10

Probabilistic primality testing

In this chapter, we discuss some simple and efficient probabilistic algorithms for testing whether a given integer is prime.

10.1 Trial division

Suppose we are given an integer $n > 1$, and we want to determine whether n is prime or composite. The simplest algorithm to describe and to program is **trial division**. We simply divide n by 2, 3, and so on, testing if any of these numbers evenly divide n. Of course, we only need to divide by primes up to $\sqrt{n}$, since if n is composite, it must have a prime factor no greater than $\sqrt{n}$ (see Exercise 1.2). Not only does this algorithm determine whether n is prime or composite, it also produces a non-trivial factor of n in case n is composite.

Of course, the drawback of this algorithm is that it is terribly inefficient: it requires $\Theta(\sqrt{n})$ arithmetic operations, which is exponential in the bit length of n. Thus, for practical purposes, this algorithm is limited to quite small n. Suppose, for example, that n has 100 decimal digits, and that a computer can perform 1 billion divisions per second (this is much faster than any computer existing today). Then it would take on the order of 10^{33} *years* to perform $\sqrt{n}$ divisions.

In this chapter, we discuss a much faster primality test that allows 100-decimal-digit numbers to be tested for primality in less than a second. Unlike the above test, however, this test does not find a factor of n when n is composite. Moreover, the algorithm is probabilistic, and may in fact make a mistake. However, the probability that it makes a mistake can be made so small as to be irrelevant for all practical purposes. Indeed, we can easily make the probability of error as small as 2^{-100} — should one really care about an event that happens with such a miniscule probability?

10.2 The Miller–Rabin test

We describe in this section a fast (polynomial time) test for primality, known as the **Miller–Rabin test**. As discussed above, the algorithm is probabilistic, and may (with small probability) make a mistake.

We assume for the remainder of this section that the number n we are testing for primality is an odd integer greater than 1.

We recall some basic algebraic facts that will play a critical role in this section (see §7.5). Suppose $n = p_1^{e_1} \cdots p_r^{e_r}$ is the prime factorization of n (since n is odd, each p_i is odd). The Chinese remainder theorem gives us a ring isomorphism

$$\begin{aligned} \theta : \quad & \mathbb{Z}_n \to \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_r^{e_r}} \\ & [a]_n \mapsto ([a]_{p_1^{e_1}}, \ldots, [a]_{p_r^{e_r}}), \end{aligned}$$

and restricting θ to $\mathbb{Z}_n^*$ yields a group isomorphism

$$\mathbb{Z}_n^* \cong \mathbb{Z}_{p_1^{e_1}}^* \times \cdots \times \mathbb{Z}_{p_r^{e_r}}^*.$$

Moreover, Theorem 7.28 says that each $\mathbb{Z}_{p_i^{e_i}}^*$ is a cyclic group, whose order, of course, is $\varphi(p_i^{e_i}) = p_i^{e_i - 1}(p_i - 1)$, where φ is Euler's phi function.

Several probabilistic primality tests, including the Miller–Rabin test, have the following general structure. Define $\mathbb{Z}_n^+$ to be the set of non-zero elements of $\mathbb{Z}_n$; thus, $|\mathbb{Z}_n^+| = n - 1$, and if n is prime, $\mathbb{Z}_n^+ = \mathbb{Z}_n^*$. Suppose also that we define a set $L_n \subseteq \mathbb{Z}_n^+$ such that:

- there is an efficient algorithm that on input n and $\alpha \in \mathbb{Z}_n^+$, determines if $\alpha \in L_n$;
- if n is prime, then $L_n = \mathbb{Z}_n^*$;
- if n is composite, $|L_n| \leq c(n-1)$ for some constant $c < 1$.

To test n for primality, we set a "repetition parameter" k, and choose random elements $\alpha_1, \ldots, \alpha_k \in \mathbb{Z}_n^+$. If $\alpha_i \in L_n$ for all $i = 1, \ldots, k$, then we output *true*; otherwise, we output *false*.

It is easy to see that if n is prime, this algorithm always outputs *true*, and if n is composite this algorithm outputs *true* with probability at most c^k. If $c = 1/2$ and k is chosen large enough, say $k = 100$, then the probability that the output is wrong is so small that for all practical purposes, it is "just as good as zero."

We now make a first attempt at defining a suitable set L_n. Let us define

$$L_n := \{\alpha \in \mathbb{Z}_n^+ : \alpha^{n-1} = 1\}.$$

Note that $L_n \subseteq \mathbb{Z}_n^*$, since if $\alpha^{n-1} = 1$, then α has a multiplicative inverse, namely,

α^{n-2}. We can test if $\alpha \in L_n$ in time $O(\text{len}(n)^3)$, using a repeated-squaring algorithm.

Theorem 10.1. *If n is prime, then $L_n = \mathbb{Z}_n^*$. If n is composite and $L_n \subsetneq \mathbb{Z}_n^*$, then $|L_n| \leq (n-1)/2$.*

Proof. Note that L_n is the kernel of the $(n-1)$-power map on $\mathbb{Z}_n^*$, and hence is a subgroup of $\mathbb{Z}_n^*$.

If n is prime, then we know that $\mathbb{Z}_n^*$ is a group of order $n-1$. Since the order of a group element divides the order of the group, we have $\alpha^{n-1} = 1$ for all $\alpha \in \mathbb{Z}_n^*$. That is, $L_n = \mathbb{Z}_n^*$.

Suppose that n is composite and $L_n \subsetneq \mathbb{Z}_n^*$. Since the order of a subgroup divides the order of the group, we have $|\mathbb{Z}_n^*| = t|L_n|$ for some integer $t > 1$. From this, we conclude that

$$|L_n| = \frac{1}{t}|\mathbb{Z}_n^*| \leq \frac{1}{2}|\mathbb{Z}_n^*| \leq \frac{n-1}{2}. \quad \square$$

Unfortunately, there are odd composite numbers n such that $L_n = \mathbb{Z}_n^*$. Such numbers are called **Carmichael numbers**. The smallest Carmichael number is

$$561 = 3 \cdot 11 \cdot 17.$$

Carmichael numbers are extremely rare, but it is known that there are infinitely many of them, so we cannot ignore them. The following theorem puts some constraints on Carmichael numbers.

Theorem 10.2. *Every Carmichael number n is of the form $n = p_1 \cdots p_r$, where the p_i's are distinct primes, $r \geq 3$, and $(p_i - 1) \mid (n-1)$ for $i = 1, \ldots, r$.*

Proof. Let $n = p_1^{e_1} \cdots p_r^{e_r}$ be a Carmichael number. By the Chinese remainder theorem, we have an isomorphism of $\mathbb{Z}_n^*$ with the group

$$\mathbb{Z}_{p_1^{e_1}}^* \times \cdots \times \mathbb{Z}_{p_r^{e_r}}^*,$$

and we know that each group $\mathbb{Z}_{p_i^{e_i}}^*$ is cyclic of order $p_i^{e_i-1}(p_i - 1)$. Thus, the power $n-1$ kills the group $\mathbb{Z}_n^*$ if and only if it kills all the groups $\mathbb{Z}_{p_i^{e_i}}^*$, which happens if and only if $p_i^{e_i-1}(p_i - 1) \mid (n-1)$. Now, on the one hand, $n \equiv 0 \pmod{p_i}$. On the other hand, if $e_i > 1$, we would have $n \equiv 1 \pmod{p_i}$, which is clearly impossible. Thus, we must have $e_i = 1$.

