1. (10 pts) The following code was supposed to return a person’s BMI (body mass index) category, using the formula \( \text{BMI} = \frac{\text{weight} \times 703}{\text{height}^2} \), where weight is in pounds and height is in inches, and the table on the right. But the code has many errors. Circle each error, and say briefly what’s wrong. On the right, write the correct code.

```scheme
(define (bmi wgt hgt)
  (bmi-cat
   703 * wgt / hgt * hgt))

(define (bmi-cat x)
  (cond
   (x < 30) "overweight"
   (x < 25) "normal"
   (x < 18.5) "underweight"
   (else "obese")))
```

1. (10 pts) What does \( \text{fn1} \) in the code below return for the following input values?

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(fn1 (list 1 2 3 4))</td>
<td>(list 4 3 2 1)</td>
</tr>
<tr>
<td>(fn1 (list 4 3 2 1))</td>
<td>(list 1 2 3 4)</td>
</tr>
<tr>
<td>(fn1 (list (list 1 2) (list 3 4)))</td>
<td>(list (list 3 4) (list 1 2))</td>
</tr>
<tr>
<td>(fn1 (list 1))</td>
<td>(list 1)</td>
</tr>
<tr>
<td>(fn1 empty)</td>
<td>empty</td>
</tr>
</tbody>
</table>

Describe what \( \text{fn1} \) returns in general: reverses top-level of list, not lists inside list

```scheme
(define (fn1 l)
  (fn2 l empty))

(define (fn2 l1 l2)
  (if (empty? l1) l2
      (fn2 (cdr l1) (cons (car l1) l2))))
```
3. (10 pts) What does \( \text{fn1} \) in the code below return for the following input values?

| \( \text{fn1 (list 1 2 3)} \) | \( \text{(list -1 0 1)} \) |
| \( \text{fn1 (list 1)} \) | \( \text{(list 0)} \) |
| \( \text{fn1 (list 4 3 2 1)} \) | \( \text{(list 1.5 0.5 -0.5 -1.5)} \) |
| \( \text{fn1 empty} \) | \( \text{error, division by zero} \) |

Describe what \( \text{fn1} \) returns in general: calculates the average of the numbers in a list and returns a list of \( x - \text{average} \), for each element \( x \) in list; not defined for empty list

\[
\text{(define (fn1 l)}
\text{ (fn2 l (/ (fn3 l) (length l)))))}
\]

\[
\text{(define (fn2 l x)}
\text{ (if (empty? l) empty (cons (- (car l) x) (fn2 (cdr l) x)))))}
\]

\[
\text{(define (fn3 l)}
\text{ (if (empty? l) 0 (+ (car l) (fn3 (cdr l))))})
\]

4. (10 pts) Define a function \( \text{(next-pow m n)} \) for non-negative integers \( m \) and \( n \) to return the smallest \( k \) such that \( m^k \geq n \). Feel free to define helper functions. Some test cases:

\[
\text{(check-expect (next-pow 2 8) 3)}
\]
\[
\text{(check-expect (next-pow 2 9) 4)}
\]
\[
\text{(check-expect (next-pow 2 16) 4)}
\]
\[
\text{(check-expect (next-pow 2 1) 0)}
\]
\[
\text{(check-expect (next-pow 3 1) 0)}
\]
\[
\text{(check-expect (next-pow 3 27) 3)}
\]
\[
\text{(check-expect (next-pow 3 80) 4)}
\]

\[
\text{(define (next-pow m n)}
\text{ (if (<= n 1) 0 (add1 (next-pow m (/ n m))))})
\]

or

\[
\text{(define (next-pow m n)}
\text{ (next-pow-iter m n 0))}
\]

\[
\text{(define (next-pow-iter m n k)}
\text{ (if (<= n (expt m k)) k (next-pow-iter m n (add1 k))))}
\]

\[\text{Comment [CKR8]: Common mistake: saying \"empty\".}\]

\[\text{Comment [CKR9]: this is both simpler and more efficient.}\]
5. (10 pts) Define a function \((\text{merge } \text{lst1 } \text{lst2})\) to return the sorted merger of two sorted lists of numbers, including duplicates. Some test cases:

\[
\begin{align*}
(\text{check-expect } (\text{merge } \text{empty } \text{empty}) & \text{ empty}) \\
(\text{check-expect } (\text{merge } (\text{list } 1 2 3) \text{ empty}) & \text{ (list } 1 2 3)) \\
(\text{check-expect } (\text{merge } \text{empty } (\text{list } 1 2 3)) & \text{ (list } 1 2 3)) \\
(\text{check-expect } (\text{merge } (\text{list } 1 2 2 8) \text{ (list } 1 1 5)) & \text{ (list } 1 1 1 2 2 5 8)) \\
\end{align*}
\]

\[
\begin{align*}
(\text{define } & (\text{merge } \text{lst1 } \text{lst2}) \\
(\text{cond } & ((\text{empty? } \text{lst1}) \text{ lst2}) \\
(\text{empty? } \text{lst2}) & \text{lst1}) \\
((< (\text{car } \text{lst1}) (\text{car } \text{lst2})) \\
(\text{cons } (\text{car } \text{lst1}) & (\text{merge } (\text{cdr } \text{lst1}) \text{lst2})))) \\
\text{else} & (\text{cons } (\text{car } \text{lst2}) & (\text{merge } \text{lst1 } (\text{cdr } \text{lst2}))))))
\end{align*}
\]

The following is less efficient, needing more comparisons and more consing.

\[
(\text{define } (\text{merge } \text{lst1 } \text{lst2}) \\
(\text{sort } (\text{append } \text{lst1 } \text{lst2}) \text{ <}))
\]

Comment [CKR10]: Common mistake: defining \text{merge}, with tests for empty, and a helper that looped and didn’t test for empty. This breaks.

Comment [CKR11]: putting an \((\text{and } (\text{empty? } \text{lst1}) (\text{empty? } \text{lst2})) \text{ empty})\) branch before this branch is redundant.

Comment [CKR12]: putting an \((\text{and } (\text{empty? } \text{lst1}) (\text{empty? } \text{lst2})) \text{ empty})\) branch after this branch is worse because it can never be executed.

Comment [CKR13]: <= is not really needed.

Comment [CKR14]: \((< (\text{car } \text{lst2}) (\text{car } \text{lst1}))\) is redundant.

Comment [CKR15]: \text{sort} requires a predicate to determine the ordering.