Answer the questions below, paying particular attention to the logic of your arguments. Short descriptions for how a function might computed (on any machine we've talked about) are sufficient to show that it is computable. Pseudocode is often appropriate for reductions.

1. Show that if A is recognizable and A reduces to $\bar{A}$, then A is decidable.

Solution:
From the definition of reducibility, it follows (see argument below) that $A \leq_M B$ if and only if $\bar{A} \leq_M \bar{B}$. In this case, we have $A \leq_M \bar{A}$, so $\bar{A} \leq_M A$. Since A is recognizable, $\bar{A}$ is also recognizable. Since both A and its complement are recognizable, A is decidable.

By definition, $A \leq_M B$ if there is a computable function f such that $w \in A \Leftrightarrow f(w) \in B$. It is true for the same function that $w \in \bar{A} \Leftrightarrow f(w) \in \bar{B}$. Therefore, $A \leq_M B \Leftrightarrow \bar{A} \leq \bar{B}$.

2. Let $B = \{< M > \mid M$ accepts exactly one of the strings 00 and 11, i.e. $|L(M) \cap \{00, 11\}| = 1 \}$.
    a. What does Rice's theorem say about B?
    b. Show that halting problem reduces to B.
    c. Show that halting problem reduces to the complement of B.
    d. Are B or its complement recognizable?

Solution:

a. From the definition of B, the criteria for whether $< M > \in B$ is determined solely by $L(M)$. Also, B is non-trivial as a machine that just accepts 00 is in the language and one that rejects all inputs is not. Thus, B meets the two conditions for Rice's theorem, so B is undecidable.
b. Create N so that if M halts on blank tape then N accepts only the string "00".
In python-like pseudocode
```
def R(<M>):
        def N(x):
                M('')
                return x == '00'
        return <N>
```
If M('') halts, then L(N) = {'00'}, so $< N > \in B$. If M('') loops, then L(N) = $\varnothing$, so <N> $\notin B$.

c. Create N that always accepts 00 and if M halts on blank tape, N accepts 11 as well.
In python-like pseudocode
```
def R(<M>):
        def N(x):
```

```
            if x == '00' return true
            M('')
            return x == '11'
        return <N>
```

If M('') halts, then L(N) = {'00','11'}, so $< N > \notin B.$ If M('') loops, then L(N) = {'00'}, so $< N > \in B.$

d. No, by the above reductions. By b, L is not co-recognizable and by c, L is not recognizable.

3. A language is *co-recognizable* if its complement is recognizable. Argue why each of the following languages is or is not recognizable and why it is or is not co-recognizable.
   a. $L_1 = \{< M > | M$ enters state q_{27} for the input string $001101\}$.
   b. $L_2 = \{< M > | L(M)$ contains at most two strings}.
   c. $L_3 = \{< M > | L(M) \subseteq \Sigma^*\}$

Solution:

a. This is similar to showing that $\{< M > | M$ accepts the input string $001101\}$ is recognizable but not co-recognizable.

L_1 may be recognized by simulating M on 001101 and monitoring to see if M ever enters its state q_{27}.

On the other hand, L_1 is not decidable since we can reduce the halting problem to this language. Create N that moves to state q27 only if M halts on blank tape.

```
def R(<M>):
    def N(x):
        M('')
        transition to state q27 # q27 is not otherwise used
        accept
    return <N>
```

b. The language L_2 is not recognizable, but it is co-recognizable. To see that $\bar{L}_2$ is recognizable consider a machine that uses dovetailing to simulate M on all strings. Such a machine will discover three strings that M accepts if three such strings exist.

On the other hand, L_2 is not recognizable as this reduction from $\bar{H}_{TM}$ (the set descriptions of machines that loop) shows. We create N that accept everything if M halts on blank tape.

```
def R(<M>):
    def N(x):
        M('')
        accept
    return <N>
```

c. L_3 is decidable, as it includes every Turing machine description. A decider need only accept in the first step on all inputs.

4. A computable verifier is a deterministic Turing machine V that takes two arguments: x (the input) and y (the proof). A computable verifier always halts. Show that a language L is recognizable if and only if there exists a computable verifier V such that
    a. if $x \in L$, then there is a string y such that V(x,y) accepts, and
    b. if $x \notin L,$ then V(x,y) rejects for every string y.

Solution:

Suppose that L is recognizable. Then there is a Turing machine M that recognizes L. Let V be a machine that simulates the computation M(x) until M halts or until a number steps equal to the length of the string y has been simulated (whichever occurs first). V accepts if the simulated machine M accepts within $|y|$ steps; otherwise, it rejects.

For any string $x \in L$, the computation $M(x)$ will halt after some finite number of steps, say k. Then, $V(x, 1^k)$ will accept. On the other hand, if $x \notin L$ then $V(x,y)$ will reject for every string y, either because it rejected or hadn't yet halted.

For the other direction, we suppose that there exists a verifier V for the language L. Consider the Turing machine M which on input x simulates $V(x,y)$ for every string y (say in lexicographical order) and accepts if it finds a y such that V(x,y) accepts. If $x \in L$, then there is some y_x, such that $V(x,y_x)$ accepts and thus $M(x)$ will accept as well. On the other hand, if $x \notin L$, then $M(x)$ will loop as it tries longer and longer strings ad infinitum.

5. Consider the following property: $P = \{L \mid L$ is the language of some Turing machine that has an odd number of states$\}$. Show that P is a trivial property. (Yes, this problem is trivial).

Solution:

For every Turing machine M that has an even number of states we can trivially another state to form a new machine M' that has an odd number of states but which accepts the same language. For example, we can add a new state, designate it as the accept state, and have the old accept state transition to this state on every symbol.

Thus, if L = L(M) for a machine M with an odd number of states, then $L \in P$ by definition. On the other hand, if $L = L(M)$ for some machine M with an even number of states, then there is

a machine M' with an odd number of states such that $L = L(M) = L(M')$. Once again $L \in P$ by definition.

[Bonus] Consider the property $P = \{L \mid L$ is accepted by some Turing machine that never halts after an even number of steps$\}$. Show that P is a trivial property.

Again, we need to take an arbitrary machine M and construct one that never halts after an even number of steps. One idea is to create two parallel states for each state of the original machine: one state for when the number of steps taken is even, one for when the number of steps take is odd. All transitions from an even state should go to the odd version of the successor state in the original machine, and vice-versa. The odd versions of the original accept and reject states should be designated as the accept and reject states of the new machine. The even versions of the original accept and rejects states should transition to the odd ones on every symbol.

6. [Bonus] Read about The Recursion Theorem in the Sipser text. One implication of the recursion theorem is that in any general purpose programming language, one can write code that outputs the code itself. Write a python program that prints its own code. Do not use any file operations.

https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-2986218594

Solution:

```
s = 's = %r\nprint(s%%s)'
print(s%s)
```

There are many others. Search for “Quine” to see a few.