Algorithm 6.6: SOLOVAY-STRASSEN(n)

choose a random integer a such that $1 \leq a \leq n-1$
$x \leftarrow (\frac{a}{n})$
if $x = 0$
  then return ("n is composite")
$y \leftarrow a^{(n-1)/2} \pmod{n}$
if $x \equiv y \pmod{n}$
  then return ("n is prime")
  else return ("n is composite")

9975 is $9975 = 3 \times 5^2 \times 7 \times 19$. Thus we have

$$\begin{aligned}\left(\frac{6278}{9975}\right) &= \left(\frac{6278}{3}\right)\left(\frac{6278}{5}\right)^2\left(\frac{6278}{7}\right)\left(\frac{6278}{19}\right) \\ &= \left(\frac{2}{3}\right)\left(\frac{3}{5}\right)^2\left(\frac{6}{7}\right)\left(\frac{8}{19}\right) \\ &= (-1)(-1)^2(-1)(-1) \\ &= -1.\end{aligned}$$

□

Suppose $n > 1$ is odd. If n is prime, then $(\frac{a}{n}) \equiv a^{(n-1)/2} \pmod{n}$ for any a. On the other hand, if n is composite, it may or may not be the case that $(\frac{a}{n}) \equiv a^{(n-1)/2} \pmod{n}$. If this congruence holds, then n is called an ***Euler pseudo-prime*** to the base a. For example, 91 is an Euler pseudo-prime to the base 10, because

$$\left(\frac{10}{91}\right) = -1 \equiv 10^{45} \pmod{91}.$$

It can be shown that, for any odd composite n, n is an Euler pseudo-prime to the base a for at most half of the integers $a \in {\mathbb{Z}_n}^*$ (see the Exercises). It is also easy to see that $(\frac{a}{n}) = 0$ if and only if $\gcd(a, n) > 1$ (therefore, if $1 \leq a \leq n-1$ and $(\frac{a}{n}) = 0$, it must be the case that n is composite).

6.4.2 The Solovay-Strassen Algorithm

We present the SOLOVAY-STRASSEN ALGORITHM, as Algorithm 6.6. The facts proven in the previous section show that this is is a yes-biased Monte Carlo algorithm with error probability at most $1/2$.

At this point it is not clear that Algorithm 6.6 is a polynomial-time algorithm. We already know how to evaluate $a^{(n-1)/2} \bmod n$ in time $O((\log n)^3)$, but how do we compute Jacobi symbols efficiently? It might appear to be necessary to first

factor n, since the Jacobi symbol $(\frac{a}{n})$ is defined in terms of the factorization of n. But, if we could factor n, we would already know if it is prime; so this approach ends up in a vicious circle.

Fortunately, we can evaluate a Jacobi symbol without factoring n by using some results from number theory, the most important of which is a generalization of the law of quadratic reciprocity (property 4 below). We now enumerate these properties without proof:

1. If n is a positive odd integer and $m_1 \equiv m_2 \pmod{n}$, then
$$\left(\frac{m_1}{n}\right) = \left(\frac{m_2}{n}\right).$$

2. If n is a positive odd integer, then
$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \equiv \pm 1 \pmod{8} \\ -1 & \text{if } n \equiv \pm 3 \pmod{8}. \end{cases}$$

3. If n is a positive odd integer, then
$$\left(\frac{m_1 m_2}{n}\right) = \left(\frac{m_1}{n}\right)\left(\frac{m_2}{n}\right).$$
In particular, if $m = 2^k t$ and t is odd, then
$$\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right)^k \left(\frac{t}{n}\right).$$

4. Suppose m and n are positive odd integers. Then
$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right) & \text{if } m \equiv n \equiv 3 \pmod{4} \\ \left(\frac{n}{m}\right) & \text{otherwise.} \end{cases}$$

Example 6.8 As an illustration of the application of these properties, we evaluate the Jacobi symbol $(\frac{7411}{9283})$ in Figure 6.2. Notice that we successively apply properties 4, 1, 3, and 2 (in this order) in this computation. □

In general, by applying these four properties in the same manner as was done in the example above, it is possible to compute a Jacobi symbol $(\frac{a}{n})$ in polynomial time. The only arithmetic operations that are required are modular reductions and factoring out powers of two. Note that if an integer is represented in binary notation, then factoring out powers of two amounts to determining the number of trailing zeroes. So, the complexity of the algorithm is determined by the number of modular reductions that must be done. It is not difficult to show that at most $O(\log n)$ modular reductions are performed, each of which can be done in time $O((\log n)^2)$. This shows that the complexity is $O((\log n)^3)$, which is polynomial

$$\begin{aligned}
\left(\frac{7411}{9283}\right) &= -\left(\frac{9283}{7411}\right) && \text{by property 4} \\
&= -\left(\frac{1872}{7411}\right) && \text{by property 1} \\
&= -\left(\frac{2}{7411}\right)^4 \left(\frac{117}{7411}\right) && \text{by property 3} \\
&= -\left(\frac{117}{7411}\right) && \text{by property 2} \\
&= -\left(\frac{7411}{117}\right) && \text{by property 4} \\
&= -\left(\frac{40}{117}\right) && \text{by property 1} \\
&= -\left(\frac{2}{117}\right)^3 \left(\frac{5}{117}\right) && \text{by property 3} \\
&= \left(\frac{5}{117}\right) && \text{by property 2} \\
&= \left(\frac{117}{5}\right) && \text{by property 4} \\
&= \left(\frac{2}{5}\right) && \text{by property 1} \\
&= -1 && \text{by property 2.}
\end{aligned}$$

FIGURE 6.2: Evaluation of a Jacobi symbol

in $\log n$. (In fact, the complexity can be shown to be $O((\log n)^2)$ by more precise analysis.)

