Intro to Computational Complexity

CSC165 Week 7

Lindsey Shorser, Winter 2021

So far we have learned…

- How to symbolize logical statements precisely with quantifiers, predicates, and logical operators connecting them.

- How to prove many of those statements using direct and indirect proof structures

So far we have learned…

- How to symbolize logical statements precisely with quantifiers, predicates, and logical operators connecting them.
- How to prove many of those statements using direct and indirect proof structures

During weeks 7-10, we will look at…

- Ways to associate a function with part of a program to represent its complexity
- How to use these functions to decide which will run “faster”
- Proofs of precise logical statements about functions that will give us this information

How can we figure out the time it takes for a program to run?

Consider this program:

```
def print_items(lst: list) -> None:
    for item in lst:
        print(item)
```

Let n = the number of items in the list.

Consider this program:

```
def print_items(lst: list) -> None:
    for item in lst:
        print(item)
```

Let n = the number of items in the list.

How long will it take to run this program as n gets bigger?

Will this depend on the number of characters being printed each time?

Measuring runtime by timing the computer

- depends on hardware
- depends on whatever software is also running at the same time
- depends on definition of “basic operations”
- depends on the relative time it takes to run different each “basic operation”

Note: Empirical measurement is useful for lower-level languages.

Consider this program:

```
def print_items(lst: list) -> None:
    for item in lst:
        print(item)
```

Let n = the number of items in the list.

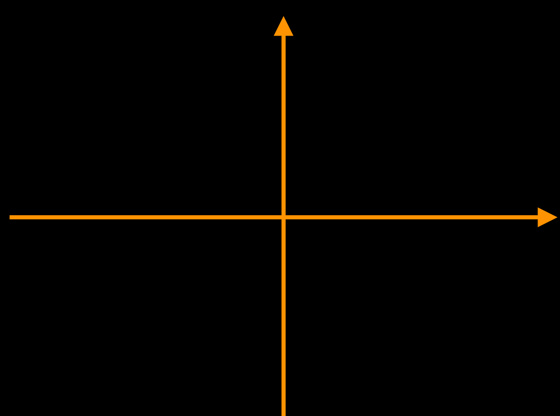
What counts as a basic operation?

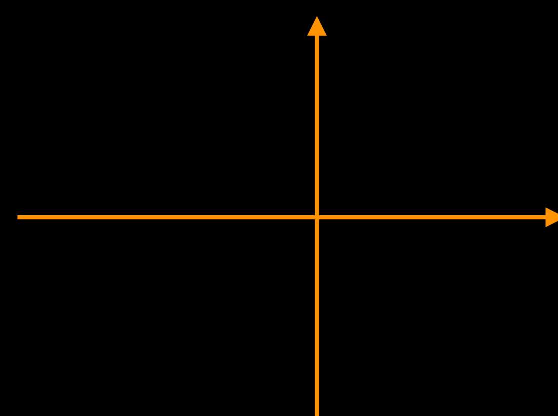
- The `print` call?
- Reassigning the `item` variable during each iteration?
- Calling `print_items` in the first place?

Total time to run the code snippet as a function of n:

Too much ambiguity!!

Solution: We can represent algorithms with a class of functions instead of using a single function.





Asymptotic Growth

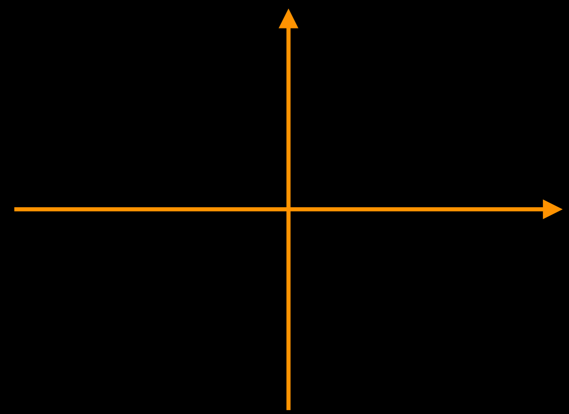
As n gets bigger, what happens to f(n)?

