You have 150 minutes. There are four problems, labeled A, B, C, D.

You may cite without proof any theorem proved in class, in the assigned sections of the textbook, or in the assigned homework. You may not cite other results without proof. You may not cite a result that trivializes (i.e. immediately solves) an exam problem. If you are unsure whether you may cite a result, then ask for clarification.

Always show your work and/or explain your answers. Partial credit is often awarded for good work without the correct final answer.

When there are multiple answers to a problem, it is understood that a simpler or more efficient answer may earn more credit.

Good luck.

A. Let M be an arbitrary nondeterministic push-down automaton. Construct a non-deterministic Turing machine N such that $L(N) = L(M)$. Be detailed; describe exactly how states, transitions, etc. of M lead to states, transitions, etc. of N. You are not required to prove that your construction works.

B. Let $A = \{M\#N : M$ and N are Turing machines and $L(M) \cap L(N)$ is finite$\}$. Prove that A is not recursive.

C. Let M be a Turing machine that, given input $N\#w$, does the following. First M runs N on w. Second, if the output of N is not a valid $L\#v$ (a Turing machine and an input for it), then M rejects; on the other hand, if the output of N is a valid $L\#v$, then M runs L on v and outputs whatever L outputs. Let $c = |M| + |\#| + 1$.

Prove that for all strings x the minimal description $d(x)$ is incompressible by c.

D. In this problem, an *expression* of SK combinatory logic is a finite nested list of Ss, Ks, and other symbols such as Ms, Ns, etc. For example, $(SN(KMK)P)N(K)$ is an expression. You will essentially show that Turing machines are at least as powerful as the SK combinatory logic, by showing that there is an algorithm for reducing expressions.

In Python or Python-like pseudocode, write a function that takes in an expression, performs one reduction step on it, and returns the reduced expression. If the expression cannot be reduced, then your function should print a message and return the expression. (By "one reduction step" I mean that if I repeatedly apply your function to an expression then that expression should gradually reduce to its normal form, if it has one.)