EECS 376 Discussion 1

~~Sec 27: Th 5:30-6:30 DOW 1017~~

Sec 12: Fri 11:30-12:30 FXB 1008 (only this week!)

IA: Eric Khiu

Agenda

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

Introduction

What is EECS 376? What should I be expecting? Is this a math class? Why is it called the *Foundations* of Computer Science?

First of all… Let me introduce myself

- Eric **Khiu** (pronounced as Q)
  - Don’t get confused with Eric **Song** (Sec 28 Th 3:30 EECS 1003)
- He/him
- Junior, Math & CS
- From Malaysia (a small country in southeast Asia)
- New here, just took this class last semester
- OH: Tue 5:30-6:30 MLB B116, Wed 1:00-2:30 DOW 1013
- E-mail: erickhiu@umich.edu
  - Would be nice if you could start the subject with [EECS 376]

Introduction to EECS 376

- **Is this a math class?** My impression of this class is 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 deceive! *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?)
- **Rumor: 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**
  - Diagonalization
  - Countability
  - 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**
  - Discrete Probabilities
  - Expectation
- **Part 5: Cryptography**
  - Modular arithmetic

Proof Methods

Goal: Review common proof strategies from EECS 203 that will be used in this class.

Direct Proof $(P \implies Q)$

- 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

$$\boxed{\textit{Prove that } x = y \textit{ if and only if } xy = \frac{(x+y)^2}{4}}$$

P Q

- Direction 1: Assume $x = y$, show $xy = \frac{(x+y)^2}{4}$
  - If $x = y$, then $xy = x^2 = \frac{(2x)^2}{4} = \frac{(x+y)^2}{4}$

- Direction 2: Assume $xy = \frac{(x+y)^2}{4}$,show $x = y$

  $4xy = (x+y)^2 = x^2 - 2xy + y^2$,so $0 = x^2 - 2xy + y^2 = (x-y)^2$,$x = y$

Examples of Direct Proofs in this class

- If the algorithm is designed as such, then it's *efficient* (runtime analysis)
  - Note: Definition of *efficient* may differ
- Problem X is solvable by computers if and only if Problem Y is solvable by computers
- If A has this information, then they can decrypt the secret message

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

- 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)

$$\boxed{\textit{Let } a, b \in \mathbb{Z}, \textit{ and suppose } \underbrace{ab \textit{ is odd}}_{P}\textit{. Then } \underbrace{a \textit{ and } b \textit{ are both odd}}_{Q}}$$

- Contrapositive: If $\underbrace{\text{at least one of } a \text{ or } b \text{ are even}}_{\neg Q}$, then $\underbrace{ab \text{ is even}}_{\neg P}$

- Without loss of generality (WLOG), assume a is even
  - Then $a = 2k$ for some integer k
  - So $ab = 2kb$, which is divisible by 2, and therefore even

An example of Proof by Contrapositive ($\neg Q \Rightarrow \neg P$) in this class

- Statement: If algorithm A solves X, then it solves Y
- Contrapositive: If algorithm A *doesn’t* solve Y, then it *doesn’t* solve X

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 this class

- 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.

Worksheet Problem 5 (if time)

Theorem. Two integers a and b are *congruent modulo* n,

$$a \equiv b \pmod{n},$$

if n divides $b - a$. Equivalently, $b = a + nk$ for some integer k.

Theorem. If $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, then $a + b \equiv a' + b' \pmod{n}$ and $ab \equiv a'b' \pmod{n}$.

