Core Objectives for Lesson 2:

- know the definitions of big-oh, theta, omega, and little-oh, and their negations, and be able to apply these to answer questions about which complexity class a given function belongs to

- be familiar with the asymptotic ordering of complexity classes from smallest to largest; for instance O(1) < O(log n) < O(sqrt(n)) < O(n) < O(nlog n) < O(n^2) < O(n^k) (k > 2) < O(2^n) < O(n!) < O(n^n)

- be able to determine asymptotic running time of a non-recursive algorithm by using rules for running time of (nested) loops and branching logic.

- be able to determine the running time of recursive algorithms (in simple cases) using any of the following approaches:
  * Guessing Method, including verification of correctness of the guess and ability to apply the "not a power of 2" theorem [The theorem says: Suppose f(n) is smooth and T(n) is eventually nondecreasing. Then if T(n) is Theta(f(n)) for n a power of 2, T(n) is Theta(f(n)).]
  * Counting self-calls
  * Using the Master Formula

- be able to show that the naive recursive algorithm for computing the Fibonacci numbers runs in exponential time and that the iterative algorithm runs in linear time

- be familiar with iterative and recursive algorithms to compute factorial and Fibonacci numbers

- be familiar with the binary search algorithm and analysis of its running time

- be able to demonstrate correctness of iterative and recursive algorithms in simple cases (factorial, fibonacci, binary search, and others of similar level of difficulty)

- be familiar with the concept of a "divide and conquer" algorithm