UNIT 2 NUMBER THEORETIC ALGORITHMS

2.1 INTRODUCTION

In this unit, we will recall some basic facts from number theory that are required in Cryptography later. You may find it useful to go through Unit 6 of MMT-003 to refresh your knowledge of congruences. In Sec. 2.2 of this unit, we discuss the structure of the groups $\mathbb{Z}_n$ and $\mathbb{Z}_n^*$. We shall introduce some notation describing the running time of algorithms and discuss some basic algorithms. In Sec. 2.3, we will discuss some primality tests. In Sec. 2.4, we will discuss primitive roots.

Objectives

After studying this unit, you should be able to

- describe the structure of $\mathbb{Z}_n^*$ and $\mathbb{Z}_n$;
- describe the extended-euclidean algorithm for integers;
- apply the repeated squaring algorithm to compute powers of elements in $\mathbb{Z}_n^*$;
- explain the concept of a pseudoprime;
- explain the Rabin-Miller strong pseudoprime test; and
- outline the AKS algorithm for primality testing.

2.2 STRUCTURE OF $\mathbb{Z}_n$ and $\mathbb{Z}_n^*$

In this section, we will recall some basic results on congruences from Unit 6 of MMT-003. Recall that, $\mathbb{Z}$ is a principal ideal domain(PID). Any ideal in $\mathbb{Z}$ is of form (n) for some integer n and we can choose n to be non-negative. For each n, we have a canonical ring homomorphism

$$\psi\colon \mathbb{Z} \longrightarrow \mathbb{Z}_n \text{ given by } \psi(a) = a + (n). \tag{1}$$

We use the notation $\bar{a}$ to denote $a + (n)$, the image of $\psi(a)$.

If $a, b \in \mathbb{Z}$, a is congruent to b modulo n if $n \mid (a - b)$. We write $a \equiv b \bmod n$ in this case. We have $a \equiv b \pmod{n}$ if and only if $\bar{a} = \bar{b}$. Thus, map ψ gives us a method of translating an assertion regarding congruences into the results about the ring $\mathbb{Z}_n$. We will frequently use this to move back and forth between results regarding congruences and results regarding the ring $\mathbb{Z}_n$.

Definition 1: We say that $\{a_1, a_2, \ldots, a_n\}$, where $a_i \in \mathbb{Z}$, is a **complete set of residues modulo n** if $a_i \not\equiv a_j \pmod{n}$ for $i \neq j$.

One natural set of complete residues modulo n is $\{0, 1, 2, \ldots, n-1\}$. We discussed the extended euclidean algorithm in Unit 6 of MMT-003. We have also seen some examples there. Also, we have discussed the extended euclidean algorithm for polynomials in Unit 1. The algorithm carries over to integers with appropriate modifications. See Algorithm 1.

Algorithm 1 Extended Euclidean Algorithm for integers.

procedure EXTENDED EUCLIDEAN ALGORITHM(m, n) ▷ Returns d, Q and R such that $Qm + Rn = d$ where $d = (m, n)$.
  $Q_1 \leftarrow 1, R_1 \leftarrow 0, Q_2 \leftarrow 0, R_2 \leftarrow 1, T_1 \leftarrow m, T_2 \leftarrow n$.
  while $T_2 \neq 0$ **do**
    $S \leftarrow \left\lfloor \frac{T_1}{T_2} \right\rfloor$. ▷ $\lfloor x \rfloor$ is largest integer less than x.
    $Q_3 \leftarrow Q_1 - SQ_2, R_3 \leftarrow R_1 - SR_2$.
    $T_3 \leftarrow T_1 - ST_2$
    $Q_1 \leftarrow Q_2, R_1 \leftarrow R_2, T_1 \leftarrow T_2$.
    $Q_2 \leftarrow Q_3, R_2 \leftarrow R_3, T_2 \leftarrow T_3$.
  end while
  return T_1, Q_1 and R_1.
end procedure

Let us now redo Example 1 in Unit 6 of MMT-003 using Algorithm 1.

Example 1: Let us find the $(141, 93)$ and the values d, Q and R using Algorithm 1. Since we have already seen similar example for polynomials in Unit 1, we will just sketch the steps.

Initial values: $Q_1 \leftarrow 1, Q_2 \leftarrow 0, R_1 \leftarrow 0, R_2 \leftarrow 1, T_1 \leftarrow 141, T_2 \leftarrow 93.$

Iteration 1 Values at the end of first iteration:

$T_1 = 93$	$Q_1 = 0$	$R_1 = 1$
$T_2 = 48$	$Q_2 = 1$	$R_2 = -1$

Iteration 2 Values at the end of second iteration:

$T_1 = 48$	$Q_1 = 1$	$R_1 = -1$
$T_2 = 45$	$Q_2 = -1$	$R_2 = 2$

Iteration 3 Values at the end of third iteration:

$T_1 = 45$	$Q_1 = -1$	$R_1 = 2$
$T_2 = 3$	$Q_2 = 2$	$R_2 = -3$

Iteration 4 Values at the end of fourth iteration:

$T_1 = 3$	$Q_1 = 2$	$R_1 = -3$
$T_2 = 0$	$Q_2 = -31$	$R_2 = 47$

Note that, $T_2 = 0$. So, there are no more iterations.

Answer: $d = 3$ $q = 2$, $r = -3$.

* * *

Try the following exercise to test your understanding of extended euclidean algorithm for integers.

E1) Find $(72, 32)$ and the values d, Q and R using extended euclidean algorithm.

One important application of euclidean algorithm is the following. Often, we have to solve the equation

$$ax \equiv b \pmod{n} \tag{2}$$

where a, b, $n \in \mathbb{Z}$. Let us see recall how we solve this equation. We have seen that Eqn. (2) has a solution if and only if $(a, n) \mid b$. Suppose this is the case. Then, we can consider the equation

$$a'x \equiv b' \pmod{n'} \tag{3}$$

where $a' = \frac{a}{d}$, $b' = \frac{b}{d}$, $n' = \frac{n}{d}$ with $(a', n) = 1$ since any solution of Eqn. (3) is also a solution to Eqn. (2). We can find x and $y \in \mathbb{Z}$ such that $a'x + n'y = 1$ using extended euclidean algorithm in polynomial time, i.e. we can solve equation Eqn. (3) in polynomial time and so Eqn. (2) also in polynomial time. See Example 2 of Unit 6 of MMT-003; in this example we have solved the equation $3x \equiv 5 \pmod{7}$.

Remark 1: Recall that, in particular, if $(a, n) = 1$, $\bar{a}$ is invertible in $\bar{a} \in \mathbb{Z}_n^*$ and we can solve the congruence $ax \equiv 1 \pmod{n}$. Conversely, if $\bar{a} \in \mathbb{Z}_n^*$, then $(a, n) = 1$.

