Closed book and closed notes. No electronic devices. Show ALL work you want graded on the test itself. There are 6 problems, worth 10 points each.

For problems that do not ask you to justify the answer, an answer alone is sufficient. However, if the answer is wrong and no derivation or supporting reasoning is given, there will be no partial credit.

GOOD LUCK!
1.(a) (5 points)
Write a Racket procedure (count-all item lst)
that takes a list lst and returns the number of
top-level elements of lst that are equal? to item.

Use only the Racket procedures and special forms:
+, define, lambda, null?, empty?, car, first, cdr, rest,
cons, append, list, list?, equal?, if, cond, integer constants,
and the quoted empty list '().

Do not define any procedures other than count-all.

Examples:
(count-all 4 '(4 5 3 4 3 0)) => 3
(count-all "hi!" '("hello" "greetings")) => 0
(count-all 'a '(a l a s (a n d))) => 2
(count-all '(0) '(1 (0) 1 (1 (0)) (0))) => 2

(define (count-all item lst)
  (cond
   [(null? lst) 0]
   [(equal? item (first lst))
    (+ 1 (count-all item (rest lst)))]
   [else
    (count-all item (rest lst))])))
1.(b) (4 points)
Draw a tree of recursive calls, with return values, for
the application (sums 0 '(2 3 8 4)), where the procedure
sums is defined as follows.

```
(define (sums n lst)
  (cond
    [(null? lst)
      '()]
    [(null? (rest lst))
      (list (+ n (first lst)))]
    [else
      (cons (+ n (first lst))
        (sums (+ n (first lst))
          (rest lst))))]))
```

```
(sums 0 '(2 3 8 4)) => '(2 5 13 17)
  /  \
 cons 2 (sums 2 '(3 8 4)) => '(5 13 17)
    /      \
   cons 5 (sums 5 '(8 4)) => '(13 17)
     /         \
    cons 13 (sums 13 '(4)) => '(17)
```

1.(c) (1 point)
Is the procedure sums in part 1(b) tail recursive?
Why or why not?

No, because a value is cons'ed onto the result of the recursive call before it is returned as the result of the procedure.
2.(a) (5 points)
Consider the Boolean function f presented by the following truth table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>f(x,y,z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Write down a Boolean expression to represent f(x,y,z) using the variables x, y, z.
(Please use dot for AND, plus for OR and prime for NOT.)

Justify the correctness of your expression.

\[ x'\cdot y'\cdot z' + x\cdot y'\cdot z + x\cdot y\cdot z' \]

by the sum-of-products algorithm.

2.(b) (5 points)
Give a Boolean expression (using AND, OR, NOT as in part 2(a)) corresponding to each of the following statements involving the Boolean variables P, Q, and R.

You DO NOT need to justify the correctness of your answers.

(i) At least two of P, Q, R are 0.

\[ P'\cdot Q' + P'\cdot R' + Q'\cdot R' \]
2. (b) continued.

(ii) Exactly two of P, Q, and R are 1.

\[ P \cdot Q \cdot R' + P \cdot Q' \cdot R + P' \cdot Q \cdot R \]

(iii) P, Q and R do not all have the same value.

\[ (P \cdot Q \cdot R + P' \cdot Q' \cdot R')' \]

(iv) P is true or the values of Q and R are different.

\[ P + Q \cdot R' + Q' \cdot R \]

(v) If we write out P, Q, R in left-to-right order, the binary number represented is one of the numbers 2, 3, 5, or 7.

\[ P' \cdot Q \cdot R' + P' \cdot Q \cdot R + P \cdot Q' \cdot R + P \cdot Q \cdot R \]
3.(a) (6 points)
Define a recursive procedure \(\text{deep-remove item lst}\) that takes
an item and a list \(\text{lst}\) and returns a list that is \(\text{lst}\) with every
occurrence of an element equal? to item removed, no matter
how deeply nested.

Use only the Racket procedures and special forms: define, lambda, if,
cond, equal?, null?, empty?, car, cdr, first, rest, cons, append,
list, list? and the quoted empty list 
'(())

Do not define any procedures other than \text{deep-remove}.

Examples:
(deep-remove 3 '()) => '()
(deep-remove 3 '(4 5 3 4 4 3 0)) => '(4 5 4 4 0)
(deep-remove "no" '("no" "yes" "maybe" "no")+)) => '("yes" "maybe")
(deep-remove 3 '((1 2 (3)) ((1 2 3)) 3 2 1)) => '((1 2 ()) ((1 2)) 2 1)
(deep-remove '(3 4 ) '((3 4 ) ((4 3 )) ((5 6 ) (3 4)))))) => '(((4 3 )) ((5 6 )))
(deep-remove '(3 ) '(3 )) => '(3)
(deep-remove '(3 ) '(((3 ))) ) => '(()

(define (deep-remove item lst)
  (cond
   [(null? lst) '()]
   [(equal? item (first lst))
    (deep-remove item (rest lst))]
   [(list? (first lst))
    (cons
     (deep-remove item (first lst))
     (deep-remove item (rest lst)))]
   [else
    (cons
     (first lst)
     (deep-remove item (rest lst)))]))
3. (b) (4 points) Draw a tree of recursive calls for the application \((\text{rec } '((1 2) 3 (4 5)))\), where the procedure \(\text{rec}\) is defined as follows. Give the final value of the application; you do not need to give intermediate values.