5. **Modular Arithmetic**
   Show that the following modular congruence is a contradiction given that x is an integer. That is, show that there is no integer x such that the following congruence is satisfied.

$$11x + 2 \equiv 3x + 3 \pmod{4}$$

Worksheet Problem 5 (if time)

Solution: Suppose toward contradiction that the stated congruence can be satisfied by some integer x. Simplifying:

$$11x + 2 \equiv 3x + 3 \pmod 4 \tag{1}$$
$$8x \equiv 1 \pmod 4 \tag{2}$$
$$0 \equiv 1 \pmod 4 \tag{3}$$

Step (1) to step (2) is justified since modular arithmetic allows any identical additions and subtractions to both sides in the same way that one could add or subtract the same quantity from each side of an equation. Step (2) to step (3) is justified because when working in $\pmod 4$, all addition or subtraction of integer multiples of 4 will have no affect on the overall equivalence. In this way, subtracting $8x$ from the LHS had no affect (i.e. (2) is true if and only if (3) is true) since $8x = 4 \cdot (2x)$ and x is an integer. Lastly, (3) is a direct contradiction since it is not true that $0 \equiv 1 \pmod 4$. Therefore, the stated congruence cannot be satisfied by any integer x.

Worksheet Problem 3

3. **Proof Misconceptions**
   For each of the following, determine what is incorrect about the proposed proof technique.

   (a) 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?

   (b) 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!

   (d) 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?

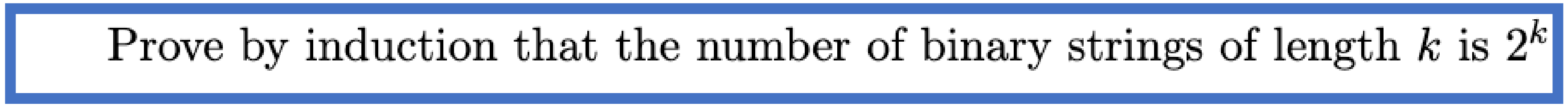
Proof By Induction

- 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

(1) $P(1)$ is true.

(2) If k is any positive integer and $P(k)$ is true, then $P(k+1)$ is true as well.

Proof By Induction- Base case


Prove by induction 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
  - We want to prove this statement for all $k \in \mathbb{Z}^+ \cup \{0\}$

- Base Case: $P(0)$
  - There is one binary string of length 0, the empty string ε, and $\mathbf{2^0 = 1}$ **so this works!**

Proof By Induction- Base case

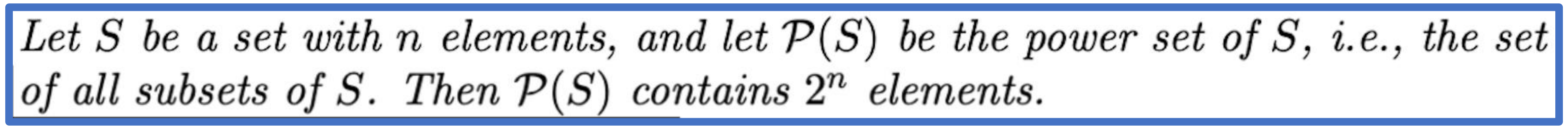
Prove by induction that the number of binary strings of length k is 2^k

- Let $P(k)$ be the statement: The number of binary strings of length k is 2^k
  - We want to prove this statement for all $k \in \mathbb{Z}^+ \cup \{0\}$

- Inductive Step: Assume $P(j)$ is true for some j
  - Let s be a binary string of length $j+1$, denoted $s_0 s_1 \dots s_j$
  - We can split s into $s_0 || s_1 \dots s_j$
  - There are 2 possible bits for s_0 0) or 1), and 2^j possible strings for $s_1 \dots s_j$

    $2 \cdot 2^j = 2^{j+1}$ so this works!

Proof By Induction Exercise (if time)



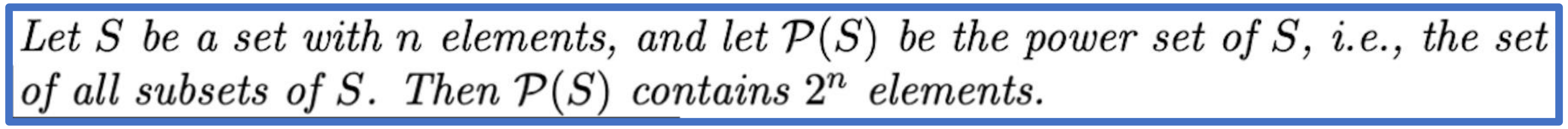
Let S be a set with n elements, and let $\mathcal{P}(S)$ be the power set of S, i.e., the set of all subsets of S. Then $\mathcal{P}(S)$ contains 2^n elements.

