J. Somaiya College of Engineering
Somaiya Vidyavihar University

Number Theory

- The goal of every cryptographic scheme is to be "crack proof"
- Cryptography is also a means to ensure the integrity and preservation of data from tampering.
- Modern cryptographic systems relies highly on functions associated with advanced mathematics, including number theory
- Number Theory explores the properties of numbers and the relationships between numbers.

Why?

- prime numbers and functions related to prime numbers

Objectives

•To study integer arithmetic, concentrating on divisibility and finding the greatest common divisor using the Euclidean algorithm

•To understand how the extended Euclidean algorithm can be used to solve linear congruent equations, and to find the multiplicative inverses

•To emphasize the importance of modular arithmetic and the modulo operator

•To emphasize and review matrices and operations on residue matrices that are extensively used in cryptography

•To solve a set of congruent equations using residue matrices

Cryptography is based on some specific areas of mathematics, including number theory, linear algebra, and algebraic structures.

INTEGER ARITHMETIC

- Concept of set and a few operations.
- Set of Integers
- Binary Operations
- Integer Division
- Divisibility
  - Euclidean Algorithm
  - The Extended Euclidean Algorithm

Set of Integers

- Set of Integers

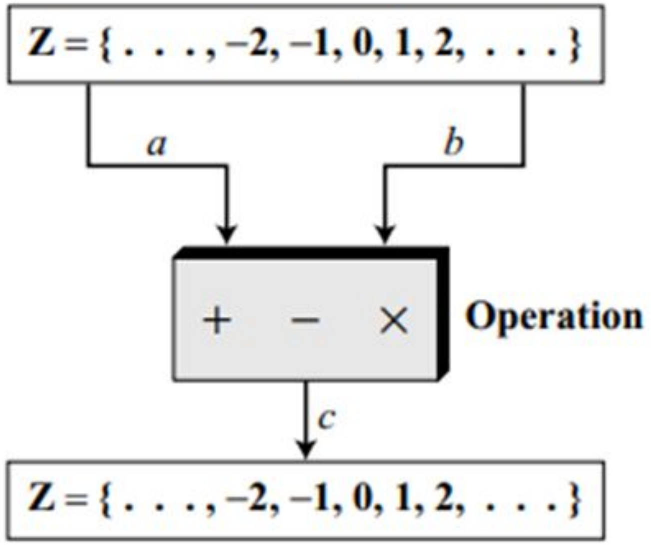
The set of integers contains all integral numbers (with no fraction) from negative infinity to positive infinity

$Z = \{ . . . , −2, −1, 0, 1, 2, . . . \}$

Binary Operations

- Three binary operations applied to the set of integers : addition, subtraction, and multiplication, division
- A binary operation takes two inputs and creates one output.
- Each operations takes two inputs (a and b) and creates one output (c)
- The two inputs come from the set of integers; the output goes into the set of integers

Figure 2.2 *Three binary operations for the set of integers*


Z = { . . . , –2, –1, 0, 1, 2, . . . }
a
b
+ – ×
Operation
c
Z = { . . . , –2, –1, 0, 1, 2, . . . }

Example 2.1

The following shows the results of the three binary operations on two integers. Because each input can be either positive or negative, we can have four cases for each operation.

Add:	$5 + 9 = 14$	$(-5) + 9 = 4$	$5 + (-9) = -4$	$(-5) + (-9) = -14$
Subtract:	$5 - 9 = -4$	$(-5) - 9 = -14$	$5 - (-9) = 14$	$(-5) - (-9) = +4$
Multiply:	$5 \times 9 = 45$	$(-5) \times 9 = -45$	$5 \times (-9) = -45$	$(-5) \times (-9) = 45$

Integer Division

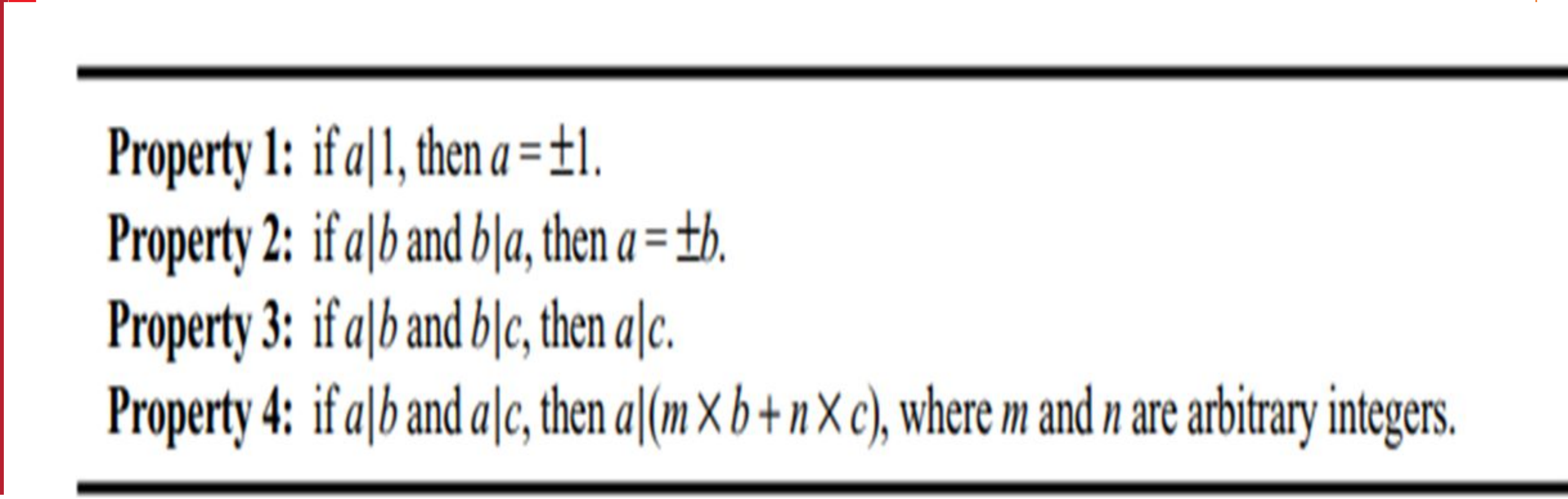
- The division produces two outputs instead of one: Quotient and Remainder
- The operation a ÷ n gives q and r.
- The relationship between these four integers:

a = q × n + r

- a => dividend;
- Q => quotient;
- N => divisor
- R => remainder.
- Division is not taken as an operation, because the result of dividing a by n is two integers, q and r.
- It is taken as division relation

Divisibility

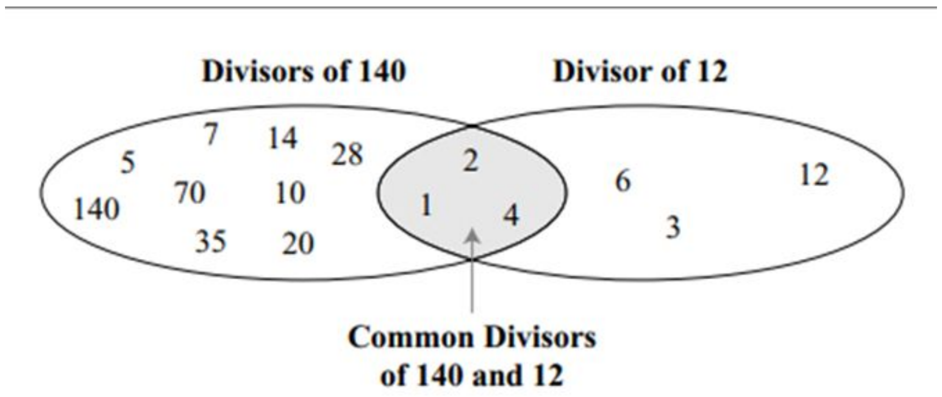
- Depends on if r=0 in the division relation
- $a|n$ if a is divisible by n
- $a\nmid n$ otherwise



Property 1: if $a|1$, then $a = \pm 1$.
Property 2: if $a|b$ and $b|a$, then $a = \pm b$.
Property 3: if $a|b$ and $b|c$, then $a|c$.
Property 4: if $a|b$ and $a|c$, then $a|(m \times b + n \times c)$, where m and n are arbitrary integers.

Greatest Common Divisor

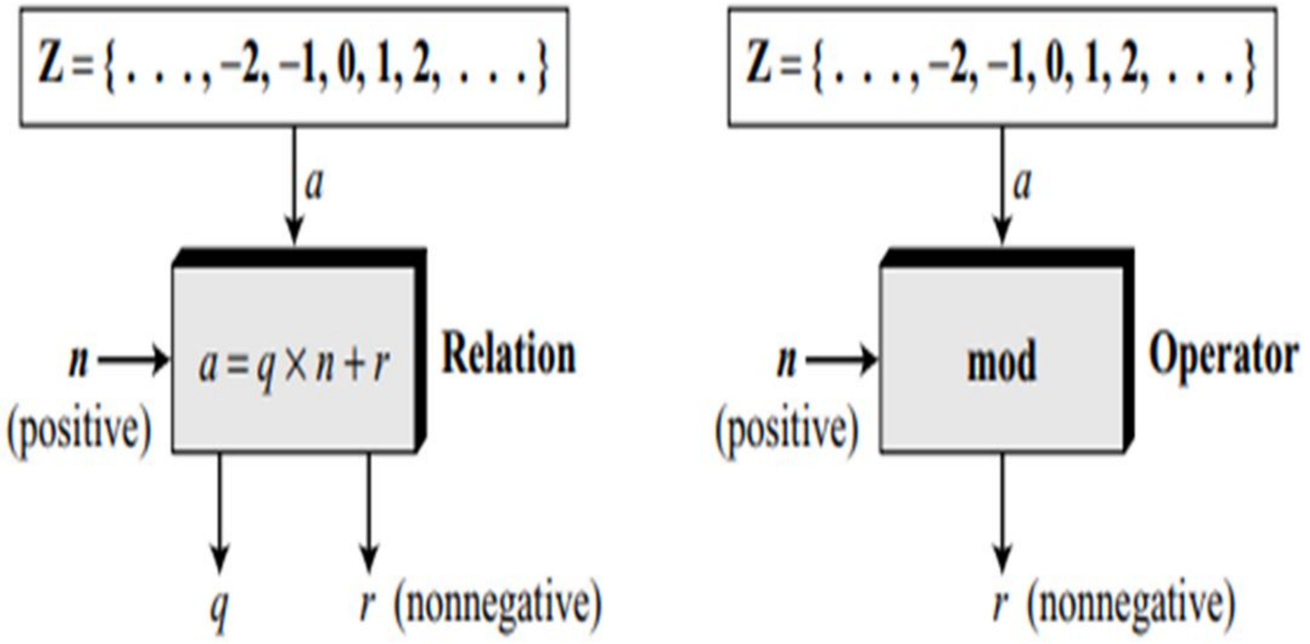
- Two positive integers may have many common divisors, but only one greatest common divisor.
- E.g. the common divisors of 12 and 140 are 1, 2, and 4. However, the greatest common divisor is 4.
- The greatest common divisor of two positive integers is the largest integer that can divide both integers.
-



MODULAR ARITHMETIC

- The modular arithmetic is interested in only one of the outputs, the remainder r.
- The quotient q is not significant
- Mod=modulo operator
- a mod n ⇒ The second input (n) is called the modulus.
- r= a mod n ⇒ The output r is called the residue.

Division relation and modulo operator



Set of Residues: Zn

- The result of the modulo operation with modulus n is always an integer between 0 and n − 1.
- i.e. result of a mod n is always a nonnegative integer < n.
- i.e. modulo operation creates a set, referred to as the set of least residues modulo n, or Z_n.

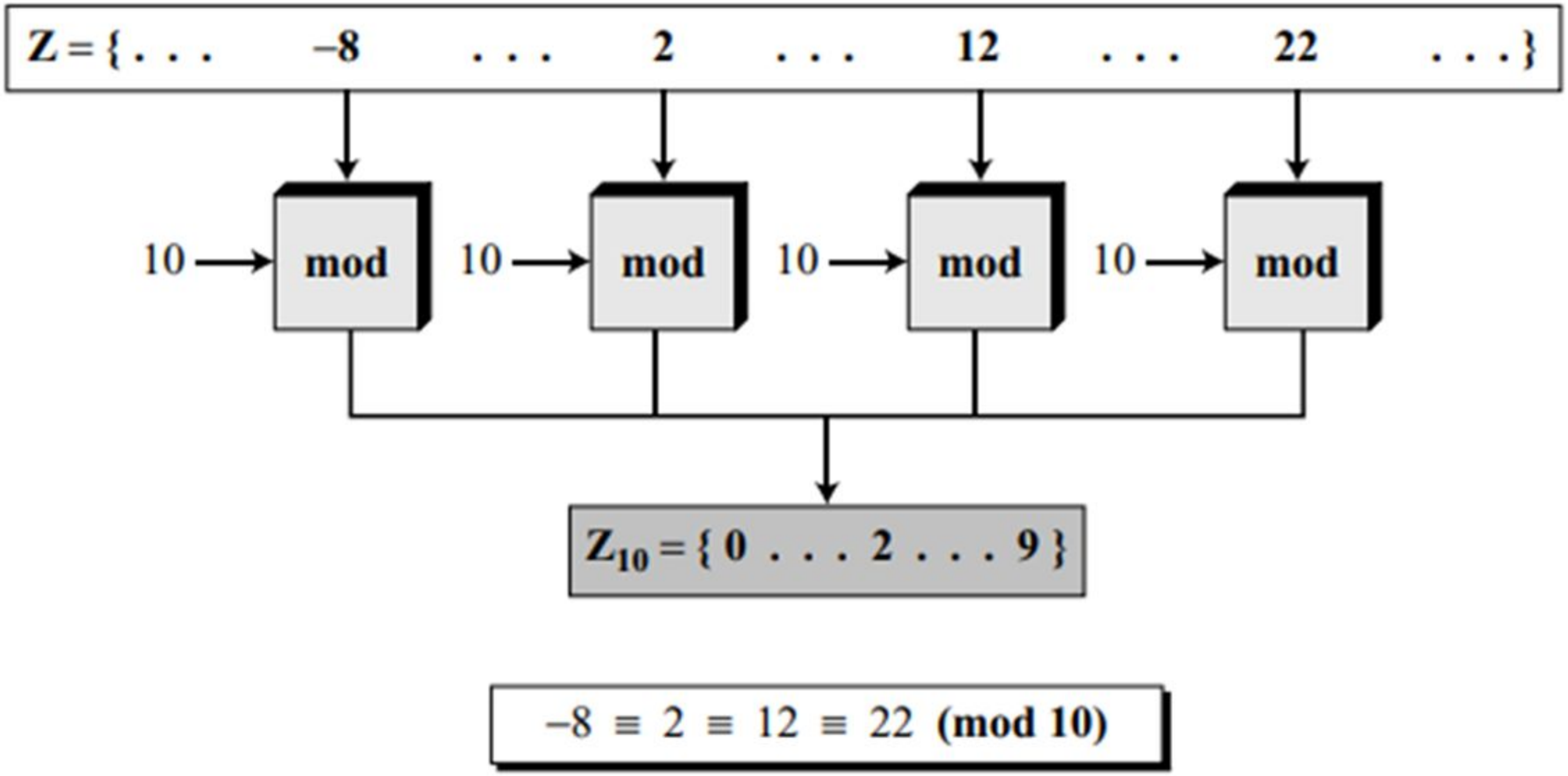
Zn = { 0, 1, 2, 3, ... , (n − 1) }

- $Z_2 = \{ 0, 1 \}$
- $Z_6 = \{ 0, 1, 2, 3, 4, 5 \}$
- $Z_{11} = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \}$

Congruence

- Cryptography uses the concept of congruence instead of equality.
- Mapping from Z to Z_n is not one-to-one.
- Infinite members of Z can map to one member of Z_n.
- E.g. , 2 mod 10 = 2, 12 mod 10 = 2, 22 mod 2 = 2, and so on.
- Hence in modular arithmetic, integers like 2, 12, and 22 are called congruent mod 10.
- Denoted with **congruence operator (≡).**
- $2 \equiv 12 \pmod{10}$, $13 \equiv 23 \pmod{10}$, $34 \equiv 24 \pmod{10}$,
- $-8 \equiv 12 \pmod{10}$, $3 \equiv 8 \pmod{5}$, $8 \equiv 13 \pmod{5}$,

$23 \equiv 33 \pmod{5}$, $-8 \equiv 2 \pmod{5}$


$Z = \{ \ldots -8 \ldots 2 \ldots 12 \ldots 22 \ldots \}$
10
mod
10
mod
10
mod
10
mod
$Z_{10} = \{ 0 \ldots 2 \ldots 9 \}$
$-8 \equiv 2 \equiv 12 \equiv 22 \pmod{10}$

Congruence Relationship

A residue class

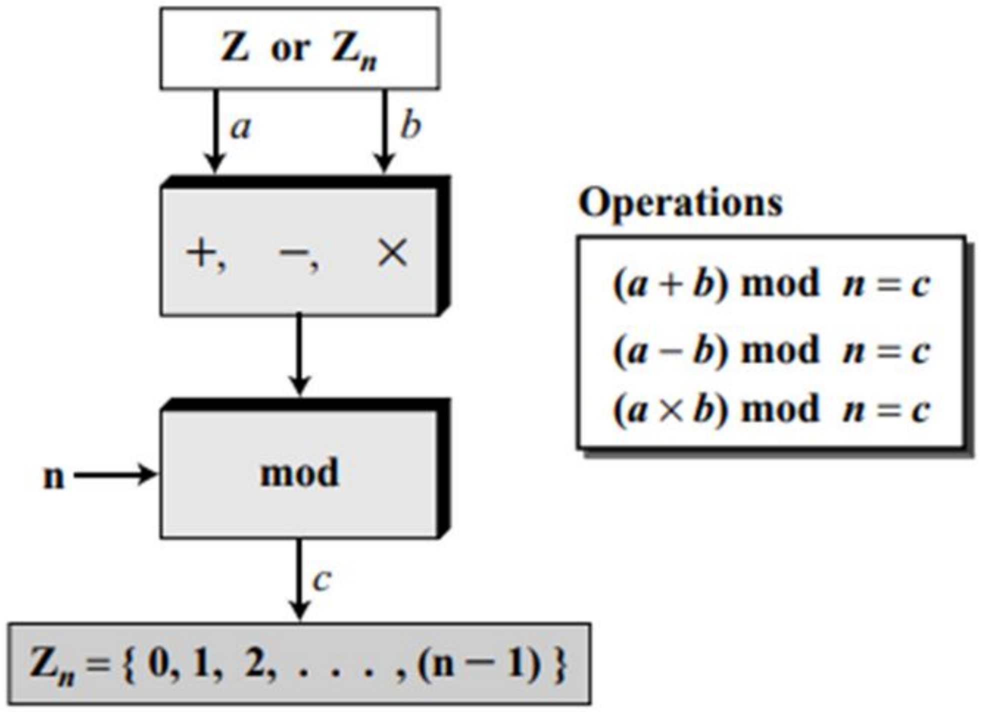
- A residue class [a] or [a]n is the set of integers congruent modulo n.
- i.e. the set of all integers x such that

$$x = a \pmod{n}.$$

- E.g. n = 5, gives five sets [0], [1], [2], [3], and [4] as :
- [0] = {…, −15, −10, −5, 0, 5, 10, 15, …}
- [1] = {…, −14, −9, −4, 1, 6, 11, 16, …}
- [2] = {…, −13, −8, −3, 2, 7, 12, 17, …}
- [3] = {…, −12, −7, −5, 3, 8, 13, 18, …}
- [4] = {…, −11, −6, −1, 4, 9, 14, 19, …}
- The set of all of these least residues: Z_5 = {0, 1, 2, 3, 4}.
- i.e, the set Z_n is the set of all least residue modulo n.