It remains to show that $r \geq 3$. Suppose $r = 2$, so that $n = p_1 p_2$. We have

$$n - 1 = p_1 p_2 - 1 = (p_1 - 1)p_2 + (p_2 - 1).$$

Since $(p_1 - 1) \mid (n-1)$, we must have $(p_1 - 1) \mid (p_2 - 1)$. By a symmetric argument, $(p_2 - 1) \mid (p_1 - 1)$. Hence, $p_1 = p_2$, a contradiction. $\square$

To obtain a good primality test, we need to define a different set L'_n, which we do as follows. Let $n-1 = t2^h$, where t is odd (and $h \geq 1$ since n is assumed odd), and define

$$L'_n := \{\alpha \in \mathbb{Z}_n^+ : \ \alpha^{t2^h} = 1 \text{ and } \alpha^{t2^{j+1}} = 1 \implies \alpha^{t2^j} = \pm 1 \text{ for } j = 0, \ldots, h-1\}.$$

The Miller–Rabin test uses this set L'_n, in place of the set L_n defined above. It is clear from the definition that $L'_n \subseteq L_n$.

Testing whether a given $\alpha \in \mathbb{Z}_n^+$ belongs to L'_n can be done using the following procedure:

$\beta \leftarrow \alpha^t$
if $\beta = 1$ then return *true*
for $j \leftarrow 0$ to $h-1$ do
    if $\beta = -1$ then return *true*
    if $\beta = +1$ then return *false*
    $\beta \leftarrow \beta^2$
return *false*

It is clear that using a repeated-squaring algorithm, this procedure runs in time $O(\text{len}(n)^3)$. We leave it to the reader to verify that this procedure correctly determines membership in L'_n.

Theorem 10.3. *If n is prime, then $L'_n = \mathbb{Z}_n^*$. If n is composite, then $|L'_n| \leq (n-1)/4$.*

Proof. Let $n-1 = t2^h$, where t is odd.

Case 1: n is prime. Let $\alpha \in \mathbb{Z}_n^*$. Since $\mathbb{Z}_n^*$ is a group of order $n-1$, and the order of a group element divides the order of the group, we know that $\alpha^{t2^h} = \alpha^{n-1} = 1$. Now consider any index $j = 0, \ldots, h-1$ such that $\alpha^{t2^{j+1}} = 1$, and consider the value $\beta := \alpha^{t2^j}$. Then since $\beta^2 = \alpha^{t2^{j+1}} = 1$, the only possible choices for β are ± 1—this is because $\mathbb{Z}_n^*$ is cyclic of even order and so there are exactly two elements of $\mathbb{Z}_n^*$ whose multiplicative order divides 2, namely ± 1. So we have shown that $\alpha \in L'_n$.

Case 2: $n = p^e$, where p is prime and $e > 1$. Certainly, L'_n is contained in the kernel K of the $(n-1)$-power map on $\mathbb{Z}_n^*$. By Theorem 6.32, $|K| = \gcd(\varphi(n), n-1)$. Since $n = p^e$, we have $\varphi(n) = p^{e-1}(p-1)$, and so

$$|L'_n| \leq |K| = \gcd(p^{e-1}(p-1), p^e - 1) = p - 1 = \frac{p^e - 1}{p^{e-1} + \cdots + 1} \leq \frac{n-1}{4}.$$

Case 3: $n = p_1^{e_1} \cdots p_r^{e_r}$ is the prime factorization of n, and $r > 1$. Let

$$\theta : \mathbb{Z}_n \to \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_r^{e_r}}$$

be the ring isomorphism provided by the Chinese remainder theorem. Also, let

$\varphi(p_i^{e_i}) = t_i 2^{h_i}$, with t_i odd, for $i = 1, \ldots, r$, and let $g := \min\{h, h_1, \ldots, h_r\}$. Note that $g \geq 1$, and that each $\mathbb{Z}^*_{p_i^{e_i}}$ is a cyclic group of order $t_i 2^{h_i}$.

We first claim that for every $\alpha \in L'_n$, we have $\alpha^{t2^g} = 1$. To prove this, first note that if $g = h$, then by definition, $\alpha^{t2^g} = 1$, so suppose that $g < h$. By way of contradiction, suppose that $\alpha^{t2^g} \neq 1$, and let j be the smallest index in the range $g, \ldots, h-1$ such that $\alpha^{t2^{j+1}} = 1$. By the definition of L'_n, we must have $\alpha^{t2^j} = -1$. Since $g < h$, we must have $g = h_i$ for some particular index $i = 1, \ldots, r$. Writing $\theta(\alpha) = (\alpha_1, \ldots, \alpha_r)$, we have $\alpha_i^{t2^j} = -1$. This implies that the multiplicative order of α_i^t is equal to 2^{j+1} (see Theorem 6.37). However, since $j \geq g = h_i$, this contradicts the fact that the order of a group element (in this case, α_i^t) must divide the order of the group (in this case, $\mathbb{Z}^*_{p_i^{e_i}}$).

For $j = 0, \ldots, h$, let us define ρ_j to be the $(t2^j)$-power map on $\mathbb{Z}^*_n$. From the claim in the previous paragraph, and the definition of L'_n, it follows that each $\alpha \in L'_n$ satisfies $\alpha^{t2^{g-1}} = \pm 1$. In other words, $L'_n \subseteq \rho_{g-1}^{-1}(\{\pm 1\})$, and hence

$$|L'_n| \leq 2|\mathrm{Ker}\, \rho_{g-1}|. \tag{10.1}$$

From the group isomorphism $\mathbb{Z}^*_n \cong \mathbb{Z}^*_{p_1^{e_1}} \times \cdots \times \mathbb{Z}^*_{p_r^{e_r}}$, and Theorem 6.32, we have

$$|\mathrm{Ker}\, \rho_j| = \prod_{i=1}^{r} \gcd(t_i 2^{h_i}, t2^j) \tag{10.2}$$

for each $j = 0, \ldots, h$. Since $g \leq h$, and $g \leq h_i$ for $i = 1, \ldots, r$, it follows immediately from (10.2) that

$$2^r |\mathrm{Ker}\, \rho_{g-1}| = |\mathrm{Ker}\, \rho_g| \leq |\mathrm{Ker}\, \rho_h|. \tag{10.3}$$

Combining (10.3) with (10.1), we obtain

$$|L'_n| \leq 2^{-r+1} |\mathrm{Ker}\, \rho_h|. \tag{10.4}$$

If $r \geq 3$, then (10.4) directly implies that $|L'_n| \leq |\mathbb{Z}^*_n|/4 \leq (n-1)/4$, and we are done. So suppose that $r = 2$. In this case, Theorem 10.2 implies that n is not a Carmichael number, which implies that $|\mathrm{Ker}\, \rho_h| \leq |\mathbb{Z}^*_n|/2$, and so again, (10.4) implies $|L'_n| \leq |\mathbb{Z}^*_n|/4 \leq (n-1)/4$. □

EXERCISE 10.1. Show that an integer $n > 1$ is prime if and only if there exists an element in $\mathbb{Z}^*_n$ of multiplicative order $n - 1$.