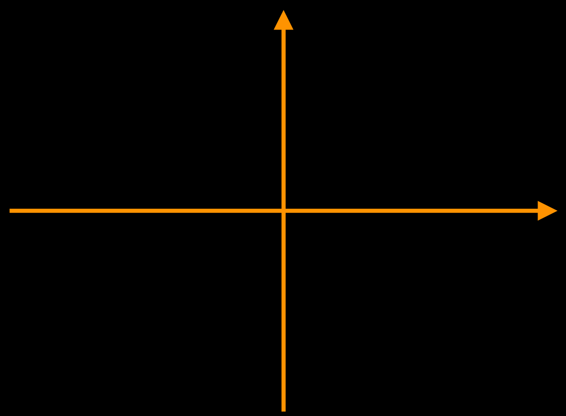
As n gets bigger, what happens to the relative positions of f(n) and g(n)?

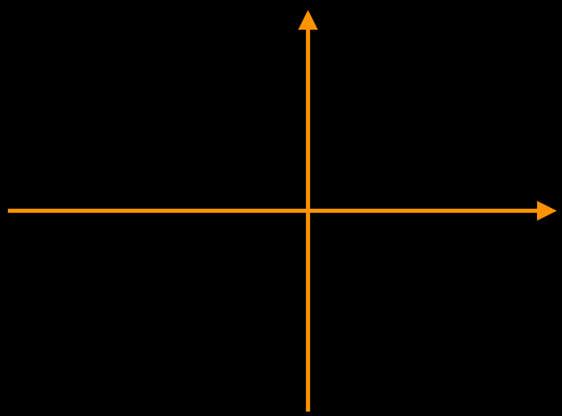
How can we describe functions that “grow together”?

How can we describe the worst case, best case, and average case scenarios for the runtime of an algorithm?

Definition 5.1. Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We say that g is **absolutely dominated by** f if and only if for all $n \in \mathbb{N}$, $g(n) \leq f(n)$.