Another important result we want to recall is the Chinese Remainder Theorem:

Theorem 1(Chinese Remainder Theorem): If $n_1, n_2, \ldots, n_k$ are pairwise relative prime integers (i.e. $(n_i, n_j) = 1$ if $i \neq j$) and $a_1, a_2, \ldots, a_k$ are any integers, there is a solution x_0 to the following simultaneous congruences:

$$\left.\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_n \pmod{n_k} \end{aligned}\right\} \tag{4}$$

If x_0 and x_0' are two solutions, then $x_0 \equiv x_0' \pmod{N}$, where $N = n_1 n_2 \cdots n_k$.

See Example 4 of Unit 6 of MMT-003 in which we have solved the simultaneous congruences $x \equiv 1 \pmod{3}$, $x \equiv 3 \pmod{5}$ and $x \equiv 6 \pmod{7}$.

The following result is a consequence of Theorem 1.

Proposition 1: Let $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$. The map given by

$$g : \mathbb{Z}_n \longrightarrow \mathbb{Z}_{p_1^{\alpha_1}} \times \mathbb{Z}_{p_2^{\alpha_2}} \cdots \mathbb{Z}_{p_k^{\alpha_k}}, m \rightsquigarrow (\phi_1(m), \phi_2(m), \ldots, \phi_k(m))$$

is an isomorphism of rings, where $\phi_i : \mathbb{Z}_n \longrightarrow \mathbb{Z}_{p_i^{\alpha_i}}$ are canonical ring homomorphisms. Also, g induces an isomorphism

$$g : \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_{p_1^{\alpha_1}}^* \times \mathbb{Z}_{p_2^{\alpha_2}}^* \cdots \mathbb{Z}_{p_k^{\alpha_k}}^*, m \rightsquigarrow (\phi_1(m), \phi_2(m), \ldots, \phi_k(m))$$

Because of Proposition 1, to understand the structure of $\mathbb{Z}_n$ and $\mathbb{Z}_n^*$, it is enough to understand the structure of $\mathbb{Z}_{p^k}$ and $\mathbb{Z}_{p^k}^*$. The additive structure of $\mathbb{Z}_{p^k}$ is simple. It is a cyclic group of order p^k. The structure of $\mathbb{Z}_{p^k}^*$ is a lot more complicated. Let C_n denote a cyclic group of order n. The important result regarding the structure of $\mathbb{Z}_{p^k}^*$ is the following:

Proposition 2: a) Let p be an odd prime and $k \geq 1$. Then, $\mathbb{Z}_{p^k}^*$ is a cyclic group.

b) If $k \leq 2$, $\mathbb{Z}_{2^k}^* \simeq C_{2^{k-1}}$. If $k \geq 3$, $\mathbb{Z}_{2^k}^*$ is a product of two cyclic groups, $\mathbb{Z}_{2^k} \simeq C_2 \times C_{2^{k-2}}$.

We now introduce some terms on time estimates that we will use later in the course.

Definition 2(The big oh notation): Let x, a be real numbers, and g a real valued function. If $g(x) > 0$ for all $x \geq a$, for a real valued function $f(x)$, we write

$$f(x) = O(g(x)) \text{ or } f = O(g)$$

(read as "f is big oh of g"), to mean that the quotient $f(x)/g(x)$ is bounded for $x \geq a$; that is, there exists a constant $M > 0$ such that

$$|f(x)| \leq Mg(x) \quad \text{for all } x \geq a.$$

Let us write $f(x) = \Omega(g(x))$ if, for any constant $c > 0$, $|f(x)| \geq cg(x)$ for infinitely many values of x. Note that, this statement is essentially a negation of the statement $f(x) = O(g(x))$.

We write $f(x) = \Theta gx)$ if there are positive constants c and d such that $dg(x) \leq f(x) \leq g(x)$ for all $x \geq a$.

There are analogous notations for functions f and g that are defined on $\mathbb{N}$. We leave it to you as an exercise to formulate this statements.

E2) Formulate the analogous big oh notation for functions defined on $\mathbb{N}$.

Let n be a large positive integer, perhaps the input for our algorithm; let γ be a real number between 0 and 1; and let $c > 0$ be a constant.

Definition 3: Let

$$L_n(\gamma; c) = O(\exp(c(\log n)^{\gamma}(\log\log n)^{1-\gamma}))$$

In particular,

$$L_n(1; c) = O(\exp(c \log n)) = O(n^c)$$

and

$$L_n(0; c) = O(\exp(c \log\log n)) = O((\log n)^c).$$

An $L(\gamma)$-algorithm is an algorithm that, when applied to the integer n has running time estimate of the form $L_n(\gamma, c)$ for some c.

In particular, a *polynomial time algorithm* is an $L(0)$-algorithm, and an *exponential time algorithm* is an $L(1)$-algorithm. By a *sub-exponential time algorithm* we mean an $L(\gamma)$-algorithm for some $\gamma < 1$.

2.2.1 Modular Exponentiation by the Repeated Squaring Method

In this section we look at an efficient method of calculating b^m mod n (that is, finding the least non negative residue) when both m and n are very large. If m and n were small, we can easily compute this by first multiplying b by itself m times, and then

reducing it modulo n. However, for large m and n, this method is not feasible, as the numbers involved would be too large. So, let us look at a quicker way of carrying out this computation.

We assume that $b < n$. In this algorithm, instead of first multiplying b repeatedly by itself and then taking the least non negative residue $\bmod n$, we immediately reduce $\bmod n$ after each multiplication, that is, at each step we replace the product by its least non negative residue. In that way we never encounter any integers greater than n^2. The algorithm is called "the repeated squaring method" because the computation involves successive squaring.

Suppose the exponent m is a power of 2, say $m = 2^k$. In this case, we can exponentiate by successively squaring.

$$b^m = b^{2^k} = \underbrace{((((\ldots(x^2)^2)^2\ldots)^2)^2)^2}_{k \text{ times}}$$

In this way we compute b^m by k squarings. For example, $b^{16} = (((x^2)^2)^2)^2$. In order to compute $b^m \mod n$, we reduce $\bmod n$ at each squaring.

If the exponent is not a power of 2, then we use its binary expansion, i.e., its expansion in base 2. The algorithm to calculate $b^m \bmod n$ is:

1. Let a denote the partial product.
2. We write m in base 2. Let $m_0, m_1, \ldots, m_{k-1}$ denote the binary digits of m, that is, $m = m_0 + 2m_1 + 4m_2 + \ldots + 2^{k-1}m_{k-1}$. Each m_i is 0 or 1.
3. If $m_0 = 1$, we change a to b, otherwise we set $a = 1$.
4. We square b and set $b_1 = b^2 \bmod n$.
5. If $m_1 = 1$, we multiply a by b_1 and reduce $\bmod n$, otherwise leave a unchanged.
6. We square b_1, and set $b_2 = {b_1}^2 \bmod n$. If $m_2 = 1$, we multiply a by b_2, otherwise we leave a unchanged. We continue in this way.
7. In the j-th step we would have computed $b_j \equiv b^{2^j} \bmod n$. If $n_j = 1$, i.e., if 2^j occurs in the binary expansion of n, then we include b_j in the product for a (if 2^j is absent from n, then we don't).
8. After the $(k-1)$-th step you will have $a \equiv b^n \bmod n$.