- Let T(n) be the statement "a set S of size n has 2^n subsets"
  - We want to prove this statement for all $n \in \mathbb{Z}^+ \cup \{0\}$

- Base case: $\mathrm{n} = 0,\ S = \emptyset$ (the empty set)

  S has one subset: $\emptyset$, $|\mathcal{P}(S)| = 1 = 2^0$

Proof By Induction Exercise (if time)


Let S be a set with n elements, and let $\mathcal{P}(S)$ be the power set of S, i.e., the set of all subsets of S. Then $\mathcal{P}(S)$ contains 2^n elements.

- Inductive Case: Assume T(k) is true for some non-negative integer k. Our goal is to show the truth of T$(k+1)$
- Let S be a set of size $k+1$, and let a be some element in S
- Every element of $\mathcal{P}(S)$ is a subset that either contains a or doesn't
  - The number of subsets **without** a is $|\mathcal{P}(S \setminus \{a\})| = 2^k$
  - The number of subsets **with** a is the same as without,
    For each subset that doesn't contain a, we form one that does by adding a
- Number of subsets = without + with = $2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$

Size of Input

Goal: Familiarize ourselves with the concept of data as bitstrings

Size of input

- In this class, we often describe the complexity of an algorithm with respect to its input size in terms of bits (not its value!)
- Algorithm A takes in an array of 10 ASCII characters and print them one by one
  - What is the input size?
- Algorithm B takes in an integer k and print “EECS 376” k times
  - What is the input size?
- 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

Breaktime!

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 Resource on Proofs

Handout 1: Good Writing and Induction

(Course google drive → Handouts)

- This handout covers general proof-writing suggestions, inductive proofs, and proofs by contradiction
- Credit to Ran Libeskind-Hadas

Asymptotic Notation

Goal: build a set of tools for proving or disproving asymptotic relationships between functions (often, but not always, runtimes)

Asymptotic Notation

- We'll use asymptotic notation to loosely describe the amount of some resource (time, space, etc.) an algorithm uses

- We do this by comparing the algorithm's approximate use of that resource to other functions

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 $n \geq n_0,\ f(n) \leq M \cdot g(n)$, then we say $f(n) = O(g(n))$
- Ex. $n = O(n^2)$, we could choose $n_0 = 1, M = 1$ to show this

Big-Ω Bounds a Function from Below

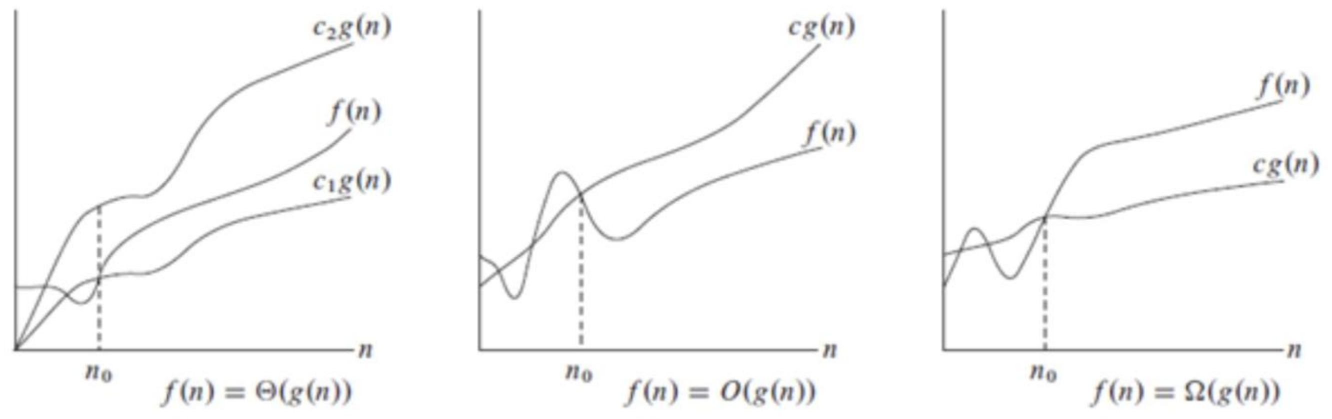
Let $f(n)$ and $g(n)$ be positive functions:

- If there exist constants $M > 0$ and n_0 such that for all $n \geq n_0,\ f(n) \geq M \cdot g(n)$, then we say $f(n) = \Omega(g(n))$
- Ex. $n^2 = \Omega(n)$, we could choose $n_0 = 1, M = 1$ to show this

Big-θ Bounds a Function in Both Directions

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

- If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then we say $f(n) = \Theta(g(n))$
- Ex. $f(n) = 3x^2 + 2, \quad g(n) = 4x^2 + 7$



Summary of Big-Θ, Big-O, and Big-Ω

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

- For applicable recurrence relations, use the Master theorem
- $\lim_{n\to\infty} \left| \frac{f(n)}{g(n)} \right| < c \text{ for a } c \in \mathbb{R}$ (L'Hopital's rule is often useful)
- Argue by definition: $\exists\, n_0, M \in \mathbb{R}\ s.t. \forall n > n_0\ |f(n)| \leq M|g(n)|$
  (show the existence of M and n_0 by listing a pair that works)
- 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

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

- If dealing with an applicable recursive relation, build your argument from the result of the master theorem
- $\lim_{n\to\infty} \left| \frac{f(n)}{g(n)} \right| = \infty$ Note that divergence isn't sufficient, consider $\lim_{n\to\infty} \left| \frac{\sin(n)}{1} \right|$
- $\lim_{n\to\infty} \left| \frac{g(n)}{f(n)} \right| = 0$ (Same idea as above: f grows strictly faster than g)
- Argue that no such c and n_0 exist according to the definition

Worksheet Problem 2

2. **Time Complexity - Applied**
   For the following pairs of $f(n)$ and $g(n)$, is $f(n) = O(g(n))$? Justify your answer by applying the definition of big-O or by applying a limit argument.

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

Worksheet Problem 2

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

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

Worksheet Problem 2

2. **Time Complexity - Applied**
   For the following pairs of $f(n)$ and $g(n)$, is $f(n) = O(g(n))$? Justify your answer by applying the definition of big-O or by applying a limit argument.

   - $f(n) = 4^n$, $g(n) = 2^n$

Worksheet Problem 2

Solution: No.

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

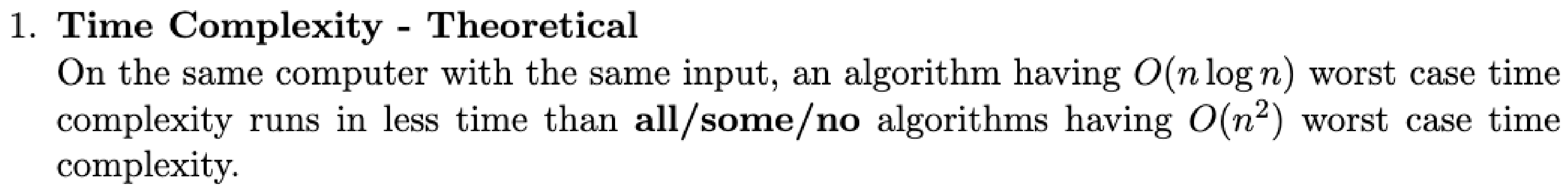
Thus, $g(n) = o(f(n))$, which implies that $f(n) \neq O(g(n))$.

Worksheet Problem 1 (if time)

1. **Time Complexity - Theoretical**
   On the 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 1



1. **Time Complexity - Theoretical**
   On the same computer with the same input, an algorithm having $O(n \log n)$ worst case time complexity runs in less time than **all/some/no** algorithms having $O(n^2)$ worst case 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

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!!