$$b^x = 2^{\log_2(b^x)} = 2^{(\log_2 b)\cdot x}$$

Fact. Suppose b and c are positive real numbers.
① Then
$$b^x = O(c^x)$$
if and only if $b \le c$.

② $b^x = c^{O(x)}$.

Because of ②, we often work base 2 in complexity theory. Often with exponential complexity, we think about the asymptotics of the exponent — it's hard and rare to get something exponential to something polynomial!

Fact. Let n be a nonneg. int. and $b > 1$ a ~~pos~~ real number. Then
$$x^n = O(b^x) \quad \text{and} \quad b^x \neq O(x^n).$$

~~The growth~~ Important: The growth rate of polynomials is dominated by exponentials.

Logs: Similar story to exponentials. From now on exponentials and logs are base 2.

Fact: $\log x = O(x)$ and $x \neq O(\log x)$.

Polynomials, logs, and exponents are the 3 most common kinds of functions that arise in analyzing complexity.

Common Pitfalls

1. Exponential Fiasco.

Many (myself included) sometimes belive that

$$4^x = O(2^x).$$

This is false since $4^x = 2^{2x}$, so it grows as the square of 2^x.

2. Constant confusion.

Every constant is $O(1)$. We'll use this (correct) fact to state and (incorrectly) prove a false proposition.

Fake Proposition. $\sum_{i=1}^{n} i = O(n)$.

Wrong Proof. Let $f(n) = \sum_{i=1}^{n} i = 1+2+3+\cdots+n.$
Since every constant is $O(1)$, we have

$$\begin{aligned} f(n) &= 1+2+3+\cdots+n \\ &= O(1)+O(1)+O(1)+\cdots+O(1) \\ &= O(n). \end{aligned}$$ □

Note that $\sum_{i=1}^{n} i = O(n^2)$ since $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

3. Equality Enigmas.

I personally dislike the notation $f = O(g)$. The use of equality is unfortunate. This equality must never be confused with the usual meaning. Otherwise...

$$0 = O(1) = 1 \Rightarrow 0 = 1.$$

Exercise. Arrange the functions in a sequence $f_1, f_2, \dots, f_{12}$ so that $f_i = O(f_{i+1})$.

1. $n \log n$ 2. n^7 3. 2^n 4. 3^n

5. $2n$ 6. $n!$ 7. 2^{n+1} 8. $2^{100} n$

9. $\frac{\log n}{n}$ 10. $\log n$ 11. $\log_{10} n$ 12. n^{64}.

Other notations.

Big O notation can only be used for upper bounds. For example, it is technically wrong to say something like "That algorithm has a running time of at least $O(n^2)$." What they mean is to use the Omega notation.

Def. Let $f: \mathbb{R} \to \mathbb{R}$ and $g: \mathbb{R} \to \mathbb{R}$ be nonnegative functions. Then $f = \Omega(g)$ is equivalent to $g = O(f)$.

All our examples and intuition from big O apply here with Omega. The correct statement is

"That algorithm has a running time of at least $\Omega(n^2)$."

Another notation is Theta.

Def. $f = \Theta(g)$ means $f = O(g)$ and $f = \Omega(g)$.

Therefore Theta comes up when two functions grow at the same rate.