Suppose that we have generated a random number n and tested it for primality using the SOLOVAY-STRASSEN ALGORITHM. If we have run the algorithm m times, what is our confidence that n is prime? It is tempting to conclude that the probability that such an integer n is prime is $1 - 2^{-m}$. This conclusion is often stated in both textbooks and technical articles, but it cannot be inferred from the given data.

We need to be careful about our use of probabilities. Suppose we define the following random variables: **a** denotes the event

"a random odd integer n of a specified size is composite,"

and **b** denotes the event

"the algorithm answers 'n is prime' m times in succession."

It is certainly the case that the probability $\mathbf{Pr}[\mathbf{b}|\mathbf{a}] \leq 2^{-m}$. However, the probability that we are really interested is $\mathbf{Pr}[\mathbf{a}|\mathbf{b}]$, which is usually not the same as $\mathbf{Pr}[\mathbf{b}|\mathbf{a}]$.

We can compute $\mathbf{Pr}[\mathbf{a}|\mathbf{b}]$ using Bayes' theorem (Theorem 3.1). In order to do this, we need to know $\mathbf{Pr}[\mathbf{a}]$. Suppose $N \leq n \leq 2N$. Applying the Prime number theorem, the number of (odd) primes between N and $2N$ is approximately

$$\frac{2N}{\ln 2N} - \frac{N}{\ln N} \approx \frac{N}{\ln N} \approx \frac{n}{\ln n}.$$

There are $N/2 \approx n/2$ odd integers between N and $2N$, so we estimate

$$\mathbf{Pr}[\mathbf{a}] \approx 1 - \frac{2}{\ln n}.$$

Then we can compute as follows:

$$\begin{aligned}
\mathbf{Pr}[\mathbf{a}|\mathbf{b}] &= \frac{\mathbf{Pr}[\mathbf{b}|\mathbf{a}]\,\mathbf{Pr}[\mathbf{a}]}{\mathbf{Pr}[\mathbf{b}]} \\
&= \frac{\mathbf{Pr}[\mathbf{b}|\mathbf{a}]\,\mathbf{Pr}[\mathbf{a}]}{\mathbf{Pr}[\mathbf{b}|\mathbf{a}]\,\mathbf{Pr}[\mathbf{a}] + \mathbf{Pr}[\mathbf{b}|\overline{\mathbf{a}}]\,\mathbf{Pr}[\overline{\mathbf{a}}]} \\
&\approx \frac{\mathbf{Pr}[\mathbf{b}|\mathbf{a}]\,\left(1 - \frac{2}{\ln n}\right)}{\mathbf{Pr}[\mathbf{b}|\mathbf{a}]\,\left(1 - \frac{2}{\ln n}\right) + \frac{2}{\ln n}} \\
&= \frac{\mathbf{Pr}[\mathbf{b}|\mathbf{a}](\ln n - 2)}{\mathbf{Pr}[\mathbf{b}|\mathbf{a}](\ln n - 2) + 2} \\
&\leq \frac{2^{-m}(\ln n - 2)}{2^{-m}(\ln n - 2) + 2} \\
&= \frac{\ln n - 2}{\ln n - 2 + 2^{m+1}}.
\end{aligned}$$

Note that in this computation, $\overline{\mathbf{a}}$ denotes the event

"a random odd integer n is prime."

It is interesting to compare the two quantities $(\ln n - 2)/(\ln n - 2 + 2^{m+1})$ and 2^{-m} as a function of m. Suppose that $n \approx 2^{1024} \approx e^{710}$, since these are the sizes of primes p and q used to construct an RSA modulus. Then the first function is roughly $708/(708 + 2^{m+1})$. We tabulate the two functions for some values of m in Table 6.1.

Although $708/(708 + 2^{m+1})$ approaches zero exponentially quickly, it does not do so as quickly as 2^{-m}. In practice, however, one would take m to be something like 50 or 100, which will reduce the probability of error to a very small quantity.

6.4.3 The Miller-Rabin Algorithm

We now present another Monte Carlo algorithm for **Composites**, which is called the MILLER-RABIN ALGORITHM (this is also known as the ***strong pseudo-prime test***). This algorithm is presented as Algorithm 6.7.

Algorithm 6.7 is clearly a polynomial-time algorithm: an elementary analysis shows that its complexity is $O((\log n)^3)$, as is the SOLOVAY-STRASSEN ALGORITHM. In fact, the MILLER-RABIN ALGORITHM performs better in practice than the SOLOVAY-STRASSEN ALGORITHM.