A. We will reduce EMPTY_{TM} to A. Suppose, for the sake of contradiction, that A is decided by a Turing machine D. Define a Turing machine C that, on input $\langle M \rangle$, where M is a Turing machine:

1. Builds a CFG G, over the same alphabet as M, for the empty language.
2. Runs D on $\langle M, G \rangle$ and outputs whatever D outputs.

If M is a Turing machine with $L(M) = \emptyset$, then D accepts $\langle M, G \rangle$, and hence C accepts $\langle M \rangle$. If M is a Turing machine with $L(M) \neq \emptyset$, then D rejects $\langle M, G \rangle$, and hence C rejects $\langle M \rangle$. Thus C is a decider for EMPTY_{TM}, which is undecidable. This contradiction implies that A is also undecidable.

B.
A. $\emptyset$.
B. $\{0^n 1^m : n, m \geq 0\}$.
C. $\{0^n 1^n : n \geq 0\}$.
D. $\{0^n 1^n 0^n : n \geq 0\}$. This is easily proved by the pumping lemma for context-free languages, using the string $0^p 1^p 0^p$.
E. HALT_{TM}.
F. $\overline{\text{HALT}_{\text{TM}}}$.
G. EQ_{TM}.

C. Define a Turing machine F that takes a Boolean formula ϕ as input, and outputs ϕ fully quantified by existential quantifiers $\exists$. More precisely, on input $\langle \phi \rangle$, where ϕ is a Boolean formula, F does these steps:

1. Scans through ϕ to examine the variables used. For each variable x used, appends $\exists x$ to the tape.
2. Copies ϕ from the start of the tape to the end of the tape, and deletes ϕ from the start of the tape. For example, if ϕ uses the variables $x_1, \ldots, x_m$, then the contents of the tape are now $\exists x_1 \ldots \exists x_m \phi$ (or rather the encoding $\langle \exists x_1 \ldots \exists x_m \phi \rangle$).
3. Accepts.

This F is deterministic, and its work can be accomplished in polynomial time, as it consists of simple scans and copies. As we discussed in class, ϕ is satisfiable if and only if $\exists x_1 \ldots \exists x_m \phi$ is true. Therefore F is a deterministic polynomial-time reduction from SAT to TQBF. Because any problem in NP can be reduced to SAT in deterministic polynomial time, it follows that any problem in NP can be reduced to TQBF in deterministic polynomial time. That is, TQBF is NP-hard.

[The problem was mis-worded to say "NP-complete" instead of "NP-hard". I graded the NP-hard part, and gave allowances to students who seemed to devote great effort to showing that TQBF is in NP also.]

D. Let M be a linear bounded automaton, with state set Q and tape alphabet Γ. For any $n \geq 0$, let $c(n) = |Q|(n+2)|\Gamma|^n$. On an input of size n, M has at most $c(n)$ possible configurations, because there are $|Q|$ possibilities for the state, $n+2$ possibilities for the location of the tape head, and $|\Gamma|^n$ possible contents of the tape. If M uses $c(n)$ time steps (or more) in its computation, then it must use $c(n)+1$ configurations, and hence it must reuse some configuration c_0, by the pigeonhole principle. Then, because M is deterministic, it must loop through c_0 indefinitely. In other words, if M does not loop indefinitely — if M accepts or rejects the given input of size n — then it will do so in less than $c(n)$ time.

E. [This is Problem 8.8 from our textbook. The proof uses ideas from Theorem 8.4 and one of our assignments. The problem is harder than I intended. It would be easier if it were phrased in terms of DFAs rather than regular expressions.] We define a nondeterministic Turing machine N to decide $\bar{A}$. On input $\langle R, S \rangle$, N does these steps:

1. Converts R and S to NFAs N_R and N_S, using our algorithm from class.
2. Computes $m = 2^{|Q_R|+|Q_S|}$, where Q_R and Q_S are the state sets of N_R and N_S.
3. Marks the start states of N_R and N_S.
4. Nondeterministically guesses a sequence of m input symbols $a_1, \ldots, a_m$, but does not store them. Rather, as it guesses each symbol, N updates the markings on the states of N_R and N_S, so that they mark all states reachable by the string of input symbols guessed thus far.
5. Inspects the markings. If an accept state is marked in one NFA and no accept state is marked in the other NFA, then N accepts. Otherwise, N rejects.

Now we define a deterministic Turing machine M that, on input $\langle R, S \rangle$, simulates N and outputs the negation of what N outputs.

First, by Savitch's theorem M requires only quadratically more space than N. So we argue that N requires only polynomial space. Brief study of the algorithm for converting regular expressions to NFAs shows that the NFA occupies space linear in the size of the regular expression. Storing m requires space logarithmic in m, and hence polynomial in $|Q_R|$ and $|Q_S|$, and hence polynomial in the size of the input. The markings require no space.

Now we argue that N decides $\bar{A}$. The crux is the fact that, if N_R and N_S agree on all strings up to length m, then they agree on all strings. This can be proved in a manner similar to Problem 4.16 from our homework. Namely, convert N_R and N_S to DFAs, which have $2^{|Q_R|}$

and $2^{|Q_S|}$ states, respectively. Using the product construction, form the symmetric difference of the two DFAs (as in Theorem 4.5). The symmetric difference DFA has m states, and hence a pumping length of m. If this DFA accepts a string of length greater than or equal to m, then it must accept a string of length less than m, by the pumping lemma applied repeatedly. Conversely, if this DFA rejects all strings of length less than m, then it rejects all strings. In other words, if N_R and N_S agree on all strings of length less than m, then they agree on all strings.