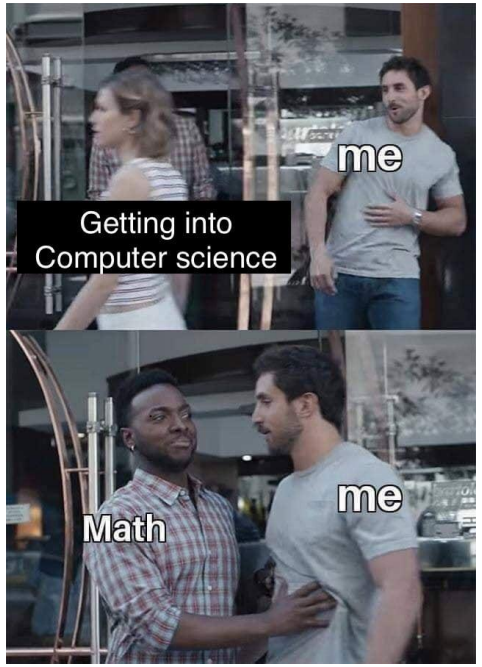
D1: Proof Methods Review and Asymptotic Notations



Sec 101: MW 3:00-4:00pm DOW 1018
IA: Eric Khiu

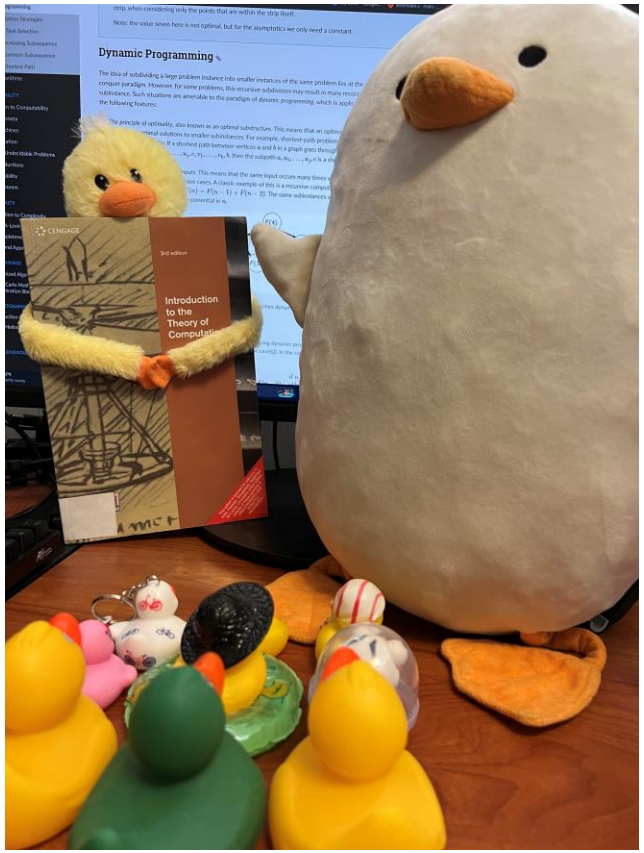
Agenda

- Introduction
- Proof Methods from EECS 203 that we need
  - Direct
  - Contrapositive
  - Contradiction
  - Induction
- Size of input: Data as bitstrings
- Asymptotic Notations
  - Definitions
  - Proof Strategies

Introduction

Me

- Eric **Khiu** (pronounced as Q)
- He/him
- Rising senior, Math & CS
- From Malaysia (a small country in southeast Asia)
- Area of interest: NLP and Computational Linguistics
- OH: Wed 4:00-6:00pm (Right here!)
- E-mail: erickhiu@umich.edu
  - Would be nice if you could start the subject with [EECS 376]



Your turn!

- This survey is anonymous. I just want to know your background to better tailor the discussion for you.

Proof Methods Review

Direct Proof ($P \Longrightarrow Q$)

- If P then Q: Assume P is true, show that P's truth implies statement Q is true.
- P if and only if Q: Show if P then Q **and** if Q then P
- **Example:** Consider the following program

```
x ←input()
while x > 10 do
    if x is odd then
        x ← (x − 1)/2
    else
        x ← x + 2
```

Prove that if the input is an even number greater than 10, then the program will not terminate.

