Problem 1

For each of the following languages, determine whether it is decidable and recognizable, undecidable but recognizable, or unrecognizable. Prove your answer.

a. $L_a = \{ \langle M \rangle \mid M$ is a TM and enterprise $\in L(M) \}$

b. $L_b = \{ \langle M \rangle \mid M$ is not a decider $\}$

c. $L_c = \{ \langle F, G \rangle \mid F$ and $G$ are DFAs s.t. $L(F) \cap L(G) = \emptyset \}$

d. $L_d = \{ \langle M \rangle \mid M$ is a TM that halts on some input $\}$
The following questions are lab problems.

Lab Problem 1

In this problem we will generalize the halting problem to a more general class of computational models.

Suppose we have a new kind of model that we call a SuperTM\textsubscript{1} that has access to the following magical black box. Given any pair \( \langle M, x \rangle \), where \( M \) is an encoding of a TM and \( x \) is a string, this black box can correctly tell us whether \( M \) will halt on input \( x \).

a. Prove that every Turing recognizable language \( L \) can be decided by a SuperTM\textsubscript{1}.

b. Consider the SuperTM\textsubscript{1} halting problem:

\[ L_1 = \{ \langle M, x \rangle \mid M \text{ is a SuperTM}_1 \text{ that halts on input } x \} \]

Prove that there does not exist a SuperTM\textsubscript{1} that decides \( L_1 \).

c. Part b. can be generalized to demonstrate that there is an infinite sequence of computational models: SuperTM\textsubscript{i}'s, SuperTM\textsubscript{i+1}'s, etc. such that for all \( i \), SuperTM\textsubscript{i}'s can magically solve the SuperTM\textsubscript{i−1} halting problem, but \( L_i \) is not SuperTM\textsubscript{i} decidable.

We say that a language \( L \) is \( \omega \)-decidable if there exists some \( n \) for which some SuperTM\textsubscript{n} decides \( L \). Give an example of a language that is not \( \omega \)-decidable (and provide a short proof that it is not \( \omega \)-decidable).

Lab Problem 2

Let TroublesomeTribble (or TT for short) be the function that takes a number \( n \) and computes the maximum finite number of steps a TM with tape alphabet \( \{\omega, 0, 1\} \) and \( n \) non-halting states can take on input \( \varepsilon \). By non-halting states, we mean all states other than \( q_{\text{accept}} \) and \( q_{\text{reject}} \).

More formally:

\[ \text{TroublesomeTribble}(n) = \max(x \mid \exists \text{ TM } M \text{ with tape alphabet } \{\omega, 0, 1\} \text{ and } n \text{ non-halting states that halts on input } \varepsilon \text{ in } x \text{ steps}) \]
To clarify, we define a step to be the action of reading an input off the tape, writing a new value, moving the head and potentially transitioning between states.

It is easy to see that $TT(1) = 1$ (make sure you understand why).

a. Show that it is impossible for a Turing machine to compute $TT(n)$. 
   *Hint: think about the halting problem.*

b. Show that if we could compute a function $f$ such that for any $n$, $f(n) > TT(n)$, we could compute $TT(n)$. Note that this means that $TT$ grows faster than any computable function.

c. The Goldbach Conjecture is a famous unproven claim in mathematics that states that every even integer greater than 2 can be expressed as the sum of two prime numbers.

Assume we have special $TM$ that has access to an oracle that can calculate $TT(n)$ for any $n$. Prove that such an augmented $TM$ could decide whether or not the Goldbach conjecture is true.

Once you’ve solved this problem you may find it interesting to consider what other open problems you could solve if only you could compute $TT(n)$ (no answer required for this).

**Lab Problem 3**

The Klingons are preparing a new type of missile to attack Earth. They’re sending programs used to operate these missiles over Klingon radio and would like to know that anyone who got a hold of these programs would be utterly confused as to what the programs do. As a result, the Klingons are hiring former CS51 students to look into obfuscators.

An *obfuscator* is a Turing machine that, on input the description of a Turing machine $\langle M \rangle$, outputs the description of another Turing machine $\langle M' \rangle$ such that $L(M) = L(M')$.

An obfuscator is successful if, given $\langle M' \rangle$, it is impossible to infer too much about $\langle M \rangle$. We omit a formal definition of a successful obfuscator here, but give you a little bit of a back story on obfuscation. The International Obfuscated C Code Contest (IOCCC) has been collecting examples of obfuscated C programs for more than twenty years now; their stated goal was to “show
the importance of programming style, in an ironic way.” A systematic, algorithmic way to construct obfuscators, however, was not discovered until very recently. In a paper that appeared in Fall 2013, Sanjam Garg, Craig Gentry, Mariana Raykova, Amit Sahai and Brent Waters gave the first algorithm for a successful obfuscator. Although this result is fairly recent, it has already spurred on a lot of follow-up research in theoretical computer science. (This result contributed to Sanjam Garg’s ACM Best Dissertation Award 2013, and to Craig Gentry’s MacArthur genius award in 2014.)

Consider the following language:

\[ Obfuscator_{TM} = \{ \langle M \rangle \mid \text{The Turing machine } M \text{ is an obfuscator} \} \]

Just as the language \( EQ_{TM} \) that we discussed in class (the textbook analyzes it in Theorem 5.30 on p. 238), neither \( Obfuscator_{TM} \), nor its complement, is Turing-recognizable.

a. Prove that \( Obfuscator_{TM} \) is not Turing-recognizable.

b. Prove that \( Obfuscator_{TM}^c \) is not Turing-recognizable.