Operations in Z_n

•Addition, subtraction, and multiplication can also be defined for the set Zn.



Inverses

- modular arithmetic often needs to find the inverse of a number relative to an operation.
- Types:
  - additive inverse (relative to an addition operation) or
  - a multiplicative inverse (relative to a multiplication

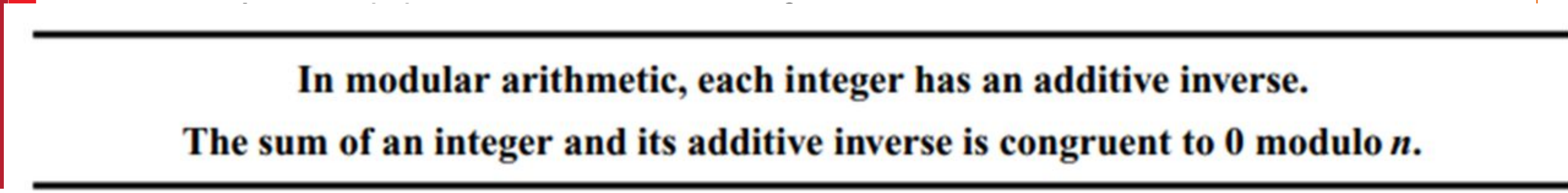
Additive Inverse In Z_n

- In Z_n, two numbers a and b are additive inverses of each other if

$$a + b \equiv 0 \pmod{n}$$

- The additive inverse of a can be calculated as

$$b = n - a.$$



In modular arithmetic, each integer has an additive inverse.

The sum of an integer and its additive inverse is congruent to 0 modulo *n*.

- Find all additive inverse pairs in Z_{10}

Solution: The six pairs of additive inverses are :

(0, 0), (1, 9), (2, 8), (3, 7), (4, 6), and (5, 5).

- 0 is the additive inverse of itself; so is 5.
- Note : the additive inverses are reciprocal;
  - E.g. if 4 is the additive inverse of 6, then 6 is also the additive inverse of 4.

Additive inverse of –ve number

- Formula : -n mod k ≡ k - (n mod k)
- e.g. -3 mod 12 = 12 - (3 mod 12) = 12 -3 = 9
- -34 mod 23 = 23 - (34 mod 23) = 23 - 11 = 12
-

Hint: if n < k, compute k-n

- else take multiple of k which is just greater than n, subtract n from that multiple
- e.g. 23 * 2 = 46 > n=34 ⇒ 46-34 = 12
-

Example operations over Z_n

1.Add 7 to 14 in Z_{15}.

Solution: (14 + 7) mod 15 → (21) mod 15 = 6

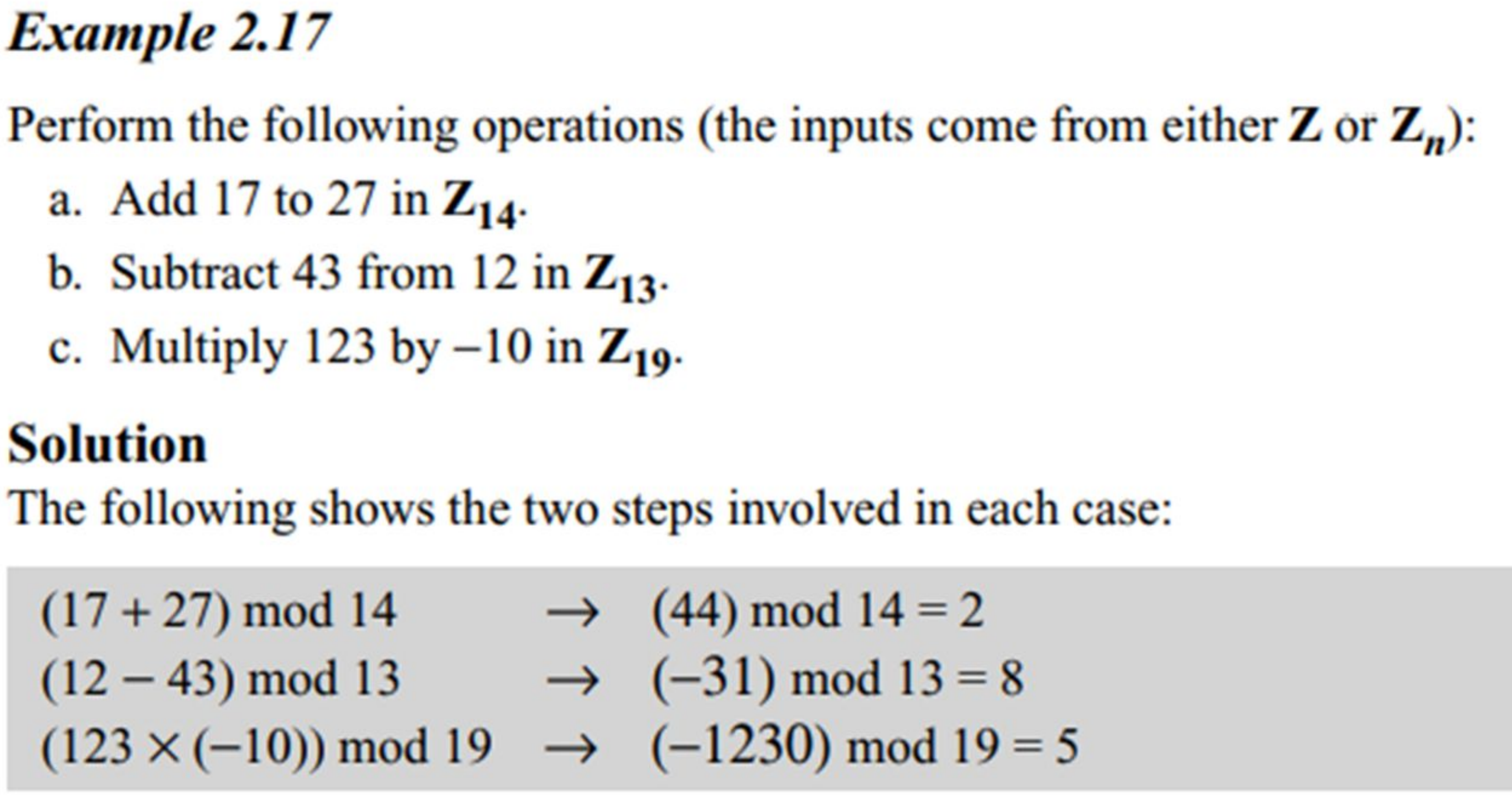
2.Subtract 11 from 7 in Z_{13}.

(7 − 11) mod 13 → (−4) mod 13 = 9 (Additive inverse)

3. Multiply 11 by 7 in Z20

(7 × 11) mod 20 → (77) mod 20 = 17

Example operations over Zn



Example 2.17

Perform the following operations (the inputs come from either $\mathbf{Z}$ or $\mathbf{Z}_n$):

a. Add 17 to 27 in $\mathbf{Z}_{14}$.

b. Subtract 43 from 12 in $\mathbf{Z}_{13}$.

c. Multiply 123 by −10 in $\mathbf{Z}_{19}$.

Solution

The following shows the two steps involved in each case:

$$\begin{aligned}
&(17 + 27) \bmod 14 &&\rightarrow &&(44) \bmod 14 = 2\\
&(12 - 43) \bmod 13 &&\rightarrow &&(-31) \bmod 13 = 8\\
&(123 \times (-10)) \bmod 19 &&\rightarrow &&(-1230) \bmod 19 = 5
\end{aligned}$$

Addition and Multiplication In cryptography

- If the sender uses an integer (as the encryption key), the receiver uses the inverse of that integer (as the decryption key).
- If the cryptographic operation is addition, Zn gives set of an additive inverse.
- if the operation is multiplication, Zn* gives multiplicative inverse

- Use Z_n for additive inverses
- use Z_{n*} when multiplicative inverses are needed.
- $Z_6 = \{0, 1, 2, 3, 4, 5\}$ $Z_{6*} = \{1, 5\}$
- $Z_7 = \{0, 1, 2, 3, 4, 5, 6\}$ $Z_{7*} = \{1, 2, 3, 4, 5, 6\}$
- $Z_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $Z_{10*} = \{1, 3, 7, 9\}$

Zp and Zp* sets

- The modulus in Zp and Zp* sets is a prime number.
- a prime number has only two divisors: integer 1 and itself.
- The set Zp is the same as Zn except that n is a prime. i.e. Zp contains all integers from 0 to p − 1.
- Each member in Zp has an additive inverse;
- each member except 0 has a multiplicative inverse.

Zp and Zp* sets

- The set Zp* is the same as Zn* except that n is a prime.
- Zp* contains all integers from 1 to p − 1.
- Each member in Zp* has an additive and a multiplicative inverse.
- Zp* is a very good candidate when we need a set that supports both additive and multiplicative inverse.

Zp and Zp* set examples

- $Z_{13} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Multiplicative Inverse

- In Zn, two numbers a and b are the multiplicative inverse of each other if

$$a \times b \equiv 1 \pmod{n}$$

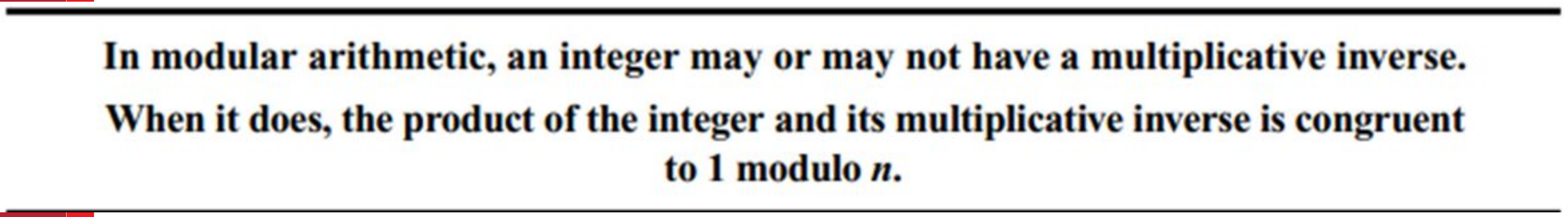
- E.g. , if the modulus is 10, then the multiplicative inverse of 3 is 7.

– i.e. $(3 \times 7) \bmod 10 = 1$.

- a has a multiplicative inverse in Zn

iff gcd (n, a) = 1. i.e, a and n are relatively prime.

Multiplicative Inverse



In modular arithmetic, an integer may or may not have a multiplicative inverse. When it does, the product of the integer and its multiplicative inverse is congruent to 1 modulo *n*.

Multiplicative inverse

- Given two integers a and m, find the modular multiplicative inverse of a under modulo m.
- The modular multiplicative inverse is an integer X such that:

$a*x \equiv 1 \pmod{m}$

If a=3, m=11 then x= 4 , 3 and 4 are called multiplicative inverses of each other modulo 11

Modular multiplicative inverse when M and A are coprime i.e. gcd(a, m)=1

Example

- Find 4^{-1} mod 23
- So 4* x = 1 (mod 23)

4* 1 = 4 mod 23 = 4

4*2 = 8 mod 23 = 8

4*3 =12 mod 23 =12

4*4 =16 mod 23 = 16

4*5 = 20 mod 23 =20

4*6 = 24 mod 23 =1

So 4 & 6 are modulo inverses of each other modulo 23

Examples

•Find the multiplicative inverse of 8 in Z_{10}.

Solution:

There is no multiplicative inverse because

gcd (10, 8) = 2 $\neq$ 1.

i.e. there exist no number between 0 and 9 such that when multiplied by 8, the result is congruent to 1.

Examples

- Find all multiplicative inverses in Z_{10}.

Solution : (1, 1), (3, 7) and (9, 9).

The numbers 0, 2, 4, 5, 6, and 8 do not have a multiplicative inverse.

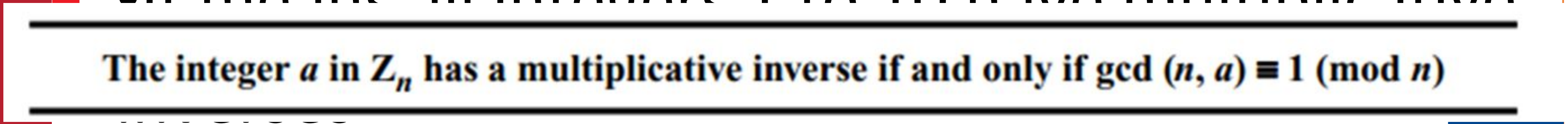
Examples

- Find all multiplicative inverse pairs in Z_{11}.

Solution: (1, 1), (2, 6), (3, 4), (5, 9), (7, 8), (9, 9), and (10, 10).

- In Z_{11}, gcd (11, a) is 1 (relatively prime) for all values of a except 0.

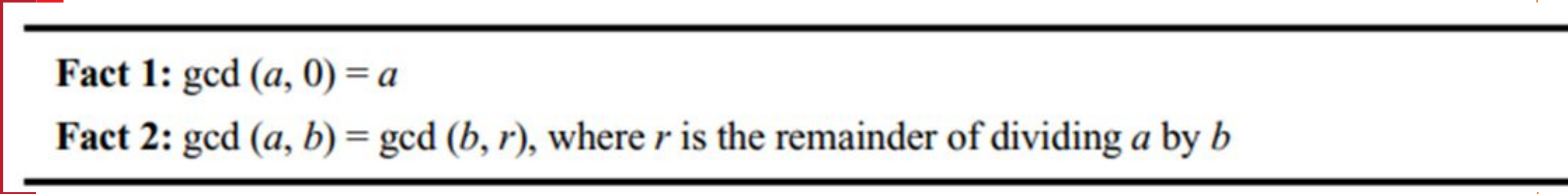
- It means all integers 1 to 10 have multiplicative



The integer a in Z_n has a multiplicative inverse if and only if gcd $(n, a) \equiv 1 \pmod{n}$

Euclidean Algorithm

- Finding gcd of two positive integers by listing all common divisors is not practical when the two integers are large.
- Euclid(BC 300) developed an algorithm to find the GCD of two positive integers.



Fact 1: gcd $(a, 0) = a$

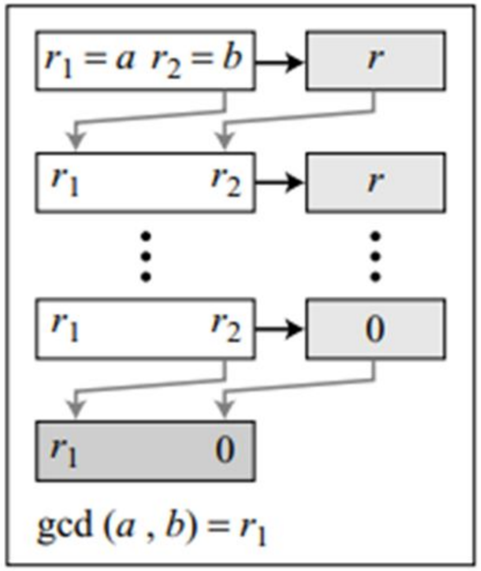
Fact 2: gcd $(a, b) =$ gcd (b, r), where r is the remainder of dividing a by b

Euclidean Algorithm

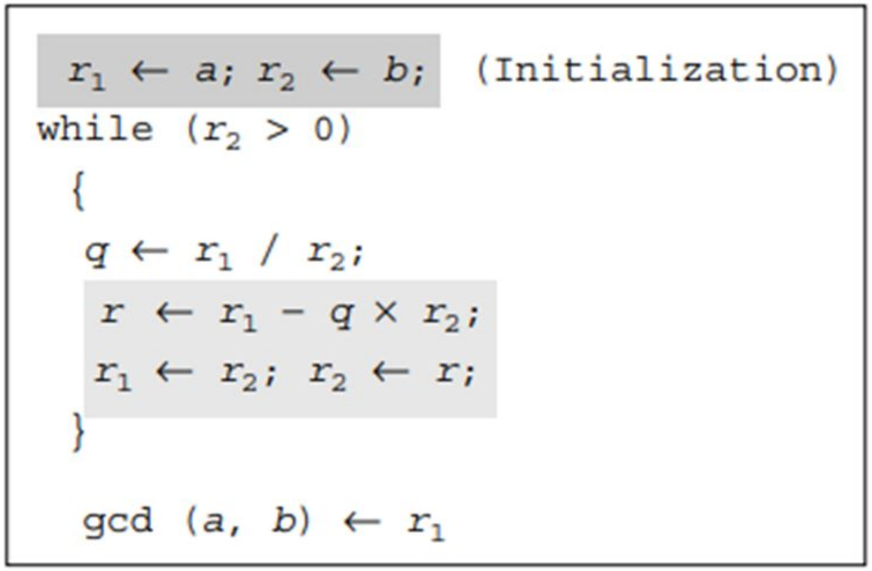
- Fact 1: if the second integer is 0, the greatest common divisor is the first one.
- Fact 2: allows us to change the value of a, b until b becomes 0
- E.g. calculate gcd (36, 10). (use the second fact several times)

gcd (36, 10) = gcd (10, 6) = gcd (6, 4) = gcd (4, 2) = gcd (2, 0) = 2

Euclidean Algorithm



a. Process



```
r₁ ← a; r₂ ← b;   (Initialization)
while (r₂ > 0)
  {
    q ← r₁ / r₂;
     r ← r₁ − q × r₂;
     r₁ ← r₂; r₂ ← r;
  }

    gcd (a, b) ← r₁
```

b. Algorithm

•When gcd (a, b) = 1, a and b are termed as relatively prime.

Euclidean Algorithm

Example 2.7

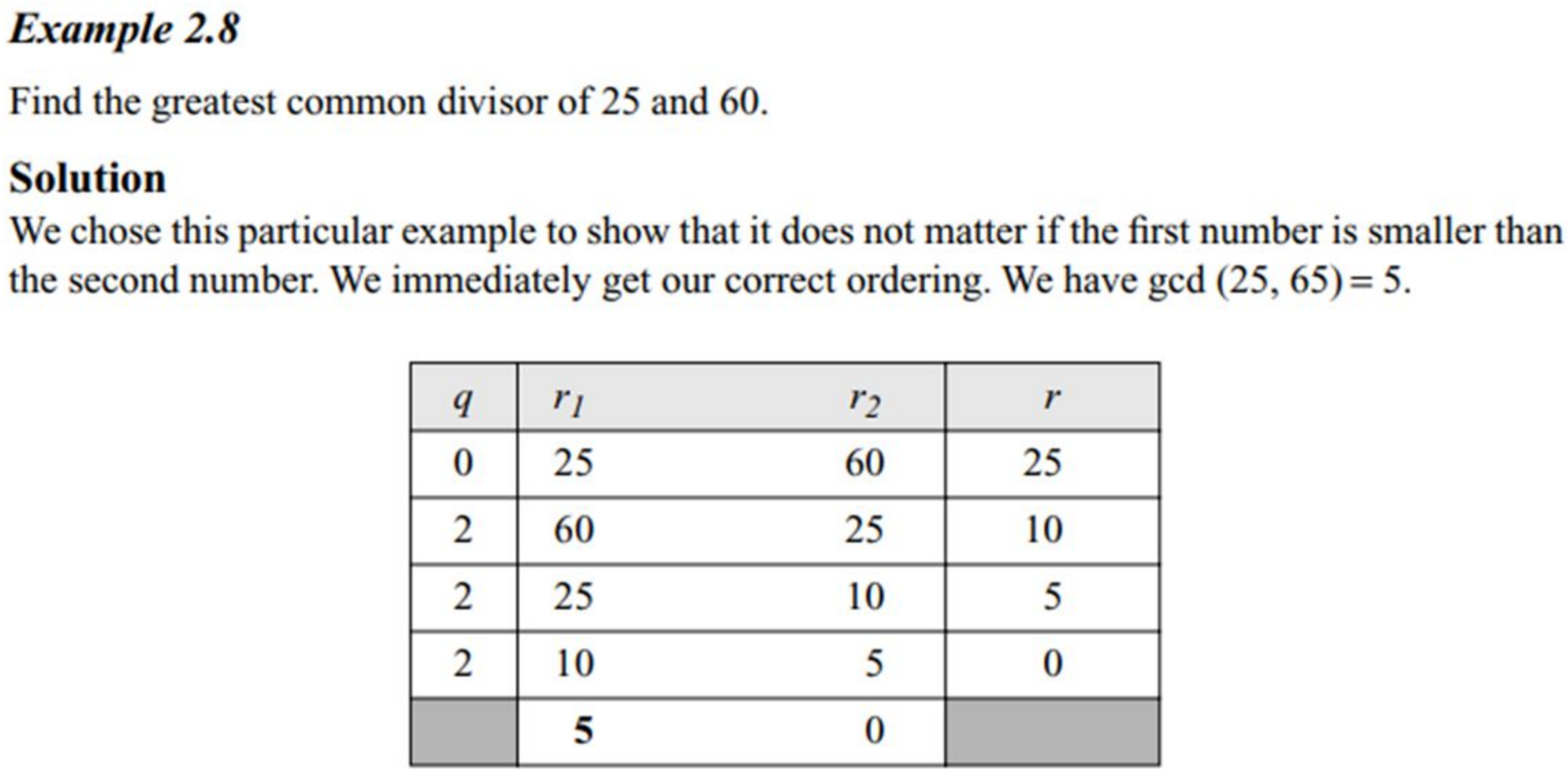
Find the greatest common divisor of 2740 and 1760.

