SICP: Exercise 1.26

August 26, 2024

Louis Reasoner is having great difficulty doing Exercise 1.24. His `fast-prime?` *test seems to run more slowly than his* `prime?` *test. Louis calls his friend Eva Lu Ator over to help. When they examine Louis's code, they find that he has rewritten the expmod procedure to use an explicit multiplication, rather than calling square:*

```
(define (expmod base exp m)
  (cond ((= exp 0) 1)
        ((even? exp)
         (remainder
          (* (expmod base (/ exp 2) m)
             (expmod base (/ exp 2) m))
          m))
        (else
         (remainder
          (* base
             (expmod base (- exp 1) m))
          m))))
```

"I don't see what difference that could make," says Louis. "I do." says Eva. "By writing the procedure like that, you have transformed the $\Theta(\log n)$ process into a $\Theta(n)$ process."
Explain.

Due to applicative-order evaluation, the argument to `square` would have been evaluated before calling `square`. That way, the recursive `expmod` call only has to happen once. However, Louis' change causes this to happen *twice*, as each argument is evaluated by itself. This doubling propagates through the call tree—since the `else` clause is only true for the first call (if at all), all subsequent calls until the exponent reaches 0 will issue the recursive call twice (which again causes the callees to issue two more calls each, i.e., 4 total, etc.), effectively requiring 2^x calls (assuming the original `expmod` required x calls). Finally, because $\Theta(\log(2^n)) = \Theta(\log_2(2^n)) = \Theta(n)$, this has transformed the process from a logarithmic one to a linear one.