The Algorithm Analysis Framework

- Read 2.1, pages 42-50
  - What are the two kinds of **efficiency** we consider? What does each measure?
  - Which of those efficiencies will we focus more on? Why?
  - True or False: For most algorithms, the runtime is more or less independent of the input, they run for about the same time regardless of the input.
  - Discuss some situations where there are multiple choices for what we might call the problem's **input size**?
  - How do we measure the input size in the case where the input is a number? Why does that make sense?
  - Why do we not use a normal unit like milliseconds to measure the running time of an algorithm?
  - What do we refer to as the **basic operation** of an algorithm? Why do we choose to measure running time by counting the number of basic operations, rather than some other metric?
  - When considering the runtime of an algorithm we often ignore multiplicative constants. Why is that?
  - What do we mean when we talk about **order of growth**?
  - Study table 2.1. How do you expect each of the numbers to change if we look at the next n power, 10^7?
  - If a computer performs a trillion (10^{12}) operations a second, how long (in years) would it take such a computer to perform $50!$ operations (this is about $3.04 \cdot 10^{64}$ operations)?
  - What effect does doubling the input size n have for each of the functions in Table 2.1? Remember that the functions are meant to represent the running time of an algorithm.
  - Explain what the terms **worst-case efficiency**, **best-case efficiency** and **average-case efficiency** mean. Demonstrate in the example of the sequential search algorithm described in the text.
    - What is the importance of the worst-case efficiency?
    - What is the importance of the average-case efficiency?
    - Explain how the average-case efficiency was computed for the example of sequential search, and what assumptions were made.
    - Why can we not compute the average-case efficiency by simply averaging the worst-case and the best-case?
  - What do we call **amortized efficiency** for an algorithm?
  - Practice problems: 2.1.1, 2.1.3, 2.1.7
  - Challenge/Fun: 2.1.4, 2.1.10

- Read 2.2, pages 52-58
  - Informally, what do the notations $O(g(n))$, $\Omega(g(n))$, $\Theta(g(n))$ mean?
    - True or False: $2n^2(n-1) \in O(n^2)$
    - True or False: $2n^2(n-1) \in \Omega(n^2)$
    - True or False: $2n^2(n-1) \in \Theta(n^2)$
  - Provide formal definitions for $O(g(n))$, $\Omega(g(n))$, $\Theta(g(n))$.
    - True of False: If $t(n) \in O(g(n))$ and $t(n) \in \Omega(g(n))$ then we also have $t(n) \in \Theta(g(n))$.
    - True or False: If $t(n) \in O(g(n))$ and c is a constant number, then $ct(n) \in O(g(n))$ also.
  - Explain the theorem on page 56: If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then $t_1(n) + t_2(n) \in O(\max c_1(g), c_2(g))$.
    - What important implication does this have on the analysis of an algorithm?
  - Study Table 2.2 on common basic efficiency classes and when they occur.
  - Practice problems: 2.2.2, 2.2.3, 2.2.5