Solution

We apply the above procedure using a table. We initialize r_1 to 2740 and r_2 to 1760. We have also shown the value of q in each step. We have gcd (2740, 1760) = 20.

q	r_1	r_2	r
1	2740	1760	980
1	1760	980	780
1	980	780	200
3	780	200	180
1	200	180	20
9	180	20	0
	20	0	

The Euclidean Algorithm



Example 2.8

Find the greatest common divisor of 25 and 60.

Solution

We chose this particular example to show that it does not matter if the first number is smaller than the second number. We immediately get our correct ordering. We have gcd (25, 65) = 5.

q	r_1	r_2	r
0	25	60	25
2	60	25	10
2	25	10	5
2	10	5	0
	5	0	

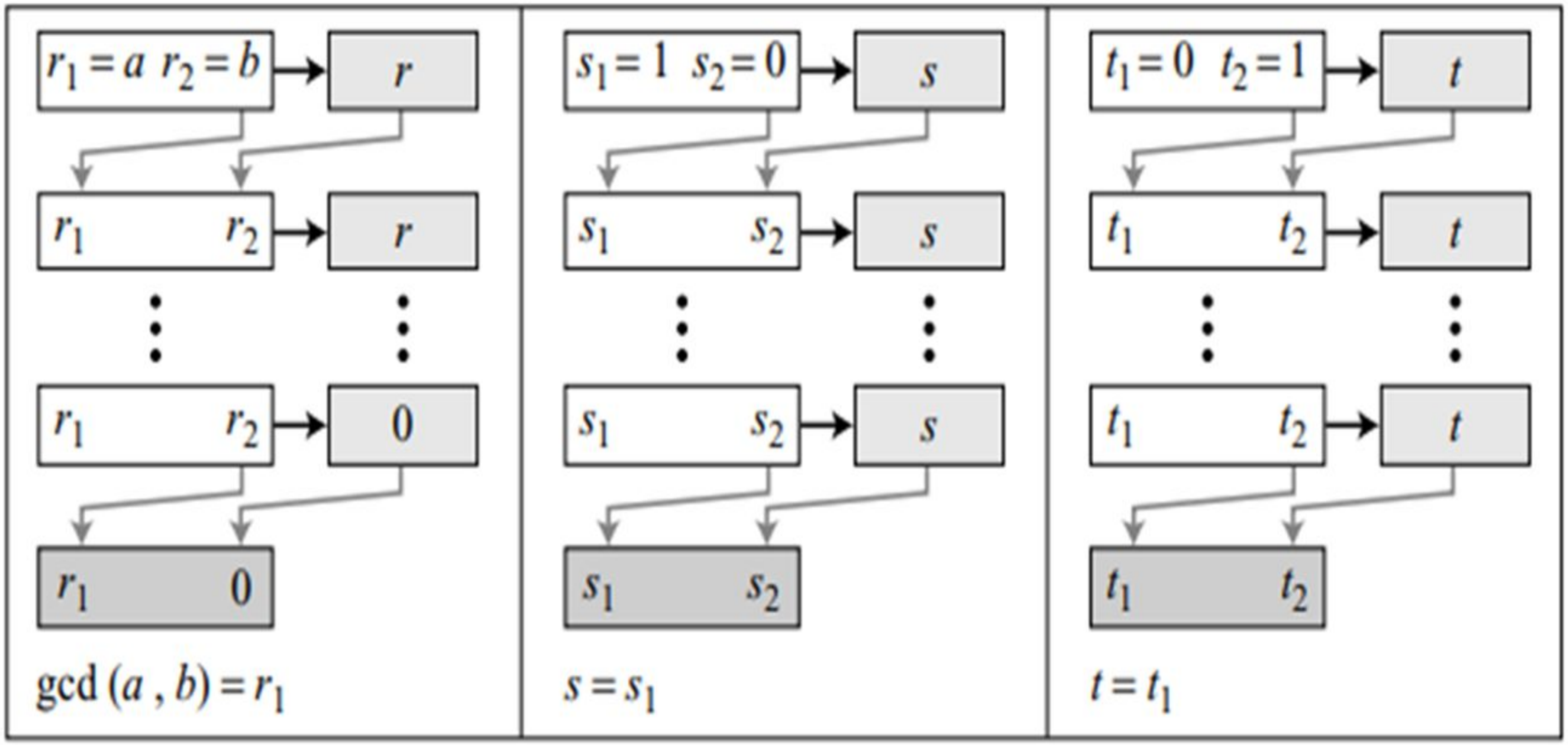
The Extended Euclidean Algorithm

- Given two integers a and b, the extended Euclidean algorithm helps to find other two integers, s and t, such that

$$s \times a + t \times b = \gcd(a, b)$$

- The extended Euclidean algorithm can calculate the gcd (a, b) and at the same time calculate the value of s and t.
- **Used extensively in computing multiplicative inverses in simple algebraic field extensions**.

Extended Euclidean Algorithm to find GCD



Extended Euclidean Algorithm

```
$r_1 \leftarrow a;\ r_2 \leftarrow b;$
$s_1 \leftarrow 1;\ s_2 \leftarrow 0;$          (Initialization)
$t_1 \leftarrow 0;\ t_2 \leftarrow 1;$
while ($r_2 > 0$)
{
    $q \leftarrow r_1 / r_2;$
    $r \leftarrow r_1 - q \times r_2;$
    $r_1 \leftarrow r_2;\ \ r_2 \leftarrow r;$       (Updating $r's$)
    $s \leftarrow s_1 - q \times s_2;$
    $s_1 \leftarrow s_2;\ \ s_2 \leftarrow s;$       (Updating $s's$)
    $t \leftarrow t_1 - q \times t_2;$
    $t_1 \leftarrow t_2;\ \ t_2 \leftarrow t;$       (Updating $t's$)
}
gcd ($a$ , $b$) $\leftarrow r_1;\ \ s \leftarrow s_1;\ \ t \leftarrow t_1$
```

Extended Euclidean Algorithm Examples

- •Given a = 161 and b = 28, find gcd (a, b). •GCD(161,28)=7

Q=r1/r2	R1	R2	R	s1	s2	s= s1 − q × s2	T1	T2	T=t1 − q × t2
5	161	28	21	1	0	1	0	1	-5
1	28	21	7	0	1	-1	1	-5	6
3	21	7	0	1	-1	4	-5	6	-23
	7	0		-1	4		6	-23	6

Solve using Extended Euclidean Algorithm

1. Find the greatest common divisor (GCD) of 120 and 23
2. Find the greatest common divisor (GCD) of 1071 and 462
3. Compute GCD(25,18)
4. Compute GCD(56,98)

- (GCD) of 120 and 23
- Multiplicative inverse of 23 modulo 120 is 47.

Q	A	B	R	S1	S2	S= S1-Q*S2	T1	T2	T=T1-Q *T2
5	120	23	5	1	0	1	0	1	-5
4	23	5	3	0	1	-4	1	-5	21
1	5	3	2	1	-4	5	-5	21	-26
1	3	2	1	-4	5	-9	21	-26	47
2	2	1	0	5	-9	23	-26	47	-120
	1	0		-9	23		47	-120	

- Compute GCD(25,18)

Q	A	B	R	t1	t2	t3
1	25	18	7	0	1	-1
2	18	7	4	1	-1	3
1	7	4	3	-1	3	-4
1	4	3	1	3	-4	7
3	3	1	0	-4	**7**	-25
	1	0		**7**	-25	

- multiplicative inverse of 18 modulo 25 is 7.

- GCD(25,18)
- GCD(25,18)=1, multiplicative inverse of 18 modulo 25 is 7.

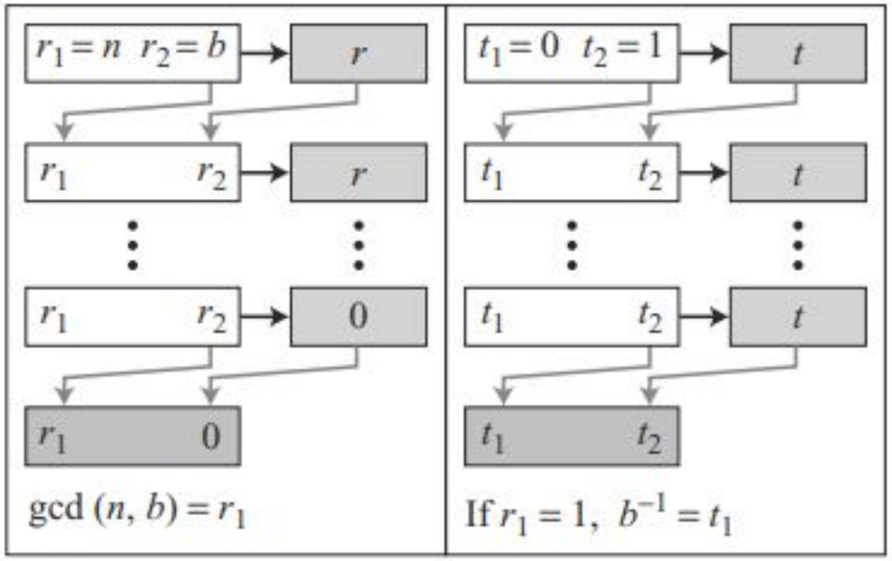
Q	A	B	R	t1	t2	t3
1	25	18	7	0	1	-1
2	18	7	4	1	-1	3
1	7	4	3	-1	3	-4
1	4	3	1	3	-4	7
3	3	1	0	-4	7	-25
	1	0		7	-25	

- GCD(56,98)

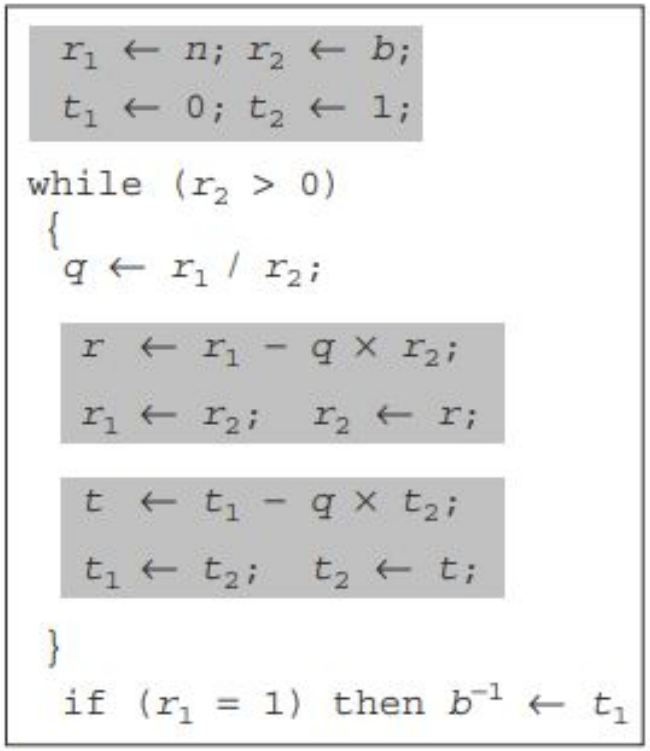
Extended Euclidean algorithm and inverse

- The extended Euclidean algorithm finds the multiplicative inverses of b in Z_n when n and b are given and gcd (n, b) = 1.
- The multiplicative inverse of b is the value of t after being mapped to Z_n.

Extended Euclidean Algorithm to compute multiplicative inverse



a. Process



```
r_1 ← n; r_2 ← b;
t_1 ← 0; t_2 ← 1;

while (r_2 > 0)
{
  q ← r_1 / r_2;

  r ← r_1 − q × r_2;
  r_1 ← r_2;   r_2 ← r;

  t ← t_1 − q × t_2;
  t_1 ← t_2;   t_2 ← t;
}
if (r_1 = 1) then b^{-1} ← t_1
```

b. Algorithm

Find multiplicative inverse using Extended Euclidean algorithm

Q=A/B	A	B	R= A % B	T1	T2	T= T1-Q*T2
				0	1	

A > B
Q=A/B
R= A%B
T1=0, T2 =1 T= T1-Q*T2

A=B
B=R
T1=T2
T2=T

Find the multiplicative inverse of 11 in Z_{26}

Q=r1/r2	r1	r2	R	T1	T2	T=t1 − q × t2
2	26	11	4	0	1	-2
2	11	4	3	1	-2	5
1	4	3	1	-2	5	-7
3	3	1	0	5	-7	26
	1	0		-7	26	

GCD(26,11)=1, i.e. multiplicative inverse exists
Multiplicative inverse = T1= -7 = 26-7= **19**

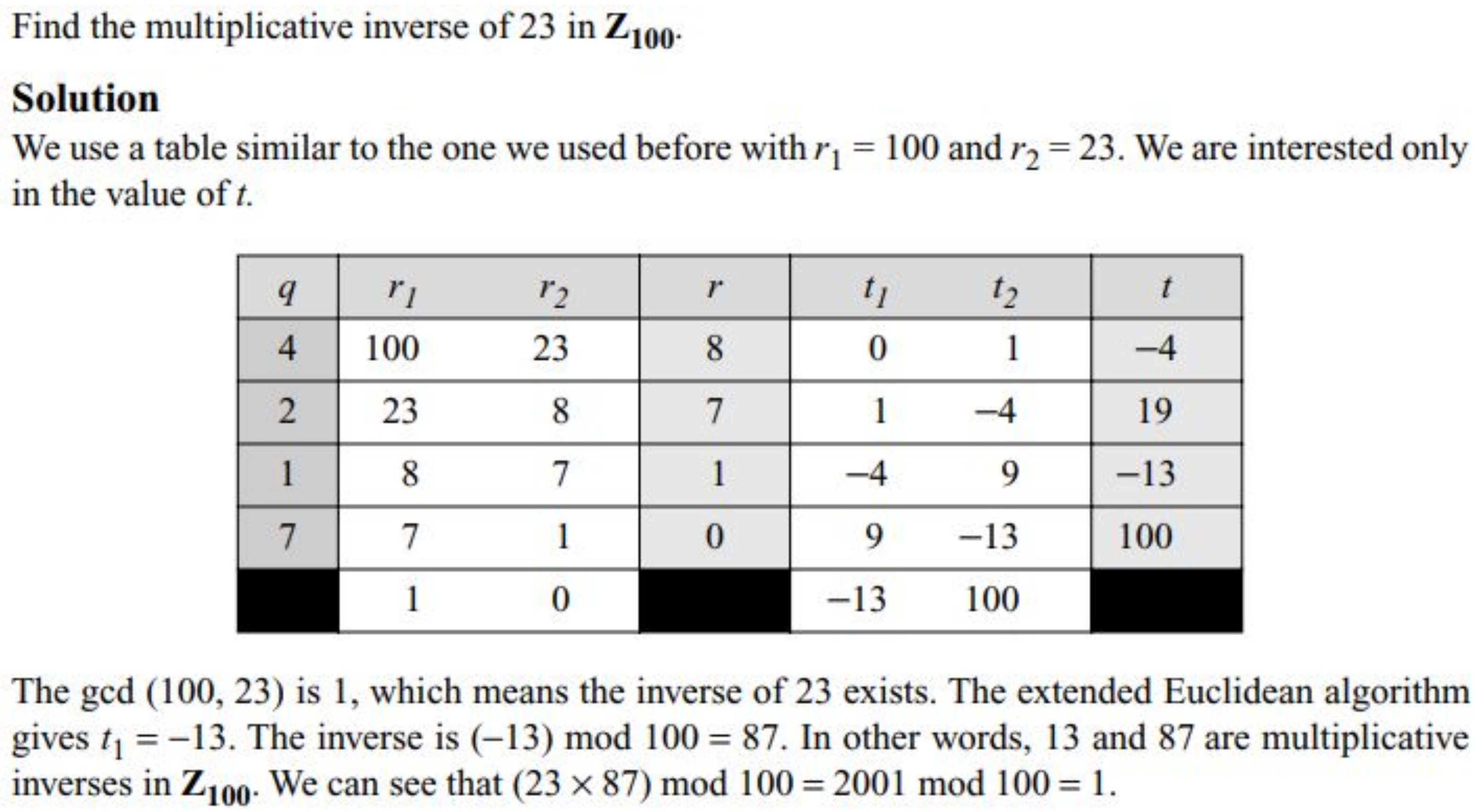
Find the inverse of 12 in $\mathbf{Z}_{26}$.

Solution

We use a table similar to the one we used before, with $r_1 = 26$ and $r_2 = 12$.

q	r_1	r_2	r	t_1	t_2	t
2	26	12	2	0	1	−2
6	12	2	0	1	−2	13
	2	0		−2	13	

The gcd (26, 12) = 2 ≠ 1, which means there is no multiplicative inverse for 12 in $\mathbf{Z}_{26}$.



Find the multiplicative inverse of 23 in $\mathbf{Z}_{100}$.

Solution

We use a table similar to the one we used before with $r_1 = 100$ and $r_2 = 23$. We are interested only in the value of t.

q	r_1	r_2	r	t_1	t_2	t
4	100	23	8	0	1	−4
2	23	8	7	1	−4	19
1	8	7	1	−4	9	−13
7	7	1	0	9	−13	100
	1	0		−13	100	

The gcd (100, 23) is 1, which means the inverse of 23 exists. The extended Euclidean algorithm gives $t_1 = -13$. The inverse is $(-13) \bmod 100 = 87$. In other words, 13 and 87 are multiplicative inverses in $\mathbf{Z}_{100}$. We can see that $(23 \times 87) \bmod 100 = 2001 \bmod 100 = 1$.

