Please note that there is more than one way to answer most of these questions. The following only represents a sample solution.

(1) (a) Give a high level description of a (deterministic) TM that, given input $w_1#w_2#\ldots#w_\ell$ with ‘elements’ $w_i \in \{0,1\}^*$ for $1 \leq i \leq \ell$, accepts if $w_i \leq w_j$ for all $1 \leq i < j \leq \ell$, and rejects otherwise.

Solution:

Since, the operator $\leq$ was never defined, there are various definitions we could choose for it. This solution is for the choice of treating each $w_i$ as a string and defining $w_i \leq w_j$ iff $|w_i| \leq |w_j|$. Thus, we will be using string length for comparisons. (Other possible choices include, using some lexicographic ordering, or treating each $w_i$ as a binary integer and taking $\leq$ as defined on integers.) The description follows:

$M = \text{"On input } w_1#w_2#\ldots#w_\ell:\$

1. Scan right until the first 0 or 1 and replace it with a special tape symbol $\$$ (that will be used to mark where the leftmost remaining 0 or 1 was located) and move right. If a $\sqcup$ is encountered before a 0 or 1, then accept the string (We processed the last symbol in the string and all the inequalities held).
2. Scan right until a $#$ is encountered. If a $\sqcup$ is encountered before a $#$ move left and go to (6.)
3. Scan to right to the next 0 or 1. If another $#$ or a $\sqcup$ is encountered before a 0 or 1 then reject (the current string is shorter than one to its left).
4. Replace the current symbol with a special tape symbol $X$ moving to the right.
5. Go to (2.).
6. Scan left until a $\$$ is encountered. Then move right and go to (1.)."

$M$ compares each string by marking off the first character of each string on its first pass, the second character of each string on its second, etc. If it marks off a character in a string to the left of a string that has no more characters to mark off, then it rejects. Otherwise it continues until it only marks off the characters in the last string. When it has marked all the characters then it accepts.

Then give a formal description of this TM.

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ with
1. $Q = \{q_0, q_1, q_2, q_3, q_a, q_r\}$,
2. $\Sigma = \{0, 1, \#\}$,
3. $\Gamma = \Sigma \cup \{\$, $X, \sqcup\}$,
4. and $\delta$ described by the following table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>$#$</th>
<th>$$</th>
<th>$X$</th>
<th>$\sqcup$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1, $, $R$</td>
<td>$q_1, $, $R$</td>
<td>$q_0, #$, $R$</td>
<td>$q_r, $, $R$</td>
<td>$q_0, X, R$</td>
<td>$q_a, \sqcup, R$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_0, 0, R$</td>
<td>$q_1, 1, R$</td>
<td>$q_0, #, R$</td>
<td>$q_r, #, R$</td>
<td>$q_r, X, R$</td>
<td>$q_3, #, L$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_1, X, R$</td>
<td>$q_1, X, R$</td>
<td>$q_r, #, R$</td>
<td>$q_r, #, R$</td>
<td>$q_2, X, R$</td>
<td>$q_r, #, R$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_3, 0, L$</td>
<td>$q_3, 1, L$</td>
<td>$q_3, #, L$</td>
<td>$q_0, $, $R$</td>
<td>$q_3, X, L$</td>
<td>$q_r, #, R$</td>
</tr>
</tbody>
</table>

This transition table performs as described by the informal description above. The entries for $q_a$ and $q_r$ are not shown because once the machine enters an accept or reject state it remains there. All of the transition to the reject state, except for the transition from $q_2$ with $\#$ and $\sqcup$ on the tape, are there because the machine should never be in that state with those symbols under the tape head if the string was formatted properly.

(b) Suppose that the TM is provided with an output tape, and at each transition it can either write a symbol to the output tape and move right, or can not write on the output tape and therefore stay in the same place. Give a high level description of TM with output tape that accepts if the input is sorted and rejects if it is not (as in the (a) part), but also writes the elements in sorted order on the output tape, again separated by $\#$.

Solution:

Since the output tape cannot move back after it has written we have to be sure to output in the sorted order before copying. We will use $M$ from part (a) as a subroutine for this part and the bubble sort to sort the strings.

