Attempt any **three** of the **six** problems below. The homework is graded on a scale of 100 points, even though you can attempt fewer or more points than that. Your recorded grade will be the total score on the problems that you attempt.

1. (**20 points**, Sipser problems 5.17 and 5.18)

   (a) (**10 points**) Prove that the Post Correspondence Problem is decidable if the alphabet is unary, i.e., $\Sigma = \{\mathtt{1}\}$.
   Solution (sketch): if the PCP instance has a tile of the form $\left[\frac{\mathtt{1}^i}{\mathtt{1}^i}\right]$, then that tile solves the problem. Otherwise, if there is at least one tile of the form $\left[\frac{\mathtt{1}^i}{\mathtt{1}^j}\right]$ with $i > j$ and one with $i < j$, then it is straightforward to solve the problem. Finally, if all tiles have more $\mathtt{1}$'s on the top than on the bottom (or all have more on the bottom than on the top), then the problem is not solvable.

   (b) (**10 points**) Prove that the Post Correspondence Problem is undecidable if the alphabet is binary, i.e., $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.
   Solution (sketch): Consider any instance of PCP with some alphabet, Σ_1. Let $k = \lceil \log_2 |\Sigma_1| \rceil$. We can encode each symbol of Σ_1 with k symbols of $\{\mathtt{0}, \mathtt{1}\}$. With this encoding, every tile will a top string whose length is a multple of k and likewise for the bottom string. This ensures that the strings of $\{\mathtt{0}, \mathtt{1}\}$ that encode symbols of Σ_1 will stay properly aligned, and the new, binary, problem is solvable iff the original problem is solvable.

2. (**30 points**, Sipser problem 5.21) Let $AMBIG_{CFG} = \{G \mid G \text{ describes an ambiguous CFG}\}$. Show that $AMBIG_{CFG}$ is undecidable. (Hint: Use a reduction from PCP. Given a PCP instance

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\},$$

   construct a CFG G with the rules

$$\begin{array}{lcl} S & \rightarrow & T \mid B \\ T & \rightarrow & t_1 T \mathtt{a}_1 \mid \ldots \mid t_k T \mathtt{a}_k \mid t_1 \mathtt{a}_1 \mid \ldots \mid t_k \mathtt{a}_k \\ B & \rightarrow & b_1 B \mathtt{a}_1 \mid \ldots \mid b_k B \mathtt{a}_k \mid b_1 \mathtt{a}_1 \mid \ldots \mid b_k \mathtt{a}_k, \end{array}$$

   where $\mathtt{a}_1 \ldots \mathtt{a}_k$ are new terminal symbols. Prove that this reduction works.)
   Solution (sketch): If the PCP problem has a solution, $i_1, i_2, \ldots i_m$ such that $t_{i_1} t_{i_2} \ldots t_{i_m} = b_{i_1} b_{i_2} \ldots b_{i_m}$, then $S \Rightarrow T \stackrel{*}{\Rightarrow} t_{i_1} t_{i_2} \ldots t_{i_k} \mathtt{a}_{i_k} \ldots \mathtt{a}_{i_2} \mathtt{a} i_1$ and $S \Rightarrow B \stackrel{*}{\Rightarrow} b_{i_1} b_{i_2} \ldots b_{i_k} \mathtt{a}_{i_k} \ldots \mathtt{a}_{i_2} \mathtt{a} i_1$. Because $t_{i_1} t_{i_2} \ldots t_{i_m} = b_{i_1} b_{i_2} \ldots b_{i_m}$, these two derivations produce the same string. Thus, G is ambiguous.

   Now, I'll show that if G is ambiguous, then the PCP problem has a solution. The critical observation is that a grammar with starts symbol T and has the rules given above is unambiguous, as there is only one way to get any suffix of $\mathtt{a}_i$ symbols. Likewise, a grammar that starts with B is unambiguous. So, if G is ambiguous, there must be a string, $x \in L(G)$ such that $S \Rightarrow T \stackrel{*}{\Rightarrow} x$ and $S \Rightarrow B \stackrel{*}{\Rightarrow} x$. The derivation gives a solution to the PCP problem (in fact, the solution is the reverse of the string of $\mathtt{a}_i$'s at the end of x).