Applications of linear Congruence

- Using the Extended Euclidean algorithm, find the greatest common divisor of:
  - 88 and 220
  - 300 and 42
  - 24 and 320
  - 401 and 700
- Find the results of the following operations.
  - 22 mod 7
  - 140 mod 10
  - −78 mod 13
  - 0 mod 15

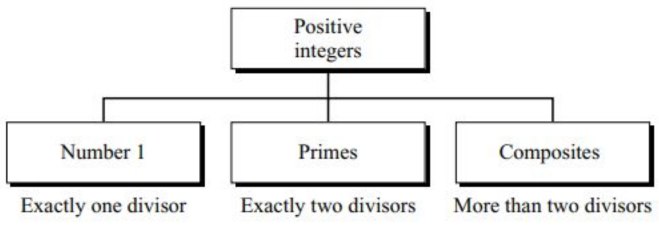
- Find the multiplicative inverse of each of the following integers in Z180 using the extended Euclidean algorithm.
  - 38
  - 7
  - 132
  - 24
- Find Zp & Zp* for p= :
  - **589**
  - **17**
  - **21**
  - **7**

Mathematics of asymmetric key cryptography

Primes

- The positive integers can be divided into three groups: the number 1, primes, and composites
- A positive integer is a prime iff it is exactly divisible by two integers, 1 and itself.
- A composite is a positive integer with more than two divisors.

Figure 9.1 *Three groups of positive integers*



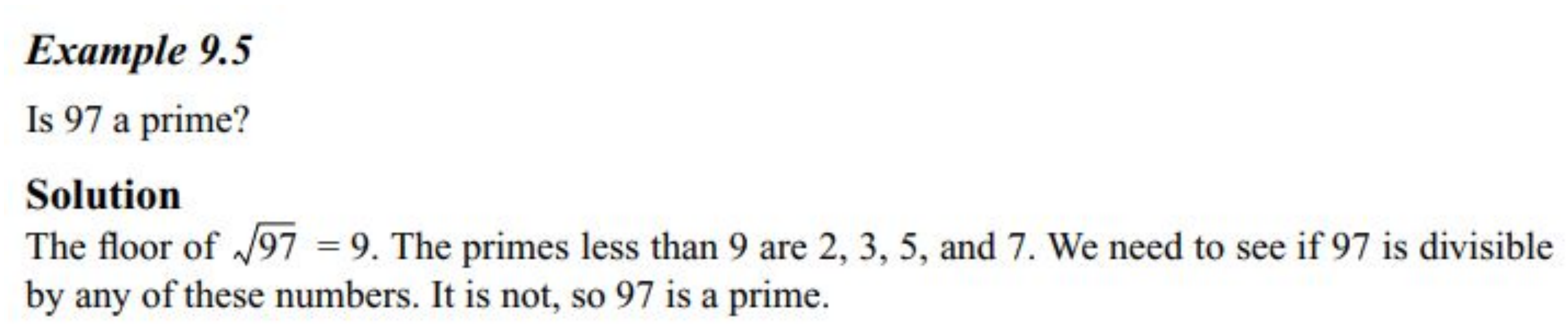
Coprimes

- Two positive integers, a and b, are relatively prime, or coprime, if gcd (a, b) = 1.
- Number 1 is relatively prime with any integer.
- If p is a prime, then all integers 1 to p − 1 are relatively prime to p.

- **<u>There is an infinite number of primes</u>**

Checking for Primality

- Given a number n, if the number is divisible by all primes less than sqrt(n) then it’s a prime number



Example 9.5

Is 97 a prime?

Solution

The floor of $\sqrt{97} = 9$. The primes less than 9 are 2, 3, 5, and 7. We need to see if 97 is divisible by any of these numbers. It is not, so 97 is a prime.

Algorithms

- Euler’s totient/ phi function
- Prime number generation
  - Sieve of Eratosthenes
  - Sieve of Atkin
  - Segmented Sieve
- Primality testing
  - Miller-Rabin, Fermat’s test, square root test – Probabilistic
  - AKS(Agrawal, Kayal, and Saxena) algorithm - deterministic
- Prime factorization
  - Trial division method
  - Pollard’s Rho
  - Pollard p – 1 Method
  - Fermat’s method

Euler's Phi-Function

- AKA Euler's totient function

1. $\Phi(1) = 0$.
2. $\Phi(p) = (p - 1)$ if p is a prime
3. $\Phi(m \times n) = \Phi(m) \times \Phi(n)$ if m and n are relatively prime.
4. $\Phi(p^e) = p^e - p^{e-1}$ if p is a prime

The difficulty of finding Φ(n) depends on the difficulty of finding the factorization of n.

Euler's ' Φ' Function

An arithmetic function Euler's Totient function 'Φ' is defined as

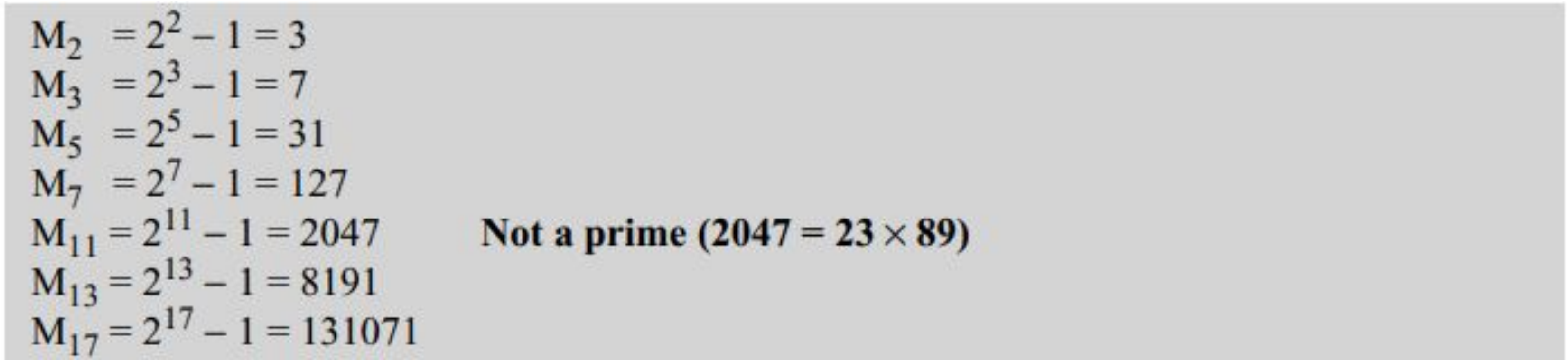
- Φ (n) = number of positive integers less than or equal to n and co-prime with n
- i.e. Φ (n) = number of positive integers ' a ' such that 1<= a <= n and GCD(a,n)=1
- Φ (15)= 8 as primes relative to 15 are given by 1, 2, 4, 7, 8, 11, 13, and 14.
- Φ(7)=6, Φ(17)=16
- Φ(m*n)= Φ(m)* Φ(n) where m and n are relatively prime

Why prime numbers?

- Primes are foundational in:
  - Hard mathematical problems (factoring, discrete logs)
  - Strong, well-understood security foundations
  - Efficient algorithms in modular arithmetic
  - Predictable group properties for encryption/signing
- Using primes ensures the modulus has fewer factors, which:
  - Prevents certain factorization shortcuts
  - Increases resistance to attacks like **Pollard's rho** or **trial division**
- Increase Randomness and Unpredictability
- Give Mathematical Proof of Security
- Primes make the math **cleaner, faster, and more secure**.

Generating Primes

- Mersenne and Fermat formula
- Mersenne numbers, can enumerate all primes for a number p
- $M_p = 2^p - 1$
- Mersenne number may or may not be a prime.



$$\begin{aligned}
M_2 &= 2^2 - 1 = 3 \\
M_3 &= 2^3 - 1 = 7 \\
M_5 &= 2^5 - 1 = 31 \\
M_7 &= 2^7 - 1 = 127 \\
M_{11} &= 2^{11} - 1 = 2047 && \textbf{Not a prime } (2047 = 23 \times 89) \\
M_{13} &= 2^{13} - 1 = 8191 \\
M_{17} &= 2^{17} - 1 = 131071
\end{aligned}$$

Fermat Primes

$$F_n = 2^{2^n} + 1$$

Fermat tested numbers up to F_4, but it turned out that F_5 is not a prime. No number

$F_0 = 3$
$F_1 = 5$
$F_2 = 17$
$F_3 = 257$
$F_4 = 65537$
$F_5 = 4294967297 = 641 \times 6700417$ **Not a prime**

Generating Primes-Sieve of Eratosthenes

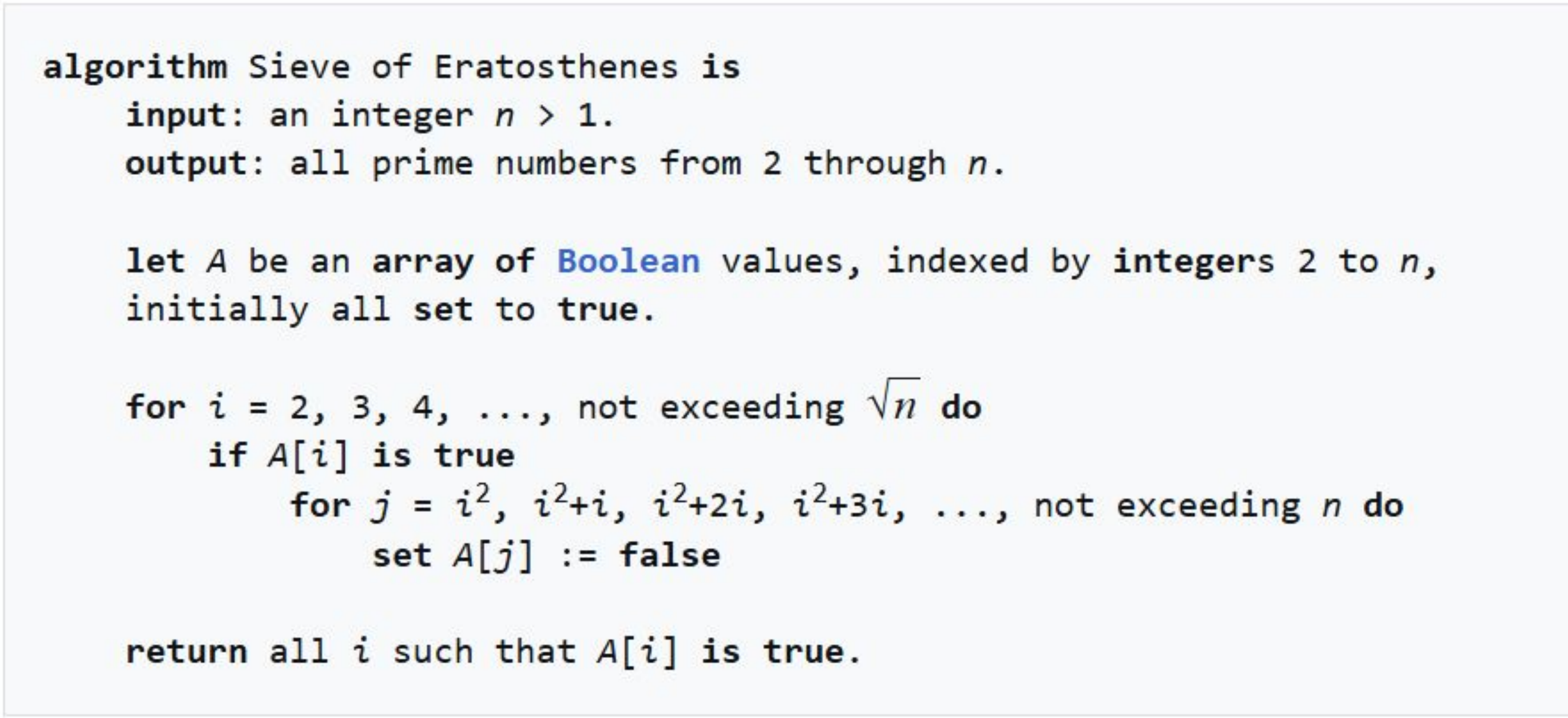
- Start with a list of numbers from 2 to n, all marked as potential primes.
- The first prime is 2 — mark all multiples of 2 (except 2 itself) as composite.
- Move to the next unmarked number (3) — mark all its multiples as composite.
- Repeat until the current number exceeds √n.
- Remaining unmarked numbers are primes.
- **Why Stop at √n?**
    - If a number x is composite, it must have at least one factor ≤ √n.
      Once all factors ≤ √n are removed, no larger composite can remain.

Eratosthenes

Etching of an ancient seal identified as Eratosthenes. Philipp Daniel Lippert [de], *Dactyliothec*, 1767.

Born	276 BC[note 1] Cyrene (in modern Libya)
Died	194 BC (around age 82)[note 2] Alexandria
Occupations	Scholar Librarian Poet Inventor
Known for	Sieve of Eratosthenes Founder of Geography

Sieve of Eratosthenes



```
algorithm Sieve of Eratosthenes is
    input: an integer n > 1.
    output: all prime numbers from 2 through n.

    let A be an array of Boolean values, indexed by integers 2 to n,
    initially all set to true.

    for i = 2, 3, 4, ..., not exceeding $\sqrt{n}$ do
        if A[i] is true
            for j = $i^2$, $i^2+i$, $i^2+2i$, $i^2+3i$, ..., not exceeding n do
                set A[j] := false

    return all i such that A[i] is true.
```

Strengths

- **Complexity:**
  - Time: **O(n log log n)**
  - Space: **O(n)**
- Strength
  - Simple to implement
  - Very fast for small to mid-size n
- Weakness
  - Space complexity
- **Use Case:**
  - When generating all primes $\leq 10^7$
  - Ideal for competitive programming and educational use

Sieve of Atkin

- Instead of crossing out multiples, it uses mathematical patterns to identify potential primes based on quadratic forms:
- Any prime > 3 is of the form **6k ± 1**.
- But Sieve of Atkin refines this further with **mod 12** because:
  - All primes > 3 are in **1, 5, 7, or 11 mod 12**.
  - Mod 12 works because LCM of the first few primes (2, 3) is 6, and multiplying by 2 (for handling even spacing in quadratic forms) leads to 12.
  - This helps skip obvious composites early.

- **The Quadratic Rules**
  - It flips the primality state for numbers satisfying:
  - $n = 4x^2 + y^2 \rightarrow$ toggle if n % 12 is 1 or 5
  - $n = 3x^2 + y^2 \rightarrow$ toggle if n % 12 is 7
  - $n = 3x^2 - y^2 \rightarrow$ toggle if $x > y$ and n % 12 is 11
- **Final Step**
  - After marking, remove all perfect squares because quadratic flipping can falsely mark them.

- Let $n = 30$.
- Initialize all is_prime[i] = False for $0 \leq i \leq n$.
- For each x and y with $1 \leq x, y \leq \sqrt{n}$:
  - Compute:
    - $n = 4x^2 + y^2$, mark if n mod 12 in {1, 5}
    - $n = 3x^2 + y^2$, mark if n mod 12 == 7
    - $n = 3x^2 - y^2$ (if $x > y$), mark if n mod 12 == 11
  - If above condition met → **flip boolean status** (True ↔ False)
- Eliminate all squares:
  - For each $i \leq \sqrt{n}$:
    - If is_prime[i], mark all multiples of i^2 as False
- Include 2, 3 manually in output.

- **Complexity:**
  - Time: **O(n / log log n)**
  - Space: **O(n)**
- Strength
  - Slightly faster than Eratosthenes for large n
  - Fewer operations on composites
- Weakness:
  - Complex to implement
- **Use Case:**
  - Efficient when n > 10^7 or 10^8
  - Useful in mathematical research or performance-sensitive scenarios

Segmented Sieve

- Generates primes in a **range [L, R]** without storing all numbers up to R.
- **Steps**
  - Generate all primes up to √R using a normal Sieve of Eratosthenes.
  - For each prime p in that list, mark multiples of p in the range [L, R] as composite.
  - Remaining unmarked numbers in [L, R] are primes.

Segmented Sieve

- **Why Only Up to √R?**
  - Any composite number in [L, R] must have a factor ≤ √R.
  - Primes larger than √R can appear in [L, R], but they **cannot** generate composites within that range (other than themselves multiplied by 1).
- **E.g.**
- If R = 120, √R ≈ 10.95 → checking primes only up to 10 works(2, 3, 5, 7).

- Let’s find primes in range [L=100, R=120].
- Compute base_primes = primes up to √120 = 10 using Eratosthenes:
  → base_primes = [2, 3, 5, 7]
- Create array is_prime_segment = [True] * (R - L + 1)
- For each p in base_primes:
  - Compute start = max(p*p, ceil(L/p)*p)
  - Mark start, start + p, …, R as False in is_prime_segment
- Remaining True values in is_prime_segment are the primes in [L, R]

- **Complexity:**
  - Time: **$O((R-L+1) \log \log R)$**
  - Space: **$O(R-L+1)$**
- Strength
  - Memory-efficient for large ranges
  - Works well when R is huge (e.g. 10^{12})
- Weakness
  - Needs primes up to $\sqrt{R}$
  - Complex to impalement
- **Use Case:**
  - Generate primes in range $[10^9, 10^9+10^6]$
  - Large-scale computations or number-theoretic simulations

Prime generation summary

Feature	Eratosthenes	Atkin (mod 12)	Segmented Sieve (√R)
Method	Cross out multiples	Mathematical toggling rules	Range sieving with base primes
Math Trick	√n limit for marking	Prime residues mod 12	Only primes ≤ √R needed
Benefit	Simple and cache-friendly	Skips many composites early	Memory efficient for large R

Why Primality Testing is Important in Applied Cryptography

- Public-key cryptography depends on large prime numbers — often hundreds or thousands of bits long.
- Security relies on:
  - Generating large random numbers
  - Quickly checking if they are prime
  - Avoiding composites that weaken encryption
- E.g. Without fast primality tests, RSA key generation would be impractically slow.
- Example: For a 2048-bit RSA key, we must test random ~2048-bit candidates until we find two large primes.

Primality testing

- Miller-Rabin, Fermat's test, square root test – Probabilistic
- AKS(Agrawal, Kayal, and Saxena) algorithm – deterministic
- Recommended Algorithm

Probabilistic Algorithms for primality testing

- A probabilistic algorithm returns either a prime or a composite based on the following rules:
  1. If the integer to be tested is actually a prime, the algorithm definitely returns a prime.
  2. If the integer to be tested is actually a composite, it returns a composite with probability $1-\varepsilon$, but it may return a prime with the probability ε.
- The probability of mistake can be improved by :
  - running the algorithm more than once with different parameters or,
  - using different methods.
- If the algorithm is run m times, the probability of error may reduce to ε^m.

Fermat’s Primality Test – Probabilistic

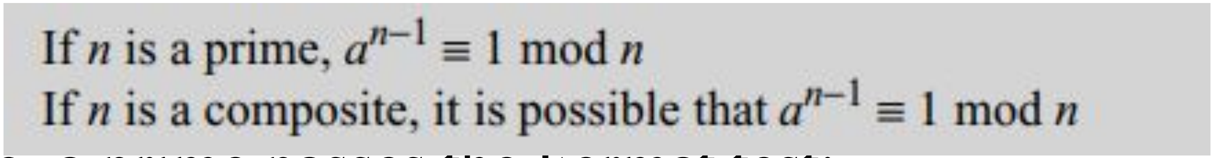
Fermat's Little Theorem:
If p is prime, and some number ‘a’ is coprime to p, then:
$a^{(p-1)} \equiv 1 \pmod p$

- **Algorithm,** for some number ‘n’:
  - Choose random a in the range [2, n-2].
  - Compute $a^{(n-1)}$ mod n.
  - If answer is not 1 → then n is composite.
  - Repeat for several a values.

Fermat Test

- Prime no property states that : If n is a prime, then

  $a^{n-1} \equiv 1 \bmod n$
- i.e. if n is a prime, the congruence holds.
- But if the congruence holds, n is not necessarily a prime.
- The integer can be a prime or composite



If n is a prime, $a^{n-1} \equiv 1 \bmod n$

If n is a composite, it is possible that $a^{n-1} \equiv 1 \bmod n$

- So, a prime passes the Fermat test;
- And a composite **<u>may</u>** pass the Fermat test with probability ε.