Direct Proof Example

- **Example:** Consider the following program

```
x ←input()
while x > 10 do
    if x is odd then
        x ← (x − 1)/2
    else
        x ← x + 2
```

Prove that if the input is an even number greater than 10, then the program will not terminate.

Suppose the input is an even number greater than 10, then on every iteration x will be incremented by 2, which keeps it even and never reach the termination condition $x \leq 10$.

Proof by Contrapositive ($\neg Q \implies \neg P$)

Definition: An integer is even if and only if $x = 2k$ for some integer k.

- When a direct proof ($P \implies Q$) seems difficult: it is **equivalent**, and sometimes **easier**, to prove the *contrapositive* (If not Q then not P)
- **Example:** Let $f(x) = x + 2$. Using proof by contrapositive, prove that $f(x)$ is even if and only if x is even.
  - Direction 1: If x is even, then $f(x)$ is even.
  - Direction 2: If $f(x)$ is even, then x is even.

Poll: What is the contrapositive of Direction 2?
A. If x is odd, then $f(x)$ is odd.
B. If $f(x)$ is odd, then x is odd.

An example of Proof by Contrapositive in EECS 376

- Statement: $\phi \in A$ if and only if $f(\phi) \in B$
  - Direction 1: $\phi \in A \Rightarrow f(\phi) \in B$
  - Direction 2: $f(\phi) \in B \Rightarrow \phi \in A$
- Contrapositive for Direction 2: $\phi \notin A \Rightarrow f(\phi) \notin B$

We'll see similar idea in Unit 3: Complexity!

Proof by Contradiction

- Assume ¬P (P is false)
- Create some false/ illogical statements, for example:
  - 1 = 2
  - a is odd (when we assumed it to be even)
  - G is acyclic (when we assumed it is not acyclic)
- Because we can arrive at some illogical conclusion with P being false, P cannot be false. Must be true.

An example of proof by contradiction in EECS 376

- Statement: “Problem P **is not** solvable by algorithm A”
- Proof by contradiction:
  - Suppose problem P **is** solvable by algorithm A
  - [something happens]
  - which means [some truth breaks down, e.g., Pigeonhole Principle]
    - 376 pigeonholes, none of the 377 pigeons share the same pigeonhole
  - Contradiction. Therefore, P must be true.

We’ll see similar idea in Unit 2: Computability

Worksheet Problem 1.2 (if time)

Definition: A number is irrational if it *cannot* be written in the form a/b where a and b are distinct integers with no common divisors.

Theorem: Let p be a prime. An integer a is divisible by p if and only if a^2 is divisible by p.

Prove by contradiction that $\sqrt{7}$ is irrational.

- Suppose for the sake of contradiction that $\sqrt{7} = a/b$ for some integers a and b
- Then $7 = a^2/b^2 \Rightarrow 7b^2 = a^2$. So a^2 is divisible by 7.
- Since 7 is a prime, then it must be that a is also divisible by 7.
- Let $a = 7k$ for some integer k. So $7b^2 = (7k)^2 = 49k^2 \Rightarrow b^2 = 7k^2$. So b^2 is divisible by 7.
- Again since 7 is a prime, then it must be that b is also divisible by 7.
- Contradiction: a and b has a common divisor.

Worksheet Problem 1.1(a)

Determine what is incorrect about the proposed proof.

- **Claim:** There are no even primes greater than 2.
- **Proof:** 3 is a prime that is greater than 2 and it is not even, so there must be no even primes greater than 2

Problem: Attempt to prove a for all statement with a counterexample, what about other primes greater than 2?

Worksheet Problem 1.1(b)

Determine what is incorrect about the proposed proof.

- **Claim:** If n^2 is even, then n is even.
- **Proof:** If n is even, then $n = 2k$ for some integer k. Therefore $n^2 = 4k^2 = 2 \cdot 2k^2$.

Problem: Wrong direction!

Worksheet Problem 1.1(d)

Determine what is incorrect about the proposed proof.

- **Claim:** 37 is the largest prime smaller than 41.
- **Proof:** 37 is prime since its only divisors are 1 and 37.

Problem: Incomplete. Why is 37 the *largest* prime smaller than 41? Do we have other prime that is in between 37 and 41?

Proof By Induction

Definition: A binary string is any sequence of zero or more binary digits (0 or 1).

