Functions in Asymptotic Notation

When we use asymptotic notation to express the rate of growth of an algorithm's running time in terms of the input size $\mathbf{n}$, it's good to bear a few things in mind.

Let's start with something easy. Suppose that an algorithm took a constant amount of time, regardless of the input size. For example, if you were given an array that is already sorted into increasing order and you had to find the minimum element, it would take constant time, since the minimum element must be at index 0. Since we like to use a function of n in asymptotic notation, you could say that this algorithm runs in $\mathbf{\Theta(n^0)}$ time. Why? Because $\mathbf{n^0 = 1}$, and the algorithm's running time is within some constant factor of 1. In practice, we don't write $\mathbf{\Theta(n^0)}$, however; we write $\mathbf{\Theta(1)}$.

Now suppose an algorithm took $\mathbf{\Theta(\log_{10} n)}$ time. You could also say that it took $\mathbf{\Theta(\lg n)}$ time (that is, $\mathbf{\Theta(\log_2 n)}$ time). Whenever the base of the logarithm is a constant, it doesn't matter what base we use in asymptotic notation. Why not? Because there's a mathematical formula that says

$$log_a n = \frac{log_b n}{log_b a}$$

for all positive numbers **a**, **b**, and **n**. Therefore, if **a** and **b** are constants, then $\mathbf{\log_a n}$ and $\mathbf{\log_b n}$ differ only by a factor of $\mathbf{\log_b a}$, and that's a constant factor which we can ignore in asymptotic notation.

Therefore, we can say that the worst-case running time of binary search is $\mathbf{\Theta(\log_a n)}$ for any positive constant **a**. Why? The number of guesses is at most $\mathbf{\lg n + 1}$, generating and testing each guess takes constant time, and setting up and returning take constant time. As a matter of practice, we write that binary search takes $\mathbf{\Theta(\lg n)}$ time because computer scientists like to think in powers of **2** (and there are fewer characters to write than if we wrote $\mathbf{\Theta(\log_2 n)}$.)

There is an order to the functions that we often see when we analyze algorithms using asymptotic notation. If **a** and **b** are constants and **a** < **b**, then a running time of $\Theta(n^a)$ grows more slowly than a running time of $\Theta(n^b)$. For example, a running time of $\Theta(n)$, which is $\Theta(n^1)$, grows more slowly than a running time of $\Theta(n^2)$. The exponents don't have to be integers, either. For example, a running time of $\Theta(n^2)$ grows more slowly than a running time of $\Theta(n^{2\sqrt{n}})$, which is $\Theta(n^{2.5})$.

Logarithms grow more slowly than polynomials. That is, $\Theta(\lg n)$ grows more slowly than $\Theta(n^a)$ for any positive constant **a**. But since the value of **lg n** increases as **n** increases, $\Theta(\lg n)$ grows faster than $\Theta(1)$.

Here's a list of functions in asymptotic notation that we often encounter when analyzing algorithms, listed from slowest to fastest growing. This list is not exhaustive; there are many algorithms whose running times do not appear here:

1. $\Theta(1)$
2. $\Theta(\lg n)$
3. $\Theta(n)$
4. $\Theta(n \lg n)$
5. $\Theta(n^2)$
6. $\Theta(n^2 \lg n)$
7. $\Theta(n^3)$
8. $\Theta(2^n)$

Note that an exponential function a^n where $a > 1$, grows faster than any polynomial function n^b where **b** is any constant.