Monday: Time Complexity

In practice, computers (and Turing machines) don't have infinite tape, and we can't afford to wait unboundedly long for an answer. "Decidable" isn't good enough - we want "Efficiently decidable".

For a given algorithm working on a given input, how long do we need to wait for an answer? How does the running time depend on the input in the worst-case? average-case? We expect to have to spend more time on computations with larger inputs.

A language (L) is **recognizable** if there's a Turing machine M that recognizes it, $L(M) = L$

A language is **decidable** if there's a (decider) Turing machine M that halts for all input strings and recognizes it.

A language is **efficiently decidable** if there's a Turing mach that halts "quickly" for all input string and recognizes the language

A function is **computable** if there's a Turing machine that computes it, i.e. halts on each input and when it does, contents of tape are $f(x)$ followed by blanks (for computation on input x)

A function is **efficiently computable** if --- "quickly" ---

Definition (Sipser 7.1): For M a deterministic decider, its **running time** is the function $f : \mathbb{N} \to \mathbb{N}$ given by

worst case

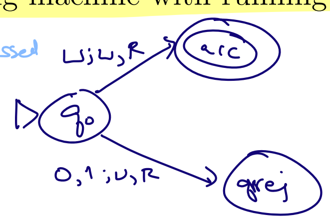
$$f(n) = \text{max number of steps } M \text{ takes before halting, over all inputs of length } n$$

Definition (Sipser 7.7): For each function $t(n)$, the **time complexity class** $TIME(t(n))$, is defined by

$$TIME(t(n)) = \{L \mid L \text{ is decidable by a Turing machine with running time in } O(t(n))\}$$

collection of functions.

An example of an element (L) of $TIME(1)$ is $\{\varepsilon\}$ witnessed by



An example of an element of $TIME(n)$ is $\{w0 \mid w \in \Sigma^*\}$ "on input w
1. Scan until first ⊔
2. Go back one cell to left, if 0 accept; if ⊔ or 1 reject"

Note: $TIME(1) \subseteq TIME(n) \subseteq TIME(n^2)$

$O(\cdot)$ upper bound

Definition (Sipser 7.12) : P is the class of languages that are decidable in polynomial time on a deterministic 1-tape Turing machine

$$P = \bigcup_k TIME(n^k)$$

Compare to exponential time: brute-force search. Usually problems in P have algorithms that are more clever than brute force.

Theorem (Sipser 7.8): Let $t(n)$ be a function with $t(n) \geq n$. Then every $t(n)$ time deterministic multitape Turing machine has an equivalent $O(t^2(n))$ time deterministic 1-tape Turing machine.

Definition (Sipser 7.1): For M a deterministic decider, its **running time** is the function $f : \mathbb{N} \to \mathbb{N}$ given by

$$f(n) = \text{max number of steps } M \text{ takes before halting, over all inputs of length } n$$

Definition (Sipser 7.7): For each function $t(n)$, the **time complexity class** $TIME(t(n))$, is defined by

$$D\ TIME(t(n)) = \{L \mid L \text{ is decidable by a Turing machine with running time in } O(t(n))\}$$

Definition (Sipser 7.12) : P is the class of languages that are decidable in polynomial time on a deterministic 1-tape Turing machine

$$P = \bigcup_k {}^{D}TIME(n^k)$$

Definition (Sipser 7.9): For N a nodeterministic decider. The **running time** of N is the function $f : \mathbb{N} \to \mathbb{N}$ given by

$$f(n) = \text{max number of steps } N \text{ takes on any branch before halting, over all inputs of length } n$$

Definition (Sipser 7.21): For each function $t(n)$, the **nondeterministic time complexity class** $NTIME(t(n))$, is defined by

$$NTIME(t(n)) = \{L \mid L \text{ is decidable by a nondeterministic Turing machine with running time in } O(t(n))\}$$

$$NP = \bigcup_k NTIME(n^k)$$

True or **False**: $TIME(n^2) \subseteq NTIME(n^2)$

True or **False**: $NTIME(n^2) \subseteq TIME(n^2)$

?

Given nondet TM, what bound can we prove on running time of deterministic TM that simulates it?