Reading: Fermat Test

- Since 5 is prime,
  - $2^4 \equiv 1 \pmod{5}$ [or $2^4\%5 = 1$],
  - $3^4 \equiv 1 \pmod{5}$ …. [2, n-2] range
- Since 7 is prime,
  - $2^6 \equiv 1 \pmod{7}$
  - $3^6 \equiv 1 \pmod{7}$
  - $4^6 \equiv 1 \pmod{7}$
  - $5^6 \equiv 1 \pmod{7}$…. [2, n-2] range
- Answer remains same with diff values of ‘a’

Fermat's Test

- Does the number 561 pass the Fermat test? Solution :
- Use base a= 2

$$2^{561-1} \equiv 1 \bmod 561$$

- The number passes the Fermat test, but it is not a prime, because $561 = 33 \times 17$

Fermat Test

- **Example: n=341**
- 341=11×31 (so it's **composite**).
- Pick base a=2 with gcd(2,341)=1.
- Compute: $2^{340} \bmod 341 = 1 \bmod 340$
- 2340≡1(mod341)
- so Fermat's test with base a=2 would label 341 as "probably prime,"
- Numbers like this are called **Fermat pseudoprimes** to base 2.
- Even stronger, numbers such as **561 = 3×11×17**—Carmichael numbers—pass the Fermat test for **every** base a that is coprime to n.

Square Root Test

- Idea:
- "Try dividing nnn by all numbers up to its square root. If none divide, it's prime."
  - A number n is **composite** if it has a factor other than 1 and itself.
    If $n=a\times b$, then at least one of a or b must be ≤ $\sqrt{}$**n**.
  - So, to test if n is prime, you only need to check divisibility up to $\sqrt{n}$
- Very easy to understand and implement.
- Works perfectly for small numbers.
- **Weaknesses**
  - Very slow for large numbers (e.g., testing a 200-digit number is impossible).
  - Not suitable for cryptography.

Square Root Test

- In modular arithmetic, if n is a prime, then square root is either +1 or −1.
- If n is composite, the square root is +1 or −1, but there may be other roots.
- This is known as the square root primality test.
- Note : in modular arithmetic, −1 means (n −1).
- If n is a prime, √ 1 mod n = ±1.
- If n is a composite, √ 1 mod n = ±1 and possibly other values

Square Root Test

- **Steps**
- If $n \leq 1$, it is not prime.
- if (n % 2 ==0 && n≠2), then n is composite.
- Check divisibility by odd numbers from 3 up to $\sqrt{n}$.
  - If any divides evenly → composite.
  - If none divide → prime.

- Example 1: Check if 37 is prime
  - $\sqrt{37} \approx 6.08 \rightarrow$ check divisibility only up to 6.
  - Divisors to test: 2, 3, 5.
  - $37 \% 2 \neq 0$, $37 \% 3 \neq 0$, $37 \% 5 \neq 0 \rightarrow$ Prime.
- Example 2: Check if 91 is prime
  - $\sqrt{91} \approx 9.53 \rightarrow$ check up to 9.
  - Test 2, 3, 5, 7.
  - $91 \% 7 = 0 \rightarrow$ divisible $\rightarrow$ Composite ($91 = 7 \times 13$).

Reading: Square Root Test

- Although this test can tell us if a number is composite, it is difficult to do the testing.
- Given a number n, all numbers less than n (except 1 and n −1) must be squared to be sure that none of them is 1.
- This test can be used for a number (not +1 or −1) that when squared in modulus n has the value 1. T

Square Root Test

- **Strengths**
  - Simple and exact.
  - Works for any n.
- **Weaknesses**
  - **O(√n)** time → hopeless for large numbers in cryptography.

Miller-Rabin Test

- Combines the Fermat test and the square root test to find a strong pseudoprime (a prime with a very high probability).
- Uses (n − 1) as the product of an odd number m and a power of 2:

$$n - 1 = m \times 2^k$$

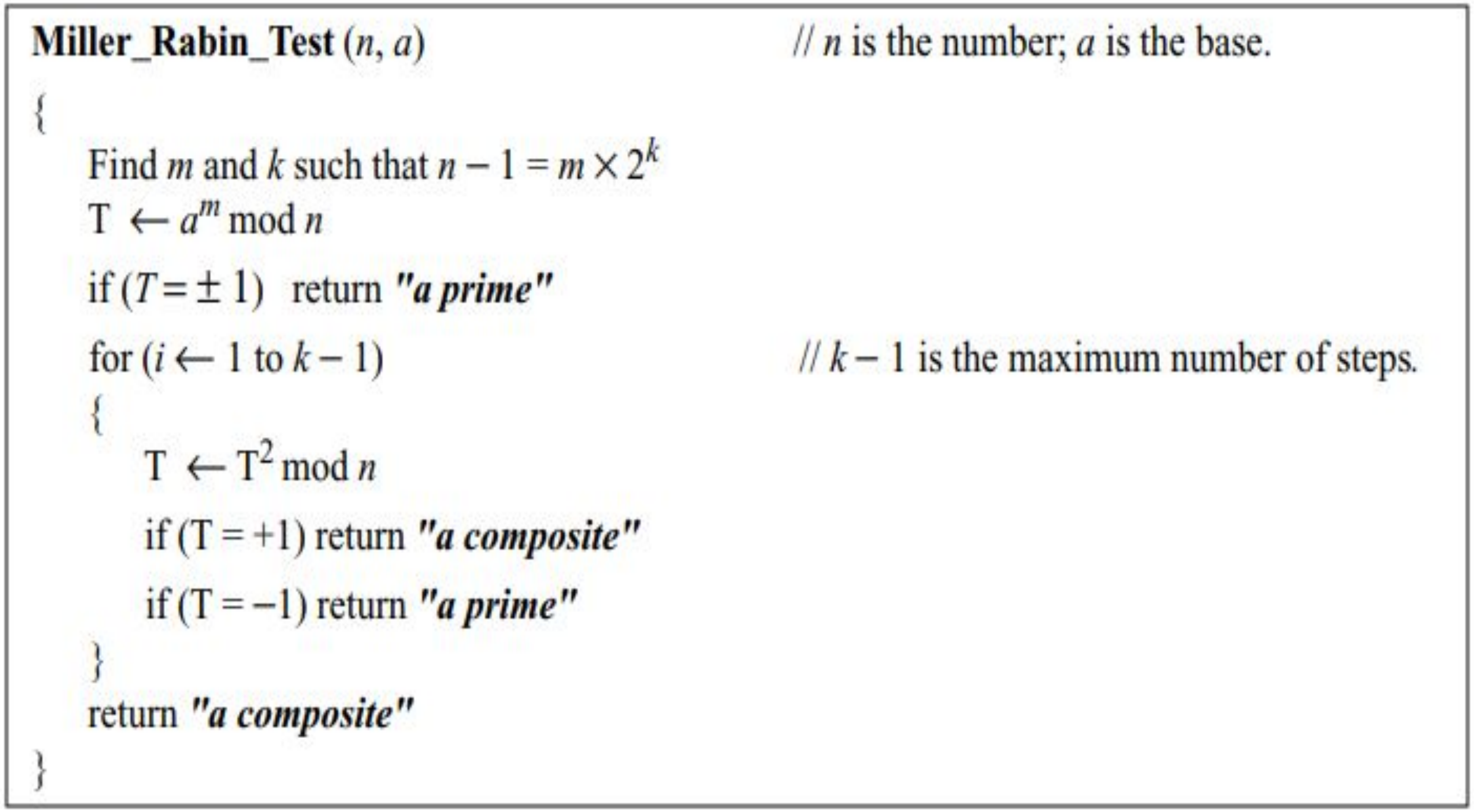
- Base ‘a’ in Fermat’s test is written as:

$$a^{n-1} = a^{m \times 2^k} = [a^m]^{2^k} = [a^m]^{\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{k\text{ times}}}$$

Reading: Miller-Rabin Test

- i.e. instead of calculating $a^{n-1} \pmod n$ in one step, it can be done in $k + 1$ steps.
- Benefit?
  - The square root test can be performed in each step.
  - If the square root test fails, stop and declare n as a composite number.
  - Each step assures that the Fermat test is passed and the square root test is satisfied between all pairs of adjacent steps, if applicable (if the result is 1).
- **The Miller-Rabin test needs from step 0 to step k – 1.**

Reading: Miller-Rabin Test



```
Miller_Rabin_Test ($n$, $a$)                      // $n$ is the number; $a$ is the base.
{
    Find $m$ and $k$ such that $n - 1 = m \times 2^k$
    T $\leftarrow a^m \bmod n$
    if ($T = \pm 1$)   return "a prime"
    for ($i \leftarrow 1$ to $k - 1$)              // $k - 1$ is the maximum number of steps.
    {
        T $\leftarrow$ T$^2 \bmod n$
        if (T = +1) return "a composite"
        if (T = −1) return "a prime"
    }
    return "a composite"
}
```

Miller-Rabbin Test

- Example : check primality of number 27 using Miller-Rabbin test

Solution:

express $n - 1$ as the product of an <u>odd number m</u> and <u>a power of 2</u> that satisfies:

$$n - 1 = m \times 2^k$$

So assume expression as: $27 - 1 = 13 \times 2^1$,

i.e. $m = 13$, $k = 1$, and $a = 2$.

As $k - 1 = 0$,

we should do only the initialization step:

$T = 2^{13} \bmod 27 = 11 \bmod 27$.

However, because the algorithm never enters the loop, it returns a composite.

Reading: Miller-Rabin test Example

Example : if the number 561 pass the Miller-Rabin test?

Solution: Assume base =2.

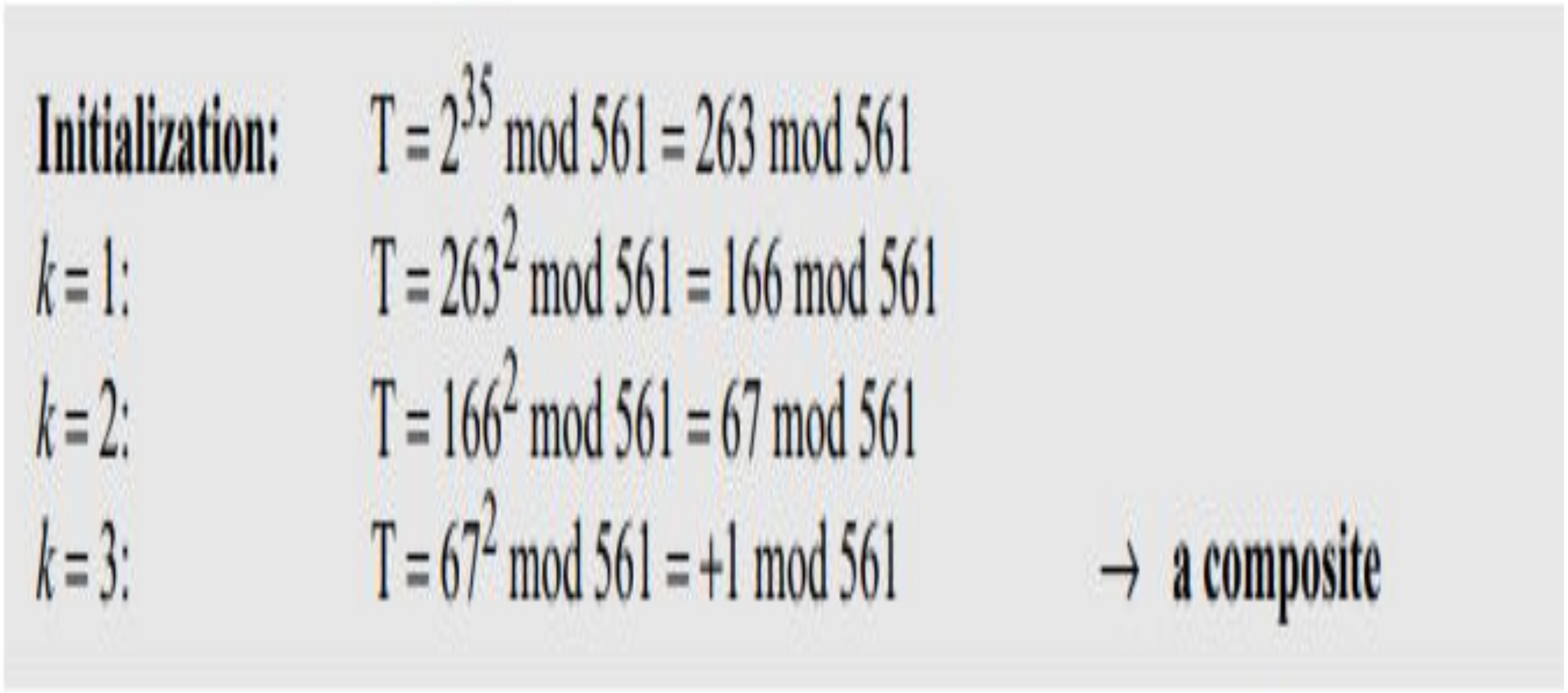
As per Miller-Rabin test , express $n - 1$ as the product of an odd number m and a power of 2 that satisfies:

$$n - 1 = m \times 2^k$$

So, let $561 - 1 = 35 \times 2^4$,

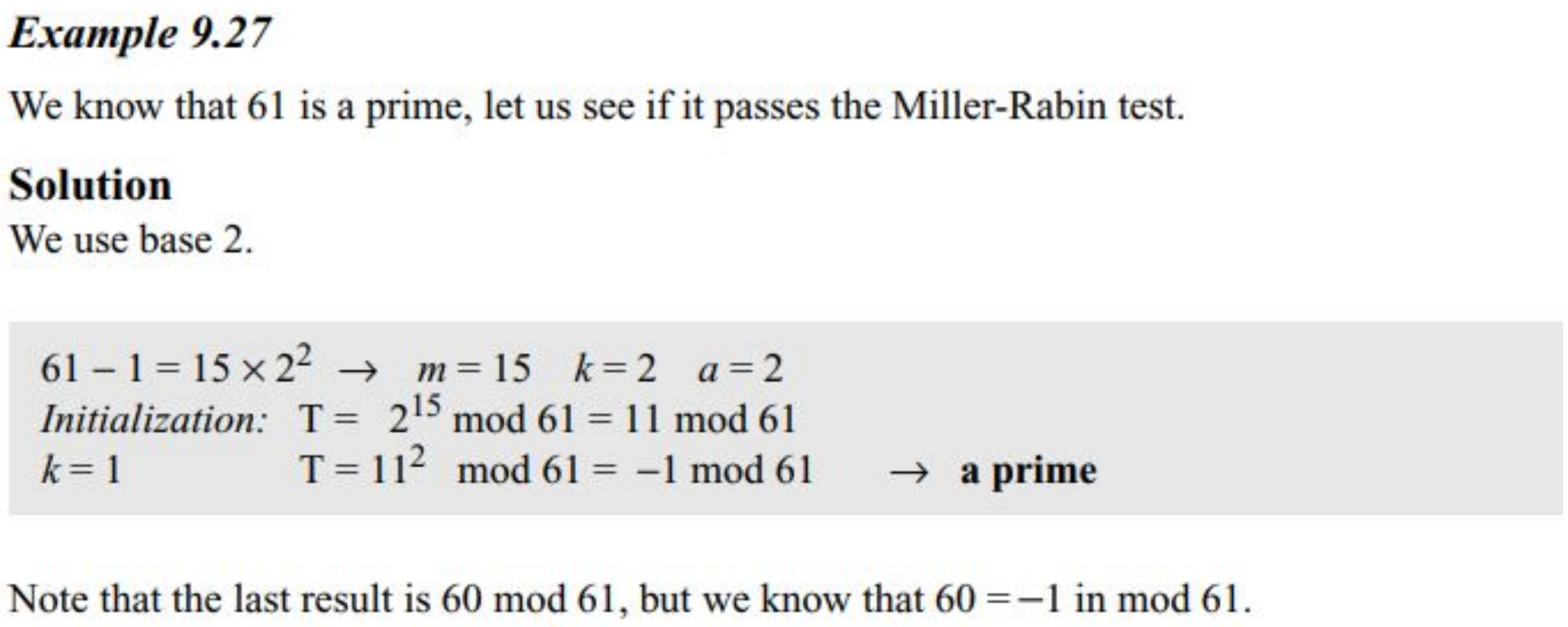
i.e $m = 35$, $k = 4$, and $a = 2$

Reading:



Initialization:	$T = 2^{35} \bmod 561 = 263 \bmod 561$
$k = 1$:	$T = 263^2 \bmod 561 = 166 \bmod 561$
$k = 2$:	$T = 166^2 \bmod 561 = 67 \bmod 561$
$k = 3$:	$T = 67^2 \bmod 561 = +1 \bmod 561$ $\rightarrow$ **a composite**

Reading:



Example 9.27

We know that 61 is a prime, let us see if it passes the Miller-Rabin test.

Solution

We use base 2.

$61 - 1 = 15 \times 2^2 \rightarrow \quad m = 15 \quad k = 2 \quad a = 2$

Initialization: $\quad T = 2^{15} \bmod 61 = 11 \bmod 61$

$k = 1 \quad T = 11^2 \bmod 61 = -1 \bmod 61 \quad \rightarrow$ **a prime**

Note that the last result is 60 mod 61, but we know that 60 = −1 in mod 61.

Rabin-Miller test

- **Strengths**
  - Very reliable in practice with a few rounds.
  - Extremely fast for large n.
- **Weaknesses**
  - Still probabilistic (error rate ~4^{-k} for k rounds).
  - Slightly more complex than Fermat.

Deterministic primality testing Algorithms

- These algorithms accept an integer and always outputs a prime or a composite.
- Until recently, all deterministic algorithms were so inefficient at finding larger primes that they were considered infeasible.
- But newer algorithm looks more promising
- Algorithms
    - Divisibility Algorithm
    - AKS(Agrawal, Kayal, and Saxena) Algorithm

AKS Algorithm

- The AKS algorithm is based on a generalization of **Fermat's Little Theorem**:
- ☞ If p is prime, then for any integer a:

$$(a+b)^p \equiv a^p+b^p \pmod p$$

- This is because all the binomial coefficients $\binom{p}{k}$ are divisible by p

AKS Algorithm

- **Mathematical Basis**
  - If n is prime $\Leftrightarrow$ $(x + a)^n \equiv x^n + a \pmod{n}$ for all a coprime to n.
- **Algorithm Outline**
  - Check if n is a perfect power.
  - Find the smallest r such that ord_r(n) > $\log^2 n$.
  - Check gcd(a, n) for $a \le r$ — if > 1 $\rightarrow$ composite.
  - For a = 1 to $\lfloor\sqrt{\varphi(r)} \log n\rfloor$:
    - Check if $(x + a)^n \equiv x^n + a \pmod{x^r - 1, n}$ fails.
  - If all pass $\rightarrow$ prime.
- E.g n = 5, $(x+1)^5 \bmod 5 = x^5 + 1$ $\rightarrow$ matches property $\rightarrow$ Prime.

How AKS Works?

1. Check if n is a perfect power
(like $n=a^b$, with $b>1$). If yes, then n composite.
2. Find a suitable r (a small number) such that the <u>order of n mod r</u> is large enough.
3. Check GCD condition for r such that,
For all $a \le r$, compute gcd(a,n)
If any nontrivial divisor found, n is composite.