In practice, we do not calculate the binary expansion separately. In Algorithm 2, at the beginning of the i^{th} iteration, the variable m holds the value

$$m_k 2^{k-i+1} + m_{k-1}2^{k-i} + \cdots + m_{i-1}.$$

We check if m is odd. Note that m is odd if m_{i-1} is 1. If m is odd, we carry out step 5 in the Algorithm 2; otherwise we don't. We then replace m by $\lfloor \frac{m}{2} \rfloor$. Note that, if

$$m = m_k 2^{k-i+1} + m_{k-1}2^{k-i} + \cdots + m_{i-1},$$

then

$$\left\lfloor \frac{m}{2} \right\rfloor = m_k 2^{k-i} m_k + m_{k-1}2^{k-i-1} + \cdots + m_i$$

so that the next iteration starts with the correct value of m. Further, we replace b by b^2.

See the pseudocode version in Algorithm 2.

Algorithm 2 Repeated Squaring Algorithm for integers.

```
1:  procedure REPEATED SQUARING ALGORITHM(b,m, n)   ▷ Returns b^m (mod n).
2:      P ← 1.
3:      while m ≠ 0 do
4:          if (m is odd) then
5:              P ← P.b (mod n)
6:          end if
7:          m ← ⌊m/2⌋.                  ▷ ⌊x⌋ is the largest integer less than or equal to x.
8:          b ← b·b (mod n).
9:      end while
10:     return P.
11: end procedure
```

Remark 2: The repeated squaring algorithm is a polynomial time algorithm. It computes $b^m \pmod n$ using $O((\log m)(\log^2 n))$ operations. This is because, there are about $\log m$ multiplications and multiplying two elements in $\mathbb{Z}_n$ takes $O\left(\log^2 n\right)$ operations. Note that, the algorithm can be used for computing powers in any group. We simply replace multiplication $\bmod n$ by the corresponding group operation. In this case, the number of operations required is $O(t \log m)$ where t is the time taken to multiply in the group G.

Let us look at an example to understand this algorithm.

Example 2: Let us find $13^{79} \pmod{97}$. We have $79 = 2^6 + 2^3 + 2^2 + 2 + 1$, so, $m_6 = 1$, $m_5 = 0$, $m_4 = 0$, $m_3 = 1$, $m_2 = 1$, $m_1 = 1$ and $m_0 = 1$.

Iteration 1

We have $m = 79$. We replace

1. P by $P \cdot b = 13$ since m is odd.
2. m = 79 by $\lfloor \frac{79}{2} \rfloor = 39$.
3. b by $b^2 = 169 \equiv 72 \bmod 97$.

So, the values at the end of iteration 1 are

$$P = 13,\ b = 72,\ m = 39$$

Iteration 2

We have $m = 39$.

1. P by $P \cdot b = 13 \cdot 72 \equiv 63 \pmod{97}$ since m is odd.
2. m = 39 by $\lfloor \frac{39}{2} \rfloor = 19$.
3. b by $b^2 = 72^2 \equiv 43 \bmod 97$.

So, the values at the end of iteration 2 are

$$P = 63,\ b = 43,\ m = 19$$

Iteration 3

We have $m = 19$.

1. P by $P \cdot b = 63 \cdot 43 \equiv 90 \pmod{97}$ since m is odd.
2. m = 19 by $\lfloor \frac{19}{2} \rfloor = 9$.
3. b by $b^2 = 43^2 \equiv 6 \bmod 97$.

So, the values at the end of iteration 3 are

$\boxed{P = 90, b = 6, m = 9.}$

Iteration 4

We have m = 9. We replace

1. P by $P \cdot b = 90 \cdot 6 = 55 \pmod{97}$ since m is odd.
2. m = 9 by $\lfloor \frac{9}{2} \rfloor = 4$.
3. b by $b^2 = 6^2 \equiv 36 \bmod 97$.

So, the values at the end of iteration 4 are

$\boxed{P = 55, b = 36, m = 4.}$

Iteration 5

We have m = 4. We do not change P since m is even. We replace

1. m = 4 by $\lfloor \frac{4}{2} \rfloor = 2$.
2. b by $b^2 = 36^2 \equiv 35 \bmod 97$.

So, the values at the end of iteration 5 are

$\boxed{P = 55, b = 35, m = 2.}$

Iteration 6

We have m = 2. We do not change P since m is even.

1. m = 2 by $\lfloor \frac{2}{2} \rfloor = 1$.
2. b by $b^2 = 35^2 \equiv 61 \bmod 97$.

So, the values at the end of iteration 6 are

$\boxed{P = 55, b = 61, m = 1.}$

Iteration 7

We have m = 1. We replace

1. P by $P \cdot b = 55 \cdot 61 = 57 \pmod{97}$ since m is odd.
2. m = 1 by $\lfloor \frac{1}{2} \rfloor = 0$.
3. b by $b^2 = 61^2 \equiv 35 \bmod 97$.

So, the values at the end of iteration 7 are

$$\boxed{P = 55,\ b = 35,\ m = 0.}$$

Since $m = 0$, there are no more iterations and $13^{79} \equiv 57 \pmod{97}$.

* * *

Try the following exercise to check your understanding of Algorithm 2.

E3) Find $5^{17} \pmod{71}$ using repeated squaring algorithm.

Recall from Unit 6 of MMT-003 the order of $\mathbb{Z}_n^*$ is $\phi(n)$, where ϕ the *Euler ϕ-function*.

Definition 4: Let $a \in \mathbb{Z}_n^*$. The order of a, denoted $o_n(a)$, is the least positive integer t such that $a^t \equiv 1 \pmod{n}$.

Example 3: The multiplicative group of residues mod 12 is $(\mathbb{Z}/12\mathbb{Z})^* = \{1+12\mathbb{Z}, 5+12\mathbb{Z}, 7+12\mathbb{Z}, 11+12\mathbb{Z}\}$, and its order is $\phi(12) = 4$.

* * *

Example 4: The multiplicative group of residues mod 21 is

$$\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}.$$

Note that $\phi(21) = \phi(3)\phi(7) = 2.6 = 12$, which is the order of the group $\mathbb{Z}_{21}^*$. The orders of the elements in $\mathbb{Z}_{21}^*$ are listed in Table 1.

* * *

Table 1: Orders of elements in $\mathbb{Z}_{21}^*$

$a \in \mathbb{Z}_{21}^*$	1	2	4	5	8	10	11	13	16	17	19	20
ord (a)	1	6	3	6	2	6	6	2	3	6	6	2

Try the following exercise to check your understanding of Example 4.

