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) NL = PSPACE.
   (b) L = P.
   (c) PH collapses.
   (d) Nothing. This is already implied by current knowledge.

   If you can reduce 3SAT to 2SAT through a logspace reduction, then NL = NP, which implies P = NP. Thus, PH collapses. This says nothing about the classes “outside” NL to NP (thus, nothing about L or PSPACE).

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) \( \exists c \geq 1, \text{PSPACE} \subseteq \text{NTIME}(n^c) \).

   The second statement implies \( \text{P} \subseteq \text{DTIME}(n^c) \), violating the time hierarchy theorem. The third implies \( \text{NP} \subseteq \text{NTIME}(n^c) \), violating the non-deterministic time hierarchy theorem.

3. Prove: if there exists a unary NP-complete language, then \( \text{P} = \text{NP} \).

   This statement is called Berman’s theorem. Let \( \mathcal{L} \) be a unary language. Consider \( \Phi \) of size \( n \).

   Start with \( \mathcal{S} = \{ \Phi \} \). Given any set \( \mathcal{S} \) of formulas with exactly \( k \) free variables, we can construct another set \( \mathcal{T} \) with exactly \((k - 1)\) free variables such that some formula in \( \mathcal{S} \) is satisfiable iff \( \mathcal{T} \) is satisfiable. For every \( \Psi \) in \( \mathcal{S} \), pick some variable, and set it to both 0 and 1 to get two formulas \( \Psi_0 \) and \( \Psi_1 \). Replace \( \Psi \) by \( \Psi_0 \) and \( \Psi_1 \).

   We can obviously keep applying this process, but the \( \mathcal{S} \) could become exponential. Critically, note that all formulas in \( \mathcal{S} \) have size at most \( n \).

   Using \( \mathcal{L} \) and the reduction \( f \) from 3SAT to \( \mathcal{L} \), we prove the main claim (in next paragraph). Observe that from the claim below, we get a polynomial time algorithm for 3SAT. Simply apply the claim any time \( |\mathcal{S}| \) becomes larger than \( p(n) \) and cut it back down to down to \( p(n) \). You should be able to argue that the whole process takes polynomial time.
4. We characterized NL as languages with a logspace-verifiable read-once certificate. Removing the read-once restriction yields the class 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 NL). Prove that the class of languages with a logspace reset verifier for read-once certificates in NP.

NP is the class of languages with a logspace-verifiable certificate. We can simulate such a verifier by a reset verifier. Add an extra logspace tape. When the usual verifier wants to move left, the reset verifier writes down the destination location, resets the head, and moves right until it reaches the destination.

5. Given the above, here’s a proof that P = NP.

Consider L ∈ NP. There exists a logspace reset machine M such that: x ∈ 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# .... (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, L has a (normal) logspace verifier for read-once certificates. Thus, NL = NP, so P = NP.

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

The verifier should be able to reject any wrong certificate. But a logspace verifier cannot check if a string is actually of the form y#y#y# .... It wouldn’t be able to distinguish (at least it’s not clear from the “proof”) y#y#y# .... from y_1#y_2# ....

6. Prove: if PSPACE ≠ EXP, then L ≠ P.

Standard padding argument. We’ll show: if L = P, then PSPACE = EXP. It suffices to prove that EXP ⊆ PSPACE. Consider L ∈ EXP, so there is a deterministic TM that runs in 2^{nc} time for L. Consider L_{pad} = \{<x, 1^{2^{nc}} \}. You can show that L_{pad} ∈ P = L. Consider the logspace machine from L_{pad}, which requires at most log(|x| + 2^{nc}) = O(nc) space. Thus, we
can run this machine on any input \( x \), and simply feed it \( 1 \) whenever it needs to some input bit from the padded part. This proves that \( \mathcal{L} \in \text{PSPACE} \).

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.

The technical name for this language is Generalized-Geography. Check out http://www.levreyzin.com/presentations/Go.pdf (slide 8) for the proof.