$N = \text{“on input } w = w_1\#w_2\#\ldots\#w_\ell:\n1. \text{ Simulate } M \text{ on the input } w \text{ (This can be done by making a copy of } w \text{ at the end of the input string and running } M \text{ on the copy and then deleting the copy so that the original } w \text{ is not altered). If } M \text{ accepts, write } w \text{ to the output tape and then accept. If } M \text{ rejects, proceed to (2).}\n2. \text{ Repeat the following step For } j = \ell \text{ down to 2 (For loops can be done by putting counters at the end of the input string):}\n   a. \text{ Repeat the following steps For } i = 1 \text{ up to } j - 1: \n      i. \text{ Simulate } M \text{ on input } w_i\#w_{i+1} \text{ (The } w_i \text{ can be found from the loop counter by zigzagging between the counter and the } \# \text{ separating the strings). If } M \text{ rejects, swap } w_i \text{ and } w_{i+1}. \text{ In either case then proceed to the the next step of the iteration.}\n   3. \text{ Write the now sorted string on the input tape to the output tape and then reject.”}
(a) **Show that 2-PDAs are more powerful than 1-PDAs.**

**Solution:**

Clearly 2-PDAs are at least as powerful as 1-PDAs, since a 2-PDA can simulate a 1-PDA, by just not using its second stack. To see that they are indeed more powerful, we need to show a language they recognize, but that 1-PDAs do not. Let $L = \{0^i1^i2^i | i \geq 0\}$. We have seen previously that $L$ is not context-free and hence that there is not a 1-PDA that recognizes it. We will now give a high level description of a 2-PDA, $P$ that recognizes $L$.

Push a special symbol on each stack to mark their bottoms. Then for each 0 seen in the input string, push a zero onto both stacks. Nondeterministically, guess when the 0s are finished. Then for each 1 seen, pop a 0 from the first stack. Nondeterministically guess when the 1s are finished. Then for each 2 seen, pop a 0 from the second stack. If the special symbol marking the bottoms of the stacks is on top of both stacks after the last 2 is read and the symbols were seen in the proper order, then accept. Otherwise reject.

Clearly, $P$ accepts $L$ since it keeps track of how many 0s were seen first and uses both stacks to ensure that the number of 1s and 2s match. Thus, 2-PDAs recognize every language 1-PDAs recognize, but also recognize at least one language that 1-PDAs cannot. Therefore, 2-PDAs are more powerful than 1-PDAs.

(b) **Show that 3-PDAs are not more powerful than 2-PDAs.**

First we will show that 2-PDAs are at least as powerful as TMs by showing how to simulate a TM, $M = (Q_M, \Sigma, \Gamma_M, \delta_M, q_0, q_a, q_r)$, with a 2-PDA, $P = (Q_M \cup Q_P, \Sigma, \Gamma_M \cup \{\$\}, \delta_M \cup \{\$\}, q'_0, \{q_a\})$. Note that the states of $P$ have all the states of $M$ plus extra ones needed for the simulation and a new start state. Also note there are two stack alphabets, that have a special symbol not $\$\$ in the TM tape alphabet to mark the bottom of the stack.

$P =$ “on input $w = w_1w_2\ldots w_n$, with each $w_i \in \Sigma$:

1. Push a $\$\$ on each stack.
2. Read in $w$ onto the first stack. The reverse of the input string is now on the stack. Note that since we have read the whole input string, every other transition of $P$ will now be $\epsilon$-transitions.
3. Pop each symbol on the first stack and push it on the second stack until the $\$\$ is on top of the first stack and move the machine to state $q_0$. Now the second stack contains $w$ in the proper order. We will now use the second stack as the top being the tape head and everything under it is what is to the right of the tape head on the tape of $M$. The first stack will be used to store what is to the left of the tape head on the tape of $M$. 

3
4. Now we will simulate $M$’s transition by showing how a left and right transition are accomplished with $P$:

a. Left transition: If $\delta_M(q_i, w_i) = (q'_i, w'_i, L)$, then we have $w_i$ on top of the second stack.
   i. If $\dollar$ is on top of the first stack, reject, since $M$ would hang by moving off the left edge of the tape.
   ii. Otherwise, pop $w_i$ from the top of the second stack and push $w'_i$ onto the second stack.
   iii. Pop the symbol of the top of the first stack and push it onto the second stack (The symbol that was to the left of the tape head is now under the tape head), switching the state of the machine to $q'_i$.
   iv. If $q'_i$ is the accept state, then $P$ accepts $w$. If $q'_i$ is the reject state, then $P$ rejects $w$. Otherwise return to (4).

