1. Consider the language:

   $CONTAINSPAL_{DFA} =$

   $$\{\langle M \rangle \mid M \text{ is a DFA and } ww^R \in L(M) \text{ for some } w \in \Sigma_M^*\}$$

   Determine whether this language is decidable, undecidable but recognizable, not recognizable but has a recognizable complement or that neither the language or its complement is recognizable. Justify your conclusion.

   This language is decidable. We can build a non-deterministic PDA P that recognizes the palindromes over the language Σ_M^*. P works by guessing the middle of the string and pushing the first half onto the stack. Then, it checks the second half with the elements of the stack, popping on a match and rejecting the string on a mismatch. When a string's first and last half match, it is accepted. Now let $L = L(P)$ be the language containing the palindromes over Σ_M^*. The intersection $L \cap L(M)$ results in the palindromes accepted by M. All we have to do now, is determine if $L \cap L(M)$ is empty. If it is, then for all $w \in \Sigma_M^*$, $ww^R \notin L(M)$, if not then we know that $L(M)$ contains a palindrome. We know that the intersection of a context-free language and a regular language is context-free, so we can find a grammar G that produces $L \cap L(M)$. From class, we know that E_{CFG} is a decidable language. Thus, we can build a Turing machine for E_{CFG}, and if it accepts G then we know $ww^R \in L(M)$ for some $w \in \Sigma_M^*$, whereas if it rejects G then $L(M)$ does not contain palindromes.

2. Consider the language:

   $CONTAINSPAL_{TM} =$

   $$\{\langle M\rangle \mid M \text{ is a TM and } ww^R \in L(M) \text{ for some } w \in \Sigma_M^*\}$$

   Determine whether this language is decidable, undecidable but recognizable, not recognizable but has a recognizable complement or that neither the language or its complement is recognizable. Justify your conclusion.

   This language is recognizable. To show this, we will first show that $CONTAINSPAL_{TM}$ is recognizable, and then show its complement is not recursively enumerable. Using dovetailing we can enumerate all strings over Σ_M^*. We run M on every string over the machine's alphabet. Then, f M rejects a string, then we ignore it. If M accepts a string, then we run the reversed of that string on M. Now, if this also gets accepted by M then we know $ww^R \in L(M)$, so we accept the machine. However, if we cannot find such string, then we will keep simulating strings on M infinitely without accepting M. Thus, this language is recognizable.

   Now, assume that $\overline{CONTAINSPAL_{TM}}$ is recognized by $M_{\overline{CONTAINSPAL_{TM}}}$ and construct a machine $M_{\overline{A_{TM}}}$ that behaves as follows:

   1 On input $\langle M, w\rangle$:
   - 1.1 Construct a description, $\langle M'\rangle$, of a TM M' that behaves as follows:
     On input w', simulate M on input w and:
     - i . accept w' if M accepts w.
     - ii . otherwise reject w or loop.
   - 1.2 Run $M_{\overline{CONTAINSPAL_{TM}}}$ on $\langle M'\rangle$ (and accept if it does).

   2 If the input is not of the form $\langle M, w\rangle$, accept

   We know that $\overline{A_{TM}}$ is not recursively enumerable so our assumption that $\overline{CONTAINSPAL_{TM}}$ is recognizable must be wrong. Thus, $\overline{CONTAINSPAL_{TM}}$ is not recursively enumerabke, so $CONTAINSPAL_{TM}$ must be recognizable.

3. Consider the language:

   $CONTAINSPAL_{CFG} =$

   $$\{\langle G \rangle \mid G \text{ is a DFG and } ww^R \in L(G) \text{ for some } w \in \Sigma_G^*\}$$

   Determine whether this language is decidable, undecidable but recognizable, not recognizable but has a recognizable complement or that neither the language or its complement is recognizable.

   Warning: This problem looks a lot like the two preceding questions, but its solution is probably much more challenging.

   You should justify your conclusion for this and each of the two preceding questions by either showing an algorithm to decide the language, showing an algorithm to recognize the language (or its complement) and using a reduction argument to show that the complement (or the language) is not recognizable or providing an argument based on reductions that neither the language nor its complement is recognizable.

   This language is undecidable but recognizable.

4. (This is a big, long problem borrowed from K Schwarz of Stanford. Don't be scared. It really is not very hard, but the result is interesting.) There are two classes of languages associated with Turing machines — the recursively enumerable languages (RE), which can each be recognized by a Turing machine, and the recursive languages (R), which can each be decided by a Turing machine. Why didn't we talk about a model of computation that accepted just the R languages and nothing else? After all, having such a model of computation would be useful — if we could reason about automata that just accept recursive languages, it would be much easier to see what problems are and are not decidable.

   It turns out, interestingly, that there is no class of automata with this property, and in fact the only way to build automata that can decide all recursive languages is to have automata that also accept some languages that are RE but not R. This problem explores why.

   Suppose, for the sake of contradiction, that there is a type of automaton called a deciding machine (or DM for short) that has the computational power to decide precisely the R languages. That is, $L \in R$ iff there is a DM that decides L. We will make the following (reasonable) assumptions about deciding machines:

   - Any recursive language is accepted by some DM, and each DM accepts a recursive language.
   - Since DMs accept precisely the recursive languages, all DMs halt on all inputs. That is, all DMs are deciders.
   - Since deciding machines are a type of automaton, each DM is finite and can be encoded as a string. For any DM D, we will let the encoding of D be represented by $\langle D \rangle$.
   - DMs are an effective model of computation.

