The final is due by 11:59PM, Friday March 18. No searching the Internet or talking to others. Refer to your notes and books freely.

For the multiple choice questions, you must provide an option and a two-sentence explanation of why you chose this. No points without an explanation. Each question is worth 5 points.

1. Suppose I could reduce 3SAT to 2SAT through a logspace reduction. What would happen?
   (a) \( \text{NL} = \text{PSPACE} \).
   (b) \( \text{L} = \mathbb{P} \).
   (c) \( \text{PH} \) collapses.
   (d) Nothing. This is already implied by current knowledge.

2. Which of the following would not contradict current knowledge? (In the following, \( c \) is an integer.)
   (a) \( \exists c \geq 1, \text{NP} \subseteq \text{SPACE}(\log^c n) \).
   (b) \( \exists c \geq 1, \text{PH} \subseteq \text{DTIME}(n^c) \).
   (c) \( \exists c \geq 1, \text{PSPACE} \subseteq \text{NTIME}(n^c) \).

3. Prove: if there exists a unary \( \text{NP} \)-complete language, then \( \mathbb{P} = \text{NP} \).
   Hint: Here’s the usual, exponential algorithm for SAT. Consider formula \( \phi \). Pick variable \( x_1 \), set it to 0 to get formula \( \phi_0 \), and set it to 1 to get formula \( \phi_1 \). Now, in each of these, set \( x_2 \) to both 0 and 1 to get the four formulas \( \phi_{00}, \phi_{01}, \phi_{10}, \phi_{11} \). \( \phi \) is satisfiable iff one of these are satisfiable. We can continue this process, and get the obvious exponential time algorithm. But now, let me reduce each of these, in polynomial time, to the unary language. Can I use this information to the cut down the exponential search?

4. We characterized \( \text{NL} \) as languages with a logspace-verifiable read-once certificate. Removing the read-once restriction yields the class \( \text{NP} \). Suppose we allowed the certificate to be read multiple times, but only “left to right”.
   In other words, define a “reset” verifier as follows. It is a normal logspace machine with a special reset state. If it enters that state, the head reading the certificate moves back the start. Other than that, the certificate head always moves to the right, as is the usual case (for \( \text{NL} \)). Prove that the class of languages with a logspace reset verifier for read-once certificates in \( \text{NP} \).

5. Given the above, here’s a proof that \( \mathbb{P} = \text{NP} \).
   Consider \( L \in \text{NP} \). There exists a logspace reset machine \( M \) such that: \( x \in L \) iff there exists a (poly-sized) certificate \( y \) such that \( M(x, y) \) accepts. Note that \( M \), being a logspace machine, runs in polynomial time (say \( p(n) \) time).
I will construct a (standard) logspace verifier \( M' \) and a read-once certificate. Simply concatenate \( y \ p(n) \) times, to get the new read-once certificate \( z = y\#y\#y\#\ldots \). (Assume \# is a new symbol.) I can simulate \( M(x, y) \) by \( M'(x, z) \) where \( M' \) makes a single left to right pass over \( z \). \( M \) behaves exactly like \( M' \) except when \( M \) goes to a reset state. In that case, \( M' \) keeps moving the certificate head to the right until it reads a \#. This simulates a reset of the head. There are at most \( p(n) \) resets, so \( z \) is large enough to do all the shifts.

Thus, \( \mathcal{L} \) has a (normal) logspace verifier for read-once certificates. Thus, \( \text{NL} = \text{NP} \), so \( \text{P} = \text{NP} \).

Something must have gone wrong. Where’s the bug?

6. Prove: if \( \text{PSPACE} \neq \text{EXP} \), then \( \mathcal{L} \neq \text{P} \).

7. There’s been much talk about Go. So let’s also (sort of) talk about Go. Consider the two-player pebble game on a directed graph \( G \). Player A begins by placing a pebble on some vertex \( v_0 \) of \( G \). Player B then places a pebble on some out-neighbor \( v_1 \) of \( v_0 \). Now, Player A has to place a pebble on some out-neighbor \( v_2 \) of \( v_1 \), where a pebble has already not been placed. (Assume both players have the same colored pebbles, so there is no distinction between them.) Then, Player B plays, so on and so forth. When some player cannot place a pebble (because the out-neighborhood of the previous move is completely occupied by pebbles), that player loses.

Prove that the language \( \{ \langle G, v \rangle | \text{Player A has a winning strategy starting from vertex } v \} \) is \( \text{PSPACE} \)-complete. You can assume that QBF over 3CNFs is \( \text{PSPACE} \)-complete.

Hint: Take inspiration from the proof that Hamiltonian Path is \( \text{NP} \)-complete. Suppose a QBF instance was in alternating form \( \exists x_1 \forall x_2 \exists x_3 \ldots \Phi \). Player A will choose all the \( \exists \) variables, and Player B will choose the \( \forall \) variables. Create a series of vertex gadgets where taking a “left” move means True, and taking a “right” move means False. Thus, every assignment of variables corresponds to a path full of pebbles, that should end at a fixed vertex regardless of the assignment. Now, set up a clause gadget. If the original QBF was true, Player A should have some way of leading to vertex \( v \) such that all out-neighbors of \( v \) already have pebbles. And vice versa.