E4) Determine the order of all the elements in $(\mathbb{Z}/15\mathbb{Z})^*$.

In the next section, we will discuss some important results about prime numbers that we will need in the later units of this course.

2.3 PRIME NUMBERS

In this section we will discuss some facts about prime numbers and primality testing in particular. Our discussion is based on [7]. We begin this section by stating some important results about prime numbers.

Some facts about prime numbers:

1. An integer $p > 1$ is a prime number if and only if its only divisors are ± 1 and $\pm p$.

2. Any integer $a > 1$ can be factored in a unique way as $a = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_t^{\alpha_t}$, where $p_1 < p_2 < \ldots < p_t$ are prime numbers and where each a_i is a positive integer.

3. **The Prime Number Theorem**. Let x be a positive real number. Consider the function $\pi(x)$, which counts the number of primes $\leq x$. Thus,

$$\pi(x) = \#\{p \leq x, p \text{ a prime}\}.$$

Then, the following holds:

$$\lim_{x \mapsto \infty} \frac{\pi(x)\log(x)}{x} = 1,$$

where $\log x$ is the natural logarithm of x.

Also, if $\vartheta(x) = \sum_{p \text{ prime}, p \leq x} \log p$, then

$$\vartheta(x) = \Theta(x) \tag{5}$$

The prime number theorem tells us that the number of primes not exceeding x is approximately $x/\log x$.

E5) Determine $\pi(100)$. Compare your result with the estimate obtained by applying the prime number theorem.

2.3.1 Primality Testing

There are many situations where one wants to know if a large number n is prime. For example, in the RSA public key cryptosystem and in various cryptosystems based on the discrete log problem in finite fields, we need to find a large "random" prime. We can do this by choosing a large odd integer n_0 and then test $n_0, n_0 + 2, \ldots$ for primality until we obtain the first prime which is $\geq n_0$. A second type of use of primality testing is to determine whether an integer of a certain very special type is a prime. For example, for some large prime p we might want to know whether $2^p - 1$ is a Mersenne prime.

A *primality test* is a criterion for a number n *not* to be prime. If n "passes" a primality test, then it *may* be prime. If it passes a whole lot of primality tests, then it is very likely to be prime. On the other hand, if n fails any single primality test, then it is definitely composite.

Most of the algorithms that are used in primality testing are *probabilistic* algorithms. So, we begin by discussing probabilistic algorithms.

2.3.2 Probabilistic Algorithms

Many problems that we want to solve can be posed as a decision problem which has YES or NO as its answer. For example, we can formulate the problem of checking whether a given natural number n is a prime as a decision problem. An algorithm that solves this decision problem, returns the value YES if n is a prime and the value NO if n is not a prime.

Definition 5: A **probabilistic algorithm** is an algorithm that uses random numbers. A probabilistic algorithm for a decision problem is called a **yes-biased Monte Carlo algorithm** if the answer YES is always correct, but a NO answer may be incorrect. We say that the algorithm has error probability ε if the probability that the algorithm will answer NO when the answer is actually YES is ε. We can similarly define a **no-biased Monte Carlo algorithm** and its error probability.

A **Las Vegas algorithm** is a probabilistic algorithm for which the answer is always correct, provided it terminates. There is a small probability that a Las Vegas algorithm may not terminate.

We can define computation complexity for probabilistic algorithms also. Here the running time is the **expected time** in which the algorithm will terminate rather than the exact time. We call a probabilistic algorithm as an expected polynomial algorithm or simply polynomial time algorithm if the expected running time of the algorithm is a polynomial function of the size of the input.

Note that, a probabilistic algorithm, as we have defined it, is really not an algorithm in the usual sense. This is because we can get different outputs for the same input if the random numbers chosen are different in the two instances. But, if we make the output of a certain number of 'coin tosses', i.e a random string of zeros and ones, also as a part of the input, then it becomes a deterministic algorithm. However, we will not be very formal in our treatment of probabilistic algorithms. You may refer to Chapter 9 of the book [11] for a more formal treatment of probabilistic algorithms. This is also a good reference for the first two units of this block.

It is often necessary, in cryptography, to generate large (e.g., 80 digit) "random primes". In practice, the way this is done is to generate large random numbers, and then test them for compositeness using a polynomial-time, yes-biased Monte Carlo algorithm. These algorithms are fast, but with probability $< \frac{1}{2}$ the algorithm may answer 'NO' and claim that the number is a prime when it is not. However, by running the algorithm enough times, the error probability can be reduced below any desired threshold.

We now discuss two *probabilistic primality tests*.

2.3.3 The Pseudoprime Test

Let n be a large odd integer, and suppose that you want to determine whether or not n is prime. The simplest primality test is "trial division." We know that, if n is composite, it has prime factor $\leq \sqrt{n}$. So, we check if n has a divisor less than n. If it doesn't have one, we know for sure that n is prime. Of course, this is an extremely time-consuming way to test whether or not n is prime. There are other tests which are much quicker.

Most of the efficient primality tests that are known are similar in general form to the following one.

According to Fermat's Little Theorem, we know that, if n is a prime, then for any b such that $\gcd(b, n) = 1$ one has

$$b^{n-1} \equiv 1 \bmod n. \tag{6}$$

If n is *not* prime, it is still possible (but probably not very likely) that the relation Eqn. (6) holds.

Definition 6: If n is an odd composite number and b is an integer such that $\gcd(b, n) = 1$ and Eqn. (6) holds, then n is called a *pseudoprime to the base* b.

In other words, a "pseudoprime" is a number n that "pretends" to be prime by satisfying Eqn. (6) although it is not a prime.

Example 5: The number $n = 91$ is a pseudoprime to the base $b = 3$, because $3^{90} \equiv 1 \bmod 91$. However, 91 is *not* a pseudoprime to the base 2, because $2^{90} \equiv 64 \bmod 91$. If we had not already known that 91 is composite, the fact that $2^{90} \not\equiv 1 \bmod 91$ would tell us that it is.

* * *

Proposition 3: Let n be an odd composite integer.

1. n is a pseudoprime to the base b, where $\gcd(b,n) = 1$, if and only if the order of b in $(\mathbb{Z}/n\mathbb{Z})^*$ (that is, the least positive power of b which is $\equiv 1 \bmod n$) divides $n-1$.
2. If n is a pseudoprime to the base b_1 and b_2 (where $\gcd(b_1,n) = \gcd(b_2,n) = 1$), then n is a pseudoprime to the base b_1b_2 and also to the base $b_1b_2^{-1}$ (where b_2^{-1} is an integer which is inverse to b_2 modulo n).
3. If n fails the test Eqn. (6) for a single base $b \in (\mathbb{Z}/n\mathbb{Z})^*$, then n fails Eqn. (6) for at least half of the possible bases $b \in (\mathbb{Z}/n\mathbb{Z})^*$.