EXERCISE 10.2. Show that Carmichael numbers satisfy Fermat's little theorem; that is, if n is a Carmichael number, then $\alpha^n = \alpha$ for all $\alpha \in \mathbb{Z}_n$.

EXERCISE 10.3. Let p be a prime. Show that $n := 2p + 1$ is a prime if and only if $2^{n-1} \equiv 1 \pmod{n}$.

EXERCISE 10.4. Here is another primality test that takes as input an odd integer $n > 1$, and a positive integer parameter k. The algorithm chooses $\alpha_1, \ldots, \alpha_k \in \mathbb{Z}_n^+$ at random, and computes

$$\beta_i := \alpha_i^{(n-1)/2} \quad (i = 1, \ldots, k).$$

If $(\beta_1, \ldots, \beta_k)$ is of the form $(\pm 1, \pm 1, \ldots, \pm 1)$, but is not equal to $(1, 1, \ldots, 1)$, the algorithm outputs *true*; otherwise, the algorithm outputs *false*. Show that if n is prime, then the algorithm outputs *false* with probability at most 2^{-k}, and if n is composite, the algorithm outputs *true* with probability at most 2^{-k}.

In the terminology of §9.7, the algorithm in the above exercise is an example of an "Atlantic City" algorithm for the language of prime numbers (or equivalently, the language of composite numbers), while the Miller–Rabin test is an example of a "Monte Carlo" algorithm for the language of *composite* numbers.

10.3 Generating random primes using the Miller–Rabin test

The Miller–Rabin test is the most practical algorithm known for testing primality, and because of this, it is widely used in many applications, especially cryptographic applications where one needs to generate large, random primes (as we saw in §4.7). In this section, we discuss how one uses the Miller–Rabin test in several practically relevant scenarios where one must generate large primes.

10.3.1 Generating a random prime between 2 and m

Suppose we are given an integer $m \geq 2$, and want to generate a random prime between 2 and m. We can do this by simply picking numbers at random until one of them passes a primality test. We discussed this problem in some detail in §9.4, where we assumed that we had a primality test *IsPrime*. The reader should review §9.4, and §9.4.1 in particular. In this section, we discuss aspects of this problem that are specific to the situation where the Miller–Rabin test is used to implement *IsPrime*. To be more precise, let us define the following algorithm:

Algorithm MR**.** On input n, k, where n and k are integers with $n > 1$ and $k \geq 1$, do the following:

if $n = 2$ then return *true*
if n is even then return *false*
repeat k times
    $\alpha \xleftarrow{\$} \mathbb{Z}_n^+$
    if $\alpha \notin L'_n$ return *false*
return *true*

So we shall implement *IsPrime*$(\cdot)$ as $\mathrm{MR}(\cdot, k)$, where k is an auxiliary parameter. By Theorem 10.3, if n is prime, the output of $\mathrm{MR}(n, k)$ is always *true*, while if n is composite, the output is *true* with probability at most 4^{-k}. Thus, this implementation of *IsPrime* satisfies the assumptions in §9.4.1, with $\varepsilon = 4^{-k}$.

Let $\gamma(m, k)$ be the probability that the output of Algorithm RP in §9.4 — using this implementation of *IsPrime* — is composite. Then as we discussed in §9.4.1,

$$\gamma(m, k) \leq 4^{-k} \cdot \frac{m-1}{\pi(m)} = O(4^{-k}\ell), \tag{10.5}$$

where $\ell := \mathrm{len}(m)$, and $\pi(m)$ is the number of primes up to m. Furthermore, if the output of Algorithm RP is prime, then every prime is equally likely; that is, the conditional distribution of the output, given that the output is prime, is (essentially) the uniform distribution on the set of primes up to m.

Let us now consider the expected running time of Algorithm RP. As discussed in §9.4.1, the expected number of iterations of the main loop in Algorithm RP is $O(\ell)$. Clearly, the expected running time of a single loop iteration is $O(k\ell^3)$, since $\mathrm{MR}(n, k)$ executes at most k iterations of the Miller–Rabin test, and each such test takes time $O(\ell^3)$. This leads to a bound on the expected total running time of Algorithm RP of $O(k\ell^4)$. However, this estimate is overly pessimistic, because when n is composite, we expect to perform very few Miller–Rabin tests — only when n is prime do we actually perform all k of them.

To make a rigorous argument, let us define random variables measuring various quantities during the *first* iteration of the main loop in Algorithm RP: N_1 is the value of n; K_1 is the number of Miller–Rabin tests actually performed; Z_1 is the running time. Of course, N_1 is uniformly distributed over $\{2, \ldots, m\}$. Let C_1 be the event that N_1 is composite. Consider the conditional distribution of K_1 given C_1. This is not exactly a geometric distribution, since K_1 never takes on values greater than k; nevertheless, using Theorem 8.17, we can easily calculate

$$\mathsf{E}[K_1 \mid C_1] = \sum_{i \geq 1} \mathsf{P}[K_1 \geq i \mid C_1] \leq \sum_{i \geq 1} (1/4)^{i-1} = 4/3.$$

Using the law of total expectation (8.24), it follows that

$$\begin{aligned}\mathsf{E}[K_1] &= \mathsf{E}[K_1 \mid C_1]\,\mathsf{P}[C_1] + \mathsf{E}[K_1 \mid \overline{C}_1]\,\mathsf{P}[\overline{C}_1] \\ &\leq 4/3 + k\pi(m)/(m-1).\end{aligned}$$

Thus, $\mathsf{E}[K_1] \leq 4/3 + O(k/\ell)$, and hence $\mathsf{E}[Z_1] = O(\ell^3\,\mathsf{E}[K_1]) = O(\ell^3 + k\ell^2)$. Therefore, if Z is the total running time of Algorithm RP, then $\mathsf{E}[Z] = O(\ell\,\mathsf{E}[Z_1])$, and so

$$\mathsf{E}[Z] = O(\ell^4 + k\ell^3). \tag{10.6}$$

Note that the above estimate (10.5) for $\gamma(m,k)$ is actually quite pessimistic. This is because the error probability 4^{-k} is a worst-case estimate; in fact, for "most" composite integers n, the probability that $\mathrm{MR}(n,k)$ outputs *true* is much smaller than this. In fact, $\gamma(m,1)$ is *very* small for large m. For example, the following is known:

Theorem 10.4. *We have*

$$\gamma(m,1) \leq \exp[-(1+o(1))\log(m)\log(\log(\log(m)))/\log(\log(m))].$$

Proof. Literature—see §10.5. □

The bound in the above theorem goes to zero quite quickly: faster than $(\log m)^{-c}$ for every positive constant c. While the above theorem is asymptotically very good, in practice, one needs explicit bounds. For example, the following *lower* bounds for $-\log_2(\gamma(2^\ell,1))$ are known:

ℓ	200	300	400	500	600
	3	19	37	55	74

Given an upper bound on $\gamma(m,1)$, we can bound $\gamma(m,k)$ for $k \geq 2$ using the following inequality:

$$\gamma(m,k) \leq \frac{\gamma(m,1)}{1-\gamma(m,1)} 4^{-k+1}. \tag{10.7}$$

To prove (10.7), it is not hard to see that on input m, the output distribution of Algorithm RP is the same as that of the following algorithm:

```
repeat
    repeat
        n' ←$ {2, ..., m}
    until MR(n', 1)
    n ← n'
until MR(n, k − 1)
output n
```

Let N_1 be the random variable representing the value of n in the first iteration of the main loop in this algorithm, let C_1 be the event that N_1 is composite, and let $\mathcal{H}_1$ be the event that this algorithm halts at the end of the first iteration of the main loop. Using Theorem 9.3, we see that

$$\gamma(m,k) = \mathsf{P}[C_1 \mid \mathcal{H}_1] = \frac{\mathsf{P}[C_1 \cap \mathcal{H}_1]}{\mathsf{P}[\mathcal{H}_1]} \leq \frac{\mathsf{P}[C_1 \cap \mathcal{H}_1]}{\mathsf{P}[\bar{C}_1]} = \frac{\mathsf{P}[\mathcal{H}_1 \mid C_1]\,\mathsf{P}[C_1]}{\mathsf{P}[\bar{C}_1]}$$
$$\leq \frac{4^{-k+1}\gamma(m,1)}{1-\gamma(m,1)},$$

which proves (10.7).

Given that $\gamma(m,1)$ is so small, for large m, Algorithm RP actually exhibits the following behavior in practice: it generates a random value $n \in \{2,\ldots,m\}$; if n is odd and composite, then the very *first* iteration of the Miller–Rabin test will detect this with overwhelming probability, and no more iterations of the test are performed on this n; otherwise, if n is prime, the algorithm will perform $k-1$ more iterations of the Miller–Rabin test, "just to make sure."

EXERCISE 10.5. Consider the problem of generating a random Sophie Germain prime between 2 and m (see §5.5.5). One algorithm to do this is as follows:

```
repeat
    n ←$ {2,...,m}
    if MR(n,k) then
        if MR(2n+1,k) then
            output n and halt
forever
```

Assuming Conjecture 5.24, show that this algorithm runs in expected time $O(\ell^5 + k\ell^4)$, and outputs a number that is not a Sophie Germain prime with probability $O(4^{-k}\ell^2)$. As usual, $\ell := \mathrm{len}(m)$.

EXERCISE 10.6. Improve the algorithm in the previous exercise, so that under the same assumptions, it runs in expected time $O(\ell^5 + k\ell^3)$, and outputs a number that is not a Sophie Germain prime with probability $O(4^{-k}\ell^2)$, or even better, show that this probability is at most $\gamma(m,k)\pi^*(m)/\pi(m) = O(\gamma(m,k)\ell)$, where $\pi^*(m)$ is defined as in §5.5.5.

EXERCISE 10.7. Suppose in Algorithm RFN in §9.6 we implement algorithm *IsPrime*$(\cdot)$ as MR$(\cdot,k)$, where k is a parameter satisfying $4^{-k}(\log m + 1) \leq 1/2$, and m is the input to RFN. Show that the expected running time of Algorithm RFN in this case is $O(\ell^5 + k\ell^4\,\mathrm{len}(\ell))$. Hint: use Exercise 9.13.

10.3.2 Trial division up to a small bound

In generating a random prime, most candidates will in fact be composite, and so it makes sense to cast these out as quickly as possible. Significant efficiency gains can be achieved by testing if a given candidate n is divisible by any prime up to a given bound s, before we subject n to a Miller–Rabin test. This strategy makes sense, since for a small, "single precision" prime p, we can test if $p \mid n$ essentially in time $O(\text{len}(n))$, while a single iteration of the Miller–Rabin test takes time $O(\text{len}(n)^3)$.

To be more precise, let us define the following algorithm:

Algorithm MRS**.** On input n, k, s, where $n, k, s \in \mathbb{Z}$, and $n > 1$, $k \geq 1$, and $s > 1$, do the following:

```
for each prime p ≤ s do
    if p | n then
        if p = n then return true else return false
repeat k times
    α ←$ Z_n^+
    if α ∉ L'_n return false
return true
```

In an implementation of the above algorithm, one would most likely use the sieve of Eratosthenes (see §5.4) to generate the small primes.

Note that $\text{MRS}(n, k, 2)$ is equivalent to $\text{MR}(n, k)$. Also, it is clear that the probability that $\text{MRS}(n, k, s)$ makes a mistake is no more than the probability that $\text{MR}(n, k)$ makes a mistake. Therefore, using MRS in place of MR will not increase the probability that the output of Algorithm RP is a composite—indeed, it is likely that this probability decreases significantly.

Let us now analyze the impact on the running time Algorithm RP. To do this, we need to estimate the probability $\sigma(m, s)$ that a randomly chosen integer between 2 and m is not divisible by any prime up to s. If m is sufficiently large with respect to s, the following heuristic argument can be made rigorous, as we will discuss below. The probability that a random integer is divisible by a prime p is about $1/p$, so the probability that it is not divisible by p is about $1 - 1/p$. Assuming that these events are essentially independent for different values of p (this is the heuristic part), we estimate

$$\sigma(m, s) \approx \prod_{p \leq s} (1 - 1/p). \tag{10.8}$$

Assuming for the time being that the approximation in (10.8) is sufficiently accurate, then using Mertens' theorem (Theorem 5.13), we may deduce that

$$\sigma(m, s) = O(1/\log s). \tag{10.9}$$

Later, when we make this argument more rigorous, we shall see that (10.9) holds provided s is not too large relative to m, and in particular, if $s = O((\log m)^c)$ for some constant c.