   Thus the Church-Turing thesis says that the Turing machine is at least as powerful as a DM. Thus there is some Turing machine UD that takes as input a description of a DM D and some string w, then accepts if D accepts w and rejects if D rejects w. Note that UD can never loop infinitely, because D is a deciding machine and always eventually accepts or rejects. More specifically, UD is the decider "On input $\langle D, w \rangle$, simulate the execution of D on w. If D accepts w, accept. If D rejects w, reject."

   Unfortunately, these four properties are impossible to satisfy simultaneously.

   (a) Consider the language $REJECT_{DM} = \{\langle D \rangle \mid D \text{ is a DM that rejects } \langle D \rangle \}$. Prove that $REJECT_{DM}$ is decidable.

   (b) Prove that there is no DM that decides $REJECT_{DM}$.

   Your result from (b) allows us to prove that there is no class of automaton like the DM that decides precisely the R languages. If one were to

exist, then it should be able to decide all of the R languages, including $REJECT_{DM}$. However, there is no DM that accepts the decidable language $REJECT_{DM}$.

Note: A word or two about what the impossible model of computation this problem talks about might look like may help you appreciate what it says. If used in simple ways (some might say appropriate ways), the header of a for loop makes it clear how often the loop will execute. That is, if you see something like

for (int x = init; x < max; x = x + increment) }

you can compute (max - init)/increment to get the number of times you expect the loop to execute. In Java, C, or C++, your estimate may be wrong because code in a loop's body can modify x, max or increment. If, however, you imagine a language in which the compiler enforces a restriction that the body of a loop cannot do anything that might modify x, max, or increment, you could get a language where for loops could never lead to infinite loops. Then, all you would have to do is eliminate while loops and recursion to get a language in which it was impossible to write an infinite loop. This question shows that it would be impossible to write certain program that always terminate in such a language.

(a) By the Churing-Thesis, we know there is some Turing machine UD that takes as input a description of a DM D and some string w, then accepts if D accepts w and rejects if D rejects w. Therefore, we can show $REJECT_{DM}$ by using UD in the following manner. It will take a description $\langle D \rangle$ and simulate D on $\langle D \rangle$. Then, accept if D rejects its own description, and reject if D accepts its own description. More specifically, UD is the decider "On input $\langle D, D \rangle$, simulate the execution of D on the description of D. If D accepts $\langle D \rangle$, accept. If D rejects $\langle D \rangle$, reject."

(a) For the sake of contradiction, assume there is a DM D that decides $REJECT_{DM}$. That is $L(D) = \{\langle M \rangle \mid M \text{ is a } DM \text{ and } \langle M \rangle \notin L(M)\}$. Now, given that we can run D on any DM, we can run D on its own description. If $\langle D \rangle \notin L(D) \implies \langle D \rangle \in L(D)$ and $\langle D \rangle \in L(D) \implies \langle D \rangle \notin L(D)$. This is clearly a contradiction, so there is no DM that decides $REJECT_{DM}$.

5. Use mapping reductions to show that neither the language

$$TOTAL_{TM} = \{\langle M \rangle \mid M \text{ is a TM that halts on all inputs}\}$$

nor its complement is recognizable. (Hint: Feel free to think about how to solve this problem without using a mapping reduction. Then, later you can figure out how to convert your "free-style" proof into one using mapping reductions.)

We will show that $TOTAL_{TM}$ is not recognizable by showing that $ALL_{TM} \leq_M TOTAL_{TM}$. To do this, we need a mapping that will take any $\langle M \rangle$ to another Turing machine description $\langle M' \rangle$ in such a way that all $N' \in L(M')$ halt on all inputs if and only if for all $N \in L(M),\ L(N) = \Sigma_M^*$. Consider the function $f(\langle M \rangle) = \langle M' \rangle$ where M' is identical to M except that all of its reject states make the machine loop forever. This means that if a machine $M \in ALL_{TM}$ then it has reached accept states in M for all inputs, meaning that no reject states have been reached in M'. Thus, M' halts on all inputs. Now, if M' halts on all inputs, then this means that no input went to a reject state and looped forever, so M' must accept all inputs over its alphabet. Given that we know that ALL_{TM} is not recognizable, we can conclude that $TOTAL_{TM}$ is not recognizable.
Similarly, we can show that $\overline{TOTAL_{TM}}$ is not recognizable by showing that $\overline{ALL_{TM}} \leq_M \overline{TOTAL_{TM}}$. We would need a mapping that will take any $\langle M \rangle$ to another Turing machine description $\langle M' \rangle$ in such a way that for all $N' \in L(M')$ there exists at least an input that makes the machine loop infinetely if and only if for all $N \in L(M),\ L(N) \neq \Sigma_M^*$. Consider the same function f. If a machine N does not contain all of the strings of Σ_M^* then it must reject an input. Thus, N' will reach a reject state and loop infinetely. Now, if a machine N' does not halt on at least one input, then we know that input reached a reject state, so the machine's language does not contain all strings over that alphabet. Given that we know that $\overline{ALL_{TM}}$ is not recognizable, we can conclude that $\overline{TOTAL_{TM}}$ is not recognizable either.