b. Right transition: If $\delta_M(q_i, w_i) = (q'_i, w'_i, R)$, then we have $w_i$ on top of the second stack.
   i. Pop $w_i$ from the top of the second stack and push $w'_i$ onto the first stack. (Now the tape head is on the symbol to the right of the previous head position)
   ii. If $\dollar$ is now on top of the second stack, push $\_|$ onto the second stack (The tape head moved right from the last non-blank symbol of the tape). In either case switch the state of the machine to $q'_i$.
   iii. If $q'_i$ is the accept state, then $P$ accepts $w$. If $q'_i$ is the reject state, then $P$ rejects $w$. Otherwise return to (4).

Since $P$ uses its stacks to simulate the transitions of $M$ and accepts and rejects as $M$ does, we see that 2-PDAs are at least as powerful as TMs.

We still have to show that 3-PDAs are no more powerful than 2-PDAs. Now we will show that 3-PDAs are no more powerful than TMs and hence by what was just shown above, no more powerful than 2-PDAs. Let $Q$ be a 3-PDA, we will simulate it with a nondeterministic 4-tape TM $N$ (which is equivalent to a TM by Corollary 3.15 and 3.18 of Sipser) as follows:

- The first tape is the input. The second tape is used as the first stack, the third tape is used as the second stack, and the fourth tape is used as the third stack.
- If a stack is altered by a push without a pop then $N$ moves the appropriate tape head to the right without changing the current symbol and then writes the symbol pushed onto the stack to the tape while staying put.
- If a stack is altered by a pop without a push, then $N$ replaces what is under the appropriate tape head with a blank and moves to the left.
- If a stack is altered by a pop and push, then $N$ alters the appropriate tape by writing the symbol pushed into the current cell and staying put.
- If a stack is not changed, then $N$ writes the same symbol under the tape head back to the tape and stays put.
- If a symbol from the input is consumed during the step then, $N$ moves the first tape head to the right.
- If a symbol is not consumed during the transition, then $N$ copies the current element under the first tape head and stays put.
• All these actions can be done in parallel on the tapes at the same time since they take only one turn. The only exception is a push without a pop as described in the second bullet. In that case all tape heads are kept the same except for the stacks effected by pushing without popping, they move to the right until a blank is under the tape head, while the machine remains in the same state. Since the entry under the tapes representing stacks without a pop doesn’t effect the transition, the computation is not changed.

• Lastly, since $N$ is nondeterministic, it can nondeterministically guess which transition to take the same way $P$ does.

Thus, we see that TMs are at least as powerful as 3-PDAs, since we can build a TM to simulate a given 3-PDA. Therefore we conclude that 3-PDAs are no more powerful than 2-PDAs (They are in fact equally powerful, and the same power as TMs, but we only needed to show one way of the equivalence here. It is easy to complete the chain by noting that 3-PDAs can simulate 2-PDAs by only using two of their stacks).

(3) 3.12 (TM with left-reset) and liberal TMs: TM Variations

(a) 3.12: TM with left-rest

Solution:

Let $M$ be a Turing machine. I will give a Turing machine with left reset (TMLR), $M_{LR}$, such that $L(M) = L(M_{LR})$. The idea used is to mark the cell to the left of the current tape head so that we can use it in finding the current position after a LEFT transition of $M$ using $M_{LR}$’s RESET.

$M_{LR} = \text{"On input } w;\text{"}$

1. If the current state $q$ is the accept or reject state of $M$, go to step 4. Otherwise, $M_{LR}$ will simulate a right or left transition of $M$.

2. Right Transition:
   a. If the current tape symbol is $a$, and the transition function of $M$, $\delta_M(q,a) = (q',b,R)$, then replace the $a$ with a $b$ and RESET.
   b. Scan right for a marked tape symbol (i.e. has a dot on it or something else to distinguish it from the symbols of $M$). If none is found RESET and mark the first tape symbol and move the tape head to the right changing the state of $M_{LR}$ to $q'$ and go to (1) (This happens if a $R$ move was performed from the leftmost tape cell). If a marked symbol is found remove the mark, move the tape head to the right.
   c. Mark the symbol under the tape head and move to the right changing the state of $M_{LR}$ to $q'$ and go to (1) (This moves the symbol left of the tape head one to the right for a $R$ transition).

