Workshop tips:

1. We can answer many of this week's questions using the limit test provided on The Asymptotic Cheat Sheet. An alternate representation that achieves the same result can be found on Stack Overflow.
2. Below is the relationship between some of the common time complexities, in order of fastest to slowest.

$$1 < \log(n) < \sqrt{n} < n < n\log(n) < n^2 < 2^3 < 2^n < 3^n < n!$$

If f is some complexity, then we can determine if f belongs to O(g), Ω(g), or Θ(g) depending on whether g is to the right, left, or in the same position as f, respectively.

Exercise 1 *Complexity Notation*

10. Right or wrong?
    (a) $n^2 + 10^6 n \in O(n^2)$

    Right → limit approaches 1 (1 < ∞).

    (b) $n \log n \in O(n)$

    Wrong → limit approaches ∞.

    (c) $n \log n \in \Omega(n)$

    True → limit approaches ∞.

    (d) $\log n \in o(n)$

    True → limit approaches 0.

11. Prove Lemma 6

Lemma 6. *The following rules hold for $\mathcal{O}$-notation:*

$$cf(n) = \Theta(f(n)) \text{ for any positive constant } c \quad (2.6)$$
$$f(n) + g(n) = \Omega(f(n)) \quad (2.7)$$
$$f(n) + g(n) = \mathcal{O}(f(n)) \text{ if } g(n) = \mathcal{O}(f(n)) \quad (2.8)$$
$$\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n)) \quad (2.9)$$

```
=== Lemma 6: 2.6 ===
    Prove:
        c*f(n) = θ(f(n)), for any c

      Proof by limits:
        lim[n->inf] c*f(n) / f(n)
                = f(n) / f(n)                  [ignore constants]
                = 1

             since limit equals a real number greater than 0, it is Big-θ
```

```
=== Lemma 6: 2.7 ===
    Prove:
        f(n) + g(n) = Ω(f(n))

    Proof by limits:
        lim[n->inf] (f(n) + g(n)) / f(n)        [ignore constants]
                >= 1                             [since all g(n) are positive,
the sum of any f and any g will be greater than any lone f ]

             since limit is at least 1, it is Big-Ω
```

```
=== Lemma 6: 2.8 ===
    Given that:     g(n) = O(f(n))
        =>    g(n) = c*f(n), for some n >= n0

    Proof by limits:
             lim[n->inf] (f(n) + c*f(n)) / f(n)
        lim[n->inf] (1+c)*f(n) / f(n)
                = f(n) / f(n)                   [ignore constants]
                = 1
                < inf
            therefore in Big-O.
```

```
=== Lemma 6: 2.9 ===
    Prove:
        O(f(n)) * O(g(n)) = O(f(n)*g(n))

    Process:
        O(f(n)) * O(g(n)) <= c1*f(n) * c2*g(n)   [definition of  big-O]
                          <= c1*c2 * f(n)*g(n)
                          <= c3 * f(n)*g(n)
                           = O(f(n)*g(n))
```

Exercise 2 *Complexity Notation*

Is it true that if f(n) = Θ (g(n)) and g(n) = Θ (h(n)), then h(n) = Θ (f(n))?

Given:

- $f = \Theta(g) \Rightarrow c_1 * g \leq f \leq c_1 * g$, where $c_1, c_2 > 0$, and n > n_0
- $g = \Theta(h) \Rightarrow c_3 * h \leq g \leq c_4 * h$, where $c_3, c_4 > 0$, and n > n_0'

Is $h = \Theta(f)$ correct?

First, define n: $n > \max(n_0, n_0')$

Now, RHS:

$$c_3 h \leq g \leq {}^{f}/_{c_1}$$

$$h \leq {}^{f}/_{c_1 c_3}$$

And, LHS:

$$ {}^{f}/_{c_2} \leq g \leq c_4 h$$

$$ {}^{f}/_{c_2 c_4} \leq h$$

Therefore:

$$ {}^{f}/_{c_2 c_4} \leq h \leq {}^{f}/_{c_1 c_3}$$

$$c_5 f \leq h \leq c_6 f, where\ c_5 = {}^{1}/_{c_2 c_4}, c_6 = {}^{1}/_{c_1 c_3}, n > \max(n_0, n_0')$$

$$\therefore h = \Theta(f)$$

Exercise 3 *Complexity Notation*

Is it true that if f(n) = O(g(n)) and g(n) = O(h(n)), then h(n) = Ω (f(n))?