The estimate (10.9) gives us a bound on the probability that a random integer passes the trial division phase, and so must be subjected to Miller–Rabin; however, performing the trial division takes some time, so we also need to estimate the expected number $\kappa(m, s)$ of trial divisions performed on a random integer between 2 and m. Of course, in the worst case, we divide by all primes up to s, and so $\kappa(m, s) \leq \pi(s) = O(s/\log s)$, but we can get a better bound, as follows. Let $p_1, p_2, \ldots, p_r$ be the primes up to s, and for $i = 1, \ldots, r$, let q_i be the probability that we perform at least i trial divisions. By Theorem 8.17, we have

$$\kappa(m, s) = \sum_{i=1}^{r} q_i.$$

Moreover, $q_1 = 1$, and $q_i = \sigma(m, p_{i-1})$ for $i = 2, \ldots, r$. From this, and (10.9), it follows that

$$\kappa(m, s) = 1 + \sum_{i=2}^{r} \sigma(m, p_{i-1}) = O\Big(\sum_{p \leq s} 1/\log p\Big).$$

As a simple consequence of Chebyshev's theorem (in particular, see Exercise 5.3), we obtain

$$\kappa(m, s) = O(s/(\log s)^2). \tag{10.10}$$

We now derive a bound on the running time of Algorithm RP, assuming that *IsPrime*$(\cdot)$ is implemented using MRS$(\cdot, k, s)$. Let $\ell := \mathrm{len}(m)$. Our argument follows the same lines as was used to derive the estimate (10.6). Let us define random variables measuring various quantities during the *first* iteration of the main loop in Algorithm RP: N_1 is the value of n; K_1 is the number of Miller–Rabin tests actually performed; Z_1 is the running time. Also, let C_1 be the event that N_1 is composite, and let D_1 be the event that N_1 passes the trial division check. Then we have

$$\begin{aligned}
\mathsf{E}[K_1] &= \mathsf{E}[K_1 \mid C_1 \cap D_1]\,\mathsf{P}[C_1 \cap D_1] + \mathsf{E}[K_1 \mid C_1 \cap \overline{D}_1]\,\mathsf{P}[C_1 \cap \overline{D}_1] \\
&\qquad + \mathsf{E}[K_1 \mid \overline{C}_1]\,\mathsf{P}[\overline{C}_1] \\
&\leq 4/3 \cdot \mathsf{P}[C_1 \cap D_1] + 0 \cdot \mathsf{P}[C_1 \cap \overline{D}_1] + k \cdot \mathsf{P}[\overline{C}_1] \\
&\leq 4/3 \cdot \mathsf{P}[D_1] + k \cdot \mathsf{P}[\overline{C}_1].
\end{aligned}$$

By (10.9) and Chebyshev's theorem, it follows that

$$\mathsf{E}[K_1] = O(1/\mathrm{len}(s) + k/\ell). \tag{10.11}$$

Let us write $Z_1 = Z_1' + Z_1''$, where Z_1' is the amount of time spent performing the Miller–Rabin test, and Z_1'' is the amount of time spent performing trial division. By (10.11), we have $\mathsf{E}[Z_1'] = O(\ell^3/\operatorname{len}(s) + k\ell^2)$. Further, assuming that each individual trial division step takes time $O(\ell)$, then by (10.10) we have $\mathsf{E}[Z_1''] = O(\ell s/\operatorname{len}(s)^2)$. Hence,

$$\mathsf{E}[Z_1] = O(\ell^3/\operatorname{len}(s) + k\ell^2 + \ell s/\operatorname{len}(s)^2).$$

It follows that if Z is the total running time of Algorithm RP, then

$$\mathsf{E}[Z] = O(\ell^4/\operatorname{len}(s) + k\ell^3 + \ell^2 s/\operatorname{len}(s)^2).$$

Clearly, we want to choose the parameter s so that the time spent performing trial division is dominated by the time spent performing the Miller–Rabin test. To this end, let us assume that $\ell \leq s \leq \ell^2$. Then we have

$$\mathsf{E}[Z] = O(\ell^4/\operatorname{len}(\ell) + k\ell^3). \tag{10.12}$$

This estimate does not take into account the time to generate the small primes using the sieve of Eratosthenes. These values might be pre-computed, in which case this time is zero, but even if we compute them on the fly, this takes time $O(s \operatorname{len}(\operatorname{len}(s)))$, which is dominated by the running time of the rest of the algorithm for the values of s under consideration.

Thus, by sieving up to a bound s, where $\ell \leq s \leq \ell^2$, then compared to (10.6), we effectively reduce the running time by a factor proportional to $\operatorname{len}(\ell)$, which is a very real and noticeable improvement in practice.

As we already mentioned, the above analysis is heuristic, but the results are correct. We shall now discuss how this analysis can be made rigorous; however, we should remark that any such rigorous analysis is mainly of theoretical interest only — in any practical implementation, the optimal choice of the parameter s is best determined by experiment, with the analysis being used only as a rough guide. Now, to make the analysis rigorous, we need prove that the estimate (10.8) is sufficiently accurate. Proving such estimates takes us into the realm of "sieve theory." The larger m is with respect to s, the easier it is to prove such estimates. We shall prove only the simplest and most naive such estimate, but it is still good enough for our purposes.

Before stating any results, let us restate the problem slightly differently. For a given real number $y \geq 0$, let us call a positive integer "y-rough" if it is not divisible by any prime p up to y. For all real numbers $x \geq 0$ and $y \geq 0$, let us define $R(x, y)$ to be the number of y-rough positive integers up to x. Thus, since $\sigma(m, s)$ is the probability that a random integer between 2 and m is s-rough, and 1 is by definition s-rough, we have $\sigma(m, s) = (R(m, s) - 1)/(m - 1)$.

Theorem 10.5. *For all real $x \geq 0$ and $y \geq 0$, we have*

$$\left| R(x, y) - x \prod_{p \leq y} (1 - 1/p) \right| \leq 2^{\pi(y)}.$$

Proof. To simplify the notation, we shall use the Möbius function μ (see §2.9). Also, for a real number u, let us write $u = \lfloor u \rfloor + \{u\}$, where $0 \leq \{u\} < 1$. Let Q be the product of the primes up to the bound y.

Now, there are $\lfloor x \rfloor$ positive integers up to x, and of these, for each prime p dividing Q, precisely $\lfloor x/p \rfloor$ are divisible by p, for each pair p, p' of distinct primes dividing Q, precisely $\lfloor x/pp' \rfloor$ are divisible by pp', and so on. By inclusion/exclusion (see Theorem 8.1), we have

$$R(x, y) = \sum_{d \mid Q} \mu(d) \lfloor x/d \rfloor = \sum_{d \mid Q} \mu(d)(x/d) - \sum_{d \mid Q} \mu(d)\{x/d\}.$$

Moreover,

$$\sum_{d \mid Q} \mu(d)(x/d) = x \sum_{d \mid Q} \mu(d)/d = x \prod_{p \leq y} (1 - 1/p),$$

and

$$\left| \sum_{d \mid Q} \mu(d)\{x/d\} \right| \leq \sum_{d \mid Q} 1 = 2^{\pi(y)}.$$

That proves the theorem. □

This theorem says something non-trivial only when y is quite small. Nevertheless, using Chebyshev's theorem on the density of primes, along with Mertens' theorem, it is not hard to see that this theorem implies that (10.9) holds when $s = O((\log m)^c)$ for some constant c (see Exercise 10.8), which implies the estimate (10.12) above, when $\ell \leq s \leq \ell^2$.

