1.13

Given that $\Phi = \frac{1+\sqrt{5}}{2}$ and $\Psi = \frac{1-\sqrt{5}}{2}$, prove that $Fib(n) = \frac{\Phi^n - \Psi^n}{\sqrt{5}}$.

(Note: $\Phi^2 = (\frac{1+\sqrt{5}}{2})^2 = \frac{1+2\sqrt{5}+5}{4} = \frac{6+2\sqrt{5}}{4} = \frac{3+\sqrt{5}}{2}$ and, similarly, $\Psi^2 = \frac{3-\sqrt{5}}{2}$.)

Proof by induction:

Base case 0: $Fib(0) = \frac{\Phi^0 - \Psi^0}{\sqrt{5}} = \frac{1-1}{\sqrt{5}} = \frac{0}{\sqrt{5}} = 0$, and $Fib(0) = 0$ by definition ✔

Base case 1: $Fib(1) = \frac{\Phi^1 - \Psi^1}{\sqrt{5}} = \frac{\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2}}{\sqrt{5}} = \frac{\frac{2\sqrt{5}}{2}}{\sqrt{5}} = \frac{\sqrt{5}}{\sqrt{5}} = 1$, and $Fib(1) = 1$ by definition ✔

Assuming that $Fib(k) = \frac{\Phi^k - \Psi^k}{\sqrt{5}}$ and $Fib(k+1) = \frac{\Phi^{k+1} - \Psi^{k+1}}{\sqrt{5}}$, show that $Fib(k+2) = \frac{\Phi^{k+2} - \Psi^{k+2}}{\sqrt{5}}$.

	$Fib(k+2)$	
=	$Fib((k+2)-1) + Fib((k+2)-2)$	By definition of $Fib(n)$
=	$Fib(k+1) + Fib(k)$	
=	$\frac{\Phi^k - \Psi^k}{\sqrt{5}} + \frac{\Phi^{k+1} - \Psi^{k+1}}{\sqrt{5}}$	By induction hypothesis
=	$\frac{(\Phi^{k+1} + \Phi^k) - (\Psi^{k+1} + \Psi^k)}{\sqrt{5}}$	
=	$\frac{\Phi^k(\Phi+1) - \Psi^k(\Psi+1)}{\sqrt{5}}$	
=	$\frac{\Phi^k(\frac{1+\sqrt{5}}{2}+1) - \Psi^k(\frac{1-\sqrt{5}}{2}+1)}{\sqrt{5}}$	By definition of Φ and Ψ
=	$\frac{\Phi^k(\frac{3+\sqrt{5}}{2}) - \Psi^k(\frac{3-\sqrt{5}}{2})}{\sqrt{5}}$	
=	$\frac{\Phi^k \Phi^2 - \Psi^k \Psi^2}{\sqrt{5}}$	See "Note" above (and the text's definition of Φ as "the number that satisfies $\Phi^2 = \Phi + 1$")
=	$\frac{\Phi^{k+2} - \Psi^{k+2}}{\sqrt{5}}$	✔

To show that $Fib(n)$ is the closest integer to $\frac{\Phi^n}{\sqrt{5}}$, consider the difference between the two:

$\left| Fib(n) - \frac{\Phi^n}{\sqrt{5}} \right| = \left| \frac{\Phi^n - \Psi^n}{\sqrt{5}} - \frac{\Phi^n}{\sqrt{5}} \right| = \left| \frac{-\Psi^n}{\sqrt{5}} \right| = \left| \frac{(\frac{\sqrt{5}-1}{2})^n}{\sqrt{5}} \right| < \frac{0.62^n}{\sqrt{5}}$. This expression assumes its largest value, ~ 0.45, when $n = 0$ and, since this difference is less than 0.5, no other integer is closer to $\frac{\Phi^n}{\sqrt{5}}$ than $Fib(n)$. For all higher values of n, $Fib(n)$ is even closer to $\frac{\Phi^n}{\sqrt{5}}$.

So what?

As the text shows, a naïve tree-recursive $Fib()$ implementation has $O(c^n)$ (i.e. exponential) time complexity, and the better linear-iterative implementation is $O(n)$. Given that we can exponentiate in $O(\log n)$ time (see e.g. Exercise 1.16), we can find $\frac{\Phi^n}{\sqrt{5}}$ (and so also the integer nearest it, which is $Fib(n)$) in $O(\log n)$ time – a big improvement.