- Used to prove that a statement $P(n)$ is true for all positive/non-neg integers n.
- Two things to prove: base case, and inductive case
- **Example:** Prove that the number of binary strings of length k is 2^k.
  - Let $P(k)$ be the statement: The number of binary strings of length k is 2^k
  - **Base Case:** $P(0)$
    - There is one binary string of length 0, the empty string ε, and $\mathbf{2^0 = 1}$
  - **Inductive Step:** Assume $P(j)$ is true for some j
    - Let s be a binary string of length $j+1$, denoted $s_1 \dots s_j || s_{j+1}$
    - We can split s into $s_1 \dots s_j || s_{j+1}$
    - There are 2 possible bits for s_{j+1} and 2^j possible strings for $s_1 \dots s_j$
    - $\mathbf{2 \cdot 2^j = 2^{j+1}}$

Example of Proof By Induction in EECS 376

- Suppose I want to compute $x!$. Consider the following algorithm:

```
FACTORIAL(x):
    if x = 0 or x = 1 then
        return 1 // Base case
    return x · FACTORIAL(x − 1)
```

- **Application of induction:**
  - **Base case:** $0! = 1! = 1$- correct
  - **Inductive step:** Assuming the recursive call $\text{FACTORIAL}(k-1)$ is correct for some k, prove that $\text{FACTORIAL}(k)$ is correct.

We'll see this soon in Unit 1: Algorithms to prove the correctness of recursion!

Another Example of Proof By Induction in EECS 376

- A coin exchange greedy algorithm determines the minimum number of coins needed to make a specific amount of change by repeatedly selecting the highest-value coin that does not exceed the remaining amount.
- **Application of induction:** Enumerate on the coins chosen in the solution
  - **Base case:** Only one coin is chosen- it is the optimal solution
  - **Inductive step:** Assuming the first k coins chosen are part of some optimal solution, prove that the first $k+1$ coins (collectively) is also part of some optimal solution

This is known as the **exchange argument**. We'll see it soon in Greedy Algorithms!

TL;DPA

- We reviewed the proof techniques (direct, contrapositive, contradiction, and induction) that will be used in EECS 376
- We went through some common mistakes in writing proofs
- We previewed some applications of the proof techniques in EECS 376

Size of Input

Size of input

Discuss: What is the size of inputs to the following algorithms?

- Algorithm A takes in an array of k ASCII characters and print them one by one
- Algorithm B takes in an integer k and print "EECS 376" k times

- **Why different?** When we deal with arrays, for simplicity we usually implicitly assume that all elements are (or bounded by) the same size (since they are the same data type), making the total size proportional to the number of elements.
  - Example: ASCII characters are represented as 8-bit bytes with the most significant bit set to 0- **so the assumption is true**
- But this assumption may not be true in general! One example we deal with is big integers
  - int $x = 5$ (binary: 101) takes $\lceil \log_2 x \rceil = 3$ bits
  - int $y = 50$ (binary: 110010) takes $\lceil \log_2 y \rceil = 6$ bits
  - int $z = 50000$ (binary 1100 0011 0101 0000) takes $\lceil \log_2 z \rceil = 16$ bits

TL;DPA

- For an integer k, the input size is $\lceil \log_2 k \rceil = O(\log k)$.
- For an array/ set with n elements of the same data type, the input size is $O(n)$.