EXERCISE 10.8. Suppose that s is a function of m such that $s = O((\log m)^c)$ for some positive constant c. Show that $\sigma(m, s) = O(1/\log s)$.

EXERCISE 10.9. Let f be a polynomial with integer coefficients. For real $x \geq 0$ and $y \geq 0$, define $R_f(x, y)$ to be the number of positive integers t up to x such that $f(t)$ is y-rough. For each positive integer m, define $\omega_f(m)$ to be the number of integers $t \in \{0, \ldots, m-1\}$ such that $f(t) \equiv 0 \pmod{m}$. Show that

$$\left| R_f(x, y) - x \prod_{p \leq y} (1 - \omega_f(p)/p) \right| \leq \prod_{p \leq y} (1 + \omega_f(p)).$$

EXERCISE 10.10. Consider again the problem of generating a random Sophie Germain prime, as discussed in Exercises 10.5 and 10.6. A useful idea is to first test if *either* n *or* $2n + 1$ are divisible by any small primes up to some bound s, before performing any more expensive tests. Using this idea, design and analyze an algorithm that improves the running time of the algorithm in Exercise 10.6 to $O(\ell^5/\operatorname{len}(\ell)^2 + k\ell^3)$—under the same assumptions, and achieving the same error probability bound as in that exercise. Hint: first show that the previous exercise implies that the number of positive integers t up to x such that both t and $2t + 1$ are y-rough is at most

$$x \cdot \frac{1}{2} \prod_{2<p\leq y} (1 - 2/p) + 3^{\pi(y)}.$$

EXERCISE 10.11. Design an algorithm that takes as input a prime q and a bound m, and outputs a random prime p between 2 and m such that $p \equiv 1 \pmod{q}$. Clearly, we need to assume that m is sufficiently large with respect to q. Analyze your algorithm assuming Conjecture 5.22. State how large m must be with respect to q, and under these assumptions, show that your algorithm runs in time $O(\ell^4/\operatorname{len}(\ell) + k\ell^3)$, and that its output is incorrect with probability $O(4^{-k}\ell)$. As usual, $\ell := \operatorname{len}(m)$.

10.3.3 Generating a random ℓ-bit prime

In some applications, we want to generate a random prime of fixed size—a random 1024-bit prime, for example. More generally, let us consider the following problem: given an integer $\ell \geq 2$, generate a random ℓ-bit prime, that is, a prime in the interval $[2^{\ell-1}, 2^\ell)$.

Bertrand's postulate (Theorem 5.8) implies that there exists a constant $c > 0$ such that $\pi(2^\ell) - \pi(2^{\ell-1}) \geq c2^{\ell-1}/\ell$ for all $\ell \geq 2$.

Now let us modify Algorithm RP so that it takes as input an integer $\ell \geq 2$, and repeatedly generates a random n in the interval $\{2^{\ell-1}, \ldots, 2^\ell - 1\}$ until *IsPrime*(n) returns *true*. Let us call this variant Algorithm RP′. Further, let us implement *IsPrime*$(\cdot)$ as MR$(\cdot, k)$, for some auxiliary parameter k, and define $\gamma'(\ell, k)$ to be the probability that the output of Algorithm RP′—with this implementation of *IsPrime*—is composite.

Then using exactly the same reasoning as in §10.3.1, we have

$$\gamma'(\ell, k) \leq 4^{-k} \frac{2^{\ell-1}}{\pi(2^\ell) - \pi(2^{\ell-1})} = O(4^{-k}\ell);$$

moreover, if the output of Algorithm RP′ is prime, then every ℓ-bit prime is equally

likely, and the expected running time is $O(\ell^4 + k\ell^3)$. By doing some trial division as in §10.3.2, this can be reduced to $O(\ell^4/\operatorname{len}(\ell) + k\ell^3)$.

The function $\gamma'(\ell, k)$ has been studied a good deal; for example, the following explicit bound is known:

Theorem 10.6. *For all $\ell \geq 2$, we have*

$$\gamma'(\ell, 1) \leq \ell^2 4^{2-\sqrt{\ell}}.$$

Proof. Literature—see §10.5. □

Upper bounds for $\gamma'(\ell, k)$ for specific values of ℓ and k have been computed. The following table lists some known *lower* bounds for $-\log_2(\gamma'(\ell, k))$ for various values of ℓ and k:

$k \backslash \ell$	200	300	400	500	600
1	11	19	37	56	75
2	25	33	46	63	82
3	34	44	55	70	88
4	41	53	63	78	95
5	47	60	72	85	102

Using exactly the same reasoning as the derivation of (10.7), one sees that

$$\gamma'(\ell, k) \leq \frac{\gamma'(\ell, 1)}{1 - \gamma'(\ell, 1)} 4^{-k+1}.$$

10.4 Factoring and computing Euler's phi function

In this section, we use some of the ideas developed to analyze the Miller–Rabin test to prove that the problem of factoring n and the problem of computing $\varphi(n)$ are equivalent. By equivalent, we mean that given an efficient algorithm to solve one problem, we can efficiently solve the other, and *vice versa*.

Clearly, one direction is easy: if we can factor n into primes, so

$$n = p_1^{e_1} \cdots p_r^{e_r}, \tag{10.13}$$

then we can simply compute $\varphi(n)$ using the formula

$$\varphi(n) = p_1^{e_1 - 1}(p_1 - 1) \cdots p_r^{e_r - 1}(p_r - 1).$$

For the other direction, first consider the special case where $n = pq$, for distinct primes p and q. Suppose we are given n and $\varphi(n)$, so that we have two equations in the unknowns p and q:

$$n = pq \quad \text{and} \quad \varphi(n) = (p - 1)(q - 1).$$

Substituting n/p for q in the second equation, and simplifying, we obtain

$$p^2 + (\varphi(n) - n - 1)p + n = 0,$$

which can be solved using the quadratic formula.

For the general case, it is just as easy to prove a stronger result: given any non-zero multiple of the *exponent* of $\mathbb{Z}_n^*$, we can efficiently factor n. In particular, this will show that we can efficiently factor Carmichael numbers.

Before stating the algorithm in its full generality, we can convey the main idea by considering the special case where $n = pq$, where p and q are distinct primes, with $p \equiv q \equiv 3 \pmod 4$. Suppose we are given such an n, along with a non-zero multiple f of the exponent of $\mathbb{Z}_n^*$. Now, $\mathbb{Z}_n^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$, and since $\mathbb{Z}_p^*$ is a cyclic group of order $p - 1$ and $\mathbb{Z}_q^*$ is a cyclic group of order $q - 1$, this means that f is a non-zero common multiple of $p - 1$ and $q - 1$. Let $f = t2^h$, where t is odd, and consider the following probabilistic algorithm:

$\alpha \xleftarrow{\$} \mathbb{Z}_n^+$
$d \leftarrow \gcd(\mathrm{rep}(\alpha), n)$
if $d \neq 1$ then output d and halt
$\beta \leftarrow \alpha^t$
$d' \leftarrow \gcd(\mathrm{rep}(\beta) + 1, n)$
if $d' \notin \{1, n\}$ then output d' and halt
output "failure"

Recall that $\mathrm{rep}(\alpha)$ denotes the canonical representative of α, that is, the unique integer a such that $[a]_n = \alpha$ and $0 \leq a < n$. We shall prove that this algorithm outputs a non-trivial divisor of n with probability at least $1/2$.