Proof: 1. We know that, in a group G, the order of any element $a \in G$ is m, then m divides any n for which $a^n = 1$.

2. We have

$$(b_1b_2)^{n-1} \equiv b_1^{n-1}b_2^{n-1} \pmod{n} \equiv 1 \pmod{n}$$

if $b_1^{n-1} \equiv 1 \pmod{n}$ and $b_2^{n-1} \equiv 1 \pmod{n}$. The proof for the case $b_1b_2^{-1}$ is similar. Note that, this condition says that set $\{b \in (\mathbb{Z}/n\mathbb{Z})^* \mid b^n \equiv 1 \pmod{n}\}$ is a group, i.e the set of all bases for which n is a pseudoprime is a subgroup of $(\mathbb{Z}/n\mathbb{Z})^*$.

3. Let $\{b_1, b_2, \ldots, b_s\}$ be the set of all bases for which n is a pseudoprime, i.e., the set of all integers $1 < b_i < n$ for which the congruence Eqn. (6) holds. Let b be a fixed base for which n is not a pseudoprime. If n were a pseudoprime for any of the bases bb_i, then by part 2 of the proposition, it would be a pseudoprime for the base $b \equiv (bb_i)b_i^{-1} \bmod n$, which is not the case. Thus, for the s distinct residues $\{bb_1, bb_2, \ldots, bb_s\}$, the integer n fails the test Eqn. (6). Hence, there are at least as many bases in $(\mathbb{Z}/n\mathbb{Z})^*$ for which n fails to be a pseudoprime as there are bases for which Eqn. (6) holds. This completes the proof.

■

It follows from Proposition 3 that unless n happens to pass the test Eqn. (6) for all possible b with $\gcd(b,n) = 1$, we have at least a 50% chance that n will fail Eqn. (6) for a randomly chosen b. That is, suppose we want to know if a large odd integer n is prime. We can do so in the following steps:

1. Choose a random integer b in the range $0 < b < n$. (It is beyond the scope of this unit to describe how to "randomly" choose an integer.)
2. Find $d = \gcd(b,n)$ using the Euclidean algorithm.
3. If $d > 1$, we know that n is not prime since we have found a nontrivial factor d of n.
4. If $d = 1$, then we raise b to the $(n-1)$-th power using the repeated squaring method of exponentiation.
5. If Eqn. (6) fails, we know that n is composite.
6. If Eqn. (6) holds, perhaps n is prime. We then try another b and go through the same process.

Algorithm 3 Pseudoprime Test.

```
1: procedure PSEUDOPRIME TEST(n) ▷ Returns YES if the number is composite, NO otherwise .
2:     Choose a random b, 0 < b < n.
3:     if (b, n) > 1 then
4:         return YES.
5:     else
6:         if b^{n-1} ≢ 1 (mod n) then
7:             return YES.
8:         else
9:             return NO.
10:        end if
11:    end if
12: end procedure
```

Based on the above discussion we have the yes-biased Monte Carlo algorithm given in Algorithm 3.

If Algorithm 3 returns YES then we can stop, since this would show that n is composite. Suppose that we run Algorithm 3 k times with k different b's and find that n is a pseudoprime for all of the k bases. By Proposition 3, the chance that n is still composite despite Algorithm 3 answering NO for k different values of b is at most 1 out of 2^k, unless n happens to have the very special property that Eqn. (6) holds for every single $b \in (\mathbb{Z}/n\mathbb{Z})^*$. If k is large, we can be sure "with a high probability" that n is prime (unless n has the property of being a pseudoprime for all bases). This method of finding prime numbers is called a *probabilistic* method. It differs from a *deterministic* method: the word "deterministic" means that the method will either reveal n to be composite or else determine with 100% certainty that n is prime.

Can it ever happen for a composite n that Eqn. (6) holds for every $b \in (\mathbb{Z}/n\mathbb{Z})^*$? In that case our probabilistic method fails to reveal the fact that n is composite (unless we are lucky and hit upon a b with $\gcd(b, n) > 1$). The answer is yes, and such a number is called a *Carmichael number*.

Definition 7: A *Carmichael number* is a composite integer n such that Eqn. (6) holds for every $b \in (\mathbb{Z}/n\mathbb{Z})^*$.

Some facts about Carmichael numbers.

Let n be an odd composite integer.

1. If n is divisible by a perfect square > 1, then n is not a Carmichael number.
2. If n is square free, then n is a Carmichael number if and only if $p - 1$ divides $n - 1$ for every prime p dividing n.
3. There exist infinitely many Carmichael numbers.

2.3.4 The Miller-Rabin Test

We saw in the preceding section that we may not get very far with the pseudoprime test because of the existence of the Carmichael numbers. We can obtain a better primality test, the Miller-Rabin test, based on the notion of a *strong pseudoprime*. It is based on the following result proved in Unit 6 of MMT-003:

Proposition 4: Let p be an odd prime and let $(a, p) = 1$. Suppose $p - 1 = t2^s$ with t odd. Then, a satisfies at least one of the following conditions:

i) $a^t \equiv 1 \pmod{p}$

ii) $a^{t2^i} \equiv -1 \pmod{p}$ for some i, $0 \leq i < s$.

Definition 8: Let n be an odd composite number and $b \in (\mathbb{Z}/n\mathbb{Z})^*$. Write $n - 1 = 2^s t$ where t is odd. If b and n satisfy the condition that either

$$b^t \equiv 1 \bmod n, \tag{7}$$

or there exists r, $0 \leq r < s$, such that

$$b^{2^r t} \equiv -1 \bmod n, \tag{8}$$

then n is called a *strong pseudoprime to the base* b.

Thus, to test whether a large odd positive integer n is prime or composite, the Miller-Rabin test goes as follows:

1. Write $n - 1 = 2^s t$, where t is odd.
2. Choose a random integer b, $0 < b < n$.
3. Compute $m = b^t \bmod n$. If we get ± 1 then we conclude that n is probably a prime.
4. Otherwise repeatedly square m mod n to get $m^2 = b^{2t} \bmod n$, $m^4 = b^{t2^2}$, ... and so on until we get -1, in which case, n is probably a prime. If we don't get -1 even after reaching $b^{t2^{s-1}}$, n is composite.

We have given the test in the form of an algorithm in Algorithm 4. This is also a yes-biased Monte Carlo algorithm.

