Problem 3.8b)

$$L = \{w \mid w \text{ contains twice as many 0s as 1s}\}$$

1. Scan the tape and mark the first 1 that hasn't been marked. If no unmarked 1's are found, go to stage 5. Else move head back to start of tape.

2. Scan the tape until an unmarked 0 is found and mark it. If no 0s are found then reject.

3. Scan the tape once more until an unmarked 0 is found and mark it. If no 0s are found then reject.

4. Move the head back to the start of the tape and go to stage 1.

5. Move the head back to the start of the tape. Scan tape for unmarked 0s. If none are found then accept. Else reject.

Problem 3.8c)

$$L = \{w \mid w \text{ does not contain twice as many 0s as 1s}\}$$

1. Scan the tape and mark the first 1 that has not been marked. If no unmarked 1's are found, go to stage 5. Else move the head back to the start of the tape.

2. Scan the tape until an unmarked 0 is found and mark it. If no 0s are found then accept.

3. Scan the tape once again until an unmarked 0 is found and mark it. If no 0s are found, accept.

4. Move the head back to the start of the tape and go to stage 1.

5. Move the head back to the start of the tape. Scan the tape to see if any unmarked 0s are found.
If none are found, reject. Else accept.
Else reject.

Problem 3.9a)

To prove that 2-PDA is more powerful than 1-PDA, we need to give a language that is accepted by 2-PDA but not 1-PDA. Suppose you have a language $L = \{a^n b^n c^n \mid n \geq 0\}$.
By using 2-PDA, you have two stacks. Initialize each stack with a $. Then, whenever a character 'a' is seen, push 'a' into both stacks. Whenever 'b' is seen, pop 'a' from the first stack. Similarly, whenever 'c' is seen, pop 'a' from the second stack. When the last 'c' is seen, accept if the symbols were seen in the correct order and if we have a $ on top of the stack, accept the language.
However, 1-PDAs cannot recognize this language because L is not context free.

Here, we showed that 2-PDA recognizes every language that 1-PDA recognize (2-PDA using 1 stack is same as 1-PDA), and also showed a language that 2-PDA can recognize but 1-PDA cannot.
Therefore, we have proved that 2-PDAs are more powerful than 1-PDAs

Problem 3.9b)

Based on the 'hint' we can simulate a Turing machine tape with two stacks. Here's a brief explanation on how to simulate a Turing machine with two stacks. Each stack (let's call it left stack and right stack) will simulate either the left half or the right half of the tape. Let the symbol on the top of the right stack denote the current position of the tape. We pop a symbol from the left stack and push a symbol on the right to move left on the tape and do the opposite for moving right. This means that 2-PDAs can do anything that a Turing machine can do.
Turing machines can simulate a k-PDAs for a finite k, which means that Turing machines can simulate 3-PDAs, 4-PDAs, and so on. Therefore, 2-PDAs is just as powerful as 3-PDAs. Therefore, we have proved that 3-PDAs are not more powerful than 2-PDAs.

Problem 3.11

A TM with a doubly infinite tape can simulate an ordinary TM. It marks the left-hand end of the input to detect and prevent the head from moving off of that end. To simulate the doubly infinite tape TM by an ordinary TM, we show how to simulate it with a 2-Tape TM. The first tape of the 2-tape TM is written with the input string and the second tape is blank. We cut the tape of the doubly infinite tape TM into two parts, at the starting cell of the input string. The portion with the input string and all the blank spaces to the right appear on the first tape of the 2-tape TM. The portion to the left of the input string appears on the second tape in reverse.

Problem 3.13

(i) Show that this Turing machine variant is not equivalent to the usual version
We can simulate any DFA on this Turing machine. This Turing machine cannot read anything that's previously written on the Tape because it cannot move left, which means given a stream of input, it only has one-way access to it.

(ii) What class of languages do these machines recognize?
This Turing machine only recognizes regular languages. We can simply construct an NFA that recognizes the same as languages this Turing machine. Construct this NFA as follows:
(i) when the Turing machine is in its control state NFA stores the symbol
(ii) when the Turing machine makes modifications, the NFA keeps track of these changes using epsilon transitions
(iii) when the Turing machine moves right, the NFA will simply read the next symbol.

Problem 3.15b)

Let A and B be decidable languages and AB be the concatenation of language A and B.
Let M_A and M_b be Turning machines that decides A and B respectively. To prove that AB is decidable, we construct a Turning Machine M that decides AB.

M = "On input w:
1. Non-deterministically partition w into strings xy
2. Input x into M_A and y into M_B
3. Accept if both M_A and M_B accept."

Problem 3.15d)

For a decidable language L, let M_L denote the Turing Machine that decides L. Then the TM for the complement is M_{L^c} which on input w, accepts if M_L rejects, and accepts otherwise.

Problem 3.16c)

For a Turing recognizable language L, we construct a non-deterministic Turing machine M_L that recognizes L we construct a Turning Machine M that decides AB.

M_L = "On input w:
1. Non-deterministically cut w into parts $w_1w_2....w_n$
2. Run M_L on w_i for all i. If M_L accepts all of them, accept. If M_L halts and rejects for any i, reject."

Problem 3.16d)

Let A and B be decidable languages. Let M_A and M_B be Turing machines that decides A and B respectively. We construct a Turing Machine MAB that recognize $A \cap B$.

$M_{A \cap B}$ = "On input w:

1. Run M_A on w. If it halts and rejects, reject. If it accepts, go to step 2.
2. Run M_B on w. If it halts and rejects, reject. If it accepts, accept."