Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

If there exists a polynomial reduction from problem A to problem B, problem B is at least as hard as problem A. From this, we can define complexity class which sort of gauge 'hardness'.

**Complexity Definitions**

- **NP**: a decision problem in which a potential solution can be verified in polynomial time.
- **P**: a decision problem which can be solved in polynomial time.
- **NP-Complete**: a decision problem in NP which all problems in NP can reduce to.
- **NP-Hard**: any problem which is at least as hard as an NP-Complete problem.

**Prove a problem is NP-Complete**

To prove a problem is NP-Complete, you must prove the problem is in NP and it is in NP-Hard. To do this, you must show there exists a polynomial verifier, and reduce an NP-Complete problem to the problem.

### 1 NP or not NP, that is the question

For the following questions, circle the (unique) condition that would make the statement true.

(a) If \( B \) is **NP**-complete, then for any problem \( A \in \text{NP} \), there exists a polynomial-time reduction from \( A \) to \( B \).

- Always True
- True iff \( P = \text{NP} \)
- True iff \( P \neq \text{NP} \)
- Always False

(b) If \( B \) is in **NP**, then for any problem \( A \in \text{P} \), there exists a polynomial-time reduction from \( A \) to \( B \).

- Always True
- True iff \( P = \text{NP} \)
- True iff \( P \neq \text{NP} \)
- Always False

(c) 2 SAT is **NP**-complete.

- Always True
- True iff \( P = \text{NP} \)
- True iff \( P \neq \text{NP} \)
- Always False

(d) Minimum Spanning Tree is in **NP**.

- Always True
- True iff \( P = \text{NP} \)
- True iff \( P \neq \text{NP} \)
- Always False
2 2-SAT and Variants

Max-2-SAT is defined as follows. Let $C_1, \ldots, C_m$ be a collection of 2-clauses and $b$ a non-negative integer. We want to determine if there is some assignment which satisfies at least $b$ clauses.

Max-Cut is defined as follows. Let $G$ be an undirected unweighted graph, and $k$ a non-negative integer. We want to determine if there is some cut with at least $k$ edges crossing it. Max-Cut is known to be NP-complete.

Show that Max-2-SAT is NP-complete by reducing from Max-Cut. Prove the correctness of your reduction.

3 Upper Bounds on Algorithms for NP Problems

Parts a and c of this problem are solo questions. You may collaborate on part b.

(a) Recall the 3-SAT problem: we have $n$ variables $x_i$ and $m$ clauses, where each clause is the OR of at most three literals (a literal is a variable or its negation). Our goal is to find an assignment of variables that satisfies all the clauses, or report that none exists.

Give a $O(2^n m)$-time algorithm for 3-SAT. Just the algorithm description is needed.

(b) Using part (a) and the fact that 3-SAT is NP-hard, give a $O(2^n c)$-time algorithm for every problem in NP, where $c$ is a constant (that can depend on the problem). Just the algorithm description and runtime analysis is needed. An informal algorithm description is fine.

(This result is known as $NP \subseteq EXP$.)

(c) Recall the halting problem from CS70: Given a program (as e.g. a .py file), determine if the program runs forever or eventually halts. Also recall that there is no finite-time algorithm for the halting problem. Let us define the input size for the halting problem to be the number of characters used to write the program.

Given an instance of 3-SAT with $n$ variables and $m$ clauses, we can write a size $O(n + m)$ program that halts if the instance is satisfiable and runs forever otherwise. So there is a polynomial-time reduction from 3-SAT to the halting problem.

Based on this reduction and part (b): Is the halting problem NP-hard? Is it NP-complete? Justify your answer.