Algorithm 4 Miller-Rabin test.

```
1:  procedure MILLER-RABIN(n, s)    ▷ Returns YES if the number is composite, NO
    otherwise. Assume that n − 1 = a2^s, a odd.
2:      Choose a random b, 0 < b < n.
3:      if (b, n) > 1 then
4:          return YES
5:      else
6:          m ← b^a
7:          if m ≡ ±1 (mod n) then
8:              return NO
9:          else
10:             for i ← 1, s − 1 do
11:                 m ← m^2
12:                 if m ≡ −1 (mod n) then
13:                     return NO
14:                 end if
15:             end for                    ▷ b^{t2^i} ≢ −1 (mod n) for any i, 0 ≤ i ≤ s − 1.
16:             return YES
17:         end if
18:     end if
19: end procedure
```

For any natural number n, if the Miller-Rabin test answers NO for k random choices of b, then the probability that n is composite is less than $1/4^k$, because of the following proposition which we shall state without a proof:

Proposition 5: If n is an odd composite integer, then n is a strong pseudoprime to the base b for at most 25% of all $0 < b < n$.

Let us now look at an example to understand Miller-Rabin test.

Example 6: Let us check whether 23297 is composite using Miller-Rabin algorithm. We have $23297 - 1 = 91 \cdot 2^8$. Let us choose $b = 2$. We have $2^{91} \equiv 22102 \not\equiv 1 \pmod{23297}$. Let us set $m = 22102$. Then, repeatedly squaring m, we get $m^2 \equiv 6908 \pmod{23937}$, $m^4 = m^{2^2} \equiv 8208 \pmod{23297}$, $m^8 = m^{2^3} \equiv 19637 \pmod{23297}$, $m^{16} = m^{2^4} \equiv 23122 \pmod{23297}$, $m^{32} = m^{2^5} \equiv 7328 \pmod{23297}$, $m^{2^6} = 23296 \equiv -1 \pmod{23297}$. So, 23297 is probably a prime.

* * *

Remark 3: From [9], we know that there is no number less than $25 \cdot 10^9$ which is a pseudoprime to all the bases 2, 3, 5, 7 and 11. In other words, we know that, if, for some n, the Algorithm 4 returns NO for $b = 2, 3, 5, 7, 11$, then n is a prime.

E6) Check whether 606893471 is composite, by using Miller-Rabin test for the bases 2, 3, 5, 7, 11.

E7) Check that 796973399 is composite.

The primality tests described above are *probabilistic*. If n survives them, then n is declared to be most probably prime. In the next section, we will discuss the first deterministic polynomial time algorithm for primality testing.

2.3.5 The Agrawal-Kayal-Saxena (AKS) Algorithm

The Agrawal-Kayal-Saxena (AKS) primality testing algorithm determines in polynomial time, whether an input number n is prime or composite, thus solving a long-standing problem concerning prime numbers. The AKS algorithm is a deterministic algorithm, i.e., it does not use random numbers. The other polynomial-time primality tests are either "probabilistic" —which means that there is a small probability of the algorithm returning a composite number as prime, or conditional —which means that the algorithm assumes some unproved hypothesis. The AKS algorithm is also significant in that the run-time and correctness proofs of the algorithm use elementary algebraic and number theoretic results, the most sophisticated being a sieve theory result due to E. Fouvry.

It was published in [1]. There are several expository articles on this algorithm available. See for example [4], [10], [16] and the book [11]. We will not discuss the complete proof. You can refer to [1] or the other expository articles we mentioned for details.

We have the following characterisation of primes.

Theorem 2: Suppose that, $n \in \mathbb{N}$, $n \geq 2$ and $a \in \mathbb{Z}$ is coprime to n. Then n is prime if and only if

$$(x + a)^p \equiv (x^p + a) \pmod{p}. \tag{9}$$

Proof: The proof is as in [1]. For $0 < i < n$, the coefficient of x^i in $((X+a)^n - (X^n + a))$ is $c(n,i)a^{n-i}$.

If n is a prime, $c(n,i) \equiv 0 \pmod{n}$ and hence all the coefficients are zero.

If n is not a prime, i.e. n is composite, let q be a prime factor n and suppose $q^k \mid n$, $q^{k+1} \nmid n$. Then q^k does not divide $c(n,i)$ and is coprime to a^{n-q} and hence the coefficient of X^q is not zero $\pmod{n}$. Thus, $((X+a)^n - (X^n + a))$ is not identically zero in $\mathbb{Z}_n$. ■

However, this characterisation is not useful for checking if n is a prime. For an input n, this would involve evaluating n coefficients in the LHS of Eqn. (9), in the worst case, which can take up to time $\Omega(n)$. Therefore, to make it feasible, we evaluate both sides of Eqn. (9) modulo a polynomial of the form $x^r - 1$; that is, we need to verify whether the following holds:

$$(x+a)^n \equiv (x^n + a) \bmod (x^r - 1) \text{ in } \mathbb{F}_p[x]. \tag{10}$$

We observe that verifying the congruence Eqn. (10) takes $O(r^2 \log^3 p)$ time by the repeated squaring method. Moreover, all primes satisfy the congruence Eqn. (10) for all values of a and r, but some composites may also pass the test.

Thus, for the algorithm to work efficiently, we need to choose r and a suitably. The algorithm consists of the following steps:

Let $n > 1$ be the input.

Step 1. Check whether n is a perfect power. This can be done by Newton's method. If n is not a perfect power, proceed to the next step.

Step 2. Find a prime $r = O(\log^6 n)$, with the property that $r - 1$ has a large prime factor $q > 4\sqrt{r}\log n$ and q divides $o_r(n)$, the order of n mod r.

Step 3. With r as obtained in Step 2, check whether the following hold for $a = 1$ to $2\sqrt{r}\log n$:

$$(x+a)^n \equiv (x^n + a) \bmod (x^r - 1) \text{ in } (\mathbb{Z}/n\mathbb{Z})[x]. \tag{11}$$

The algorithm will declare n to be **composite** if Eqn. (11) does not hold for some a; n is declared to be **prime** if Eqn. (11) hold for all a in the range specified in Step 3.

We are assured of finding such a prime r in Step 2 because of the following lemma, together with an application of the prime number theorem.

Lemma 1: Let $P(n)$ denote the greatest prime divisor of n. There exist constants $c > 0$ and n_0 such that, for all $x > n_0$,

$$\#\{p \mid p \textit{ is prime},\ p \le x \textit{ and } P(p-1) > p^{\frac{2}{3}}\} \ge c\frac{x}{\log x}.$$

It is clear that if n is prime, then the algorithm will declare it to be prime.

Let us consider the case when n is composite. Then there exists a prime factor p of n such that q divides $o_r(p)$, where q is the largest prime factor of $r - 1$. Let us see why this is true.

Suppose $n = p_1 p_2 \cdots p_k$ is the prime factorisation where p_is need not be distinct. Then, $o_r(n) \mid \operatorname{lcm}(o_r(p_1), o_r(p_2), \ldots, o_r(p_k))$. If $q \nmid o_r(p_i)$ for all i, $1 \le i \le k$, then

$q \nmid \text{lcm}(o_r(p_1), o_r(p_2), \ldots, o_r(p_k))$, so it can't divide $o_r(n)$ either since $o_r(n) \mid \text{lcm}(o_r(p_1), o_r(p_2), \ldots, o_r(p_k))$. This contradicts the choice of r in Step 2.

