Exercise1.9

the first one

recursive

```
(define (+ a b)
(if (= a 0) b (inc (+ (dec a) b))))

(+ 4 5)
(inc (+ 3 5))
(inc (inc (+ 2 5)))
(inc (inc (inc (+ 1 5))))
(inc (inc (inc (inc (+ 0 5)))))
(inc (inc (inc (inc 5))))
(inc (inc (inc 6)))
(inc (inc 7))
(inc 8)
9
```

the second one

interative

```
(define (+ a b)
(if (= a 0) b (+ (dec a) (inc b))))

(+ 4 5)
(+ 3 6)
(+ 2 7)
(+ 1 8)
(+ 0 9)
9
```

Exercise1.10

```
(define (A x y)
(cond ((= y 0) 0)
((= x 0) (* 2 y))
((= y 1) 2)
(else (A (- x 1) (A x (- y 1))))))
(A 3 3)
```

(A 1 10)=2^{10} (A 2 4)=2^{16} (A 3 3)=2^{16}

```
(define (f n) (A 0 n))
(define (g n) (A 1 n))
(define (h n) (A 2 n))
(define (k n) (* 5 n n))
```

the first: $2n$ the second: 2^n, 0 for n=0 the third: 2^{2^n}, 0 for n=0

Exercise1.11

recursive

```
(define (f n)
  (if (< n 3)
      n
      (+ (f (- n 1)) (* 2 (f (- n 2))) (* 3 (f (- n 3))))))
(f 6)
```

iterative, c is the value to return

```
(define (f n)
  (f-iter 0 1 2 n))
(define (f-iter a b c n)
  (if (< n 3)
      c
      (f-iter b c (+ (* 3 a) (* 2 b) c) (- n 1))))
(f 6)
```

Exercise1.12

```
(define (pascal row col)
  (cond ((= col 1) 1)
        ((= row col) 1)
        (else (+ (pascal (- row 1) (- col 1)) (pascal (- row 1) col))))
    )

(pascal 5 3)
```

Exercise1.13

1. it is true for n = 0, 1

   $fib(0) = (\varphi^0 - \psi^0)/\sqrt{5} = 0$

$fib(1) = (\varphi^1 - \psi^1)/\sqrt{5} = 1$

2. assume it is true for n < k $fib(k) = (\varphi^k - \psi^k)/\sqrt{5}$
3. prove it is true for n = k + 1

$$\begin{aligned} fib(k+1) &= fib(k) + fib(k-1) \\ &= (\varphi^k - \psi^k)/\sqrt{5} + (\varphi^{k-1} - \psi^{k-1})/\sqrt{5} \\ &= \frac{(\varphi+1)(\varphi^{k-1}) - (\psi+1)(\psi^{k-1})}{\sqrt{5}} \\ &= \frac{(\varphi^2)(\varphi^{k-1}) - (\psi^2)(\psi^{k-1})}{\sqrt{5}} \\ &= \frac{\varphi^{k+1}) - \psi^{k+1})}{\sqrt{5}} \end{aligned}$$

so $fib(n) = (\varphi^n - \psi^n)/\sqrt{5}$ note that $0 < \psi < 1$ so $\psi^n/\sqrt{5} < 0.5$ fib(n) is the closest integer to $\varphi^n/\sqrt{5}$

Exercise1.14

(11 5) (11 4) (-39 5)=0 (11 3) (-14 3)=0 (11 2) (1 3) (11 1) (6 2) (1 2) (-9 3)=0 (11 0)=0 (10 1) (6 1) (1 2) (1 1) (-4 2)=0 (10 0)=0 (9 1) (6 0)=0 (5 1) (1 1) (-4 2)=0 (1 0)=0 (0 1)=1 … (0 1)=1 (0 1)=1 (0 1)=1

4 ways of count-change

Exercise1.15

1. 5 times
2. $O(log_3(10n))$

Exercise1.16

```
(define (even? n) (= (remainder n 2) 0))
(define (square n) (* n n))
(define (exp b n)
    (fast-exp 1 b n))
(define (fast-exp a b n)
```

```
    (cond ((= n 0)
           a)
          ((even? n)
           (fast-exp a (square b) (/ n 2)))
          (else (fast-exp (* a b) b (- n 1))))))
(exp 2 12)
```

