1) a) $q_0,000$
   $L \times e_2 \times 0 \times L$  $L \times e_5 \times x \times L$
   $L \times e_2 \times 0 \times L$  $L \times e_5 \times x \times L$
   $L \times e_3 \times 0$  $L \times e_2 \times 0 \times L$  $L \times e_5 \times x \times L$
   $L \times e_4 \times 0$  $L \times e_2 \times 0 \times L$  $L \times e_5 \times x \times L$
   $L \times e_5 \times x \times L$  $L \times e_2 \times 0 \times L$

b) $q_1,00000$

2) The flaw in step 2 is that we commit to running $M$ on $s_i$ until it finishes. The problem is that it may never terminate while running on a particular string, meaning that certain strings in $L$ never get tried. If they never get tried, our proposed enumerator won't ever produce them. Thus, the proposal doesn't satisfy the requirement of an enumerator to print out any string in $L$. For example, if $1 \in L$ and $0 \notin L$ and $M$ loops on $0$, assuming we test $0$ before $1$, our proposed enumerator will $\text{NEVER}$ print $1$, even though $1 \in L$. 

3) We will prove that the two models are equivalent by showing that a Stay TM can simulate any steps that a regular TM can, and that a regular TM can simulate any steps that a Stay TM executes.

The first direction of the proof is fairly simple. A Stay TM can simulate a regular TM simply by not using its Stay option, since any transition allowed on a regular TM is allowed on a Stay TM.

For the other direction, each move left or right in a Stay TM can be replicated identically on a regular TM. But, we must find a way to simulate the move:

\[ \delta(q, a) = (r, b, S) \]

on a Stay TM on a regular TM. We will replace each move of this type in a regular TM as follows:

\[ \delta(q, a) = (\text{temp-state},, b, R) \]

\[ \delta(\text{temp-state},, X) = (r, X, L), \text{ for each character } X \in \Gamma \cup B. \]

Note that the temporary state is different for each destination state \( r \) that has a stay transition in the stay machine. Without this distinction, there’s no way of knowing which state to proceed to from the temp-state. Also, note that you need several transitions, for each alphabet character and the blank symbol, because the transitions must handle whatever character appears on the square to the right. Finally, it’s important to do the right transition before the left transition instead of the other way around because we’re only guaranteed that a square will be available on the right.

Creating this substitution will allow a regular TM to simulate the behavior of a Stay TM so that outcome of all input strings in the Stay machine is preserved in this regular TM that simulates it.
4.

a) Let $T_{M_1}$ decide $L_1$ and $T_{M_2}$ decide $L_2$. We describe how to build a $T_{M_3}$ that decides $L_1 \cup \overline{L_2}$. Let $T_{M_3}$ run the same steps as $T_{M_1}$. If this accepts, accept. Otherwise, run the input on the same directions as $T_{M_2}$. If this accepts, accept. Otherwise, reject. For specifics, describe $T_{M_3}$ as a multi-tape machine that first copies its input onto its second tape before running $T_{M_1}$'s steps. After finishing those steps, if necessary, copy the contents from the second tape to the first. Assume these details for parts (b), (c), and (d).

b) Keep everything the same as part (a) except if the steps of $T_{M_2}$ reject, reject. If they accept, continue and run the steps of $T_{M_2}$ on the input. Accept if this accepts. Reject otherwise.

c) Let $T_{M_1}$, decide $L$. We describe how to build $T_{M_3}$ that decides $L$. Run the steps of $T_{M_1}$. If they accept, reject. Otherwise, accept.

d) Make $T_{M_3}$ a multi-tape machine to decide $L_1 \cup \overline{L_2}$. Use one tape as a counter that starts at 0 and ends at $n$, where $n$ is the length of the input. Use another tape to store a copy of the original input.
5) For each value, \( \ell \), of the counter, copy the first \( \ell \) letters of the input onto the main tape. Run the steps of \( T_{M1} \) on this value. If it rejects, increment the counter. If it accepts, copy the last \( n-\ell \) letters from the original input onto the main tape and run the same steps as \( T_{M2} \) on this value. If these accept, accept. If not, increment the counter. If the counter exceeds \( n \), reject.

Basically, we are taking a string \( w_1w_2...w_n \) and checking if any of the following are true:

\[
\ell \in L_1 \land w_1w_2...w_\ell \in L_2 \\
\ell \in L_1 \land w_1...w_\ell \in L_2 \\
w_1w_2\ell \in L_1 \land w_3...w_n \in L_2 \\
:\
\ell \in L_1 \land w_1w_2...w_\ell w_{\ell+1}...w_n \in L_2 \\
w_1w_2...w_\ell \in L_1 \land \ell \in L_2 
\]

6) Use a multitape machine for both. Copy the input onto a second tape. Run one step from \( T_{M1} \) on tape 1, followed by running one step from \( T_{M2} \) on tape 2. Continue alternating steps between tapes. For unique acceptance, accept if either tape accepts, reject only if both tapes do. Alternatively, we may loop on both tapes. But if a string is in \( L_1 \cup L_2 \), this machine will accept it.
6) For intersection, only accept if both tapes do. Reject if at least one tape does. This machine may loop as well, but it accepts all strings in $L_1 \cap L_2$.

The clever twist to this proof was alternating steps between the two simulations. The reason the previous proof didn't suffice if because simulating the steps of $TM_1$ on the input may never halt. This would be a problem in trying to recognize $L_1 \cup L_2$, because the input string may belong to $L_2$, but if the steps of $TM_1$ never finished, we'd never find out that the input string was in $L_2$. (Technically, running one after the other would suffice to show that Turing recognizable languages are closed under intersection.)

7) Let $L$ be a Turing recognizable language. Assume that $\Sigma$ is also Turing recognizable. Let $TM_1$ recognize $L$ and $TM_2$ recognize $\Sigma$. We will create a $TM$ $TM_3$ that DECIDES $L$. Let $TM_3$ simulate $TM_1$ and $TM_2$ alternating steps as described in question 6. Since this string MUST belong to either $L$ or $\Sigma$, we are guaranteed that one of our two simulations will accept and halt. If $TM_1$ accepts, $TM_3$ accepts. If $TM_2$ accepts, $TM_3$ rejects. Thus the trouble is, if the assertion were true, then all Turing recognizable languages are Turing decidable.