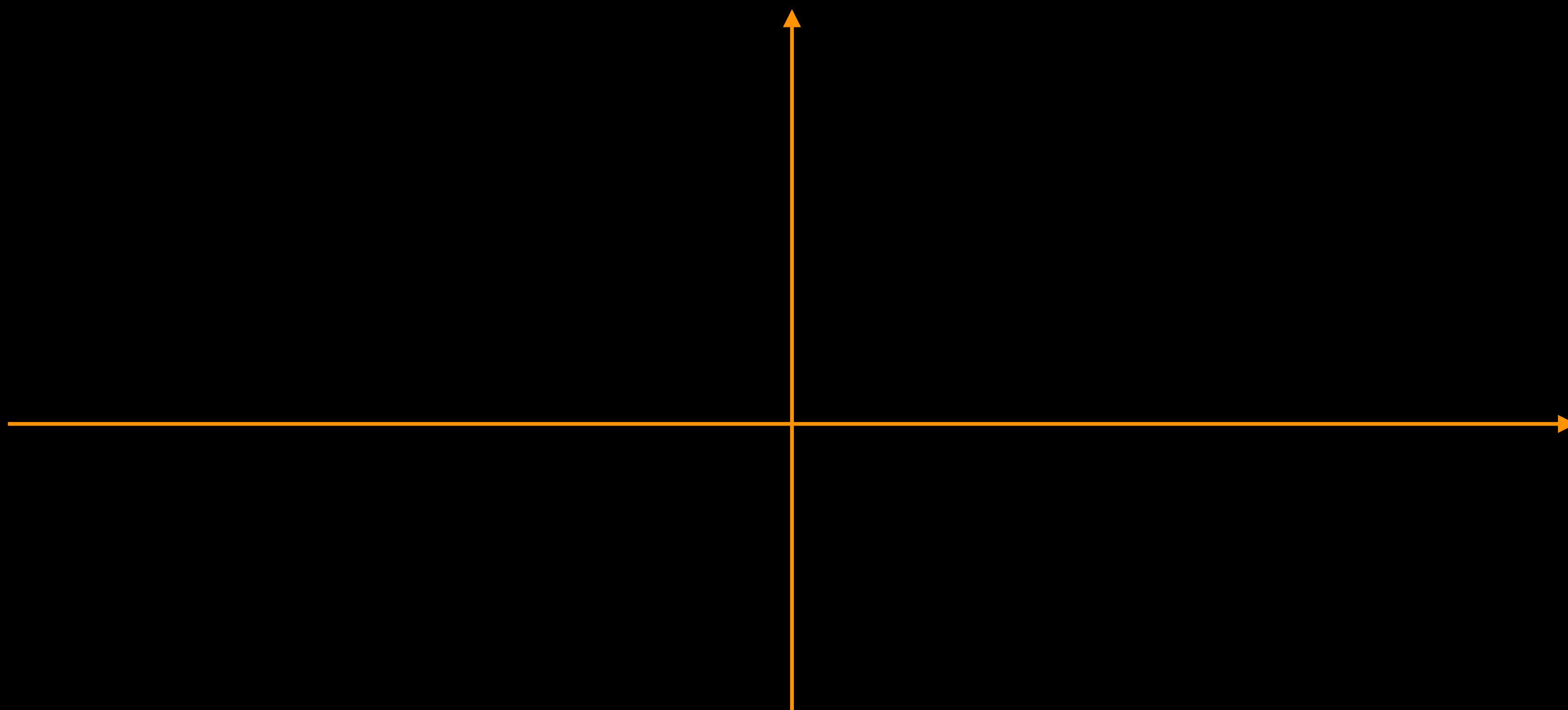


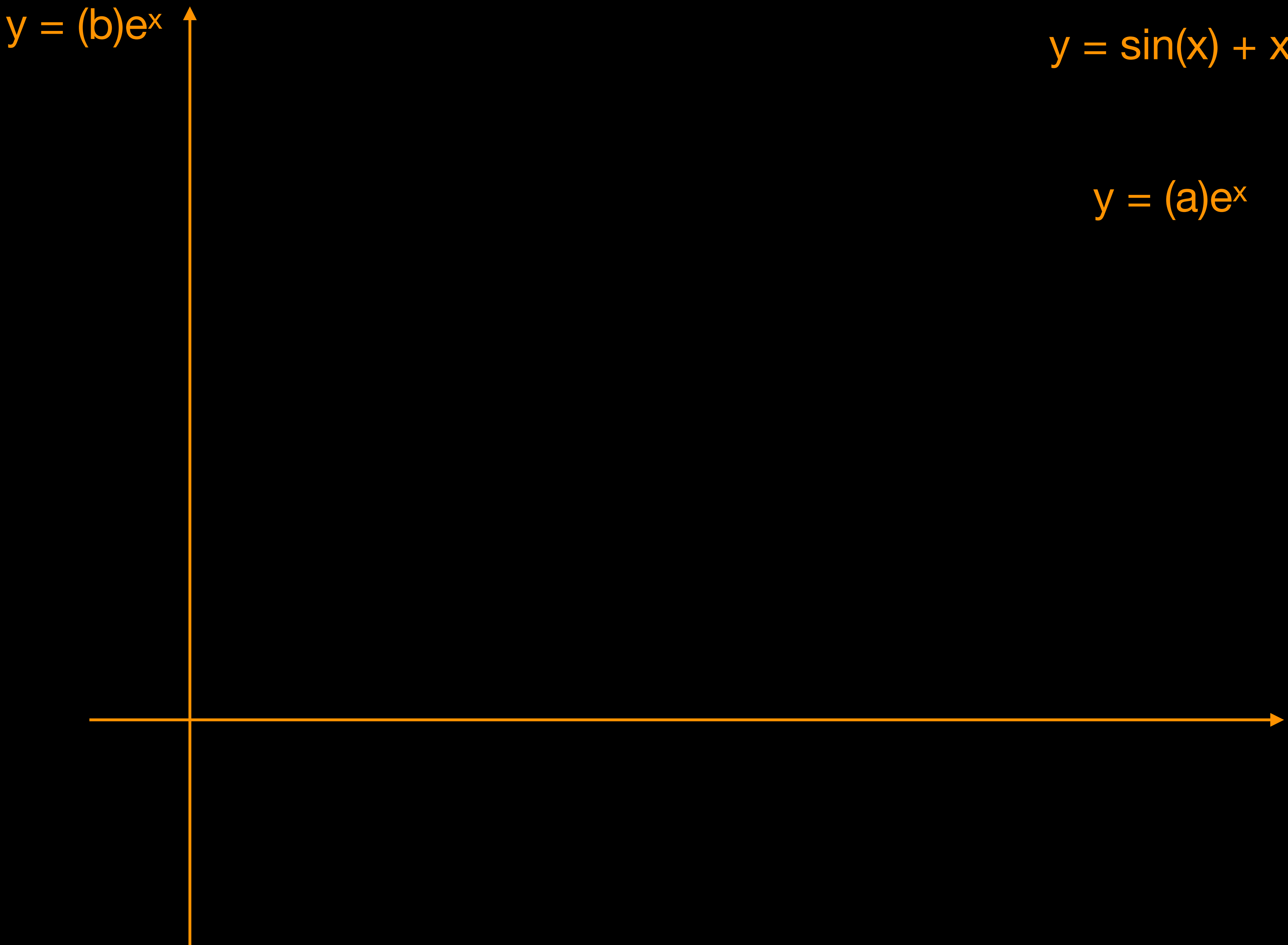


[redacted]

Definition 5.2. Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We say that g **is dominated by** f **up to a constant