4. Main Congruence Test

Verify for several a, up to $(\sqrt{\phi(r)}\log n)$:

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

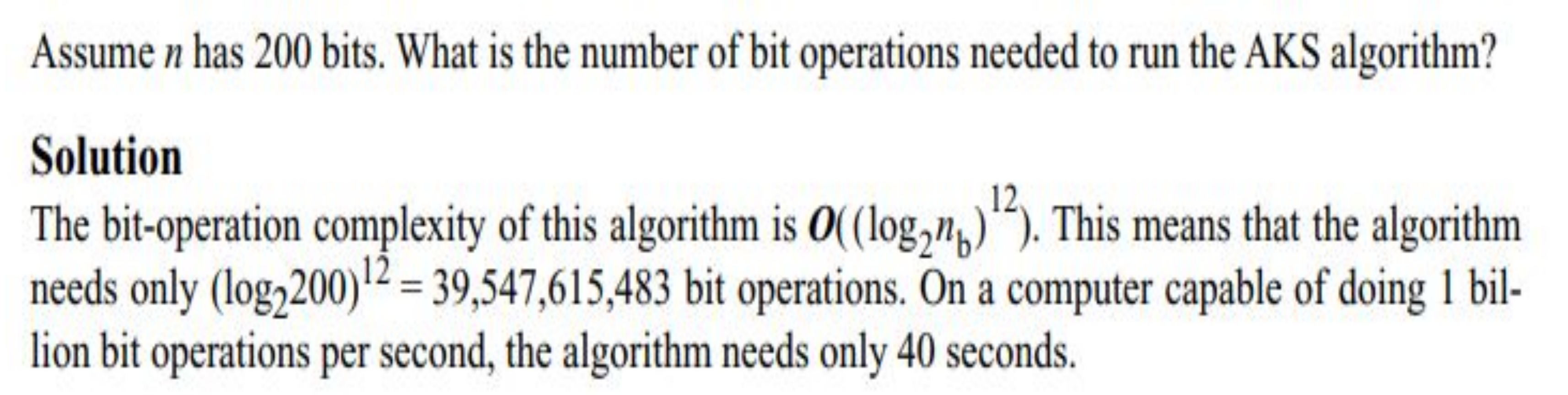
If this holds for all tested a, then n is **prime**.
If not, n is composite.

5. If passes all tests → PRIME.

Reading: AKS Deterministic primality testing Algorithm

- primality testing algorithm with polynomial bit-operation time complexity of $O((log_2 n_b)^{12})$
- algorithm uses the fact

$$(x - a)^p \equiv (x^p - a) mod\ p$$



Assume *n* has 200 bits. What is the number of bit operations needed to run the AKS algorithm?

Solution

The bit-operation complexity of this algorithm is $\boldsymbol{O}((\log_2 n_b)^{12})$. This means that the algorithm needs only $(\log_2 200)^{12} = 39{,}547{,}615{,}483$ bit operations. On a computer capable of doing 1 billion bit operations per second, the algorithm needs only 40 seconds.

AKS Algorithm

- **Strengths**
  - Deterministic and polynomial time.
  - First breakthrough in showing primality testing in P-time.
- **Weaknesses**
  - Very slow in practice compared to Miller–Rabin.
  - Used mainly for theoretical proofs.

Reading: Recommended Primality Test

- One of the most popular primality test is a combination of the divisibility test and the Miller-Rabin test.
- Recommended steps:

1. Choose an odd integer, because all even integers (except 2) are definitely composites.
2. Do some trivial divisibility tests on some known primes such as 3, 5, 7, 11, 13, be sure that its really a composite. If the number passes all of these tests, move to the next step. If the number fails any of these tests, go back to step 1 and choose another odd number.
3. Choose a set of bases for testing. A large set of bases is preferable.
4. Do Miller-Rabin tests on each of the bases. If any of them fails, go back to step 1 and choose another odd number. If the test passes for all bases, declare the number a strong pseudoprime.

Factorization

Fundamental Theorem of Arithmetic

- "Any positive integer greater than one can be written uniquely in the following prime factorization form where $p_1, p_2, \ldots, p_k$ are primes and $e_1, e_2, \ldots, e_k$ are positive integers"

$$n = p_1^{e1} \times p_2^{e2} \times \ldots \times p_k^{ek}$$

Applications of factorization

- Calculation of the greatest common divisor
- Calculation of the least common multiplier
- Cryptographic and cryptanalytics solutions in cryptology

Greatest Common Divisor

- Fact 1: gcd (a, 0) = a
- Fact 2: gcd (a, b) = gcd (b, r), where r is the remainder of dividing a by b
- Fact 3: gcd(a,b)=1 then a and b are co-prime or relatively prime

Reading: Factorization

- The Euclidean algorithm can give value of gcd(a,b)
- It Can also be computed if the factorization of a and b is known

$$a = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k} \qquad b = p_1^{b_1} \times p_2^{b_2} \times \cdots \times p_k^{b_k}$$

$$\gcd(a, b) = p_1^{\min(a_1, b_1)} \times p_2^{\min(a_2, b_2)} \times \cdots \times p_k^{\min(a_k, b_k)}$$

Reading: Least Common Multiplier

- The least common multiplier, lcm (a, b), is the smallest integer that is a multiple of both a and b.
- Using factorization, we also find lcm (a, b).

$$a = p1^{a1} \times p2^{a2} \times \ldots \times pk^{ak}$$

$$b = p1^{b1} \times p2^{b2} \times \ldots \times pk^{bk}$$

Then,

$$\mathrm{lcm}(a,b) = p1^{\max(a1,b1)} \times p2^{\max(a2,b2)} \times \ldots \times pk^{\max(ak,bk)}$$

Reading: Relation between GCD and LCM

- lcm (a, b) × gcd (a, b) = a × b

Factorization Methods

- No such perfect algorithm for factorization
- There are several algorithms that can factor a number
- None are capable of factoring a very large number in a reasonable amount of time

Factorization Methods

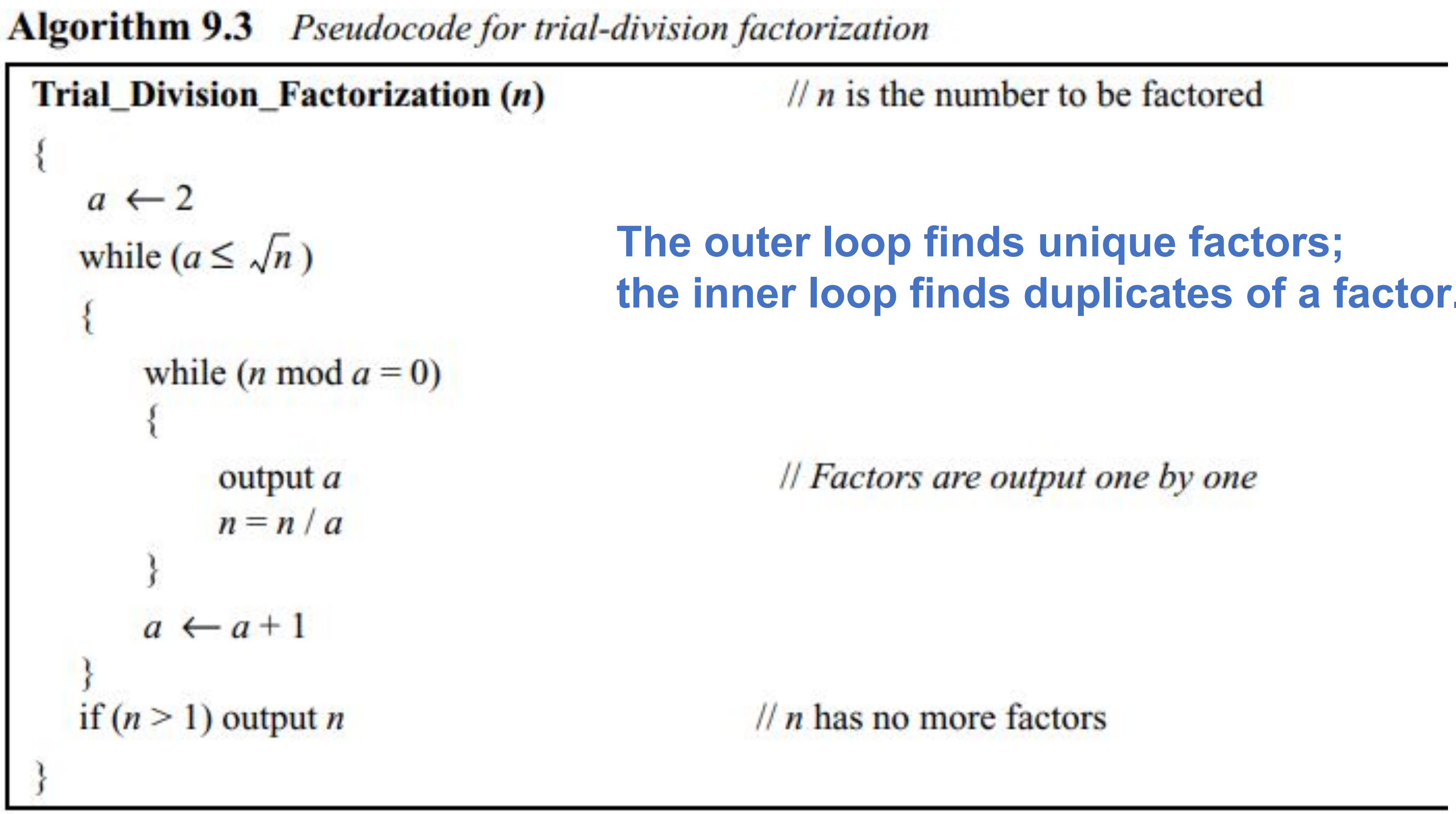
- Trial Division Method
- Fermat Method
- Pollard p – 1 Method
- Pollard rho Method

- <u>Study Fermat's method and Pollard rho method</u>

Trial Division Method

- The simplest and least efficient algorithm
- Try all the positive integers, starting with 2, to find one that divides n.
- if n is composite, then it will have a prime $p \leq \sqrt{n}$
- The outer loop finds unique factors; the inner loop finds duplicates of a factor.
- For example, $2^4 = 2^3 \times 2$. The outer loop finds the factors 2 and 3.
- The inner loop finds that 2 is a multiple factor

Trial Division algorithm



Algorithm 9.3 *Pseudocode for trial-division factorization*

```
Trial_Division_Factorization (n)                    // n is the number to be factored
{
    $a \leftarrow 2$
    while ($a \leq \sqrt{n}$)
    {
        while ($n \bmod a = 0$)
        {
            output $a$                                  // Factors are output one by one
            $n = n / a$
        }
        $a \leftarrow a + 1$
    }
    if ($n > 1$) output $n$                             // n has no more factors
}
```

The outer loop finds unique factors;
the inner loop finds duplicates of a factor.

Trial Division Method

- For example,
- N=24
- while (a ≤√24) i.e. a = 2,3,4 □ outer loop
- while (n mod a = 0) (true for 2,3,4 i.e. all numbers) □ inner loop gives all factors
- Complexity
  - The trial-division method is normally good if $n < 2^{10}$,
  - but it is very inefficient and infeasible for factoring large integers.
  - The complexity of the algorithm is exponential.

Trial Division Method

- Use the trial division algorithm to find the factors of 1233.
- N= 1233, √1233= 35
- Outer loop a= 2 to 34
- Inner loop (1233 mod 3 =0)
  - $\frac{n}{a}=\frac{1233}{3}=411$
  - (411 mod 3=0)
  - $\frac{411}{3}=137$
- i.e. $1233 = 137 * 3^2$

- Use the trial division algorithm to find the factors of 1523357784.
- Solution
- $1523357784 = 2^3 \times 3^2 \times 13 \times 37 \times 43987$

Fermat Method

- The Fermat factorization method divides a number n into two positive integers a and b (not necessarily a prime) so that $n = a \times b$.
- It is based on the fact that if we can find x and y such that $n = x^2 - y^2$, then we have

$$n = (x^2 - y^2) = a \times b$$

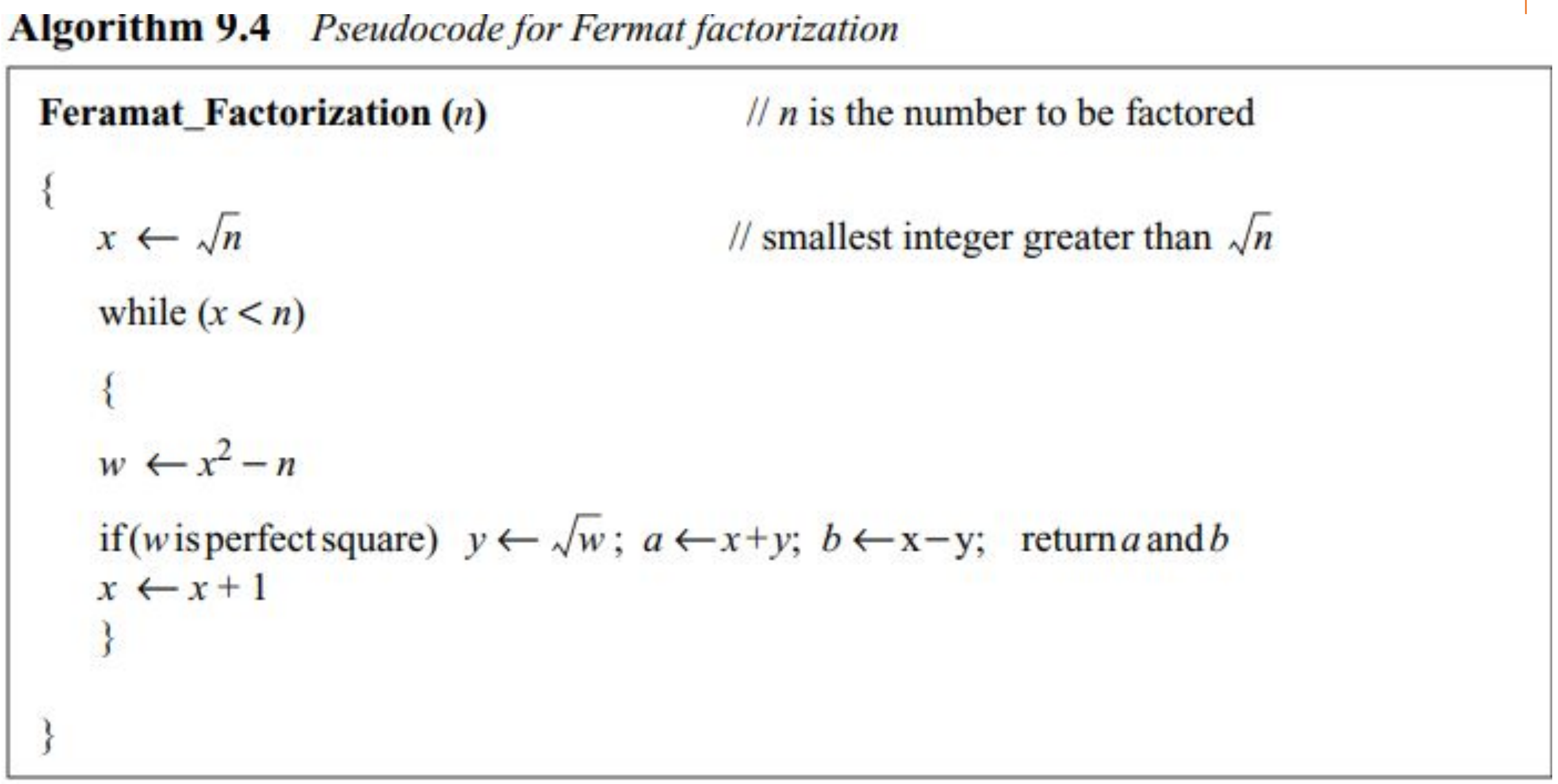
$$\text{with } a = (x + y)$$

$$\text{and } b = (x - y)$$

Fermat Method

- Finds two integers a and b close to each other ($a \approx b$).
- It starts with the smallest integer greater than $x = \sqrt{n}$ and
- Searches for another integer y that holds the relation $y^2 = x^2 - n$
- in each iteration, result of $y = x^2 - n$ has to be a perfect square.
- If such a value for y is found, calculate a and b and break from the loop.
- If such y is not found, do another iteration.

Fermat Method



Algorithm 9.4 *Pseudocode for Fermat factorization*

```
Feramat_Factorization (n)                    // n is the number to be factored
{
    x ← √n                                   // smallest integer greater than √n
    while (x < n)
    {
    w ← x^2 − n
    if (w is perfect square)  y ← √w ;  a ← x+y;  b ← x−y;   return a and b
    x ← x + 1
    }
}
```

Fermat Method

- The method does not necessarily find a prime factorization;
- the algorithm must be recursively repeated for each value a and b until the prime factors are found.
- Complexity :
    - The complexity of the Fermat method is close to subexponential

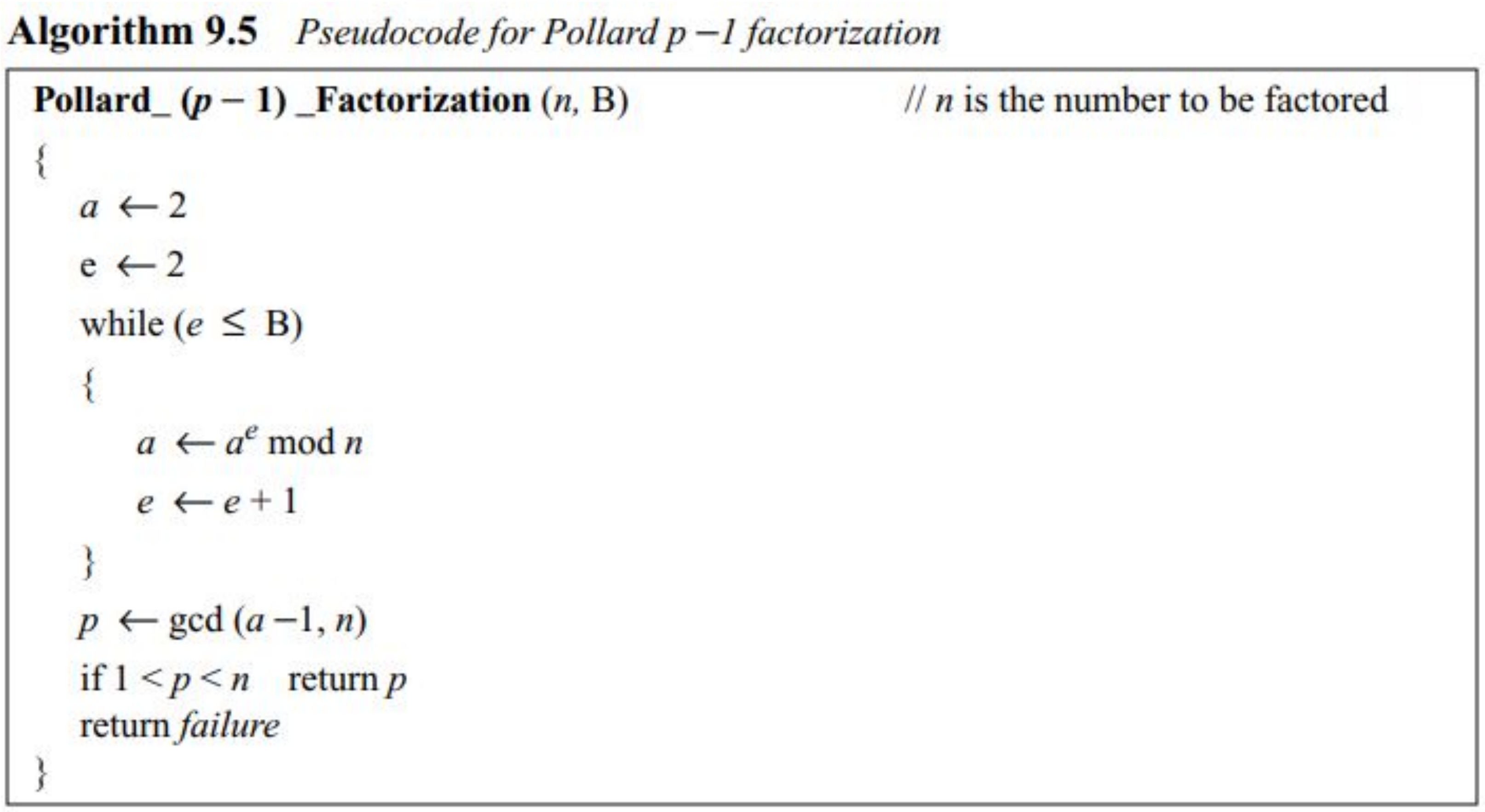
Pollard p – 1 Method

- Works on a predefined **bound** value B
- Finds a prime factor p of a number based on the condition that p−1 has no factor larger than B.

$$p = \gcd(2^{B!} - 1, n)$$

- P is a prime number
- P-1 has no factor greater than bound B

Pollard p – 1 Method



Algorithm 9.5 *Pseudocode for Pollard p −1 factorization*

```
Pollard_ (p − 1) _Factorization (n, B)          // n is the number to be factored
{
    a ← 2
    e ← 2
    while (e ≤ B)
    {
        a ← a^e mod n
        e ← e + 1
    }
    p ← gcd (a −1, n)
    if 1 < p < n    return p
    return failure
}
```

Pollard p – 1 Method

- Example: n=1233, assume b=4
- Initialization a=2, e=2
- While (e<=B=4) i.e.
    - $a=a^e$ mod n = 2^2 mod 1233= 4 ; e=3
    - $a=4^3$ mod 1233 = 64 ; e= 4
    - $A= 64^4$ mod 1233 = 1018; e =5 end loop
- P= gcd(1233,1017(p-1)) =9
- 9 is factor, hence 1233=9*137
- (p-1) has no factor is greater than 4
- i.e. $8= 2^2 \times 2$

Pollard p – 1 Method

- Use the Pollard p − 1 method to find a factor of 57247159 with the bound B = 8
- Solution: p = 421.
- So, 57247159 = 421 × 135979.
- Note that 421 is a prime and p − 1 has no factor greater than 8 ($421 - 1 = 2^2 \times 3 \times 5 \times 7$).