3. (**35 points**, Sipser problem 5.26) Define a *two-headed finite automaton* (2HDFA) to be a deterministic finite automaton that has two read-only, bidirectional heads that start at the left-hand end of the input tape and can be independently controlled to move in either direction. The tape of a 2HDFA is finite and is just large enough to contain the input plus a left-endmarker, $\vdash$, and a right-endmarker, $\dashv$. A 2HDFA may not move either of its heads beyond either delimeter. A 2HDFA accepts by entering a special accept state.

   (a) (**10 points**) Describe a 2HDFA that recognizes the language $\{\mathtt{a}^n \mathtt{b}^n \mathtt{c}^n \mid n \geq 0\}$. You don't need to specify all the details of the transition function. Just write a few sentences explaining how it works.
   Solution (sketch): Build a *2HDFA* that starts by moving its first head one square to the right to reach a $\mathtt{a}$. If it reaches a $\mathtt{b}$ or $\mathtt{c}$ it can immediately reject and if it reaches a $\dashv$ it can immediately accept. Now, move the second head to the right until it reaches a $\mathtt{b}$. If it reaches a $\mathtt{c}$ or a $\dashv$ first, the machine rejects. Scan

both heads to the right thereby pairing up a and b symbols. When the left head reaches a b, the right head should be at its first c; otherwise, reject. Now, compare the number of b's and c's in the same way. If they are equal, accept; otherwise, reject.

(b) **(10 points)** Let $A_{2HDFA} = \{M\#w \mid M \text{ describes a 2HDFA that accepts } w\}$. Show that A_{2HDFA} is Turing-decidable.
Solution (sketch): Because the tape is bounded, a $2HDFA$ has a finite number of configurations: $(|w| + 2)^2|Q|$ because each tape head has $|w| + 2$ possible positions, and the machine has $|Q|$ possible states (assuming Q is the set of states for the machine). Thus, it is sufficient to simulate that $2HDFA$ for that many steps. If it accepts by then, then accept. Otherwise, the $2HDFA$ must be in a loop and we can reject.

(c) **(15 points)** Let $E_{2HDFA} = \{M \mid M \text{ describes a 2HDFA such that } L(M) = \emptyset\}$. Show that E_{2HDFA} is not Turing-decidable. (Hint: use computational histories.)

Solution (sketch): We can basically use the same method as for part (a). The $2HDFA$ moves its first head one square to the right (of the $\vdash$) to the first symbol of the first configuration. It moves its second head to the right to the first symbol of the second configuration. Now, it scans the two heads to the right making sure that the configuration under the right head is a valid successor of the configuration under the left head. The $2HDFA$ also notes if the configuration under the right head is an accepting configuration. If it reaches the right-endmarker having seen a valid sequence of configurations with the final one being in an accepting sate for M, then the $2HDFA$ accepts. Otherwise, the $2HDFA$ rejects.

4. (**35 points**, See Sipser problem 5.31) Let

$$f(x) = \begin{cases} 3x+1, & \text{if } x \text{ is odd} \\ x/2, & \text{if } x \text{ is even} \end{cases}$$

for any integer $x \geq 0$. Starting from x, obtain the sequence x, $f(x)$, $f(f(x))$, ... Stop if you ever reach 1. This sequence is known as the "hailstone" sequence for x. For example if $x = 23$, then you get the sequence: 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1. Extensive comptuer tests have showen tha every starting point from 1 through 2.88×10^{18} produces a seqence that ends in 1 (see http://en.wikipedia.org/wiki/Collatz_conjecture). The Collatz conjecture is that all positive starting points end up at 1, and this conjecture is unsolved.

Show that the Collatz conjecture is Turing-reducible to $TOTAL_{TM}$.

Note: Sipser seems to be asking to show a reduction to A_{TM} but I couldn't think of any way to do that.

Solution (sketch): Build a TM, M, whose input strings are encodings of integers. On input w, M computes the hailstone sequence and halts (e.g.i accepts) if it reaches 1. Otherwise, M loops. M is total (halts on all inputs) iff the Collatz conjecture is true.