Let ρ be the t-power map on $\mathbb{Z}_n^*$, and let $G := \rho^{-1}(\{\pm 1\})$. We shall show that

- $G \subsetneq \mathbb{Z}_n^*$, and
- if the algorithm chooses $\alpha \notin G$, then it splits n.

Since G is a subgroup of $\mathbb{Z}_n^*$, it follows that $|G|/|\mathbb{Z}_n^+| \leq |G|/|\mathbb{Z}_n^*| \leq 1/2$, and this implies the algorithm succeeds with probability at least $1/2$.

Let $\theta : \mathbb{Z}_n \to \mathbb{Z}_p \times \mathbb{Z}_q$ be the ring isomorphism from the Chinese remainder theorem. The assumption that $p \equiv 3 \pmod 4$ means that $(p - 1)/2$ is an odd integer, and since f is a multiple of $p - 1$, it follows that $\gcd(t, p - 1) = (p - 1)/2$, and hence the image of $\mathbb{Z}_p^*$ under the t-power map is the subgroup of $\mathbb{Z}_p^*$ of order 2, which is $\{\pm 1\}$. Likewise, the image of $\mathbb{Z}_q^*$ under the t-power map is $\{\pm 1\}$. Thus,

$$\theta(\mathrm{Im}\,\rho) = \theta((\mathbb{Z}_n^*)^t) = (\theta(\mathbb{Z}_n^*))^t = (\mathbb{Z}_p^*)^t \times (\mathbb{Z}_q^*)^t = \{\pm 1\} \times \{\pm 1\},$$

and so $\mathrm{Im}\,\rho$ consists of the four elements:

$$1 = \theta^{-1}(1, 1),\ -1 = \theta^{-1}(-1, -1),\ \theta^{-1}(-1, 1),\ \theta^{-1}(1, -1).$$

By the observations in the previous paragraph, not all elements of $\mathbb{Z}_n^*$ map to ± 1 under ρ, which means that $G \subsetneq \mathbb{Z}_n^*$. Suppose that the algorithm chooses $\alpha \in \mathbb{Z}_n^+ \setminus G$. We want to show that n gets split. If $\alpha \notin \mathbb{Z}_n^*$, then $\gcd(\mathrm{rep}(\alpha), n)$ is a non-trivial divisor of n, and the algorithm splits n. So let us assume that $\alpha \in \mathbb{Z}_n^* \setminus G$. Consider the value $\beta = \alpha^t = \rho(\alpha)$ computed by the algorithm. Since $\alpha \notin G$, we have $\beta \neq \pm 1$, and by the observations in the previous paragraph, we have $\theta(\beta) = (-1, 1)$ or $\theta(\beta) = (1, -1)$. In the first case, $\theta(\beta + 1) = (0, 2)$, and so $\gcd(\mathrm{rep}(\beta) + 1, n) = p$, while in the second case, $\theta(\beta + 1) = (2, 0)$, and so $\gcd(\mathrm{rep}(\beta) + 1, n) = q$. In either case, the algorithm splits n.

We now consider the general case, where n is an arbitrary positive integer. Let $\lambda(n)$ denote the exponent of $\mathbb{Z}_n^*$. If the prime factorization of n is as in (10.13), then by the Chinese remainder theorem, we have

$$\lambda(n) = \mathrm{lcm}(\lambda(p_1^{e_1}), \ldots, \lambda(p_r^{e_r})).$$

Moreover, for every prime power p^e, by Theorem 7.28, we have

$$\lambda(p^e) = \begin{cases} p^{e-1}(p-1) & \text{if } p \neq 2 \text{ or } e \leq 2, \\ 2^{e-2} & \text{if } p = 2 \text{ and } e \geq 3. \end{cases}$$

In particular, if $d \mid n$, then $\lambda(d) \mid \lambda(n)$.

Now, assume we are given n, along with a non-zero multiple f of $\lambda(n)$. We would like to calculate the complete prime factorization of n. We may proceed recursively: first, if $n = 1$, we may obviously halt; otherwise, we test if n is prime, using an efficient primality test, and if so, halt (if we are using the Miller–Rabin test, then we may erroneously halt even when n is composite, but we can ensure that this happens with negligible probability); otherwise, we split n as $n = d_1 d_2$, using an algorithm to be described below, and then recursively factor both d_1 and d_2; since $\lambda(d_1) \mid f$ and $\lambda(d_2) \mid f$, we may use the same value f in the recursion.

So let us assume that $n > 1$ and n is not prime, and our goal now is to use f to obtain a non-trivial factorization of n. If n is even, then we can certainly do this. Moreover, if n is a perfect power—that is, if $n = a^b$ for some integers $a > 1$ and $b > 1$—we can also obtain a non-trivial factorization of n (see Exercise 3.31).

So let us assume not only that $n > 1$ and n is not prime, but also that n is odd, and n is not a perfect power. Let $f = t2^h$, where t is odd. Consider the following probabilistic algorithm:

$\alpha \xleftarrow{\$} \mathbb{Z}_n^+$
$d \leftarrow \gcd(\mathrm{rep}(\alpha), n)$
if $d \neq 1$ then output d and halt
$\beta \leftarrow \alpha^t$
for $j \leftarrow 0$ to $h - 1$ do
    $d' \leftarrow \gcd(\mathrm{rep}(\beta) + 1, n)$
    if $d' \notin \{1, n\}$ then output d' and halt
    $\beta \leftarrow \beta^2$
output "failure"

We want to show that this algorithm outputs a non-trivial factor of n with probability at least $1/2$. To do this, suppose the prime factorization of n is as in (10.13). Then by our assumptions about n, we have $r \geq 2$ and each p_i is odd. Let $\lambda(p_i^{e_i}) = t_i 2^{h_i}$, where t_i is odd, for $i = 1, \ldots, r$, and let $g := \max\{h_1, \ldots, h_r\}$. Note that since $\lambda(n) \mid f$, we have $1 \leq g \leq h$.

Let ρ be the $(t2^{g-1})$-power map on $\mathbb{Z}_n^*$, and let $G := \rho^{-1}(\{\pm 1\})$. As above, we shall show that

- $G \subsetneq \mathbb{Z}_n^*$, and
- if the algorithm chooses $\alpha \notin G$, then it splits n,

which will prove that the algorithm splits n with probability at least $1/2$.