Pollard rho Method

Pollard rho factorization method is based on :

1. Assume that there are two integers, x1 and x2, such that p divides x1 − x2, but n does not.
2. It can be proven that p = gcd (x1 − x2, n). Because p divides x1 − x2,
   - it can be written as x1 − x2 = q × p.
   - But because n does not divide x1− x2, it is obvious that q does not divide n.
   - This means that gcd (x1 − x2, n) is either 1 or a factor of n.

Pollard rho Method

The algorithm repeatedly selects x1 and x2 until it finds an appropriate pair.

1. Choose x1, a small random integer called the seed.
2. Use a function to calculate x2 such that n does not divide x1 − x2.

   A function that may be used here is

   $x2 = f(x1) = x1^2 + a$ (a is normally = 1).

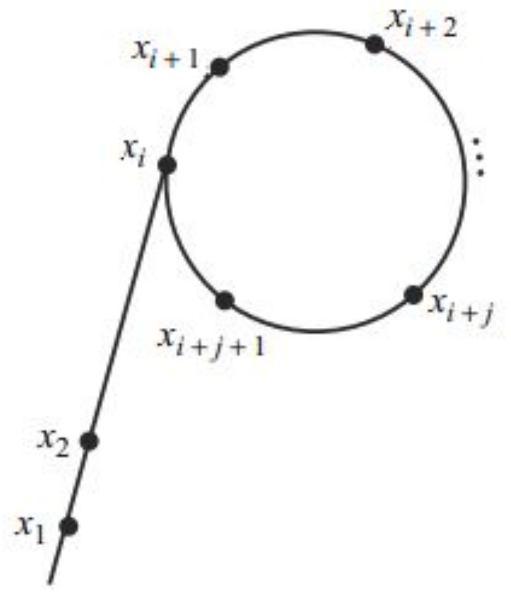
3. Calculate gcd (x1 − x2, n). If it is not 1, the result is a factor of n; stop.

   If it is 1, return to step 1 and repeat the process with x2. calculate x3 and so on.

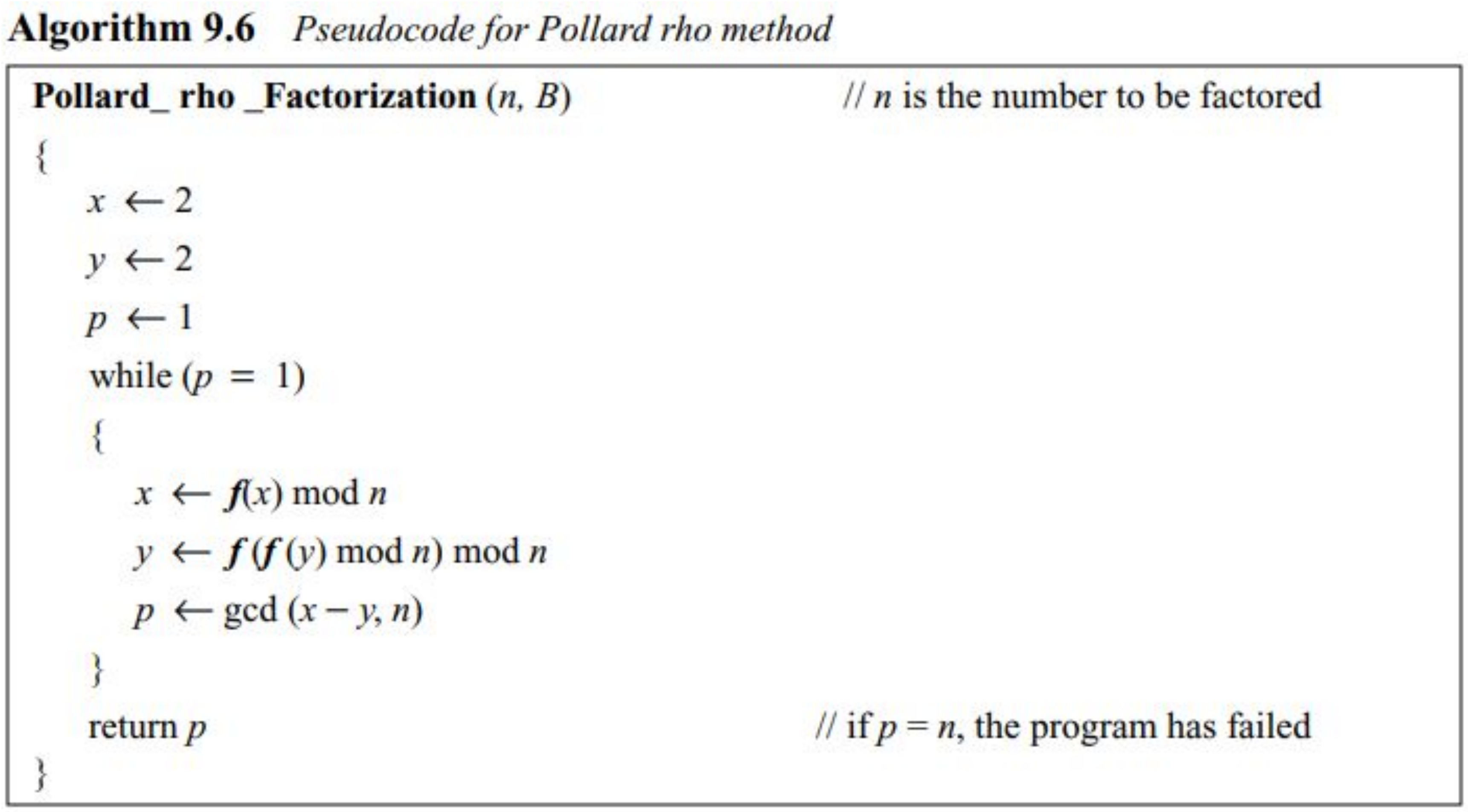
   Listing the values of x's using the Pollard rho algorithm, its obsrved that that the values are eventually repeated, creating a shape similar to the Greek letter rho (ρ)

Pollard rho Method

Figure 9.3 *Pollard rho successive numbers*



Pollard rho Method



Algorithm 9.6 *Pseudocode for Pollard rho method*

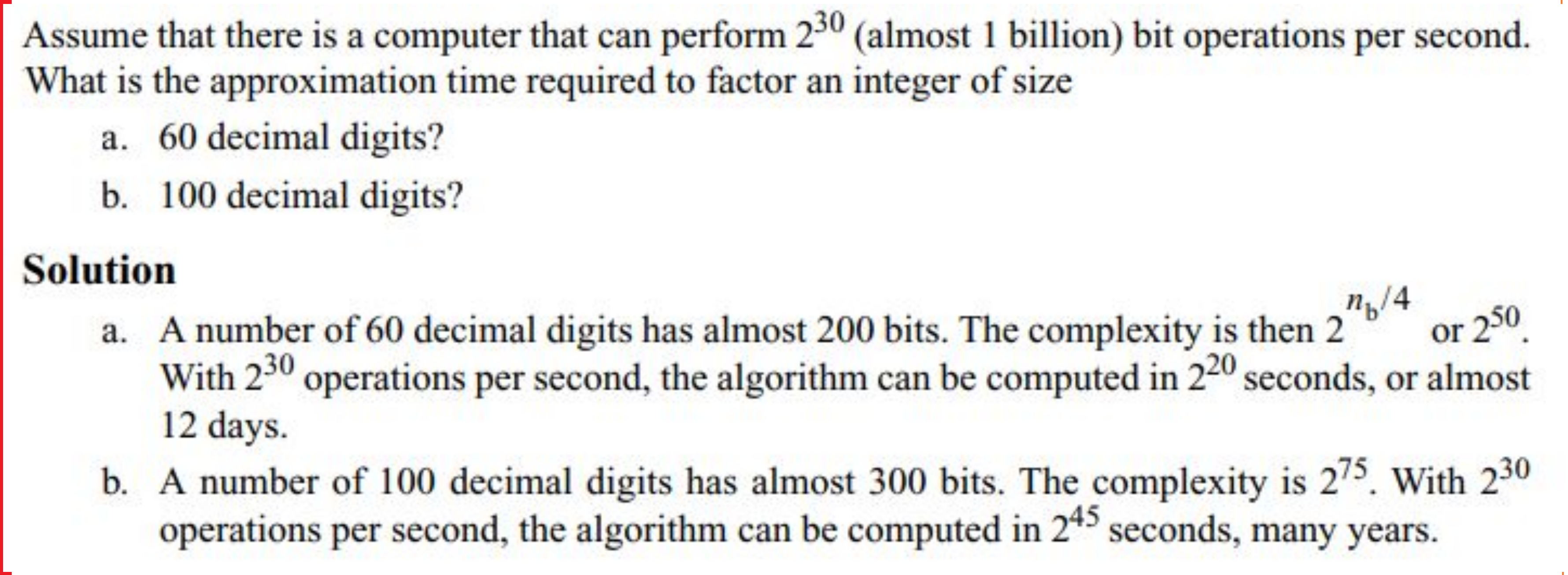
```
Pollard_ rho _Factorization ($n$, $B$)                // $n$ is the number to be factored
{
    $x \leftarrow 2$
    $y \leftarrow 2$
    $p \leftarrow 1$
    while ($p = 1$)
    {
        $x \leftarrow f(x) \bmod n$
        $y \leftarrow f(f(y) \bmod n) \bmod n$
        $p \leftarrow \gcd(x - y, n)$
    }
    return $p$                                       // if $p = n$, the program has failed
}
```

Pollard rho Method

Complexity

- The method requires $\sqrt{p}$ arithmetic operations.
- However, because p is expected to be smaller or equal to $\sqrt{n}$, expected number of arithmetic operations = $n^{1/4}$.
- This means that the bit-operation complexity is $O(2^{nb/4})$, i.e. exponential.

Pollard rho Method



Assume that there is a computer that can perform 2^{30} (almost 1 billion) bit operations per second. What is the approximation time required to factor an integer of size

a. 60 decimal digits?

b. 100 decimal digits?

Solution

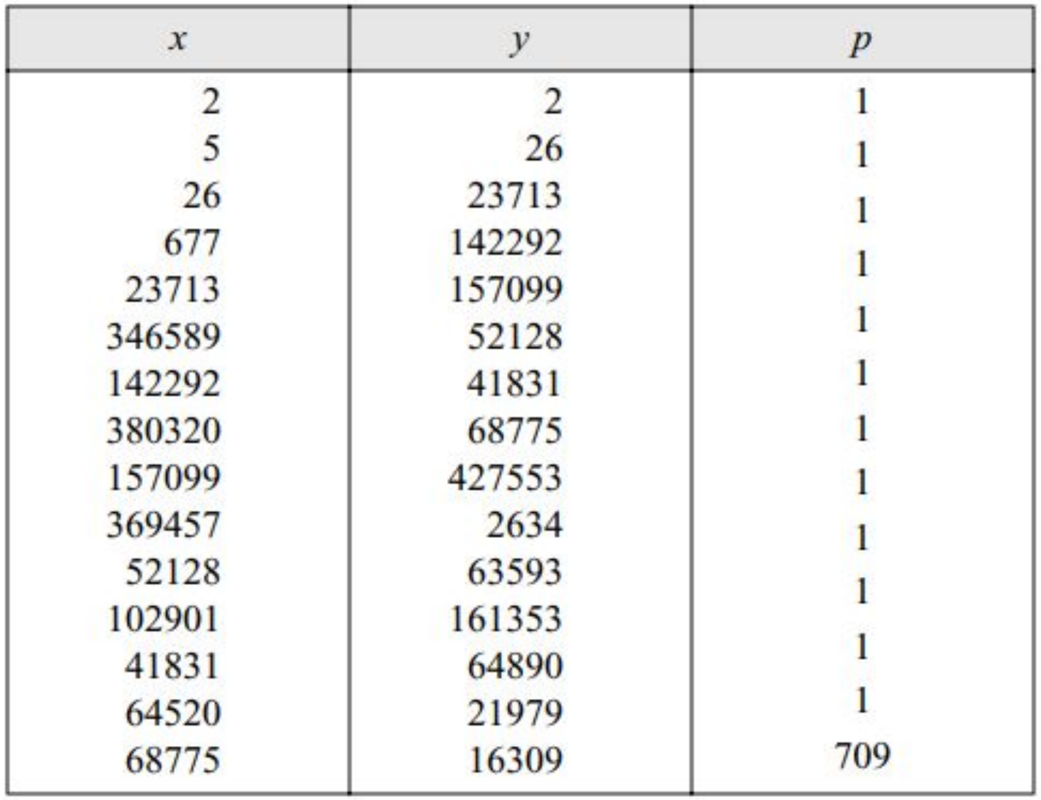
a. A number of 60 decimal digits has almost 200 bits. The complexity is then $2^{n_b/4}$ or 2^{50}. With 2^{30} operations per second, the algorithm can be computed in 2^{20} seconds, or almost 12 days.

b. A number of 100 decimal digits has almost 300 bits. The complexity is 2^{75}. With 2^{30} operations per second, the algorithm can be computed in 2^{45} seconds, many years.

Pollard rho Method

- Example: Factorize 434617 using Pollard rho method
- Ans: 709
- i.e.
- 434617= 709*613

Table 9.2 *Values of x, y, and p in Example 9.33*



x	y	p
2	2	1
5	26	1
26	23713	1
677	142292	1
23713	157099	1
346589	52128	1
142292	41831	1
380320	68775	1
157099	427553	1
369457	2634	1
52128	63593	1
102901	161353	1
41831	64890	1
64520	21979	1
68775	16309	709

Reading: Discrete Logarithm

The idea of Discrete Logarithm

- Discrete Logarithm problem is to compute x given g^x (mod p).
- The problem is **hard for a large prime p**

The idea of Discrete Logarithm

The group $G = \langle Z_n^*, \times \rangle$ has several interesting properties:

1. Its elements include all integers from 1 to $p - 1$.
2. It always has primitive roots.
3. It is cyclic.
4. The elements can be created using g^x where x is an integer from 1 to $\varphi(n) = p - 1$.
5. The primitive roots can be thought as the base of logarithm.
    1. If the group has k primitive roots, calculations can be done in k different bases.
    2. Given $x = \log_g y$ for any element y in the set, there is another element x that is the log of y in base g.
    3. This type of logarithm is called discrete logarithm.
    4. A discrete logarithm Lg to shows that the base is g (the modulus is understood).

Modular Logarithm Using Discrete Logs

- solve problems to find x where y = a^x (mod n) and y is given. □ x= $\log_a y$
- Solution : use a table for each Zp* and different bases. E.g. solve for Z_7*
- Zp* has two primitive roots= 3 and 5

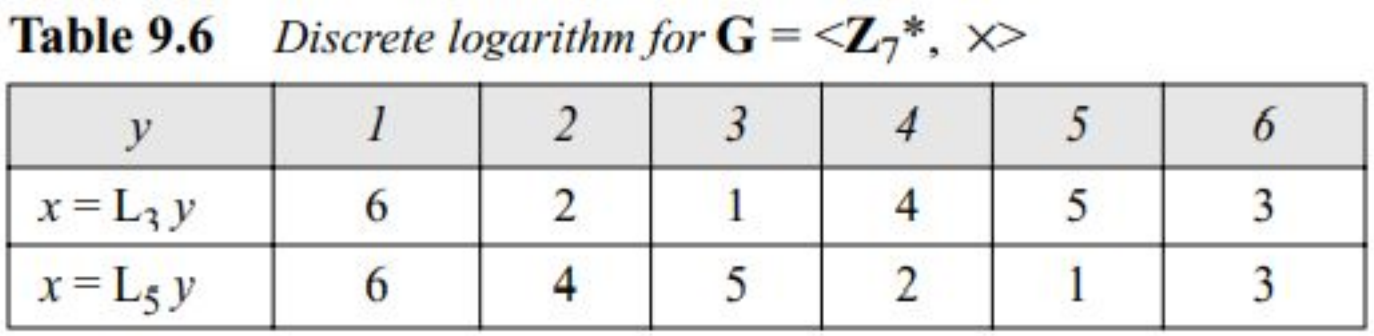


Table 9.6 *Discrete logarithm for* $\mathbf{G} = \langle \mathbf{Z}_7^*, \times\rangle$

y	1	2	3	4	5	6
$x = L_3\, y$	6	2	1	4	5	3
$x = L_5\, y$	6	4	5	2	1	3

Example: Discrete logarithms

- Find x where $6 \equiv 5^x \pmod 7$
- Solution: use the tabulation of the discrete logarithm for Z_7^*
- $6 \equiv 5^x \bmod 7 \rightarrow x = L_5 6 \bmod 7 = 3 \bmod 7$

Example

- Find x where a $4 \equiv 3^x \pmod 7$
- Solution: use the tabulation of the discrete logarithm for Z_7^*
- $4 \equiv 3^x \bmod 7 \rightarrow x = L_3 4 \bmod 7 = 4 \bmod 7$

Discrete Logarithm solution

- Let's revisit the discrete logarithm and examples at the end of slides.
- What all we need for computation of discrete logarithm solution?
  1. Finite groups,
  2. order of group,
  3. order of elements
  4. Primitive roots
  5. Cyclic groups

Discrete Logarithm solution

- Borrows concepts from Finite Multiplicative Group
- G =<Z_n*,X>
    - operation is multiplication
    - set Zn* contains integers from 1 to n−1 that are relatively prime to n;
    - the identity element is e = 1.
    - Note that when the modulus of the group is a prime, we have G =<Z_p*,X>

Order of a finite group

- order of a finite group denoted as |G|,
- i.e the number of elements in the group G.
- As per number theory: $|G| = \varphi(n)$
- $\varphi(n)=\varphi(p)*\varphi(q)$ where p and q are prime factors.
- E.g. order of group G = $=<Z_{15}*, X>$?
- $|G| = \varphi(15) = \varphi(5)*\varphi(3) = 8$

Order of an Element

- In finite group $G = <Z_n^*, X>$
- The order of an element, ord(a), is the smallest integer i such that $a^i \equiv e \pmod{n}$.
- The identity element e is 1

- Example:
- Find the order of all elements in G = $<Z_{10}^*, X>$

$\varphi(10) = \varphi(5)*\varphi(2) = 4 = \{1,3,7,9\}$

- As per Lagrange theorem: order of an element divides the order of the group
- So, integers that divide 4 are 1, 2, and 4,
- i.e. check only these powers to find the order of the element
  - $1^1 \equiv 1 \bmod (10) \rightarrow \text{ord}(1) = 1$.
  - $3^1 \equiv 3 \bmod (10)$; $3^2 \equiv 9 \bmod (10)$; $3^4 \equiv 1 \bmod (10) \rightarrow \text{ord}(3) = 4$.
  - $7^1 \equiv 7 \bmod (10)$; $7^2 \equiv 9 \bmod (10)$; $7^4 \equiv 1 \bmod (10) \rightarrow \text{ord}(7) = 4$.
  - $9^1 \equiv 9 \bmod (10)$; $9^2 \equiv 1 \bmod (10) \rightarrow \text{ord}(9) = 2$.