Asymptotic Notations

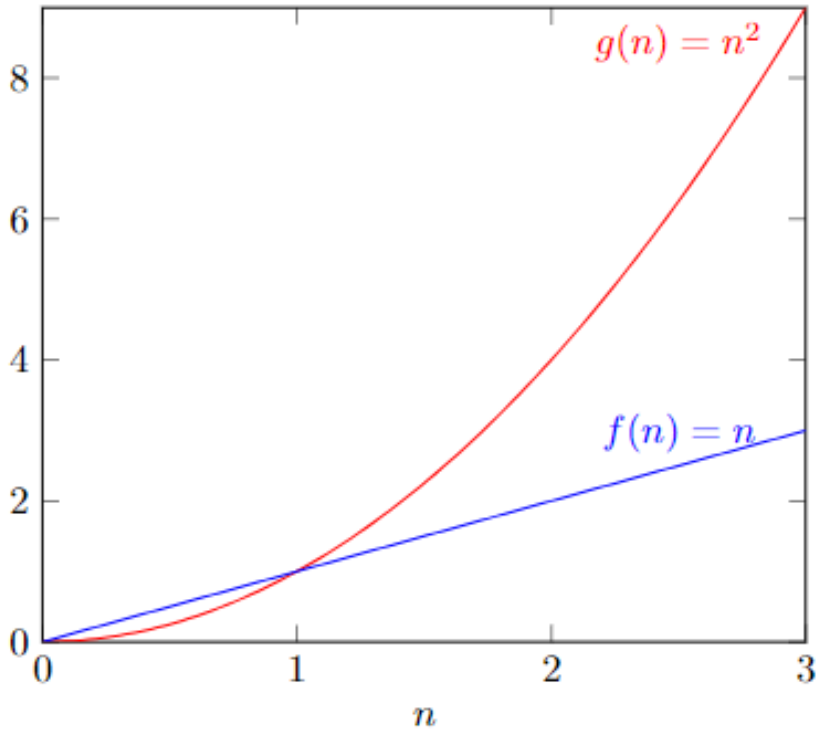
Big-O Bounds a Function from Above

Let $f(n)$ and $g(n)$ be positive functions:

- If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0$, we have $f(n) \leq M \cdot g(n)$, then we say

$$f(n) = O(g(n))$$

- For example, on the right we could pick $M = 1$ and $n_0 = 1$ to show $n = O(n^2)$

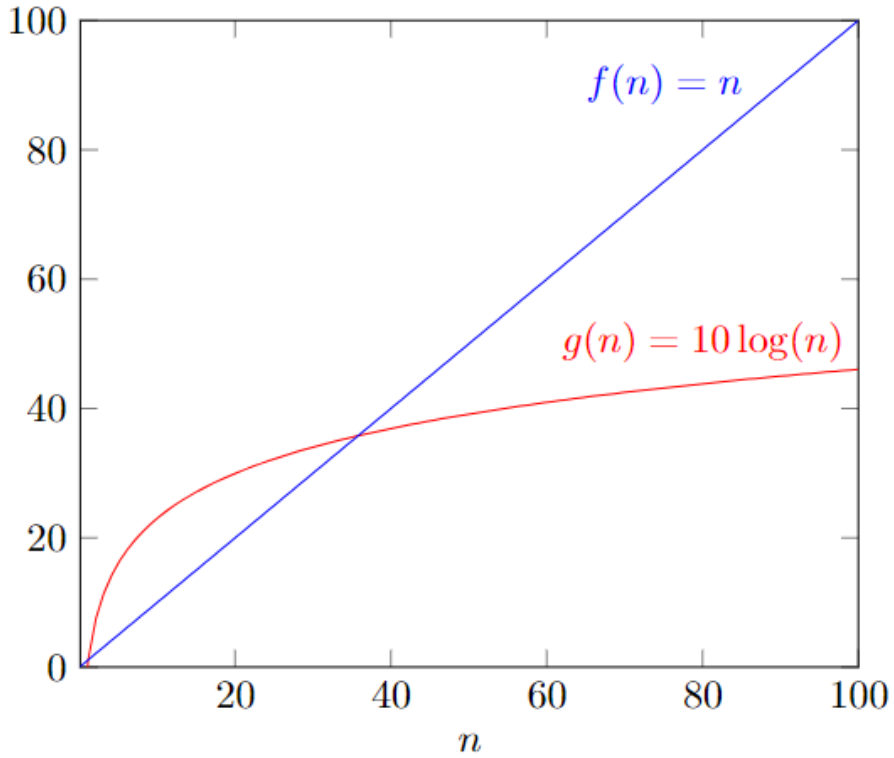


Big-Ω Bounds a Function from Below

Let $f(n)$ and $g(n)$ be positive functions:

- If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0$, we have $f(n) \geq M \cdot g(n)$, then we say

$$f(n) = \Omega(g(n))$$



Discuss: Come out with **two** pairs of n_0 and M that proves $f(n) = \Omega\big(g(n)\big)$ here.

