Eighth Quiz

You have 15 minutes from the start of class to complete this quiz. Read the questions with care; work with deliberate speed. Don’t give us more than we ask for. The usual instructions apply. Good luck!

Problem 1 (5 points)

For each of the algorithms or operations described below, check the box corresponding most closely to its complexity (i.e., its O-notation).

(a) \((\text{map } f \ (\text{filter } p? \ L))\) where \(f\) and \(p?\) are functions and \(L\) is a list of \(n\) items.
   - [ ] Constant—O(1)
   - [ ] Logarithmic—O(\(\log n\))
   - [ ] Linear—O(\(n\))
   - [ ] Quadratic—O(\(n^2\))

(b) Print out each element in a binary search tree containing \(n\) items, in order.
   - [ ] Constant—O(1)
   - [ ] Logarithmic—O(\(\log n\))
   - [ ] Linear—O(\(n\))
   - [ ] Quadratic—O(\(n^2\))

(c) \((\text{vector-ref } V \ N)\) where \(V\) is a vector and \(N\) is a number between 0 and \((\text{vector-length } V)\).
   - [ ] Constant—O(1)
   - [ ] Logarithmic—O(\(\log n\))
   - [ ] Linear—O(\(n\))
   - [ ] Quadratic—O(\(n^2\))

(d) Binary search for an item in an ordered vector containing \(n\) items.
   - [ ] Constant—O(1)
   - [ ] Logarithmic—O(\(\log n\))
   - [ ] Linear—O(\(n\))
   - [ ] Quadratic—O(\(n^2\))

(e) Search for an element in a (balanced) binary search tree containing \(n\) items.
   - [ ] Constant—O(1)
   - [ ] Logarithmic—O(\(\log n\))
   - [ ] Linear—O(\(n\))
   - [ ] Quadratic—O(\(n^2\))

Problem 2 (20 points)

Suppose a course requires five lab projects, each of which gets a score from 0 to 100. Given the definition below, \((\text{get-scores } 5)\) creates a vector containing the five scores for one student; it reads each of the five scores (e.g., from an input file).

\[
\text{;; get-scores: number -> vector of number}
\]
\[
\text{;; Return a vector whose size is specified by the input number; fill the vector}
\]
\[
\text{;; with that number of scores from the input. (Note that \(\text{read}\) will read one}
\]
\[
\text{;; expression, e.g., one number, so this reads the next \(n\) numbers into the vector.)}
\]
\[
\text{(define get-scores}
\]
\[
\text{\quad (lambda (number-of-assts)
\text{\quad \quad (build-vector number-of-assts (lambda(i) (read))))})
\]

(a) (2 points) If we have \((\text{define myscores (get-scores 15)})\), write a Scheme expression whose value is the 12th score in myscores. (Don’t forget zero-based indexing.)

(b) (2 points) Suppose you already have the function \(\text{vector-sum}\) that takes a vector of numbers and returns the sum of the values in the vector. Write a Scheme expression whose value is the average score in myscores (assuming that all the scores are weighted equally).
(c) (2 points) Define the function `vector-average` as specified below. You may call `vector-sum` as needed. This is nearly the same as part (b).

```scheme
;; vector-average: vector-of-numbers -> number
;; Input is a vector of numbers (of any length); return average of values in vector
(define vector-average (lambda (v) (/ (vector-sum v) (vector-length v))))
```

(d) (4 points) Complete the definition of `vector-sum` below.

```scheme
;; vector-sum: vector-of-numbers -> number
(define vector-sum
  (lambda (v)
    (vector-sum-aux v 0 (sub1 (vector-length v))))
)

(define vector-sum-aux
  (lambda (v start end)
    (cond
      ((or equal? start end) (vector-ref v end))
      (else (+ (vector-ref v ____________) start
                  (vector-sum-aux v (add1 ____________) __________))))))
```

(e) (2 points) If these scores were in a conventional Scheme list instead of a vector, we could compute their sum with the one-line expression `(reduce 0 + my-score-list)`. Give one good reason (in one short sentence) why we might choose to use vectors instead; in other words, what’s a situation where some advantage of vectors over lists would be useful?

(f) (3 points) Complete the definition below of the function `build-gradebook`.

```scheme
;; build-gradebook: number number -> vector (of vectors)
;; Return a vector containing score vectors created by get-scores; the first argument
;; is the number of rows (i.e., the number of students); the second is the
;; number of columns (i.e., the number of scores for each student).
(define build-gradebook
  (lambda (num-students num-scores)
    (build-vector ____________ (lambda(i) (____________  ____________)))))
```

(g) (2 points) If we have `define classgrades (build-gradebook 50 15)`, write a Scheme expression whose value is the 35th student’s score on the 5th assignment.

```
(vector-ref (vector-ref classgrades 34) 4) — no double jeopardy for missing zero-based indexing.
```

[It didn’t quite fit on one piece of paper this week; sorry! But there’s just one more part.]
Write a contract and purpose statement for the function below, giving it a better name than just \( f \). Your purpose statement should describe clearly and precisely what the arguments mean and what the function does in terms of the arguments. [Don’t just say something like, “It divides total of \( v \), num, zero, and something by vector-length ….”]

```
;; ____________________:

;;
;;
;;
;;
;;
;;
;;
;;
(define f
  (lambda (v num)
    (local ((define total
                (lambda (v num current last)
                  (cond
                    ((= current last) (vector-ref (vector-ref v last) num))
                    (else (+ (vector-ref (vector-ref v current) num)
                             (total v num (add1 current) last))))))
     (/ (total v num 0 (sub1 (vector-length v)))
        (vector-length v))))
```