Exercise1.17 and Exercise1.18

step: O(log(n)) space: O(1)

```
(define double (lambda (a) (+ a a)))

(define (even? n) (= (remainder n 2) 0))

(define halve (lambda (a) (/ a 2)))


(define (* b n)
   (fast* 0 b n))

(define (fast* a b n)
  (cond ((= n 0)
     a)
    ((even? n)
     (fast* a (double b) (halve n)))
    (else (fast* (+ a b) b (- n 1)))))

(* 3 8)
```

Exercise1.19

Transform matrix

```
(define (fib n)
  (fib-iter 1 0 0 1 n))
(define square (lambda (x) (* x x)))
(define (fib-iter a b p q count)
(cond ((= count 0) b)
      ((even? count)
       (fib-iter a
             b
             (+ (square q) (square p))
                 (+ (* 2 (* q p)) (square q))
             (/ count 2)))
```

```
      (else (fib-iter (+ (* b q) (* a q) (* a p))
              (+ (* b p) (* a q))
              p
              q
              (- count 1)))))
(fib 14)
```

Exercise1.20

```
(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (remainder a b))))
```

normal order: fully expand and then reduce, use remainder 1+2+4+7+2+1+1=18 times

```
(gcd 206 40)
(gcd 40 (remainder 206 40))
(gcd (remainder 206 40) (remainder 40 (remainder 206 40))) ; remainder
   1 in if
(gcd (remainder 40 (remainder 206 40)) (remainder (remainder 206 40) (
   remainder 40 (remainder 206 40)))) ; remainder 2 in if
(gcd (remainder (remainder 206 40) (remainder 40 (remainder 206 40))) (
   remainder (remainder 40 (remainder 206 40)) (remainder (remainder
   206 40) (remainder 40 (remainder 206 40))))) ;remainder 4 in if
(remainder (remainder 206 40) (remainder 40 (remainder 206 40))) ;
   remainder 7 in if
(remainder 6 (remainder 40 6)) ;remainder 2
(remainder 6 4) ;remainder 1
2 ;remainder 1
```

applicative order: evaluate the arguments and then apply, use remainder operation four times

```
(gcd 206 40)
(gcd 40 6)
(gcd 6 4)
(gcd 4 2)
(gcd 2 0)
```

Exercise1.21

```
(define square (lambda (x) (* x x)))
(define (smallest-divisor n) (find-divisor n 2))
(define (find-divisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
```

```
    ((divides? test-divisor n) test-divisor)
    (else (find-divisor n (+ test-divisor 1)))))
(define (divides? a b) (= (remainder b a) 0))
(smallest-divisor 19999)
```

Exercise1.22

$\sqrt{10}$

```
(define square (lambda (x) (* x x)))
(define (prime? n)
  (= n (smallest-divisor n)))
(define (smallest-divisor n) (find-divisor n 2))
(define (find-divisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
    ((divides? test-divisor n) test-divisor)
    (else (find-divisor n (+ test-divisor 1)))))
(define (divides? a b) (= (remainder b a) 0))
(define (even? n)
  (= (remainder n 2) 0))
(define (timed-prime-test n)
  ;; (newline)
  ;; (display n)
  (start-prime-test n (runtime)))
(define (start-prime-test n start-time)
  (if (prime? n)
      (report-prime n (- (runtime) start-time))))
(define (report-prime n elapsed-time)
  (newline)
  (display n)
  (display " *** ")
  (display elapsed-time))
(define (search-for-primes start end)
  (cond ((even? start) (search-for-primes (+ start 1) end))
    ((<= start end) (timed-prime-test start) (search-for-primes (+
        start 2) end))))

(search-for-primes 1000000000 1000000021)       ; 0.02
(search-for-primes 10000000000 10000000061)     ; 0.06
(search-for-primes 100000000000 100000000057)   ; 0.21
(search-for-primes 1000000000000 1000000000063) ; 0.54
```