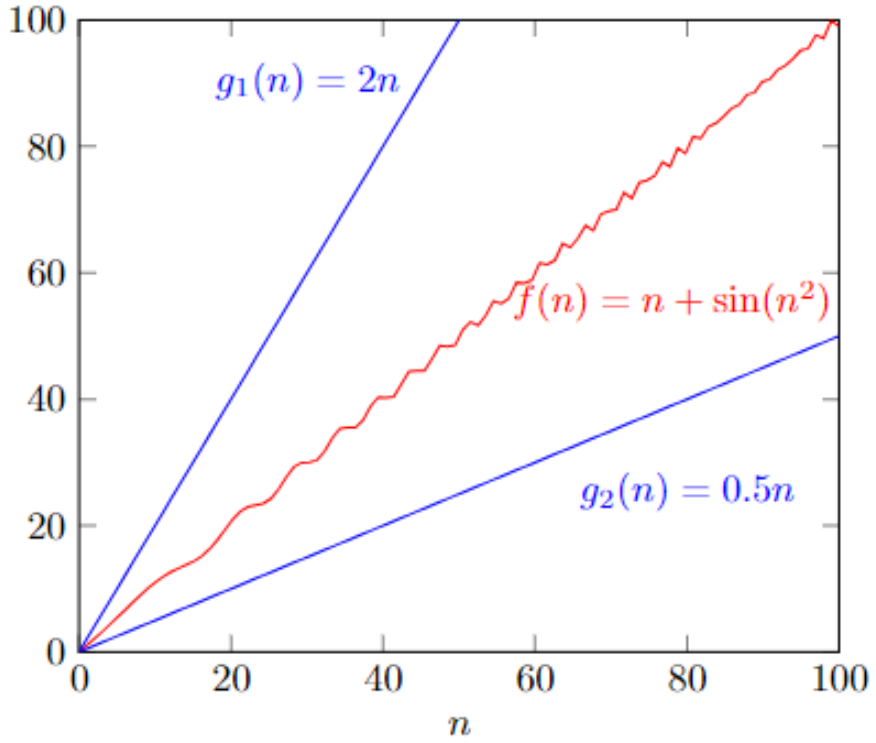
Big-Θ Bounds a Function from Both Directions

Let $f(n)$ and $g(n)$ be positive functions:

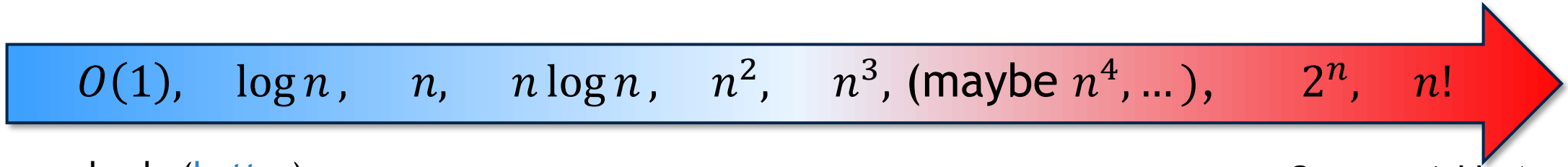
- If $f(n) = O\big(g(n)\big)$ AND $f(n) = \Omega(g(n))$, then we say

$$f(n) = \Theta(g(n))$$

- For example, if we want to prove $n + \sin(n^2) = O(n)$, set $f(n) = n + \sin(n^2)$ and $g(n) = n$, we pick $n_0 = 0$ and
  - $M = 2$ to show $\forall n \geq n_0$, $f(n) \leq M \cdot g(n) \Rightarrow f(n) = O\big(g(n)\big)$
  - $M = 0.5$ to show $\forall n \geq n_0$, $f(n) \geq M \cdot g(n) \Rightarrow f(n) = \Omega\big(g(n)\big)$



Properties of Asymptotic Notations



- Consider positive-valued functions $f_1(n) = \Theta(g_1(n))$ and $f_2(n) = \Theta(g_2(n))$
- **Addition:**

$$(f_1 + f_2)(n) = \Theta(\max\{g_1(n), g_2(n)\})$$

Keep the bigger one

- **Scalar multiplication:**

$$kf(n) = \Theta(f(n))$$

Doesn't change

- **Product:**

$$(f_1 \cdot f_2)(n) = \Theta(g_1(n) \cdot g_2(n))$$

Multiply their runtime

Methods of Proving $f(n) = O(g(n))$

- **Argue by definition:** Show the existence of M and n_0 by listing a pair that works
- **"Ratio test":** If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < c$ for some $c \in \mathbb{R}$, then $f(n) = O(g(n))$
  - Hint: L'Hopital's Rule may be helpful
- Start with a known asymptotic relationship and build a proof from there
  - It is very helpful to look at dominating terms when dealing with complex functions
- For applicable recurrence relations, use the Master theorem