Let

$$\theta : \mathbb{Z}_n \to \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_r^{e_r}}$$

be the ring isomorphism of the Chinese remainder theorem. We have

$$\theta(\mathrm{Im}\,\rho) = G_1 \times \cdots \times G_r,$$

where

$$G_i := \left(\mathbb{Z}_{p_i^{e_i}}^*\right)^{t2^{g-1}} \quad \text{for } i = 1, \ldots, r.$$

Let us assume the p_i's are ordered so that $h_i = g$ for $i = 1, \ldots, r'$, and $h_i < g$ for $i = r' + 1, \ldots, r$, where we have $1 \leq r' \leq r$. Then we have $G_i = \{\pm 1\}$ for $i = 1, \ldots, r'$, and $G_i = \{1\}$ for $i = r' + 1, \ldots, r$.

By the observations in the previous paragraph, and the fact that $r \geq 2$, the image of ρ contains elements other than ± 1; for example, $\theta^{-1}(-1, 1, \ldots, 1)$ is such an element. This means that $G \subsetneq \mathbb{Z}_n^*$. Suppose the algorithm chooses $\alpha \in \mathbb{Z}_n^+ \setminus G$. We want to show that n gets split. If $\alpha \notin \mathbb{Z}_n^*$, then $\gcd(\mathrm{rep}(\alpha), n)$ is a non-trivial divisor of n, and so the algorithm certainly splits n. So assume $\alpha \in \mathbb{Z}_n^* \setminus G$. In loop iteration $j = g - 1$, the value of β is equal to $\rho(\alpha)$, and writing $\theta(\beta) = (\beta_1, \ldots, \beta_r)$, we have $\beta_i = \pm 1$ for $i = 1, \ldots, r$. Let S be the set of indices i such that $\beta_i = -1$.

As $\alpha \notin G$, we know that $\beta \neq \pm 1$, and so $\emptyset \subsetneq S \subsetneq \{1, \ldots, r\}$. Thus,

$$\gcd(\mathrm{rep}(\beta) + 1, n) = \prod_{i \in S} p_i^{e_i}$$

is a non-trivial factor of n. This means that the algorithm splits n in loop iteration $j = g - 1$ (if not in some earlier loop iteration).

So we have shown that the above algorithm splits n with probability at least $1/2$. If we iterate the algorithm until n gets split, the expected number of loop iterations required will be at most 2. Combining this with the above recursive algorithm, we get an algorithm that completely factors an arbitrary n in expected polynomial time.

EXERCISE 10.12. Suppose you are given an integer n of the form $n = pq$, where p and q are distinct, ℓ-bit primes, with $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are themselves prime. Suppose that you are also given an integer t such that $\gcd(t, p'q') \neq 1$. Show how to efficiently factor n.

EXERCISE 10.13. Suppose there is a probabilistic algorithm A that takes as input an integer n of the form $n = pq$, where p and q are distinct, ℓ-bit primes, with $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are prime. The algorithm also takes as input $\alpha, \beta \in (\mathbb{Z}_n^*)^2$. It outputs either "failure," or integers x, y, not both zero, such that $\alpha^x \beta^y = 1$. Furthermore, assume that A runs in expected polynomial time, and that for all n of the above form, and for randomly chosen $\alpha, \beta \in (\mathbb{Z}_n^*)^2$, A succeeds in finding x, y as above with probability $\varepsilon(n)$. Here, the probability is taken over the random choice of α and β, as well as the random choices made during the execution of A on input (n, α, β). Show how to use A to construct another probabilistic algorithm A' that takes as input n as above, runs in expected polynomial time, and that satisfies the following property:

if $\varepsilon(n) \geq 0.001$, then A' factors n with probability at least 0.999.

10.5 Notes

The Miller–Rabin test is due to Miller [67] and Rabin [79]. The paper by Miller defined the set L'_n, but did not give a probabilistic analysis. Rather, Miller showed that under a generalization of the Riemann hypothesis, for composite n, the least positive integer a such that $[a]_n \in \mathbb{Z}_n \setminus L'_n$ is at most $O((\log n)^2)$, thus giving rise to a deterministic primality test whose correctness depends on the above unproved hypothesis. The later paper by Rabin re-interprets Miller's result in the context of probabilistic algorithms.

Bach [10] gives an explicit version of Miller's result, showing that under the same assumptions, the least positive integer a such that $[a]_n \in \mathbb{Z}_n \setminus L'_n$ is at most $2(\log n)^2$; more generally, Bach shows that the following holds under a generalization of the Riemann hypothesis:

> For every positive integer n, and every subgroup $G \subsetneq \mathbb{Z}_n^*$, the least positive integer a such that $[a]_n \in \mathbb{Z}_n \setminus G$ is at most $2(\log n)^2$, and the least positive integer b such that $[b]_n \in \mathbb{Z}_n^* \setminus G$ is at most $3(\log n)^2$.

The first efficient probabilistic primality test was invented by Solovay and Strassen [99] (their paper was actually submitted for publication in 1974). Later, in Chapter 21, we shall discuss a recently discovered, deterministic, polynomial-time (though not very practical) primality test, whose analysis does not rely on any unproved hypothesis.

Carmichael numbers are named after R. D. Carmichael, who was the first to discuss them, in work published in the early 20th century. Alford, Granville, and Pomerance [7] proved that there are infinitely many Carmichael numbers.

Exercise 10.4 is based on Lehmann [58].

Theorem 10.4, as well as the table of values just below it, are from Kim and Pomerance [55]. In fact, these bounds hold for the weaker test based on L_n.

Our analysis in §10.3.2 is loosely based on a similar analysis in §4.1 of Maurer [65]. Theorem 10.5 and its generalization in Exercise 10.9 are certainly not the best results possible in this area. The general goal of "sieve theory" is to prove useful upper and lower bounds for quantities like $R_f(x, y)$ that hold when y is as large as possible with respect to x. For example, using a technique known as Brun's pure sieve, one can show that for $\log y < \sqrt{\log x}$, there exist β and β', both of absolute value at most 1, such that

$$R_f(x, y) = (1 + \beta e^{-\sqrt{\log x}})x \prod_{p \le y} (1 - \omega_f(p)/p) + \beta' \sqrt{x}.$$

Thus, this gives us very sharp estimates for $R_f(x, y)$ when x tends to infinity, and y is bounded by any fixed polynomial in $\log x$. For a proof of this result, see §2.2 of Halberstam and Richert [44] (the result itself is stated as equation 2.16). Brun's pure sieve is really just the first non-trivial sieve result, developed in the early 20th century; even stronger results, extending the useful range of y (but with larger error terms), have subsequently been proved.

Theorem 10.6, as well as the table of values immediately below it, are from Damgård, Landrock, and Pomerance [32].

The algorithm presented in §10.4 for factoring an integer given a multiple of $\varphi(n)$ (or, for that matter, $\lambda(n)$) is essentially due to Miller [67]. However, just as for his primality test, Miller presents his algorithm as a deterministic algorithm, which

he analyzes under a generalization of the Riemann hypothesis. The probabilistic version of Miller's factoring algorithm appears to be "folklore."