5. (**35 points**, Sipser problem 5.34) Let P be a PDA and $WW = \{ww \mid w \in \{\mathtt{0},\mathtt{1}\}^*\}$. Use computational histories to show that the question of whether P accepts some string in WW is undecidable.
Solution (sketch): Given a string $M\#x$ where M describes a TM and x describes an input to M, we'll build a PDA that accepts a string of the form $h\$h\$$ iff $x \in L(M)$. As we described in the Nov. 10 notes, we'll write the history as $C_0C_1^{\mathcal{R}}C_2C_3^{\mathcal{R}}\ldots C_n$, where C_0 is the initial configuration, C_n is the final configuration (reversed if n is odd, even-indexed configurations are written in normal order, and odd-indexed configurations are reversed. Our PDA will push C_0 onto its stack while confirming that it is the initial configuration for M running with input w. For each odd indexed configuration, the PDA pops the previous configuration off of its stack and verifies that the new configuration is the successor of the previous one. When the PDA reads the first \$, it skips configuration C_0 and then pushes C_1 onto the stack. This time, it reads the configurations confirming that each even-indexed configuration is the successor of the previous odd-indexed configuration. The PDA accepts if all of these checks for valid successors of configurations pass and if the final configuration is an accepting one. This PDA accepts a string of the form ww iff M accepts x.

   Note that this PDA may accept strings of the form $y\$z\$$ where $y \neq z$ even if M does not accept x. This is because our construction relies on the separte check that the input is of the form ww (that can't be done by a PDA) to ensure that the two passes over the history are checking the *same* history.

6. (**45 points**) The RSA encryption method relies on the assumption that it is difficult to factor large numbers. However, no one knows whether or not there is a polynomial time algorithm for factoring. However, there does exist a polynomial time algorithm that will determine whether or not the integer represented by N (a binary string) is prime or composite, but if N is composite, this algorithm doesn't determine the factors.

   Now, consider a TM, M, that does the following when run with input N, a binary encoding of an integer:

   1. Check if N is prime (e.g., using the algorithm described above). If N is prime, M writes the string "$\mathtt{0}$" on its tape and halts. Otherwise, it continues to step 2.
   2. Find f a factor of N with $1 < f < N$. M then writes the binary encoding of f on its tape and halts.

   Describe how you can implement the second step of this algorithm with a method that will find f in polynomial time iff there is a polynomial time algorithm for factoring. In other words, your algorithm should be one that runs in polynomial time if factoring is polynomial, and in super-polynomial time otherwise.

   Hints: (1) Use diagonalization. (2) Remember that big-O analysis ignores constant factors, even ***really big*** ones.

   Solution (sketch): If factoring is polynomial time, then there exists some TM, M that when run with the encodng of an integer n on its tape, computes some non-trivial factor of n and halts with that factor written on its tape. Furthermore, there is such a TM that performs its computation in polynomial time.

   Let M_i be the i^{th} TM is some enumeration sequence. Now, run the following program:

```
for i = 1 to ∞ do
    for j = 1 to i do
        run M_j with input n for i steps.
        if M_j halts with in i steps
            check to see if the string on its tape is a non-trivial factor of n.
            if so, accept.
    od
od
```

   This computation is guaranteed to terminate because n is composite From step 1 of the machine described in the problem, and there is some TM that will compute the factors of n. Now, consider the case if factoring is

polynomial time. Then there is some TM M_j that finds a factor of n in polynomial time. Note that the existence of M_j does not depend on whether or not we know that factory is polynomial time. It only depends on whether or not factoring actually is polynomial time. Let $\ell(n)$ denote the length of the string that encodes n, and note that $\ell(n) \approx \log n$. If M_j takes time $f(\ell(n))$ time to find a factor, where f is a polynomial (the length of the string that describes n is $\Theta(\log n)$, then the total time to perform step 2 is roughly $O((j + f(\ell(n))^3))$. However, j is a constant. So this is $O(f^3(\ell(n)))$ which is a polynomial in $\ell(n)$. Thus, this is a factoring algorithm that is guaranteed to run in polynomial time iff factoring is in polynomial time.