This homework is due on **November 14, 9am ET**.

You are welcome to work with others, however you must explicitly list all collaborators and materials that you used. You must write up your own solution and your own code to every problem. See Georgetown University Honor System. When in doubt, ask the instructor what is allowed.

Problem 1 (**NP**-completeness of 3-SAT)**.** In Lecture 15, we proved that the Satisfiability problem (SAT) is **NP**-complete. The goal of this exercise is to prove that even the special case of SAT where all clauses have length at most 3 is still **NP**-complete. For this, we need to design a polynomial-time reduction from SAT to 3-SAT.

Provide a polynomial-time algorithm that takes as input a SAT formula ϕ (with arbitrarily long clauses) and outputs an equisatisfiable 3-SAT formula ψ. Note that this algorithm is allowed to introduce new variables, the only requirements are that (i) the algorithm runs in polynomial time, (ii) every clause of ψ has length at most 3, (iii) there exists an assignment to the variables of ϕ satisfying ϕ if and only if there exists an assignment to the variables of ψ satisfying ψ. Prove that your algorithm satisfies all three requirements.

Problem 2 (Search version of SAT)**.** Assume that there is a polynomial-time algorithm $\mathcal{A}$ that *decides* if a SAT formula is satisfiable. That is, for every SAT formula ϕ, if there is an assignment to the n variables of ϕ satisfying ϕ, then $\mathcal{A}(\phi) = 1$, otherwise $\mathcal{A}(\phi) = 0$. Use the algorithm $\mathcal{A}$, to design a polynomial-time algorithm $\mathcal{A}'$ that *finds* a satisfying assignment $x \in \{0, 1\}^n$ for any satisfiable SAT formula (and outputs 0 for unsatisfiable formulas). Prove that your algorithm is correct.

Problem 3 (Error correcting codes)**.** Use results from Lecture 18 (on Error Correcting Codes) to design an algorithm for the following game. I choose a number from 1 to 16. You can ask yes-no questions, and I'll lie on all or all but one question. Design an algorithm that finds my number using as few questions as possible. (For the full score, prove that there is an algorithm that always asks at most 7 questions.)

Problem 4 (From Las Vegas to Monte Carlo)**.** In this exercise, we'll show that a Las Vegas algorithm for a problem implies a Monte Carlo algorithm for the same problem.

Assume that there is a randomized algorithm $\mathcal{A}$ that always correctly computes the function $f\colon \{0,1\}^n \to \{0,1\}$ in expected time $T(n)$. That is,

- for every input $x \in \{0,1\}^n$, $\mathcal{A}(x) = f(x)$;
- for every input $x \in \{0,1\}^n$, let the random variable $T(x)$ be the running time of the algorithm $\mathcal{A}$ on input x. Then for every $x \in \{0,1\}^n$, $\mathbb{E}_x[T(x)] = n^c$ for a constant $c > 0$.

Use Markov's inequality (from Lecture 3) to design a randomized algorithm $\mathcal{A}'$ that *always* stops after $O(n^c)$ operations, and *for every* x, computes $f\colon \{0,1\}^n \to \{0,1\}$ with probability $\geq 2/3$. Prove that your algorithm works for every input x (rather than for $2/3$ of the inputs).

Problem 5 (Circuits for Addition. **Extra credit**)**.** Design a Boolean circuit that performs addition of two n-bit long numbers written in binary.

- Design a circuit of linear size $O(n)$.
- Design a circuit of linear size $O(n)$ and logarithmic depth $O(\log n)$.