CS 135 2015 – Homework 8

Submit one file that contains the Scheme definitions (and any tests you want to include) for the following exercises. Submit another document with all answers

1. (Review) Define a Scheme function expo so that (expo n i) is n^i for natural numbers n and i. Use the standard math definition

$$n^0 = 1 \quad \text{and} \quad n^i = n * n^{i-1} \quad \text{for all } i > 0$$

2. Using the standard math definition, prove that

$$n^i * n^j = n^{i+j}$$

   for all natural numbers i, j and all natural numbers n. Use induction on i.

3. As a consequence of what you proved, we know that $n^{2*i} = n^i * n^i$. In other words, if j is even then $n^j = n^{j/2} * n^{j/2}$. Use this observation to add an additional case in your definition of expo. Call the new Scheme function expo-fast.

   Is it faster? justify your answer somehow.

4. Define another version, expo-trec, that works by calling a tail-recursive helper function. It does not need to use the improvement from expo-fast. The helper function should be called expo-help and it should use an additional parameter as an accumulator, so that (expo-help n i r) accumulates the result in r.

5. Our goal is to prove that (expo-trec n i) = (expo n i) for any natural n and i. But (expo-trec n i) probably just calls (expo-help n i 1), so what we really need is to come up with an equation of this form:

   LEMMA (expo-help n i r) = ???

   And it has to be such that if we plug in 1 for r then the right side simplifies to (expo n i).

   Your job: figure out that equation and prove it by induction on i. Then use the lemma to prove (expo-trec n i) = (expo n i) for any n, i.

6. Consider this function.

```
(define powsum
  ; Takes a list, lon, of natural numbers.  Returns the sum of
  ; the powers 2^n where n ranges over elements of lon.
  (lambda (lon)
    (cond [(null? lon) 0]
          [else (+ (expt 2 (car lon)) (powsum (cdr lon)))])))
```

For example, (powsum '(2 3 1)) returns $2^2 + 2^3 + 2^1$ which is 14.

Write another version, powsum-trec, that satisfies the same specification but works by calling a tail recursive helper. (It can use the built-in function expt or one of your expo functions, whatever you like.)

Here is some code you could use for testing.

```
(define test-powsum
  (and (equal? (powsum '(1 2 3)) (powsum-trec '(1 2 3)))
       (equal? (powsum '()) (powsum-trec '()))
       (equal? (powsum '(5 2 49 0)) (powsum-trec '( 5 2 49 0)))))
```