Let us suppose that the algorithm declares n to be a prime. To arrive at a contradiction, we consider the group G generated by $\{(x-a) | 1 \leq a \leq 2\sqrt{r}\log n\}$ in $\mathbb{F}_p[x]/(h(x))$, where $h(x)$ is an irreducible factor of $x^r - 1$. Then G is a cyclic group with order greater than $n^{2\sqrt{r}}$. Let $g(x)$ be a generator of G, whose order we denote by o_g.

Consider the set

$$I_{g(x)} = \{m \in \mathbb{Z} | g(x)^m \equiv g(x^m) \bmod (x^r - 1) \textit{ in } \mathbb{F}_p[x]\}.$$

Then $I_{g(x)}$ has the following properties:

1. $I_{g(x)}$ is closed under multiplication.
2. $m_1 \equiv m_2 \pmod{r}$ implies that $m_1 \equiv m_2 \pmod{o_g}$, whenever $m_1, m_2 \in I_{g(x)}$.

These properties allow us to construct a set

$$E = \{n^i p^j | 0 \leq i, j \leq [\sqrt{r}]\}$$

which is a subset of $I_{g(x)}$.

Now $\#E > r$, which implies that there exist two distinct elements in E which are congruent modulo r, hence are congruent modulo o_g, by a property of $I_{g(x)}$. Since $o_g > n^{2\sqrt{r}}$, the congruence reduces to an equality, from which we deduce that $n = p^k$ for some integer k, which contradicts the fact that we have already ruled out perfect powers in Step 1 of the algorithm.

Therefore, we conclude that the algorithm will declare an input n to be prime if and only if n is a prime.

E8) Find the smallest pseudoprime to the base 2.

E9) Prove that a Carmichael number has at least three distinct prime factors.

E10) Show that 65 is a strong pseudoprime to the base 8 and to the base 18, but not to the base 14, which is the product of 8 and 18 modulo 65.

We have seen in Sec. 2.2 that $(\mathbb{Z}/n\mathbb{Z})^*$ need not be cyclic. In the next section, we will discuss the case when $(\mathbb{Z}/n\mathbb{Z})^*$ is cyclic.

2.4 PRIMITIVE ROOTS

Let us first discuss the notion of a *reduced residue system*.

Definition 9: By a *reduced residue system modulo* m we mean any set of $\phi(m)$ integers, incongruent modulo m, each of which is relatively prime to m.

Example 7: The set $\{1, 5, 7, 11\}$ is a reduced residue system modulo 12.

* * *

Theorem 3: If $\{a_1, a_2, \ldots, a_{\phi(m)}\}$ is a reduced residue system modulo m and $\gcd(k, m) = 1$, then $\{ka_1, ka_2, \ldots, ka_{\phi(m)}\}$ is also a reduced residue system modulo m.

Proof: It is clear that no two of the numbers ka_i are congruent modulo m. Also, since $\gcd(a_i, m) = \gcd(k, m) = 1$, we have $\gcd(ka_i, m) = 1$, so each ka_i is relatively prime to m. ■

We have seen Theorem 4 and its generalisation Theorem 5 in Unit 6 of MMT-003.

Theorem 4: If a prime p does not divide a, then

$$a^{p-1} \equiv 1 \pmod{p}$$

Theorem 5(The Euler-Fermat Theorem): Let a and m be relatively prime integers, with $m \geq 1$. Then we have

$$a^{\phi(m)} \equiv 1 \pmod{m}. \tag{12}$$

As above, let a and m be relatively prime integers, with $m \geq 1$. Consider all the positive powers of a:

$$a, a^2, a^3, \ldots$$

We know, from Theorem 5 that $a^{\phi(m)} \equiv 1 \pmod{m}$. However, there may be an earlier power $a^f \equiv 1 \pmod{m}$. We are interested in the smallest positive f with this property.

Definition 10: The smallest positive integer f such that

$$a^f \equiv 1 \pmod{m}$$

is called the exponent of a modulo m, and is denoted by writing

$$f = \exp_m(a)$$

If $\exp_m(a) = \phi(m)$, then a is called a *primitive root* mod m.

Another way of looking at the notion of a primitive root is in the context of $\mathbb{Z}_n^*$. We follow the notation as in Definition Definition 4.

Definition 11: Let $a \in \mathbb{Z}_m^*$. If $o_m(a) = \phi(m)$, then α is said to be a *generator* or a *primitive element* of $\mathbb{Z}_m^*$, or a *primitive root* mod m.

Remark 4: If $\mathbb{Z}_m^*$ has a generator, then $\mathbb{Z}_m^*$ is *cyclic*.

We have seen that if p is a prime, then $\mathbb{Z}_p$ is a field, and $\mathbb{Z}_p^*$ is a cyclic group.

Definition 12: Let p be a prime. The integer a for which the residue class $a + p\mathbb{Z}$ generates $\mathbb{Z}_p^*$ is called a *primitive root* mod p.

Theorem 5 tells us that $\exp_m(a) \leq \phi(m)$. The next theorem shows that $\exp_m(a)$ divides $\phi(m)$.

Theorem 6: Given $m \geq 1$, $\gcd(a, m) = 1$, let $f = \exp_m(a)$. Then we have:

(a) $a^k \equiv a^h \pmod{m}$ if, and only if, $k \equiv h \pmod{f}$.

(b) $a^k \equiv 1 \pmod{m}$ if and only if $k \equiv 0 \pmod{f}$. In particular, f divides $\phi(m)$.

(c) The numbers $1, a, a^2, \ldots, a^{f-1}$ are incongruent mod m.

Proof: Parts (b) and (c) follow at once from (a), so we need to only prove (a).

If $a^k \equiv a^h \pmod{m}$, then $a^{k-h} \equiv 1 \pmod{m}$. Write

$$k - h = qf + r, \quad \text{where } 0 \leq r < f.$$

Then $1 \equiv a^{k-h} = a^{qf+r} \equiv a^r \pmod{m}$, so $r = 0$ and $k \equiv h \pmod{f}$.

Conversely, if $k \equiv h \pmod{f}$, then $k - h = qf$. So $a^{k-h} \equiv 1 \pmod{m}$ and hence $a^k \equiv a^h \pmod{m}$. ■

Theorem 7: Let $\gcd(a, m) = 1$. Then a is a primitive root $\bmod m$ if, and only if, the numbers

$$a, a^2, \ldots, a^{\phi(m)} \tag{13}$$

form a reduced residue system $\bmod m$.

Proof: If a is a primitive root, the numbers in Eqn. (13) are incongruent $\bmod m$, by Theorem 6(c). Since there are $\phi(m)$ such numbers, they form a reduced residue system $\bmod m$.