3. Left Transition:
   a. If $\delta_M(q,a) = (q',b,L)$, replace the $a$ with a $b$ and RESET.
   b. If the first symbol is marked, remove the mark, RESET and change the state of $M_{LR}$ to $q'$ and go to (1) (This happens if the $L$ move was to the first tape position).
c. Otherwise, scan right for a marked tape symbol. If none is found reject (This happens if a \( L \) move is tried from the leftmost tape cell so the machine hangs).
d. If a marked symbol is found RESET and mark the first symbol.
e. If the next symbol is marked unmark it, RESET, and move right to the second tape cell changing the state of \( M_{LR} \) to \( q' \) and go to (1) (This happens if the \( L \) move is to the second position).
f. If the second cell is not marked repeat the following loop:
   i. RESET. Move right to the first marked cell. Unmark it and move right.
   ii. Mark the current cell and move right.
   iii. If the cell is unmarked return to (i).
   iv. Otherwise unmark it and RESET.
   v. Move right back to the first marked cell. Move right and change the state of \( M_{LR} \) to \( q' \) and go to (1) (This loop only halts when there is a state of the tape when two marked cells are next to each other marking the two cells left of the cell we started the \( L \) transition on. The cell that was initially to the left is unmarked at the end and returned to with \( M_{LR} \) in the proper state).

4. If \( q' \) is the accept state of \( M \), accept. If \( q' \) is the reject state of \( M \), reject.”

Clearly, \( L(M) = L(M_{LR}) \) since \( M_{LR} \) emulates the transitions of \( M \) and only accepts or rejects when \( M \) does. Thus, a TMLR recognizes the class of Turing-recognizable languages.

(b) **Liberal TM**: A TM with Stay put (\( S \)) instead of right move \( R \).

**Solution:**

A *liberal* TM (LTM) can only ever view the first symbol of the input string (since the tape head always starts at the leftmost symbol of the input string and a LTM cannot move to the right). Any actions it can take are based on that symbol alone. Thus, a LTM recognizes an even smaller set of languages than regular languages. The family of languages, \( \mathcal{L} \), that a LTM can recognize are for each \( X \subseteq \Sigma_\epsilon \), there is an LTM that can recognize \( \{xy \mid x \in X \text{ and } y \in \Sigma^* \text{ if } x \neq \epsilon \text{ and } y = \epsilon \text{ if } x = \epsilon \} \). Thus, LTMs can recognize languages that start with any subset of symbols from the input alphabet plus the empty string (If one of those symbols is seen at the start of the tape accept, otherwise reject). Some special cases of \( \mathcal{L} \) include: \( \{\epsilon\} \) (\( X = \{\epsilon\} \)) which accepts only if the tape is blank; \( \emptyset \) (\( X = \emptyset \)) which always rejects; and \( \Sigma^* \) (\( X = \Sigma_\epsilon \)) which always accepts. LTMs cannot recognize any other language because they require being able to view more than the first position on the tape. For example, an LTM cannot recognize the regular language \( \{a\} \), since it cannot ensure that nothing follows the \( a \). Thus, it cannot distinguish between the strings \( a \) and \( ab \). Thus, LTMs are not as powerful as regular languages.
(4) 3.15 (c), (e) and 3.16 (b), (c): Closure properties of Turing-decidable and Turing-recognizable languages

Solution:

Most of our solution will use nondeterministic TMs to partition the input string into pieces (by placing marks on the string). In this way, we need only consider one way to partition up the string, as separate branches will handle each case. If we used determinism, we would have to try all possible ways. (This could be done too, but is more complicated to explain.)

3.15(c): Star for Turing-decidable languages

Let $L$ be a Turing-decidable language. Then there is some deterministic 1-tape TM, $M$, that decides $L$. We form a nondeterministic 2-tape TM, $M'$ that decides $L^*$.