Exercise1.23

```
(define square (lambda (x) (* x x)))
```

```
(define (smallest-divisor n) (find-divisor n 2))
(define (find-divisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
    ((divides? test-divisor n) test-divisor)
    (else (find-divisor n (next test-divisor)))))
(define (divides? a b) (= (remainder b a) 0))
(define next (lambda (x) (cond ((= x 2) 3)
                    (else (+ x 2)))))
(smallest-divisor 19999)
```

Exercise1.24

```
(define square (lambda (x) (* x x)))
(define (expmod base exp m)
  (cond ((= exp 0) 1)
    ((even? exp)
     (remainder
      (square (expmod base (/ exp 2) m))
      m))
    (else
     (remainder
      (* base (expmod base (- exp 1) m))
      m))))
(define (fermat-test n)
  (define (try-it a)
    (= (expmod a n n) a))
  (try-it (+ 1 (random (- n 1)))))
(define (fast-prime? n times)
  (cond ((= times 0) true)
    ((fermat-test n) (fast-prime? n (- times 1)))
    (else false)))
(define (timed-prime-test n)
  ;; (newline)
  ;; (display n)
  (start-prime-test n (runtime)))
(define (start-prime-test n start-time)
  (if (fast-prime? n 2)
      (report-prime n (- (runtime) start-time))))
(define (report-prime n elapsed-time)
  (newline)
  (display n)
  (display " *** ")
  (display elapsed-time))
 (newline)
 (timed-prime-test 1000000007)
 (timed-prime-test 1000000009)
 (timed-prime-test 1000000021)
 (timed-prime-test 10000000019)
```

```
 (timed-prime-test 10000000033)
 (timed-prime-test 10000000061)
 (timed-prime-test 100000000003)
 (timed-prime-test 100000000019)
 (timed-prime-test 100000000057)
 (timed-prime-test 1000000000039)
 (timed-prime-test 1000000000061)
(timed-prime-test 1000000000063)
(timed-prime-test 1000000000063)
```

Exercise1.25

```
(define (square x) (* x x))
(define (even? n)
(= (remainder n 2) 0))
(define (fast-expt b n)
(cond ((= n 0) 1)
((even? n) (square (fast-expt b (/ n 2))))
(else (* b (fast-expt b (- n 1))))))
(define (expmod base exp m)
  (remainder (fast-expt base exp) m))
(expmod 2 4 3)
```

Exercise1.26

do it better when use n (* (expmod base n m) (expmod base n m)) take k step (* (expmod base 2n m) (expmod base 2n m)) (* (* (expmod base n m) (expmod base n m)) (* (expmod base n m) (expmod base n m))) take 2k step

Exercise1.27

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

```
(define (carmi n)
  (carmichael n 1))
(define (carmichael n count)
  (cond ((= n count) true)
    ((= (expmod count n n) count) (carmichael n (+ count 1)))
    (else false)))
(carmi 561) ; t
(carmi 1105) ; t
(carmi 1729) ; t
(carmi 12454) ; f
```

Exercise 1.28

Miller-Rabin test

```
(define (square x)
  (* x x))
(define (miller-rabin-square-remainder x y)
  (cond ((and (= (remainder (square x) y) 1) (not (= x 1)) (not (= x (-
      y 1)))) 0)
    (else (remainder (square x) y))))
(define (expmod base exp m)
  (cond ((= exp 0) 1)
    ((even? exp)
     (miller-rabin-square-remainder
      (expmod base (/ exp 2) m)
      m))
    (else
     (remainder
      (* base (expmod base (- exp 1) m))
      m))))
(define (miller-rabin-test n)
  (define (try-it a)
    (define (true-or-false x)
      (cond ((= x 0) false)
        ((= x 1) true)))
    (true-or-false (expmod a (- n 1) n)))
  (try-it (+ 1 (random (- n 1)))))
(define (fast-prime? n times)
  (cond ((= times 0) true)
    ((miller-rabin-test n) (fast-prime? n (- times 1)))
    (else false)))
(fast-prime? 561 2)
```