Definition: If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0$, $f(n) \leq M \cdot g(n)$, then we say $f(n) = O(g(n))$

Theorem (L'Hopital's Rule): Suppose functions f and g are differentiable, for $a \in \mathbb{R}^+$ (extended reals) when

- $\lim_{x\to a} f(x) = \pm\infty$ and $\lim_{x\to a} g(x) = \pm\infty$, or
- $\lim_{x\to a} f(x) = 0$ and $\lim_{x\to a} g(x) = 0$,

then assuming the limit exists,

$$\lim_{x\to a} \frac{f(x)}{g(x)} = \lim_{x\to a} \frac{f'(x)}{g'(x)}.$$

Methods of Proving $f(n) \neq O(g(n))$

Definition: If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0$, $f(n) \leq M \cdot g(n)$, then we say $f(n) = O(g(n))$

- **Argue by definition:** Argue no M and n_0 pairs that works
- **"Ratio test":** If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty$, then $f(n) \neq O(g(n))$
  - Note: Divergence isn't sufficient! Consider $\lim_{n\to\infty} \left|\frac{\sin(n)}{1}\right|$
- Equivalently, we can also show $\lim_{n\to\infty} \left|\frac{g(n)}{f(n)}\right| = 0$.
- For applicable recurrence relations, use the Master theorem

Worksheet Problem 2.2

Determine if $f(n) = O\big(g(n)\big)$ for the following pairs of $f(n)$ and $g(n)$. Justify your answer.

- $f(n) = 4^n$ and $g(n) = 2^n$
- $f(n) = \log_a n\,, g(n) = \log_b n$ with $a, b > 0$ and $a, b \neq 1$

Definition: If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0$, $f(n) \leq M \cdot g(n)$, then we say $f(n) = O(g(n))$

Theorem: If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < c$ for some $c \in \mathbb{R}$, then $f(n) = O\big(g(n)\big)$

Theorem: If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty$, or equivalently $\lim_{n\to\infty} \left|\frac{g(n)}{f(n)}\right| = 0$, then $f(n) \neq O\big(g(n)\big)$

Worksheet Problem 2.2(a) Solution

Determine if $f(n) = O\big(g(n)\big)$ for the following pairs of $f(n)$ and $g(n)$. Justify your answer.

- $f(n) = 4^n$ and $g(n) = 2^n$
- $f(n) = \log_a n\,, g(n) = \log_b n$ with $a, b > 0$ and $a, b \neq 1$

Solution: No.

$$\begin{aligned}\lim_{n\to\infty} \frac{g(n)}{f(n)} &= \lim_{n\to\infty} \frac{2^n}{2^n \cdot 2^n} \\ &= \lim_{n\to\infty} \frac{1}{2^n} \\ &= 0.\end{aligned}$$

Thus, $g(n) = o(f(n))$, which implies that $f(n) \neq O(g(n))$.

Definition: If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0$, $f(n) \leq M \cdot g(n)$, then we say $f(n) = O(g(n))$

Theorem: If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < c$ for some $c \in \mathbb{R}$, then $f(n) = O\big(g(n)\big)$

Theorem: If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty$, or equivalently $\lim_{n\to\infty} \left|\frac{g(n)}{f(n)}\right| = 0$, then $f(n) \neq O\big(g(n)\big)$

Worksheet Problem 2.2(b) Solution

Determine if $f(n) = O(g(n))$ for the following pairs of $f(n)$ and $g(n)$. Justify your answer.

