1. (30 pts.) Answer briefly in English using a minimum of mathematics.

   (a) What is the Church-Turing thesis regarding Turing machines?

   Ans. Turing machines can do the same things computers can do. In other words, any computer algorithm can be done on a Turing machine.

   (b) How do certificates and verifiers relate the class NP to ordinary Turing machines?

   Ans. Various answers are possible. The basic idea is that L is in NP if and only if there is a Turing machine V (a verifier) such that every $w \in L$ has a certificate $c(w)$ and V accepts the input $w, c(w)$ precisely for those $w \in L$.

   (c) What does "M accepts the string w" mean when M is a nondeterministic automaton or Turing machine?

   Ans. It means that there is *some* way for M to reach an accept state. (For an automaton, the accept state must be reached at the end of the input; for a Turing machine, it can be reached anytime.)

2. (30 pts.) Write regular expressions for the following when $\Sigma = \{0, 1\}$.

   (a) $\overline{(\Sigma^* 1)} \cap \overline{(1\Sigma^*)}$.
   Hint: First describe the strings in the language without using "not."

   Ans. This is the strings that do not end in 1 and do not start in 1. A regular expression is $\epsilon \cup 0\Sigma^* 0$.

   (b) $\{w \mid w$ has an even number of 0's, or 1's, or both$\}$.
   (For example, ϵ, 010, 110, and 1010 are in the language, but 01 is not.)

   Ans. An even number of 0's: $(1^*01^*01^*)^*$ or $1^*(01^*01^*)^*$ are possibilities. Interchanging 0 and 1 gives the result for an even number of 1's. Taking the union gives both:

$$(1^*01^*01^*)^* \ \cup \ (0^*10^*10^*)^*.$$

3. (45 pts.) Beware of guessing:

correct answer +5pts. incorrect answer −3pts. no answer 0pts

F A nondeterministic Turing machine can recognize more languages than a standard Turing machine.

T Context free grammars can generate languages that DFAs cannot recognize.

F Context free grammars can generate languages that Turing machines cannot recognize.

T $\{a^k b^k \mid 0 \leq k \leq 5\}$ is a regular language.

F There are polynomial time algorithms for some NP-complete problems.

F If L is an NP-complete language and M is polynomial-time reducible to L, then M is also an NP-complete language.

T The class of regular languages is closed under complementation.

F The class of context-free languages is closed under complementation.

F The class of Turing-recognizable languages is closed under complementation.

4. (30 pts.) Construct CFGs that generate the following languages when $\Sigma = \{0, 1\}$.

(a) $\{ww^{\mathcal{R}} \mid w \in \Sigma^*\}$, where $w^{\mathcal{R}}$ is the reverse of the string w.

Ans. $S \to \epsilon \mid 0S0 \mid 1S1$.

(b) $\{0^i 1^j 0^k \mid \text{where } i + j = k\}$.

Ans. $S \to 0S0 \mid A \qquad A \to \epsilon \mid 1A0$.

5. (30 pts.) Construct PDAs that recognize the following languages when $\Sigma = \{0, 1\}$.

(a) $\{w \mid w$ contains at least two 1's$\}$.

Ans. Here's a verbal description. This machine doesn't even need a stack!

- Loop in the start state until a 1 is seen, then move to q_1.
- Now loop in q_1 until a 1 is seen and then move to q_2, which is the only accept state.
- Loop in q_2 until the input is read.

(b) $\{0^i 1^j 0^k \mid$ where $i + k = j\}$. *Warning*: This is *not* the same language as in 4(b).

Ans. Here's a verbal description. (As usual, if the PDA gets "stuck" in a state, that's a reject.)

- Mark the start of the stack with \$.
- Push 0's onto the stack as long as 0's are read (a single 0 for each 0 read).
- When 1's start being read, pop 0's off the stack as long as 0's are present, popping a single 0 for each 1.
- Pop a \$ off the stack and push it back on.
- Push 1's on the stack as long as 1's are read (one for one).
- Pop 1's off the stack as long as 0's are read (one for one).
- Pop \$ off the stack and move to the accept state.

6. (30 pts.) NEQ_{CFG} is the set of pairs G_1, G_2 of CFGs such that G_1 and G_2 generate different languages. Prove that NEQ_{CFG} is Turing-recognizable. That is, prove that you can build a Turing machine that will take two CFGs and accept them if and only if they produce different languages.
Remark: To "build a Turing machine," it is sufficient to give a high level description of an algorithm — you need not give details such as state transitions and tape reading/writing.
Hint: Since CFGs can be put in Chomsky normal form, assume that G_1 and G_2 are in Chomsky normal form.

Ans. Let G_1 and G_2 be in Chomsky normal form. For each $n > 0$:
- Generate all possible strings from G_1 that involve at most $2n$ substitutions.
- Discard all strings of length greater than n.
- Discard all strings that contain variables.
- Repeat the above steps for G_2.
- If the two sets of strings differ, accept.

(This will work because all strings of length n is a language are derivable in less than $2n$ steps when the grammar is in Chomsky normal form.)

7. (30 pts.) Prove that P (the class of languages decidable in polynomial time) is closed under complementation and union.

Ans. Suppose L is in P. To decide $\overline{L}$, run the Turing machine that decides L. It will either accept or reject in polynomial time. Do the reverse.

Ans. Suppose L_1 and L_2 are in P. To decide $L_1 \cup L_2$, run the Turing machines that decide L_1 and L_2. Since each decides in polynomial time, this takes polynomial time. If both machines reject, then reject; otherwise, accept.