Every problem in NP is decidable with an exponential-time algorithm

Nondeterministic approach: guess a possible solution, verify that it works.

Brute-force (worst-case exponential time) approach: iterate over all possible solutions, for each one, check if it works.

choices of NTM TM.

Examples in P

data structure

Can't use nondeterminism; Can use multiple tapes; Often need to be "more clever" than naïve / brute force approach

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is digraph with } n \text{ nodes there is path from s to t}\}$$

Use breadth first search to show in P

A_{DFA}

E_{DFA}

EQ_{DFA}.

$$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime integers}\}$$

Use Euclidean Algorithm to show in P

$$L(G) = \{w \mid w \text{ is generated by } G\}$$

(where G is a context-free grammar). Use dynamic programming to show in P.

CSE101.

Examples in NP

"Verifiable" i.e. NP, Can be decided by a nondeterministic TM in polynomial time, best known deterministic solution may be brute-force, solution can be verified by a deterministic TM in polynomial time.

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is digraph with n nodes, there is path from s to t that goes through every node exa

$$VERTEX-COVER = \{\langle G, k\rangle \mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-node vertex cover}\}$$

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-clique}\}$$

$$SAT = \{\langle X\rangle \mid X \text{ is a satisfiable Boolean formula with } n \text{ variables}\}$$

Undecidable

Decidable langs

Problems in P	**Problems in NP**
(Membership in any) regular language	Any problem in P
(Membership in any) context-free language	
A_{DFA}	SAT
E_{DFA}	$CLIQUE$
EQ_{DFA}	$VERTEX-COVER$
$PATH$	$HAMPATH$
$RELPRIME$	...
...	

Notice: $NP \subseteq \{L \mid L \text{ is decidable}\}$ so $A_{TM} \notin NP$

Million-dollar question: Is $P = NP$?

One approach to trying to answer it is to look for *hardest* problems in NP and then (1) if we can show that there are efficient algorithms for them, then we can get efficient algorithms for all problems in NP so $P = NP$, or (2) these problems might be good candidates for showing that there are problems in NP for which there are no efficient algorithms.

efficiently decidable languages

Wednesday: P vs. NP

← efficiently verifiable language.

Definition (Sipser 7.29) Language A is **polynomial-time mapping reducible** to language B, written $A \leq_P B$, means there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for every $x \in \Sigma^*$

$$x \in A \quad \text{iff} \quad f(x) \in B.$$

running time of algorithm that halts and outputs f(x) when given x is polynomial.

The function f is called the polynomial time reduction of A to B.

Theorem (Sipser 7.31): If $\overset{①}{A \leq_P B}$ and $\overset{②}{B \in P}$ then $A \in P$.

Proof:

Recall { If $A \leq_m B$ and B decidable then A is decidable
If $A \leq_m B$ and B recognizable then A is recognizable

Goal: Build a TM that runs in polynomial time & decides A

Given ① TM F that computes the function witnessing $A \leq_p B$

i.e. for each x, run F on x and in polynomial many step F halts and output $F(x)$ so that $x \in A$ iff $F(x) \in B$.

and ② TM M_B that runs in polynomial time & decides B.

Define M_A = " On input x

polynomial 1. Compute $F(x)$.

polynomial 2. Run M_B on $F(x)$ — length of $F(x)$ is polynomial in length of x

constant 3. If accepts, accept; if rejects, reject "

Claim ① $L(M_A) = A$
② M_A decides
③ M_A halt in polynomial time for each input.

Definition (Sipser 7.34; based in Stephen Cook and Leonid Levin's work in the 1970s): A language B is **NP-complete** means (1) B is in NP **and** (2) every language A in NP is polynomial time reducible to B.

Theorem (Sipser 7.35): If B is NP-complete and $B \in P$ then $P = NP$.

Proof: Goal: $P = NP$, i.e. $\underbrace{P \subseteq NP}_{\text{known}}$ and $\underbrace{NP \subseteq P}_{\text{need}}$.

Consider arbitrary set in NP, call it A. WTS $A \in P$.

Given B is NP-complete and $B \in P$.

By definition of NP-complete, since A is in NP

$A \leq_p B$.

Use theorem 7.31 to combine $A \leq_p B$ and assumption that $B \in P$ to get $A \in P$.