1.16

```
(define (even? n) (= (remainder n 2) 0))
(define (square n) (* n n))

(define (exp-iter a b n)
  (cond ((= n 0) a)
        ((even? n) (exp-iter a (square b) (/ n 2)))
        (else (exp-iter (* a b) b (- n 1)))))

(define (exp b n) (exp-iter 1 b n))
```

See how the process evolves for 2^5

a	**b**	**n**
1	2	5
2	2	4
2	4	2
2	16	1
32	16	0

and for 3^8

a	**b**	**n**
1	3	8
1	9	4
1	81	2
1	6561	1
6561	6561	0

1.17

```
(define (double n) (+ n n))
(define (even? n) (= (remainder n 2) 0))
(define (halve n) (/ n 2))

(define (show a b)
  (display (string-append
            "a=" (number->string a) " "
            "b=" (number->string b) "\n")))

(define (mult a b)
  (show a b)
  (cond ((= b 1) a)
        ((even? b) (mult (double a) (halve b)))
        (else (+ a (mult a (- b 1))))))

(let* ((args (cdr (command-line))))
  (display (mult (string->number (car args)) (string->number (car (cdr args)))))
  (newline))
```

```
% guile 1.17.scm
a=3 b=9
a=3 b=8
a=6 b=4
a=12 b=2
a=24 b=1
27

% for n in 1 10 100 1000 10000; do guile 1.17.scm 2 $n | wc -l; done
       2
       6
      10
      16
      19
```

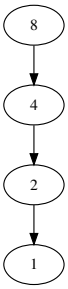
1.18

```
(define (double n) (* n 2))
(define (even? n) (= (remainder n 2) 0))
(define (halve n) (/ n 2))

(define (show x a b)
  (display (string-append
            "x=" (number->string x) " "
            "a=" (number->string a) " "
            "b=" (number->string b)
            "\n")))

(define (mult-iter x a b)
  (show x a b)
  (cond ((= b 0) x)
        ((even? b) (mult-iter x (double a) (halve b)))
        (else (mult-iter (+ x a) a (- b 1)))))

(define (mult a b) (mult-iter 0 a b))

(let* ((args (cdr (command-line))))
  (display (mult (string->number (car args)) (string->number (car (cdr args)))))
  (newline))
```

```
% guile 1.18.scm 3 9
x=0 a=3 b=9
x=3 a=3 b=8
x=3 a=6 b=4
x=3 a=12 b=2
x=3 a=24 b=1
x=27 a=24 b=0
27

% for n in 1 10 100 1000 10000; do guile 1.18.scm 2 $n | wc -l; done
       3
       7
      11
      17
      20
```

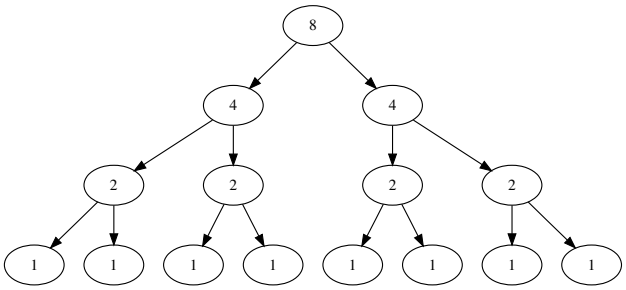
1.26

The original `expmod` function manages to perform only $\Theta(\log n)$ multiplications by making a single recursive call per iteration, halving the exponent each time until it equals 1. (It's slightly more complicated with odd numbers, but ignore that for now.) With $n = 8$, the exponent value changes as in this "tree":



One multiplication (via `square`) is performed by each parent node, so that the number of multiplications (a good proxy for the total number of operations) is proportional to $\log n$.

The modified `expmod` function, however, makes *two* recursive calls per iteration, producing a tree like this:



Like the original version, it performs one multiplication per parent node, but the number of parent nodes doubles at each level of the (binary) tree. Since the number of nodes in any binary tree is $2^k - 1$, where k is the depth of the tree, and since the subtree consisting only of parent nodes has depth $\log_2 n$, the number of parent nodes is $2^{\log_2 n} - 1$, or $n - 1$, which is $\Theta(n)$. So the number of operations is likewise $\Theta(n)$.