Given:

- $f = O(g) \Rightarrow f \le c_1 * g$, where c_1 > 0, and n > n_0 [1]
- $g = O(h) \Rightarrow g \le c_2 * h$, where c_2 > 0, and n > n_0' [2]

Is $h = \Omega(f)$ correct?

Again, let's define n: $n > \max(n_0, n_0')$

$f \le c_1 * \boldsymbol{g} \le c_1 * (\boldsymbol{c_2} * \boldsymbol{h})$ [chain 1 and 2]

$^1/_{c_1} f \le g \le c_2 h$ [divide all by c_1]

$^1/_{c_1 c_2} f \le h$ [ignore middle, and isolate h]

$c_3 f \le h, where\ c_3 = {}^1/_{c_1 c_2}, n > \max(n_0, n_0')$

$\therefore h = \Omega(f)$

Exercise 4 *Complexity Notation*

Is it true that a Θ (n^2) algorithm always takes longer to run than a Θ (logn) algorithm?

No, for two reasons:

1. When we have a small n, then the terms other than the fastest growing terms are significant and cannot be ignored.
2. Different programming languages and computer architecture can also affect how long an algorithm runs for.

We can only say that the n^2 algorithm will *always* take longer when all other factors are constant or ignored – which is a big reason as to why we use Big-O notation – and again, requires n to exceed a threshold n_0.

Exercise 5 *Complexity Notation*

For each pair of functions given below, point out the asymptotic relationships that apply: $f = O(g)$, $f = \Theta(g)$, $f = \Omega(g)$.

- $f(n) = \sqrt{n}$ and $g(n) = log(n)$
- $f(n) = 1$ and $g(n) = 2$
- $f(n) = 1000 \cdot 2^n$ and $g(n) = 3^n$
- $f(n) = 4^{n+4}$ and $g(n) = 2^{2n+2}$
- $f(n) = 5nlog(n)$ and $g(n) = nlog(5n)$
- $f(n) = n!$ and $g(n) = (n+1)!$

To solve these we can either graph them or use the limit test, however that adds unnecessary steps, instead we can use tip #2.

- f=Ω(g) "f grows at least as fast as g"
- f=Θ(g) (limit=.5) "f grows at the same rate as g"
- f=O(g) "f grows no faster than g"
- f=Θ(g) (limit=64) "f grows at the same rate as g"
- f=Θ(g) (limit= 5) "f grows at the same rate as g"
- f=O(g) "f grows no faster than g"

Exercise 6 *Complexity Notation*

Prove that $n^k = o(c^n)$ for any integer k and any c > 1.

```
PROVE: n^k = o(c^n)

GOAL: Show that for all c1 > 0, there exists a n0 such that n >= n0 and a^k <=
c1*c^n
      Let c1 > 1, c >0, and k be constants.

Let's start by ASSUMING the statement is true.
      n^k = o(c^n)

      n^k <= c1 * c^n                    c1>0 & n>n0  [definition of little-o]
IFF   log[c](n^k) <= log[c](c1 * c^n)                 [log base c both sides]
IFF   log[c](n^k) <= log[c](c1) + log[c](c^n)         [log(AB) = log(A) + log(B)]
IFF   k*log[c](n) <= log[c](c1) + n*log[c](c)         [log(A^n) = n*log(A)]
IFF   k*log[c](n) <= log[c](c1) + n                   [log[A](A) = 1]
IFF   k*log[c](n) - log[c](c1) <= n

Now let's swap n with n0:
      n0 >= k*log[c](n0) - log[c](c1)
This tells us how to find n0. Remember for little-o we need EVERY c1 > 0 to have
an appropriate threshold (n0) to hold true (this is different than big-o).

From this we can see that no matter what c1 is, we will be able to find an
appropriate threshold. And we can only get this if our original assumption was
true, meaning the mathematical form is correct. And if that's correct, the big-O
notation must also be correct.

Test this proof using limits:
      lim[n->inf] n^k / c^n = {small} / {massive} = 0
```