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.

1 NP Basics

Assume A reduces to B in polynomial time. In each part you will be given a fact about one of the problems. What information can you derive of the other problem given each fact? Each part should be considered independent; i.e., you should not use the fact given in part (a) as part of your analysis of part (b).

(a) A is in **P**.

(b) B is in **P**.

(c) A is **NP**-hard.

(d) B is **NP**-hard.

2 Exact 4-SAT

The Exact 4-SAT problem is defined as follows.

Input: n boolean variables $\{x_1, \ldots, x_n\}$ and clauses $\{C_1, \ldots, C_m\}$ with each containing exactly {four distinct} literals. For example, the following is an instance of Exact 4-SAT,

$$(x_1 \vee x_2 \vee \overline{x_4} \vee \overline{x_5}) \wedge (x_3 \vee \overline{x_4} \vee \overline{x_1} \vee x_2) \wedge (x_1 \vee x_3 \vee x_4 \vee x_5)$$

Note that all the 4 literals within a clause have to be distinct.

Goal: Find an assignment to the variables $x_1, \ldots, x_n$ that satisfies all the clauses.

(a) Give a polynomial time reduction from 3-SAT to Exact 4-SAT.

(b) Give a polynomial time reduction from Exact 4-SAT to 3-SAT.

3 Cycle Cover

In the cycle cover problem, we have a directed graph G, and our goal is to find a set of directed cycles $C_1, C_2, \ldots C_k$ in G such that every vertex appears in exactly one cycle (a cycle cannot revisit vertices, e.g. $a \to b \to a \to c \to a$ is not a valid cycle, but $a \to b \to c \to a$ is), or declare none exists.

In the bipartite perfect matching problem, we have a undirected bipartite graph (a graph where the vertices can be split into L, R, and there are no edges between two vertices in L or two vertices in R), and our goal is to find a set of edges in this graph such that every vertex is adjacent to exactly one edge in the set, or declare none exists.

Give a reduction from cycle cover to bipartite perfect matching.

(Hint: In a cycle cover, every vertex has one incoming and one outgoing edge.)

4 Public Funds

You are looking to build a new fence for your mansion, to keep out pesky people protesting profligate purchases. You have m bank accounts at your disposal to use to pay for your fence; each account i has a balance of b_i. You must choose one of n options for your fence; each fence j costs c_j dollars. You would like to withdraw from at most k of the bank accounts to build the fence, and due to peculiar UC accounting rules, if you use a particular bank account, you must use the whole balance (all b_m dollars.)

Determine whether it is possible to exactly pay for some fence j; that is, whether there is a j between 1 and n such that you can withdraw exactly c_j dollars given the bank account balances $b_1, \ldots, b_m$, the fence costs $c_1, \ldots, c_n$, and k.

Your task is to prove that Public Funds is **NP**-complete.

(a) Prove that Public Funds is in **NP**.

(b) Prove that Public Funds is **NP**-hard by providing a reduction from Subset Sum.

Note: to rigorously prove the correctness of a reduction from A to B, you must show two things:

1. *If an instance of A has a solution, then the transformed instance of B has a solution.*
2. *If an instance of B in the format of the transformation has a solution, then the corresponding instance of A has a solution.*

5 SAT and Integer Programming

Consider the 3-SAT problem, where the input is a set of clauses and each one is a OR of 3 literals. For example, $(x_1 \vee \overline{x}_4 \vee \overline{x_7})$ is a clause which evaluated to true iff one of the literals is true. We say that the input is satifiable if there is an assignment to the variables such that all clauses evaluate to true. We want to decide whether the input is satifiable.

On the other hand, consider the integer linear programming feasibility problem: We are given a set of variables and constraints in terms of these variables (we are not given an objective). The constraints are either linear inequalities, or the 0-1 constraints $x_i \in \{0, 1\}$. We want to decide if it possible to assign the variables values that satisfy all the constraints.

Give a reduction from 3-SAT to integer linear programming feasibility, and briefly justify its correctness. No runtime analysis needed.

6 (Optional) Upper Bounds on Algorithms for NP Problems

(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.

*Hint: Since 3-SAT is **NP**-hard, you can use it to solve any problem in **NP**!*

Note: 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.