\[
\text{(define (rec lst)}
\begin{array}{l}
\text{(cond)} \\
\text{\qquad [\leq (length lst) 1]} \\
\text{\qquad \quad lst]} \\
\text{\qquad [list? (first lst)]} \\
\text{\qquad \quad (append (rec (first lst)) (rec (rest lst)))]} \\
\text{\qquad [else]} \\
\text{\qquad \quad (append (rec (rest lst)) (list (first lst)))]}
\end{array}
\]

\[
\text{(rec '((1 2) 3 (4 5))) => '(2 1 (4 5) 3)}
\]

\[
/ \quad | \quad \backslash \\
\text{append (rec '((1 2) 3 (4 5))) } \\
/ \quad | \quad \backslash \\
\text{append (rec '((1 2) 3 (4 5))) } \\
/ \quad \backslash \\
\text{append '(2) } (1) \text{ append '(4 5)) } (3)
\]
4. For each of the following three Turing machines, assume that it is started in a configuration with a nonempty contiguous sequence of 0’s and 1’s on the tape, surrounded by blanks, with the head on the leftmost non-blank symbol, in state q1. (Recall that the blank symbol is denoted by b.)

For each of the three machines do the following.

(i) Show the first four configurations starting with the initial configuration for the input 100 (counting the initial configuration as the first of the four.)

(ii) Specify for every input string of 0’s and 1’s whether the Turing machine will eventually halt, and the final contents of the tape if and when it does halt.

4.(a) (3 points)

(q1, 0, q1, 0, R)
(q1, 1, q1, 1, R)
(q1, b, q2, b, L)

(1) b 1 0 0 b  (3) b 1 0 0 b
   ^      ^
   q1     q1

(2) b 1 0 0 b  (4) b 1 0 0 b
   ^      ^
   q1     q1

This machine halts for every input, and the contents of the tape are the same as they were in the initial configuration.
4. (b) (3 points)

(q1, 0, q1, 1, R)
(q1, 1, q1, 0, R)
(q1, b, q2, 1, L)
(q2, 1, q2, 1, L)
(q2, b, q1, b, R)

(1) b 1 0 0 b  (3) b 0 1 0 b
      ^    ^
     q1    q1

(2) b 0 0 0 b  (4) b 0 1 1 b
      ^    ^
     q1    q1

This machine halts on every input. If the input contains at least one 1, then the output is equal to the input with the bits flipped (0 for 1, and 1 for 0), with a 1 appended at the end. If the input is a string of 0’s, then the output is equal to the input with 01 appended at the end.

4. (c) (4 points)

(q1, 0, q1, x, R)
(q1, 1, q1, 1, R)
(q1, b, q2, b, L)
(q1, x, q1, x, R)
(q2, x, q2, x, L)
(q2, 1, q2, x, L)
(q2, b, q1, b, R)

(1) b 1 0 0 b  (3) b 1 x 0 b
      ^    ^
     q1    q1

(2) b 1 0 0 b  (4) b 1 x x b
      ^    ^
     q1    q1

This machine does not halt for any input. The first pass of q1 changes all the 0’s to x’s until the right end of the input. Moving left in q2, the machine changes the 1’s to x’s. Finally, it loops back and forth over the x’s, running right in q1 and left in q2.
5. (10 points)

5.(a) State the theorem we proved in class about the Halting Problem for Racket programs. (Note: DO NOT prove the theorem, just state it.)

There is no procedure (halts? proc expr) that for any Racket procedure proc and Racket expression expr will always return #t if (proc expr) halts and #f if (proc expr) does not halt.

5.(b) Give an example of the rm command in Linux, and briefly describe what it does.

> rm temp.txt
>
This removes the file with the name temp.txt from the current directory.

5.(c) Give a Racket expression, using only cons, numbers, and the quoted empty list '(), that evaluates to '((1) (2 3)).

(cons (cons 1 '()) (cons (cons 2 (cons 3 '())) '()))
5. continued.

5.(d) What does the following procedure do when called on positive integers x and y?

(define (what x y)
  (if (>= x y)
      (list x y)
      (what y (+ x 2)))))

When x is greater than or equal to y, the list \((x y)\) is returned. When x is one less than y, the procedure doesn’t halt, for example, 
\((\text{what} 3 \ 4) => (\text{what} 4 \ 5) => (\text{what} 5 \ 6) => \ldots\) When x is less than or equal to \(y-2\), the list \((y \ x+2)\) is returned.

5.(e) For the following procedure find the values of \((f 1)\), \((f 2)\), and \((f 3)\). Rewrite the procedure using the special form let so that there is only ONE recursive call \((f (- n 1))\).

(define (f n)
  (if (= n 1)
      4
      (* (f (- n 1))
          (f (- n 1)))))

\((f \ 1) => 4, \ (f \ 2) => (* \ 4 \ 4) => 16, \ (f \ 3) => (* \ 16 \ 16) => 256\)

(define (f n)
  (if (= n 1)
      4
      (let ((prev (f (- n 1))))
        (* prev prev))))
6. (10 points) Evaluate the following Scheme expressions. 
(Remember to show work to permit partial credit!)

Example:

(cons 1 '(6 5)) => '(1 6 5)

(a) (cons '(1 2) '(3 4)) => '((1 2) 3 4)

(b) (first (rest '((apple book) (cat) (doubt eel)))) => '(cat)

(c) (map length '((3 (2 1)) (5 4 6) ((7) 8))) => '(2 3 2)

(d) (let ((lst1 '(3 4 5 6 7 8 9)))
    (let ((lst2 (filter (lambda (x) (and (even? x) (< x 8)))
                   lst1)))
      (append lst2 lst2))) => '(4 6 4 6)

(e) (let ((p (lambda (x) (lambda (q) (q (q x))))))
    (r (lambda (y) (* y y))))
    (list (r 2) ((p 3) r))) => '(4 81)