3SAT: A literal is a Boolean variable (e.g. x) or a negated Boolean variable (e.g. $\bar{x}$). A Boolean formula is a **3cnf-formula** if it is a Boolean formula in conjunctive normal form (a conjunction of disjunctive clauses of literals) and each clause has three literals.

$$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$$

string coding

Example string in $3SAT$

$$\langle(\underbrace{x \vee \bar{y} \vee \bar{z}}_{\text{ORs}}) \wedge (\underbrace{\bar{x} \vee y \vee z}_{\text{ORs}}) \wedge (\underbrace{x \vee y \vee z}_{\text{ORs}})\rangle$$

AND AND

$x = T/F$
$y = T/F$
$z = T/F$

Example string not in $3SAT$

$$\langle(x \vee y \vee z) \wedge (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})\rangle$$

all assignment of T/F to 3 variables evaluate to F.

Cook-Levin Theorem: $3SAT$ is NP-complete.

Are there other NP-complete problems? To prove that X is NP-complete

- *From scratch*: prove X is in NP and that all NP problems are polynomial-time reducible to X.
- *Using reduction*: prove X is in NP (1) and that a known-to-be NP-complete problem (2) is polynomial-time reducible to X.

i.e. WTS $3\text{-SAT} \leq_p X$

Why enough? $\leq_p$, $\leq_m$ are transitive!

i.e. $A \leq_m B$ and $B \leq_m C$ guarantees $A \leq_m C$

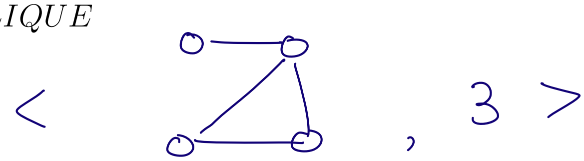
and $A \leq_p B$ and $B \leq_p C$ guarantees $A \leq_p C$.

nodes, edges.

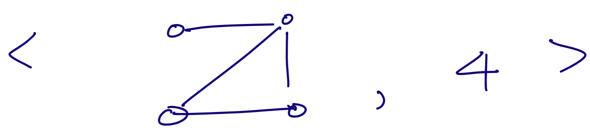
CLIQUE: A ***k*-clique** in an undirected graph is a maximally connected subgraph with k nodes.

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Example string in $CLIQUE$



Example string not in $CLIQUE$



hence CLIQUE is NP-complete.

Theorem (Sipser 7.32):

$$3SAT \leq_P CLIQUE$$

i.e. WTS there's an eff. comp. F <formulas> to <graphs, num>

Given a Boolean formula in conjunctive normal form with k clauses and three literals per clause, we will map it to a graph so that the graph has a clique if the original formula is satisfiable and the graph does not have a clique if the original formula is not satisfiable.

The graph has $3k$ vertices (one for each literal in each clause) and an edge between all vertices except

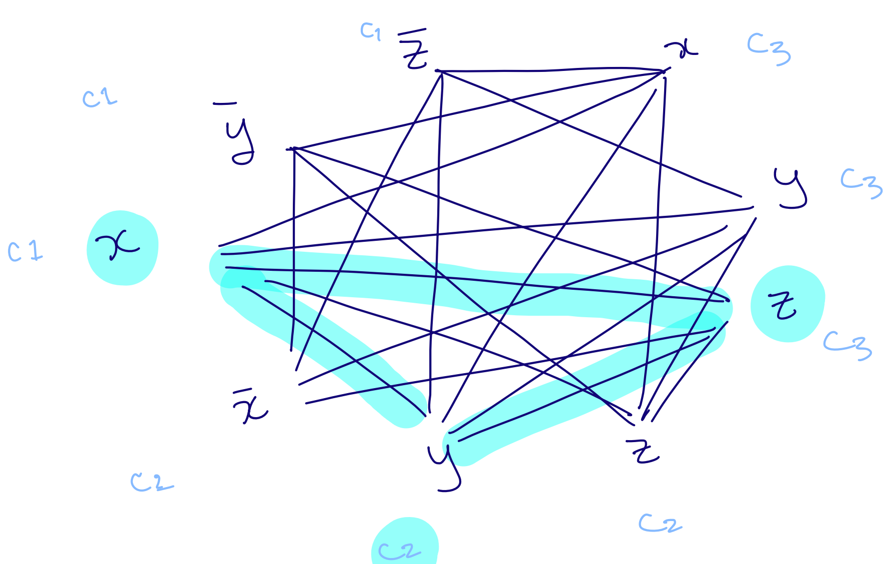
- vertices for two literals in the same clause
- vertices for literals that are negations of one another

clauses.

