

## Solutions to Quiz 1

### *Problem 1*

15  
9  
21  
13  
5

### *Problem 2*

Note: 0 is not an even number. However, you were not penalized for missing this case.

```
(define (even? x)
  (cond ((= x 0) #f)
        ((= (remainder x 2) 0) #t)
        (else #f)))
```

Somewhat more elegantly:

```
(define (even? x)
  (or (= x 0) (= (remainder x 2) 0)))
```

### *Problem 3*

```
(caddar a) or (car (cdr (cdr (car a))))
```

```
(cadr b) or (car (cdr b))
```

**Problem 4**

2  
4  
4

**Problem 5**

- Fully expand, then reduce. Don't evaluate any arguments until they are needed.
- Evaluate all arguments, then apply the first to the rest. This is what Scheme does.
- 6 multiplications (4 for arguments, 1 in each of the 2 calls to the square procedure)
- 4 multiplications (2 for arguments, 1 in each of the 2 calls to the square procedure)
- Some reasons: You don't need to worry about evaluating errors in conditional statements. You don't need to have special forms to create special evaluation rules. You won't evaluate any arguments unless they're needed.
- Some reasons: You are likely to perform fewer computations, since each argument is evaluated only once.

**Problem 6**

```
(define (repeat2 f)
  (lambda (x) (f (f x))))
```

**Problem 7**

```
(define (car x) (x 0))
(define (cdr x) (x 1))
```

**Problem 8**

$T(n)=\Theta(n)$

$S(n)=\Theta(n)$

N is dependent upon b-a

Recursive process

```
(define (prod-of-squares a b)
  (prod square
    a
    (lambda (x) (+ x 1))
    b))
```

```
(define (fact n)
  (prod (lambda (x) x)
    1
    (lambda (x) (+ x 1))
    n))
```

```
(define (prod term a next b)
  (define (prod-iter a ans)
    (if (> a b)
        ans
        (prod-iter (next a) (* ans (term a)))))
  (prod-iter a 1))
```