

# NINTH 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** (8 points)

Below are three versions of a function that takes a list of numbers and returns the sum of the items on the list:

```
(define sum1
  (lambda (L)
    (foldr + 0 L)))
```

```
(define sum2
  (lambda (L)
    (cond
      ((empty? L) 0)
      (else (+ (first L) (sum2 (rest L)))))))
```

```
(define sum3
  (lambda (L)
    (sum3-acc L 0)))
```

```
(define sum3-acc
  (lambda (L accumulated-sum)
    (cond
      ((empty? L) accumulated-sum)
      (else (sum3-acc (rest L) (+ (first L) accumulated-sum))))))
```

Which of the following are accurate statements about the functions above? (Choose *one or more* of the following.)

- A. The execution times of `sum1`, `sum2`, and `sum3` are all  $O(\log n)$  (for a list of  $n$  items).
- B. The tail-recursive function(s) require constant ( $O(1)$ ) space on the call stack.
- C. `sum2` requires  $O(n)$  space on the call stack.
- D. From the code given, we can't tell the space efficiency of `sum1` because we don't know how `foldr` is implemented.
- E. `sum2` and `sum3-acc` are accumulator-style functions.
- F. `sum3` (with its helper function `sum3-acc`) is more space-efficient than `sum2`.
- G. The execution times of `sum2` and `sum3` are both  $O(n \log n)$ .
- H. If the last line of `sum3-acc` were `(else (+ (sum3-acc (rest L) accumulated-sum) (first L))))`, it would return the same results and have the same efficiency.

**Problem 2** (4 points)

There are four functions defined above. Mark each function as either:

- TR, if the function is tail recursive
- RO, if the function is recursive (only) and not tail recursive
- N, if the function is not recursive at all

**Problem 3** (8 points)

(a) What is redundant information (in just a couple of words) and how does it relate to data compression (in just a few more words)?

(b) Give two examples of redundancy in real life. Only one of them has to be redundant *information*.

(c) There are two categories of data compression: lossless (where the compressed information can be expanded or reconstructed perfectly, with no loss of information) and lossy (where some of the information is thrown away, so the expanded/reconstructed information is not identical to the original). Why would anybody ever want to use lossy data compression?

(d) Below are listed four kinds of data. Some of them are possible candidates for lossy compression; others are not. Circle “could use lossy compression” or “would not use lossy compression” as appropriate.

- A photograph.     COULD USE LOSSY COMPRESSION /  WOULD NOT USE LOSSY COMPRESSION
- Scheme code.     COULD USE LOSSY COMPRESSION /  WOULD NOT USE LOSSY COMPRESSION
- A spreadsheet.     COULD USE LOSSY COMPRESSION /  WOULD NOT USE LOSSY COMPRESSION
- An audio recording.     COULD USE LOSSY COMPRESSION /  WOULD NOT USE LOSSY COMPRESSION