* k = 3

size of clique.

Example: $\underbrace{(x \vee \bar{y} \vee \bar{z})}_{c1} \wedge \underbrace{(\bar{x} \vee y \vee z)}_{c2} \wedge \underbrace{(x \vee y \vee z)}_{c3}$



Friday: Review

	Model of Computation	Class of Languages
Ch1	**Deterministic finite automata**: formal definition, how to design for a given language, how to describe language of a machine? **Nondeterministic finite automata**: formal definition, how to design for a given language, how to describe language of a machine? **Regular expressions**: formal definition, how to design for a given language, how to describe language of expression? *Also*: converting between different models.	**Class of regular languages**: what are the closure properties of this class? which languages are not in the class? using **pumping lemma** to prove nonregularity.
Ch2	**Push-down automata**: formal definition, how to design for a given language, how to describe language of a machine? **Context-free grammars**: formal definition, how to design for a given language, how to describe language of a grammar?	**Class of context-free languages**: what are the closure properties of this class? which languages are not in the class?
Ch7	Turing machines that always halt in polynomial time Nondeterministic Turing machines that always halt in polynomial time	P NP
Ch3 +4 +5	**Deciders** (Turing machines that always halt): formal definition, how to design for a given language, how to describe language of a machine?	**Class of decidable languages**: what are the closure properties of this class? which languages are not in the class? using diagonalization and mapping reduction to show undecidability
Ch3 +4 +5	**Turing machines** formal definition, how to design for a given language, how to describe language of a machine?	**Class of recognizable languages**: what are the closure properties of this class? which languages are not in the class? using closure and mapping reduction to show unrecognizability

Given a language, prove it is regular

Strategy 1: construct DFA recognizing the language and prove it works.

Strategy 2: construct NFA recognizing the language and prove it works.

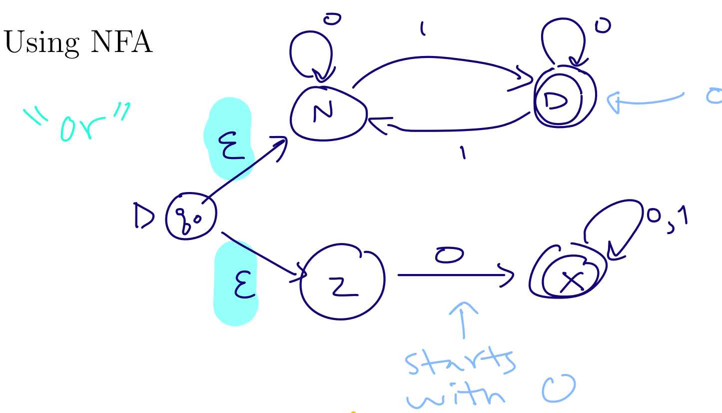
Strategy 3: construct regular expression recognizing the language and prove it works.

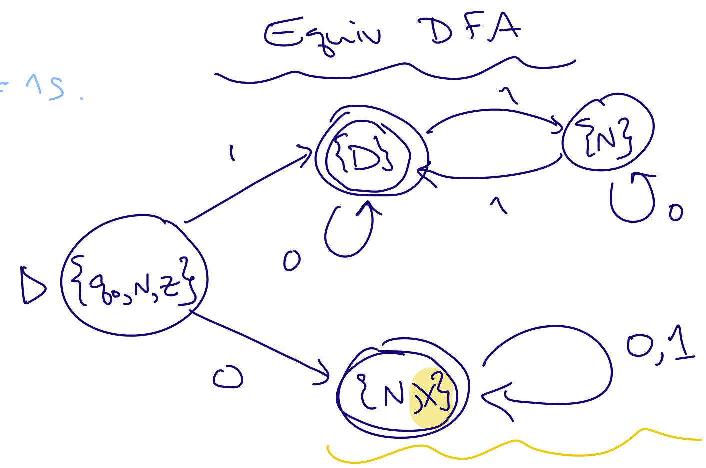
"Prove it works" means ...