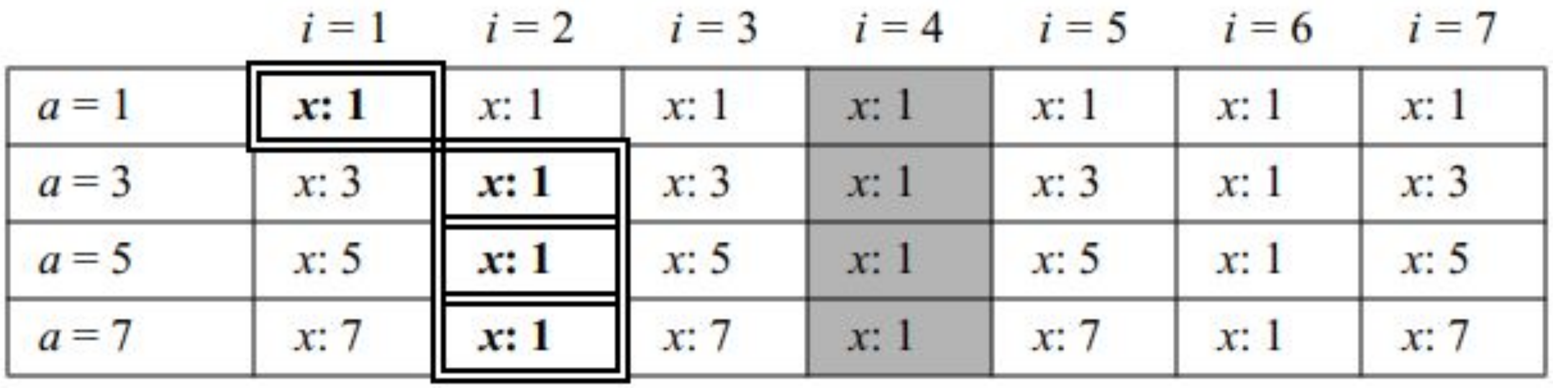
Euler's Theorem and finite groups

- As per Euler's theorem if a is the member of $G = \langle Z_n^*, X \rangle$, then $a^{\varphi(n)} = 1 \bmod n$
- i.e. relationship $a^i \equiv 1 \pmod{n}$ holds when $i = \varphi(n)$

Euler’s Theorem and finite groups

- shows the result of $a^i \equiv x \pmod 8$ for the group $G = \langle Z_n^*, \times \rangle$.
- Note that $\varphi(8) = 4$. elements $=\{1, 3, 5, 7\}$.

Table 9.4 *Finding the orders of elements in Example 9.48*



	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$
$a = 1$	**x: 1**	x: 1	x: 1	x: 1	x: 1	x: 1	x: 1
$a = 3$	x: 3	**x: 1**	x: 3	x: 1	x: 3	x: 1	x: 3
$a = 5$	x: 5	**x: 1**	x: 5	x: 1	x: 5	x: 1	x: 5
$a = 7$	x: 7	**x: 1**	x: 7	x: 1	x: 7	x: 1	x: 7

Euler’s Theorem and finite groups

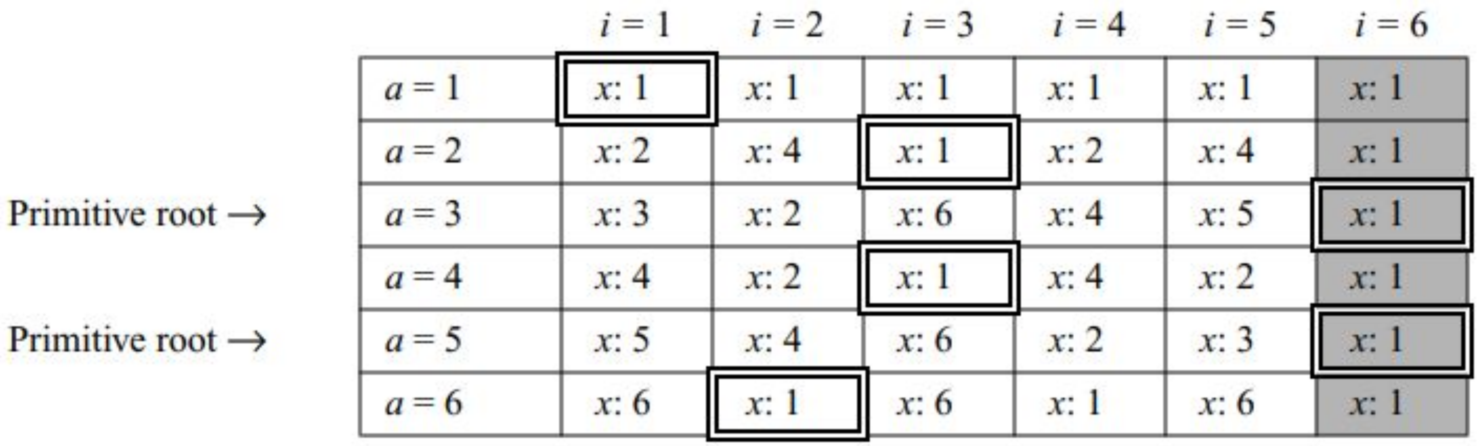
- From the table:
  - When i = φ(8) = 4, the result is x = 1 for every a.
  - Value of x can be 1 for more values of I
  - Solutions:
    - ord(1) = 1,
    - ord(3) = 2,
    - ord(5) = 2, and
    - ord(7) = 2.

Primitive Roots

- In the group $G = <Z_n^*, X>$, when the order of an element is the same as $\varphi(n)$, that element is called the primitive root of the group.
- E.g. $G = <Z_8^*, X>$
  - Z_8^* elements={1,3,5,7}
  - ord(1) = 1, ord(3) = 2, ord(5) = 2, and ord(7) = 2
  - i.e. no element has the order equal to $\varphi(8) = 4$.
  - The order of elements are all smaller than 4.
  - Thus, G Has no primitive roots

Primitive Roots

- E.g. G =<Z_7*,X>
  - Z7* elements={1,2,3,4,5,6}
  - φ(7) = 6



		i = 1	i = 2	i = 3	i = 4	i = 5	i = 6
	a = 1	$\boxed{x: 1}$	x: 1	x: 1	x: 1	x: 1	x: 1
	a = 2	x: 2	x: 4	$\boxed{x: 1}$	x: 2	x: 4	x: 1
Primitive root →	a = 3	x: 3	x: 2	x: 6	x: 4	x: 5	$\boxed{x: 1}$
	a = 4	x: 4	x: 2	$\boxed{x: 1}$	x: 4	x: 2	x: 1
Primitive root →	a = 5	x: 5	x: 4	x: 6	x: 2	x: 3	$\boxed{x: 1}$
	a = 6	x: 6	$\boxed{x: 1}$	x: 6	x: 1	x: 6	x: 1

ord(1) = 1, ord(2) = 3, **ord(3) = 6**, ord(4) = 3, **ord(5) = 6**, and ord(6) = 1.
So, group has only two primitive roots: 3 and 5 as ord(3)=ord(5)= φ(7)

Primitive Roots

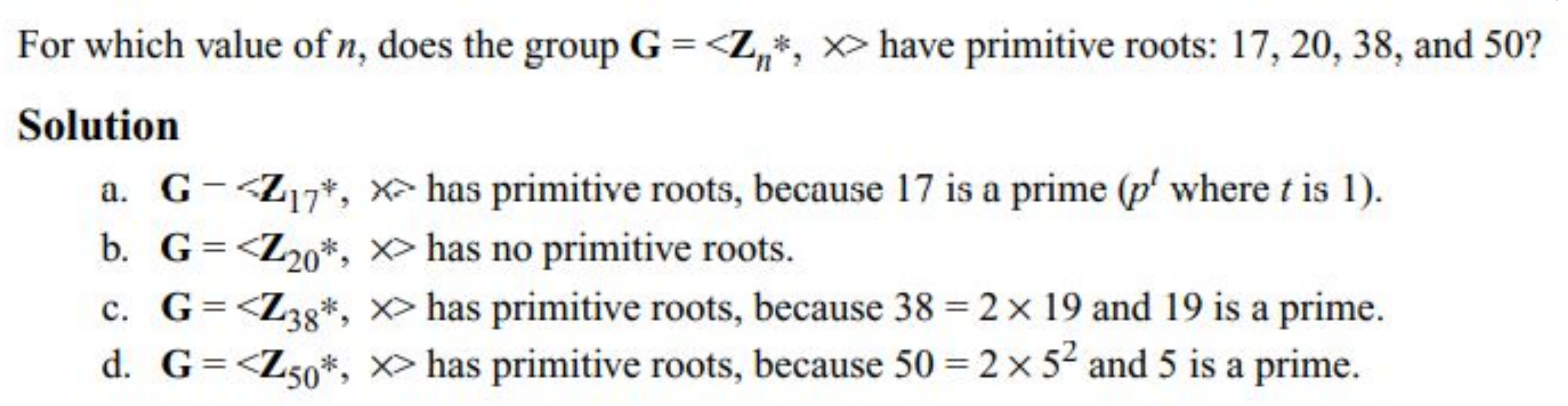
- The group $G = \langle Z_n^*, X \rangle$ has primitive roots only if n is 2, 4, p^t, or $2p^t$.
- If a group has a primitive root, then it normally has several of them.
- The number of primitive roots are calculated as $\varphi(\varphi(n))$.
- E.g. the number of primitive roots $G = \langle Z_{17}^*, X \rangle$ are: $\varphi(\varphi(17)) = \varphi(16) = 8^{\#}$

#cant be divided in prime factors.

$Z_{16}^* = \{1,3,5,7,9,11,13,15\}$

Primitive Roots

- Given: The group $G = <Z_n^*, X>$ has primitive roots only if n is 2, 4, p^t, or $2p^t$.



For which value of n, does the group $\mathbf{G} = <\mathbf{Z}_n^*, \times>$ have primitive roots: 17, 20, 38, and 50?

Solution

a. $\mathbf{G} - <\mathbf{Z}_{17}^*, \times>$ has primitive roots, because 17 is a prime (p^t where t is 1).

b. $\mathbf{G} = <\mathbf{Z}_{20}^*, \times>$ has no primitive roots.

c. $\mathbf{G} = <\mathbf{Z}_{38}^*, \times>$ has primitive roots, because $38 = 2 \times 19$ and 19 is a prime.

d. $\mathbf{G} = <\mathbf{Z}_{50}^*, \times>$ has primitive roots, because $50 = 2 \times 5^2$ and 5 is a prime.

Challenges and issues with Primitive Roots

1. Given an element a and the group $G = \langle Z_n^*, X \rangle$, how can we find out whether a is a primitive root?
   a. Computation of φ(n) is as difficult as factorization of n.
   b. Checking whether ord(a) = φ(n).
2. Given a group $G = \langle Z_n^*, X \rangle$ how to check all primitive roots of G?

   This is more difficult than the first task because we need to check if **ord(a) = φ(n)** for all elements of the group.
3. Given a group $G = \langle Z_n^*, X \rangle$, how to select a primitive root of G?
   a. Cryptography needs at least one primitive root in the given group.
   b. However value of n is chosen by the user and user knows the value of φ(n).
   c. The user has to try several elements until they find the first one.

Cyclic Groups

- if the group $G = \langle Z_n^*, \times \rangle$ has primitive roots, it is cyclic.
- Each primitive root is a generator and can be used to create the whole set.
- In other words, if g is a primitive root in the group, we can generate the set Zn* as

$$Z_n^* = \{g^1, g^2, g^3, \ldots, g^{\varphi(n)}\}$$

- The group $G = \langle Z_n^*, \times \rangle$ is a cyclic group if it has primitive roots.
- The group $G = \langle Z_p^*, \times \rangle$ is always cyclic. (Prime no property)

Example: Cyclic group

- The group G =<Z_{10}*,X> has two primitive roots because φ(10) = 4 and φ(φ(10)) = 2.
- Primitive roots= 3,7.
- using each primitive root, generate a group of Z_{10}*.

Using g=3 □ g^1 mod 10 = 3 g^2 mod 10 = 9 g^3 mod 10 = 7 g^4 mod 10 = 1

using g=7□ g^1 mod 10 = 7 g^2 mod 10 = 9 g^3 mod 10 = 3 g^4 mod 10 = 1

- Cyclic group

Discrete Logarithm..... again

- Discrete Logarithm problem is to compute x given $g^x \pmod{p}$.
- The problem is **hard for a large prime p**

The idea of Discrete Logarithm

The group $G = \langle Z_n^*, \times \rangle$ has several interesting properties:

1. Its elements include all integers from 1 to $p - 1$.
2. It always has primitive roots.
3. It is cyclic.
4. The elements can be created using g^x where x is an integer from 1 to $\varphi(n) = p - 1$.
5. The primitive roots can be thought as the base of logarithm.
    1. If the group has k primitive roots, calculations can be done in k different bases.
    2. Given $x = \log_g y$ for any element y in the set, there is another element x that is the log of y in base g.
    3. This type of logarithm is called discrete logarithm.
    4. A discrete logarithm Lg to shows that the base is g (the modulus is understood).

Modular Logarithm Using Discrete Logs

- solve problems to find x where y = a^x (mod n) and y is given. □ x= $\log_a$ y
- Solution : use a table for each Zp* and different bases. E.g. solve for Z_7*
- Zp* has two primitive roots= 3 and 5

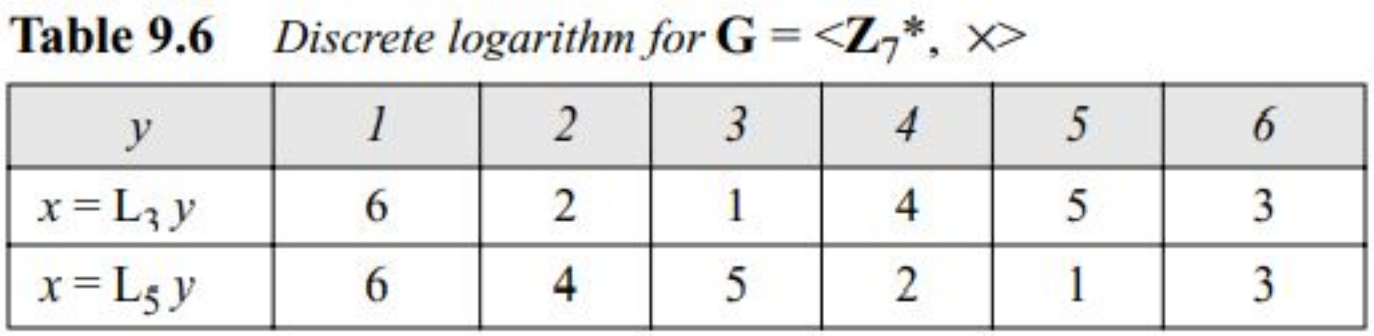


Table 9.6 *Discrete logarithm for* $\mathbf{G} = \langle \mathbf{Z}_7^*, \times\rangle$

y	*1*	*2*	*3*	*4*	*5*	*6*
$x = L_3\, y$	6	2	1	4	5	3
$x = L_5\, y$	6	4	5	2	1	3

Example: Discrete logarithms

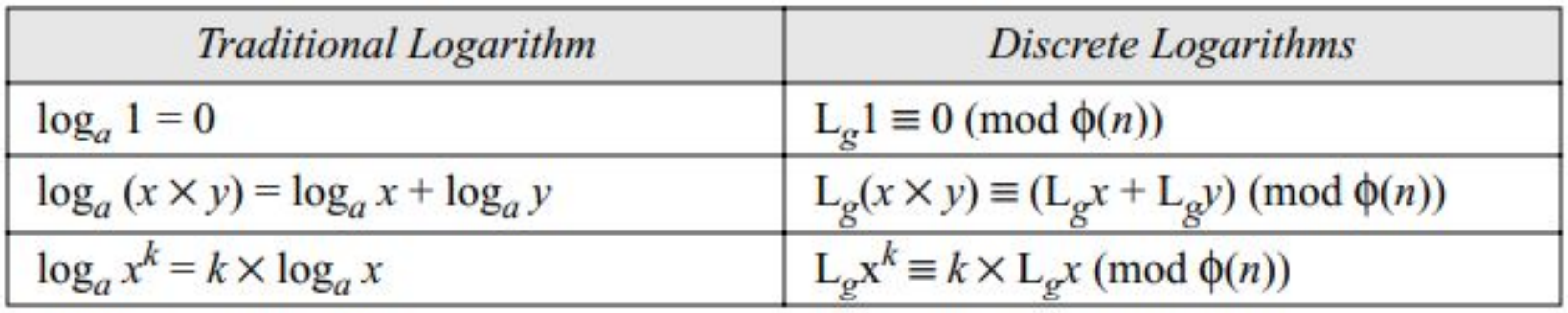
- Find x where $6 \equiv 5^x \pmod 7$
- Solution: use the tabulation of the discrete logarithm for Z_7*
- $6 \equiv 5^x \bmod 7 \rightarrow x = L_5 6 \bmod 7 = 3 \bmod 7$

- Find x where a $4 \equiv 3^x \pmod 7$
- Solution: use the tabulation of the discrete logarithm for Z_7*
- $4 \equiv 3^x \bmod 7 \rightarrow x = L_3 4 \bmod 7 = 4 \bmod 7$

Traditional Vs Discrete Logarithm

- The discrete logarithm problem has the same complexity as the factorization problem

Table 9.7 *Comparison of traditional and discrete logarithms*



Traditional Logarithm	*Discrete Logarithms*
$\log_a 1 = 0$	$L_g 1 \equiv 0 \pmod{\phi(n)}$
$\log_a (x \times y) = \log_a x + \log_a y$	$L_g(x \times y) \equiv (L_g x + L_g y) \pmod{\phi(n)}$
$\log_a x^k = k \times \log_a x$	$L_g x^k \equiv k \times L_g x \pmod{\phi(n)}$

Applications of discrete logarithms

- fundamental to a number of public-key algorithms,
  - E.g. Diffie-Hellman key exchange and the digital signature algorithm (DSA).
- Currently, the best-known algorithm for computing discrete logarithms in Z*p (for p prime) is the **general number field sieve(NFS)**

Questions?