factor** if and only if there exists a positive real number c such that for all $n \in \mathbb{N}$, $g(n) \leq c \cdot f(n)$.

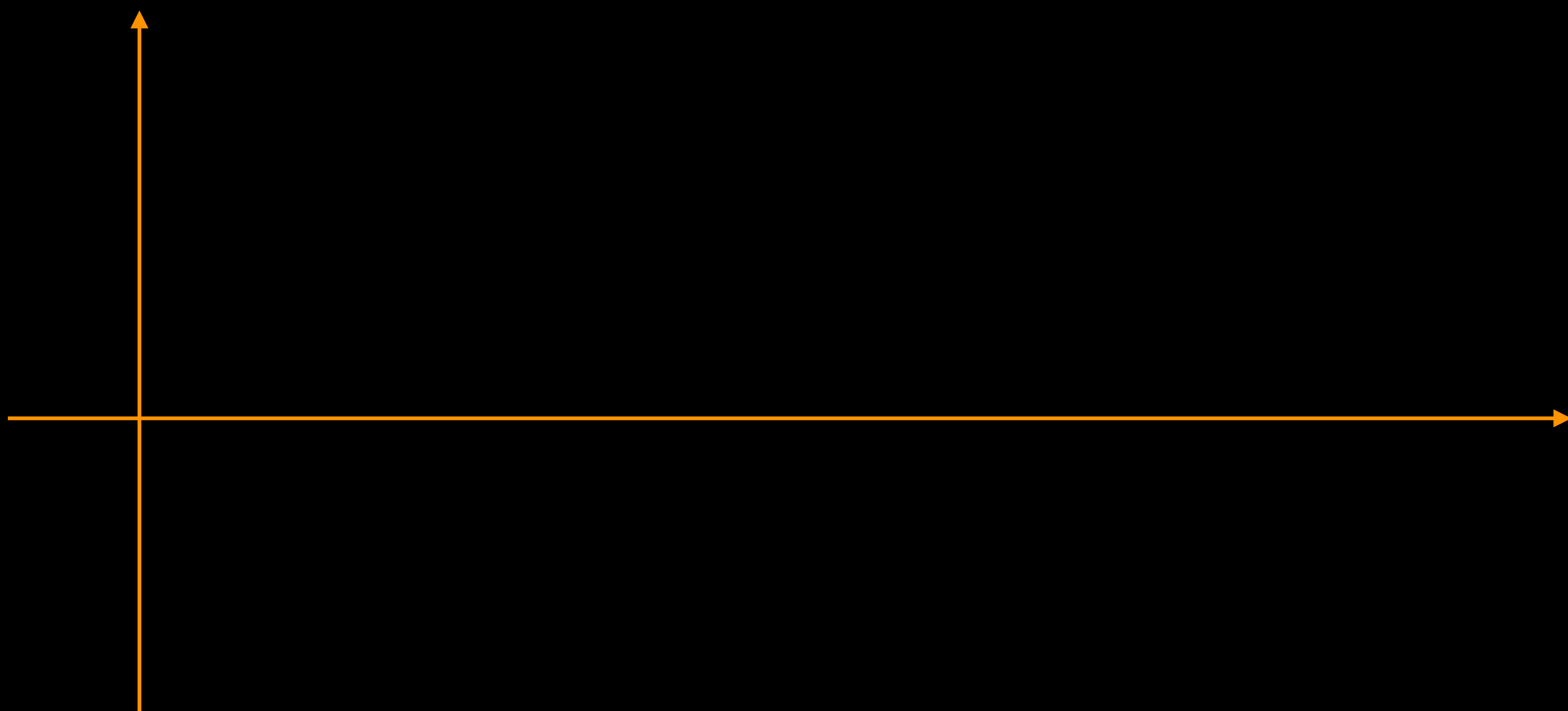
https://www.desmos.com/calculator/dfg5bpuupq