extra practice

Example: $L = \{w \in \{0,1\}^* \mid w \text{ has odd number of 1s or starts with } 0\}$

Using NFA



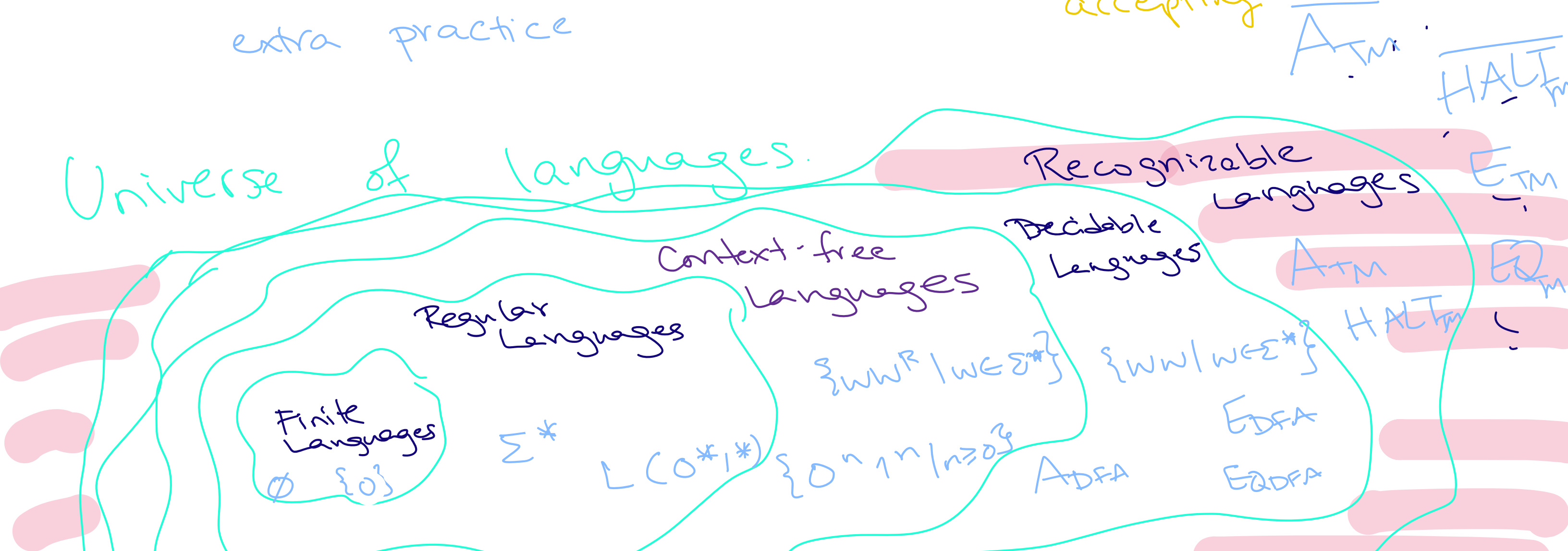


Could be in q_0 or N or Z before reading any input

optimization: macro state construction gives more states here but can collapse b/c all accepting

Using regular expressions

extra practice



non-context-free.

Example: Select all and only the options that result in a true statement: "To show a language A is not regular, we can..."

there is no Turing machine that decides A

- ~~a.~~ Show A is finite
- ~~b.~~ Show there is a CFG generating A
- c.? Show A has no pumping length
- d. Show A is undecidable

MOST FREQUENTLY USED STRATEGY.

"If A is $\boxed{\text{undecidable}}$ then A is not regular"

EQ_{TM}

In fact, all finite languages are regular.

– Build DFA.

Pumping Lemma: Every regular language has a pumping length

NOTE: There are nonregular languages that have a pumping length.

