Exercise 1.25. Alyssa P. Hacker complains that we went to a lot of extra work in writing `expmod`. After all, she says, since we already know how to compute exponentials, we could have simply written

```
(define (expmod base exp m)
  (remainder (fast-expt base exp) m))
```

Is she correct? Would this procedure serve as well for our fast prime tester? Explain.

Answer. In order to indentify the correctness of the algorithm given by Alyssa, we'd better test it by embedding it into the `search-for-primes` procedure as in exercise 1.22.

As you can see in Table 1, although Alyssa's algorithm is correct, but the time it requires to perform computation

Magnitude	Prime	Time (s) (Original Method)	Time (s) (Alyssa's Method)
10^3	1009	3.0000000000001137e-2	6.0000000000002274e-2
	1013	3.0000000000001137e-2	0.04999999999999716
	1019	3.0000000000001137e-2	6.0000000000002274e-2
10^4	10007	3.0000000000001137e-2	4.670000000000002
	10009	0.03999999999999915	4.609999999999999
	10037	0.02999999999999403	4.650000000000006
10^5	100003	0.03999999999999915	744.45
	100019	0.03999999999999915	746.82
	100043	0.03999999999999915	739.0699999999999

Table 1. Comparison of Performance between Two Versions of `expmod`

is horribly giant when compared with the original method. But why?

Well, our original `expmod` procedure constrains the complexity of computation by using a decomposition:

$$(x \cdot y) \bmod m = [(x \bmod m) \cdot (y \bmod m)] \bmod m$$

as is indicated by the footnote in section 1.2.6.[1] This strategy ensure that the computation involved to evaluate $(x \cdot y) \bmod m$ reaches $(x \bmod m) \cdot (y \bmod m) \leqslant m^2$ at most on every level of recursion, so that the number of steps grows $\Theta(\log n)$.

Whereas, in Alyssa's algorithm, the first operand in the body of `expmod` procedure accumulates as `fast-expt` been performed over and over again. Supposed that we perform the `expmod` operantion on an integer whose magnitude is 10^n, that is, we assume the value of `exp` here to be n. By the definition of `fermat-test`, the value of `base` will be picked randomly within the interval $[0,10^n]$ and we take $\frac{1}{2} \times 10^n$ on average for analyzing. Hence, the number genereted by procedure `fast-expt` turns out to be on a scale of:

$$\left(\frac{1}{2} \times 10^n\right)^n$$

Hence, the order of growth is $n\,\Theta(10^n)$. Thus, when the primitive operation `remainder` is finally applied, the number has become so giant that it takes an incredibly long time to work it out as the magnitude raises up.

---

1. The reduction steps in the cases where the exponent e is greater than 1 are based on the fact that, for any integers x, y and m, we can find the remainder of x times y modulo m by computing separately the remainders of x modulo m and y modulo m, multiplying these, and then taking the remainder of the result modulo m. For instance, in the case where e is even, we compute the remainder of $b^{e/2}$ modulo m, square this, and take the remainder modulo m. This technique is useful because it means we can perform our computation without ever having to deal with numbers much larger than m. (Compare exercise 1.25.)