Informatics 41 • Fall 2010 • David G. Kay • UC Irvine • Your name ____________________

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 (6 points)

(a) (4 points) For each of the following expressions, draw a five-cell box showing the contents of the vector and the element number (i.e., the index or the subscript) of each element. (Recall that `identity` is a predefined function that returns its argument unchanged; it's equivalent to `(lambda (x) x)`.)

```
(vector "Huey" "Dewey" "Louie" "Daisy" "Donald")
```

```
(build-vector 5 identity)
```

```
(build-vector 5 (lambda (n) (+ 2 n)))
```

(b) (2 points) Suppose we have this definition: `(define V (vector 10 20 30 40 50))`. What is the value of each of these expressions?

```
(vector-ref V 3)

(vector-ref V 0)

(vector-length V)

(vector? V)
```

Problem 2 (3 points)

In class a few weeks ago, I did a little demonstration where a student picked a single word in the textbook, didn't tell me where it was, and I located it in fewer than 20 yes-or-no questions. I did it using the binary search algorithm: I started by opening the book to the middle and asking, "Is the word before this point or after?" Then I kept going along those lines until I'd narrowed it down to the chosen word.

(a) What's the O-notation for the binary search algorithm if you're searching through *n* items?

(b) If I could find the word in a 500-page book with 20 questions, about how long a book (in pages) could I search (using binary search) with 21 questions?

(c) If I can search a list of *n* items in 15 seconds using the linear search algorithm (not binary search), about how long (in seconds) will it take me to search a list that has twice as many (2*n*) items?

Problem 3 (5 points)

(a) (2 points) Suppose you have an old digital camera that takes only black-and-white pictures (technically, "gray-scale" or "monochrome" pictures, with pixels that can be black, white, or various shades of gray). If the camera uses 4 bits for each pixel, how many different shades of gray can it accommodate (in each pixel, including pure black and pure white)?

(b) (3 points) If the same camera captures an image that's 200 pixels wide and 150 pixels high, how many bytes (not bits) of storage does each image require (assuming no compression)? (Showing your work will help us give you partial credit.)

Problem 4 (6 points)

Suppose we have a binary search tree with nodes defined as usual:

```
      (define-struct treenode (rootvalue leftsubtree rightsubtree))
```

(a) (1 point) Fill in the blanks below to complete this definition:

```
;; BST->list:  BST  ->  list
;; Create a list from the contents of the BST, using inorder traversal
(define BST->list
  (lambda (T)
    (cond ((empty? T) _______________)

          (else (_______________
                 (BST->list (treenode-leftsubtree T))
                 (list (treenode-rootvalue T))
                 (BST->list (treenode-rightsubtree T)))))))
```

(b) (1 point) Fill in the blanks below to complete this definition. The predefined function `identity` just returns its argument unchanged.

```
;; BST-sum:  BST-of-number  ->  number
;; Return the sum of all the root values in the BST
(define BST-sum
  (lambda (T)
    (cond ((empty? T) _______________)

          (else (_______________
                 (BST-sum (treenode-leftsubtree T))
                 (identity (treenode-rootvalue T))
                 (BST-sum (treenode-rightsubtree T)))))))
```

(c) Since these two functions use the same algorithm, we can refactor them into one generalized traversal function by adding three parameters for the three places where the above two functions are different:

```
;; BST-traverse:  BST  X  (X X X -> X)  (Y -> X)  ->  X
;; Traverse the BST in order, using the second argument as the value for an empty
;; treenode, the third argument to combine values from the subtrees and the root, and
;; the fourth argument to convert a root value into the right type to be combined.
(define BST-traverse
  (lambda (T emptyvalue combine convert)
    (cond
      ((empty? T) emptyvalue)
      (else (combine
             (BST-traverse (treenode-leftsubtree T) emptyvalue combine convert)
             (convert (treenode-rootvalue T))
             (BST-traverse (treenode-rightsubtree T) emptyvalue combine convert))))))
```

(c.1) (2 points) Suppose we have a BST of strings called `names`. Fill in the blanks below to call `BST-traverse` to produce the equivalent of `(BST->list names)`.

```
(BST-traverse _______________  _______________  _______________  _______________  )
```

(c.2) (2 points) Suppose we have a BST of numbers called `figures`. Fill in the blanks below to call `BST-traverse` to produce the equivalent of `(BST-sum figures)`.

```
(BST-traverse _______________  _______________  _______________  _______________  )
```