Conversely, if the numbers in Eqn. (13) form a reduced residue system, then $a^{\phi(m)} \equiv 1 \pmod{m}$, but no smaller power is congruent to 1, so a is a primitive root. ■

The importance of primitive roots is explained by Theorem 7. If m has a primitive root, then each reduced residue system modulo m can be expressed as a geometric progression. In other words, if $\mathbb{Z}_m^*$ has a generator, then all its elements can be expressed as powers of a single element. Unfortunately, not all moduli have primitive roots.

Some facts about primitive roots:

We shall state without proof some properties of primitive roots.

1. Primitive roots exist only for the following moduli: $m = 1, 2, 4, p^\alpha$, and $2p^\alpha$, where p is an odd prime and $\alpha \geq 1$.
2. If a is a generator of $\mathbb{Z}_m^*$, then $\mathbb{Z}_m^* = \{a^i \bmod m | 0 \leq i \leq \phi(m) - 1\}$.
3. Suppose that a is a generator of $\mathbb{Z}_m^*$. Then $b = a^i \bmod m$ is also a generator of $\mathbb{Z}_m^*$ if and only if $\gcd(i, \phi(m)) = 1$. It follows that if $\mathbb{Z}_m^*$ is cyclic, then the number of generators is $\phi(\phi(m))$.
4. a is a generator of $\mathbb{Z}_m^*$ if and only if $a^{\phi(m)/p} \not\equiv 1 \pmod{m}$ for each prime divisor p of $\phi(m)$.

Example 8: $\mathbb{Z}_{21}^*$ is not cyclic since it does not contain an element of order $\phi(21) = 12$ (see Table 1.); note that 21 does not satisfy the condition 1 above on the existence of primitive roots.

* * *

Try the next exercise to check your understanding of primitive roots.

E11) For $g = 2, 3, 5, 7, 11$ determine a prime number $p > g$ such that g is a primitive root mod p.

Let us now summarise the discussion in this unit.

2.5 SUMMARY

In this Unit, we have discussed the following:

1. The structure of $\mathbb{Z}_n^*$ and $\mathbb{Z}_n$;
2. The extended-euclidean algorithm for integers;
3. The repeated squaring algorithm to compute powers of elements in $\mathbb{Z}_n^*$;
4. The concept of a pseudoprime;
5. The Rabin-Miller strong pseudoprime test; and
6. An outline of the AKS algorithm for primality testing.

2.6 SOLUTIONS/ANSWERS

E1) Initial values: $\boxed{Q_1 \leftarrow 1, Q_2 \leftarrow 0, R_1 \leftarrow 0, R_2 \leftarrow 1, T_1 \leftarrow m.}$

Iteration 1 Values at the end of first iteration:

$T_1 = 32$	$Q_1 = 0$	$R_1 = 1$
$T_2 = 8$	$Q_2 = 1$	$R_2 = -2$

Iteration 2 Values at the end of second iteration:

$T_1 = 8$	$Q_1 = 1$	$R_1 = -2$
$T_2 = 0$	$Q_2 = -4$	$R_2 = 9$

We see that $T_2 = 0$. So, there are no more iterations.

Answer: $T_1 = 8, Q_1 = 1, R_1 = -2$

E2) The definition is as follows:

Definition 13: (The big oh notation). Let n, n_0 be integers, and g a real valued function defined on $\mathbb{N}$. If $g(n) > 0$ for all $n \geq n_0$, we write

$$f(n) = O(g(n)) \text{ or } f = O(g)$$

(read as "f is big oh of g"), to mean that the quotient $f(n)/g(n)$ is bounded for $n \geq n_0$; that is, there exists a constant $M > 0$ such that

$$|f(n)| \leq Mg(n) \quad \text{for all } n \geq n_0.$$

Let us write $f(n) = \Omega(g(n))$ if, for any constant $c > 0$, $|f(n)| \geq cg(n)$ for infinitely many values of n. Note that, this statement is essentially a negation of the statement $f(n) = O(g(n))$.

We write $f(n) = \Theta g(n)$ if there are positive constants c and d such that $dg(x) \leq f(x) \leq g(x)$.

E3) We just give the values of P, b and m after each iteration. We leave it to you check the details. Values at the end of iteration 1:

$$P = 5, b = 25, m = 8$$

Values at the end of iteration 2:

$$P = 5, b = 57, m = 4$$

Values at the end of iteration 3:

$$P = 5, b = 54, m = 2$$

Values at the end of iteration 4:

$$P = 5, b = 5, m = 1$$

Values at the end of iteration 5:

$$P = 25, b = 25, m = 0$$

So, $5^{21} \pmod{71} = 25$ using Algorithm 2.

E4)

$a \in \mathbb{Z}_{15}^*$	1	2	4	7	8	11	13	14
ord (a)	1	4	2	4	4	2	4	2

E5) $\pi(100) = 25$, while the prime number theorem tells us that $\pi(100) \sim 100/\log 100$, where log denotes the natural logarithm. Direct calculation gives $100/\log 100 \sim 21.7$.

E6) We have $606893471 = 2 \cdot 303446735$. So, $t = 303446735$, $s = 2$. We have $2^t \equiv 1 \pmod{606893471}$, so it is probably a prime. $3^t \equiv 1 \pmod{606893471}$, $5^t \equiv 1 \pmod{606893471}$, $7^t \equiv -1 \pmod{606893471}$ and $11^t \equiv 1 \pmod{606893471}$. So, 606893471 is a strong pseudoprime to the bases 2, 3, 5, 7, 11. By our remark, since $606893471 < 25 \cdot 10^9$, 606893471 is a prime.

E7) We have $796973399 = 2 \cdot 398486699$. Further, $m = 2^{398486699} \equiv 52785661 \pmod{796973399}$ and $m^2 \equiv 209867455 \pmod{796973399}$. So, 796973399 is composite.

E8) The smallest pseudoprime to the base 2 is 341. We have $341 = 11.31$ and $2^{340} \equiv 1 \bmod 341 = 1$.

E9) Let n be a Carmichael number. By definition, it is not a prime number, and by one of the properties stated in Sec. 2. 2, it is square-free, hence not a prime power. Therefore, n has at least two prime divisors. Let $n = pq$ with prime factors $p, q, p > q$. Another property of the Carmichael numbers states that $p - 1$ divides $n - 1$. Now $n - 1 = pq - 1 = (p-1)q + q - 1$. Therefore, $p - 1$ is a divisor of $q - 1$. This is impossible since $0 < q - 1 < p - 1$. Hence a Carmichael number has at least three distinct prime divisors.

E10) $8^2 \equiv 18^2 \equiv -1 \bmod 65$; $14^2 \equiv 1 \bmod 65$, but $14^1 \not\equiv \pm 1 \bmod 65$.

E11) $2, 3, 5, 7, 11$ are primitive roots mod $3, 5, 7, 11, 13$ respectively.