- $f(n) = 4^n$ and $g(n) = 2^n$
- $f(n) = \log_a n, g(n) = \log_b n$ with $a, b > 0$ and $a, b \neq 1$

Solution: Yes. Setting $M = \frac{\ln b}{\ln a}$ and $n_0 = 1$;

$$\begin{aligned} f(n) &= \log_a n \\ &= \frac{\ln n}{\ln a} \\ &= \frac{\ln n}{\ln a} \cdot \frac{\ln b}{\ln b} \\ &= \frac{\ln b}{\ln a} \cdot \frac{\ln n}{\ln b} \\ &= \frac{\ln b}{\ln a} \cdot g(n) \end{aligned}$$

Definition: If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0$, $f(n) \leq M \cdot g(n)$, then we say $f(n) = O(g(n))$

Theorem: If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < c$ for some $c \in \mathbb{R}$, then $f(n) = O(g(n))$

Theorem: If $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty$, or equivalently $\lim_{n\to\infty} \left|\frac{g(n)}{f(n)}\right| = 0$, then $f(n) \neq O(g(n))$

Worksheet Problem 2.1 (if time)

- On ethe same computer with the same input, an algorithm having $O(n \log n)$ time complexity runs in less time than all/ some/ no algorithms having $O(n^2)$ time complexity.
  - Hint: Consider small n

Worksheet Problem 2.1 (if time)

- On ethe same computer with the same input, an algorithm having $O(n \log n)$ time complexity runs in less time than all/ some/ no algorithms having $O(n^2)$ time complexity.
  - Some
  - Say algorithm A takes $376n \log n = O(n \log n)$ steps and algorithm B takes $n^2 = O(n^2)$ steps
    - Algorithm A takes more steps for small values of n
    - Ex for $n = 2$, A takes 752 steps while B takes 4 steps
  - Say algorithm A takes $n \log n = O(n \log n)$ steps and algorithm B takes $376n^2 = O(n^2)$ steps
    - Algorithm A takes less steps for small values of n
    - Ex for $n = 2$, A takes 2 steps while B takes 1504 steps

Let’s talk more about EECS 376!

An Overview of EECS 376

- **Is this a math class?** We are borrowing this nice tool called *proof* from math to explain ideas in CS
- **How much math do I need?** Little to none. We don't expect you to do two-pages-long algebra or calculus in this class. It's more about the formality of math language.
- **Why is this class called *Foundations* of CS?** Don't get deceived! *Foundation ≠ introductory*! It means we are dealing with the *foundational problems* in the discourse of CS (e.g., what problems can/ can't computer solve?)

Myth or truth? EECS 376 is part 2 of EECS 203

Is EECS 376 part 2 of EECS 203?

We DO need stuff from 203, but the two courses are fundamentally different

- **Throughout the course**
  - Proof methods- will see today
  - Asymptotic notation- will see today
  - Logical expressions
  - Set theory
  - Graph theory
- **Part 1: Design and Analysis of Algorithm**
  - Recursion + Master Theorem
- **Part 2: Computability**
  - Countability and Diagonalization
  - Pigeonhole Principle

Is EECS 376 part 2 of EECS 203?

We DO need stuff from 203, but the two courses are fundamentally different

- **Part 3: Complexity**
  - Function (mapping)
- **Part 4: Randomness in Algorithms**
  - Counting
  - Discrete Probabilities
  - Expectation
- **Part 5: Cryptography**
  - Modular Arithmetic
  - Divisibility and Modular Inverses

Back Matter

What's a Good Proof?

- The number one concern in the first HW is always "Am I writing this proof right?"
- There's no one right way to write a proof!
  - All that matters is that it explains clearly and logically how you've solved the problem
  - This often requires a mix of math and written words, you choose the balance!
- Some suggestions
  - Start by claiming what you'll prove and with what method
  - When creating notation, choose names that are intuitive to follow
  - Read over your proof out loud (where no one will hear)
    and make sure it makes sense to you

Additional Resources

- Handout 0: Running Time and Asymptotics
- Handout 1: Good Writing and Induction
- Desmos for visualizing asymptotics

The Fight Against Imposter Syndrome

- The beginning of a new semester is extremely overwhelming
  - You're entering a class that has a reputation for being difficult
  - There are a million other factors in your life like work, family, health, etc

- **Always remember that you can do it!**
- **You belong in this class and in the EECS program**

- Every member of the course staff believes you can succeed and it's our job to help make that happen
  - Please take advantage of all the resources in this class – piazza, office hours, discussions, lecture notes, discussion worksheets
  - If/when you feel lost, come to office hours and talk to us! A lot of this material will not make sense the first time you see it, but it's very rewarding to understand so let us help you!!