In fact, all regular languages are context-free.

e.g. $\emptyset$ is regular and context free

and $\{ww^R \mid w \in \Sigma^*\}$ is nonregular and context-free

To prove L is nonregular, can consider arbitrary positive integer p and work to prove that it is not a pumping length. WTS that there is a string that "should be pumpable" relative to L and p, but isn't

① $s \in L$ ② $|s| \geq p$ ③ $\forall x \forall y \forall z \left(s = xyz \text{ and } |xy| \leq p \text{ and } |y| > 0 \rightarrow \exists i \; xy^iz \notin L \right)$

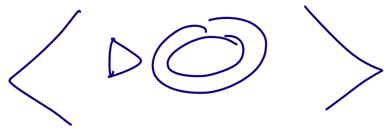
Example: What is the language generated by the CFG with rules

$$\begin{aligned} S &\to aSb \mid bY \mid Ya \\ Y &\to bY \mid Ya \mid \varepsilon \end{aligned}$$

extra practice

Example: Prove that the language $T = \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) \text{ is infinite}\}$ is undecidable.

example string in T



To show a language is undecidable can find some set known to be undecidable and prove that it mapping reduces to T.

A_{TM}, $\overline{A_{TM}}$, $HALT_{TM}$, $\overline{HALT_{TM}}$, E_{TM}, $\overline{E_{TM}}$, EQ_{TM}, $\overline{EQ_{TM}}$

Goal: Prove that $A_{TM} \leq_m T$.

We need a computable function $f : \Sigma^* \to \Sigma^*$ where $x \in A_{TM}$ iff $f(x) \in T$.

Goal

- $x \neq \langle M, w \rangle$ ... $f(x) \notin T$ if $f(x) = \langle \quad \rangle$ — $\emptyset$, lang finite.
- $x = \langle M, w \rangle$ and $w \notin L(M)$ ... $f(x) \notin T$ ... if $f(x) = \langle \quad \rangle$ — lang infinite
- $x = \langle M, w \rangle$ and $w \in L(M)$ ... $f(x) \in T$ ... $f(x) = \langle M\_ \rangle$ — Σ^*

Refer to $A_{TM} \leq_m E_{TM}$.

Define $F =$ " On input x

1. If $x = \langle M, w \rangle$ for some TM M, string w go to step 3.
2. If not, output $\langle$ "On input y, 1. Reject." $\rangle$
3. Let $x = \langle M, w \rangle$. Define

   $M'_x =$ " On input y,
   1. Run M on w
   2. If M accepts w, accept y.
   3. If M rejects w, reject y."
4. Output $\langle M'_x \rangle$ "

Claim: F computes function satisfying goal Pf: ...

Example: Prove that the class of decidable languages is closed under concatenation.

Extra practice

Week 10 at a glance

Textbook reading: Chapter 7

For Monday: Definition 7.1 (page 276)

For Wednesday: Definition 7.7 (page 279)

Make sure you can:

- Classify the computational complexity of a set of strings by determining whether it is decidable or undecidable and recognizable or unrecognizable.
  - Distinguish between computability and complexity
  - Articulate motivating questions of complexity
  - Define NP-completeness
  - Give examples of PTIME-decidable, NPTIME-decidable, and NP-complete problems
- Use mapping reduction to deduce the complexity of a language by comparing to the complexity of another.
  - Distinguish between computability and complexity
  - Articulate motivating questions of complexity
  - Use appropriate reduction (e.g. mapping, Turing, polynomial-time) to deduce the complexity of a language by comparing to the complexity of another.
  - Use polynomial-time reduction to prove NP-completeness

TODO:

Student Evaluations of Teaching forms: Evaluations are open for completion anytime BEFORE 8AM on Saturday, March 16. Access your SETs from the Evaluations site

https://academicaffairs.ucsd.edu/Modules/Evals

You will separately evaluate each of your listed instructors for each enrolled course.

NEW WINTER 2024 SET INCENTIVE LOTTERY: In Winter 2024, students who complete all of their student evaluation forms for their undergraduate course will be entered into a lottery to win one of 5 $100 Visa gift cards! To be entered into the lottery, students must complete at least one instructor evaluation for EACH of their undergraduate courses. They will be automatically entered when they have completed an instructor evaluation for all of their undergraduate courses.

Review quizzes based on class material each day; review quiz for Friday includes opportunity for feedback for course.

Homework assignment 5 due Thursday.