$y = (b)e^x$
$y = \sin(x) + x$
$y = (a)e^x$

Definition 5.3. Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We say that g **is eventually dominated by** f if and only if there exists $n_0 \in \mathbb{R}^+$ such that $\forall n \in \mathbb{N}$, if $n \geq n_0$ then $g(n) \leq f(n)$.



Big-Oh Notation

A way to describe the set of functions g that will eventually dominate f up to a constant factor

Big-Oh Notation

A way to describe the set of functions g that will

eventually be dominated by f up to a constant factor

$$\mathcal{O}(f) = \{g \mid g : \mathbb{N} \to \mathbb{R}^{\geq 0},\ \exists c, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \leq c \cdot f(n)\}.$$

We can talk about the runtime of an algorithm that is represented by function g(n) by finding f(n) so that $g \in \mathcal{O}(f)$

Definition 5.4. Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We say that g **is eventually dominated by** f **up to a constant factor** if and only if there exist $c, n_0 \in \mathbb{R}^+$, such that for all $n \in \mathbb{N}$, if $n \geq n_0$ then $g(n) \leq c \cdot f(n)$.

Definition 5.3. Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We say that g **is eventually dominated by** f if and only if there exists $n_0 \in \mathbb{R}^+$ such that $\forall n \in \mathbb{N}$, if $n \geq n_0$ then $g(n) \leq f(n)$.

Example: Let $f(n) = n^3$ and $g(n) = n^3 + 100n + 500$

Strategy:

If $a \leq x$, $b \leq y$, and $c \leq z$, then $a+b+c \leq x+y+z$

$$
\begin{aligned}
n^3 &\leq c_1 n^3 \\
100n &\leq c_2 n^3 \\
5000 &\leq c_3 n^3
\end{aligned}
$$

Approach 1: focus on choosing n_0.

It turns out we can satisfy the three inequalities even if $c_1 = c_2 = c_3 = 1$:

- $n^3 \le n^3$ is always true (so for all $n \ge 0$).
- $100n \le n^3$ when $n \ge 10$.
- $5000 \le n^3$ when $n \ge \sqrt[3]{5000} \approx 17.1$

Why?

We can pick n_0 to be the largest of the lower bounds on n, $\sqrt[3]{5000}$, and then these three inequalities will be satisfied!

Approach 2: focus on choosing c.

Another approach is to pick c_1, c_2, and c_3 to make the right-hand sides large enough to satisfy the inequalities.

- $n^3 \leq c_1 n^3$ when $c_1 = 1$.
- $100n \leq c_2 n^3$ when $c_2 = 100$.
- $5000 \leq c_3 n^3$ when $c_3 = 5000$, *as long as* $n \geq 1$.

Note 1: We want to look at multiples of n^3 so that we can collect "like terms" at the end to get $c \cdot n^3$.

Note 2: $n = 0$ will never satisfy the last inequality for ANY value of c_3 so we actually needed the word "eventually".

Proof. (*Using Approach 2*) Let $c = 5101$ and $n_0 = 1$. Let $n \in \mathbb{N}$, and assume that $n \geq n_0$. We want to show that $n^3 + 100n + 5000 \leq cn^3$.

First, we prove three simpler inequalities:

- $n^3 \leq n^3$ (since the two quantities are equal).
- Since $n \in \mathbb{N}$, we know that $n \leq n^3$, and so $100n \leq 100n^3$.
- Since $1 \leq n$, we know that $1 \leq n^3$, and then multiplying both sides by 5000 gives us $5000 \leq 5000n^3$.

Adding these three inequalities gives us:

$$n^3 + 100n + 5000 \leq n^3 + 100n^3 + 5000n^3 = 5101n^3 = cn^3.$$

What does it mean if g $\in \mathcal{O}(1)$?

$$\mathcal{O}(f) = \{g \mid g : \mathbb{N} \to \mathbb{R}^{\geq 0},\ \exists c, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \leq c \cdot f(n)\}.$$