$M' = \text{“On input } w:"
1. Scan the input tape (tape 1) from left to right until a blank is encountered. For each tape cell read, nondeterministically choose either to write the same symbol, or to write the symbol with a mark on it. Any number of tape cells can be marked in this process.
2. Position the tape head on the first tape at the left hand end.
3. While the symbol on the first tape under the tape head is not a blank:
   a. Make the second tape entirely blank. (Start at the left hand end, and move right writing blanks until the first blank is read.)
   b. Copy all symbols from the first tape starting at the current position of the tape head up to and including the marked symbol onto the second tape or blank if the end of the input is reached (starting at the left hand end of the second tape), removing the mark on the last symbol. Leave the tape head on the first tape positioned on the cell right after the last (marked) symbol copied.
   c. Run $M$ on the contents of the second tape. If it rejects, then $M'$ rejects (that branch terminates) and stops. If it accepts, continue from (3).
4. Accept.”

$M'$ will only accept if if there is a branch of the nondeterminism that accepts and hence it exits the loop. That means it is possible to partition the input string into substrings that are all in $L$. Otherwise all branches will reject, which means that $M'$ will reject.

3.15(e): Intersection for Turing-decidable languages

Let $L_1$ and $L_2$ be Turing-decidable languages. Then there exists 1-tape deterministic TMs $M_1$ and $M_2$ such that $M_1$ decides $L_1$ and $M_2$ decides $L_2$. We will construct a TM, $M$ such that $M$ decides $L_1 \cap L_2$.

$M = \text{“On input } w:"
1. Run $M_1$ on input $w$. If it rejects, then $M$ rejects. If it accepts, continue to (2).
2. Run $M_2$ on input $w$. If it rejects, then $M$ rejects. If it accepts, then $M$ accepts.”

Clearly, $M$ accepts $w$ iff both $M_1$ and $M_2$ accept $w$ iff $w \in L_1 \cap L_2$. $M$ rejects $w$, otherwise. Therefore, $M$ is a decider for $L_1 \cap L_2$. 


3.16(b): Concatenation for Turing-recognizable languages

Let \( L_1 \) and \( L_2 \) be Turing-decidable languages. Then there exists TMs \( M_1 \) and \( M_2 \) such that \( M_1 \) recognizes \( L_1 \) and \( M_2 \) recognizes \( L_2 \). We will construct a TM, \( M \) such that \( M \) recognizes \( L_1 \circ L_2 \).

\( M = \) "On input \( w \):
1. Scan the input tape (tape 1) from left to right until a blank is encountered. For each tape cell read, nondeterministically choose either to write the same symbol, or to write the symbol with a mark on it. Any number of tape cells can be marked in this process.
2. If more than one cell on the input tape is marked by the process in (1), then \( M \) rejects (that branch terminates).
3. Position the tape head on the first tape at the left hand end.
4. If the symbol on the first tape under the tape head is not a blank, copy all symbols from the first tape starting at the current position of the tape head up to and including the marked symbol onto the second tape (starting at the left hand end of the second tape), removing the mark on the last symbol. Leave the tape head on the first tape positioned on the cell right after the last (marked) symbol copied.
5. Run \( M_1 \) on the contents of the second tape. If it rejects, then \( M \) rejects (that branch terminates) and stops. If it runs forever then that branch will run forever. If it accepts, continue.
6. Make the second tape entirely blank. (Start at the left hand end, and move right writing blanks until the first blank is read.)
7. Copy all symbols from the first tape starting at the current position of the tape head up to the first blank (starting at the left hand end of the second tape).
8. Run \( M_2 \) on the contents of the second tape. If it rejects, then \( M \) rejects (that branch terminates) and stops. If it runs forever then that branch will run forever. If it accepts, then \( M \) accepts."

If it is possible to break the string into two pieces such that the first piece is accepted by \( M_1 \) and the second piece by \( M_2 \), then it will be accepted by \( M \). If none of the possible branches on input \( w \) has both \( M_1 \) and \( M_2 \) accept, by either rejecting or running forever, then \( w \) will not be accepted \( M \). We cannot guarantee it will be rejected though. Hence, \( M \) recognizes \( L_1 \circ L_2 \).

3.15(c): Star for Turing-recognizable languages

Let \( L \) be a Turing-recognizable language. Then there is some TM, \( M \), that recognizes \( L \). We form a nondeterministic 2-tape TM, \( M' \) that recognizes \( L^* \).

\( M' = \) "On input \( w \):
1. Scan the input tape (tape 1) from left to right until a blank is encountered. For each tape cell read, nondeterministically choose either to write the same symbol, or to write the symbol with a mark on it. Any number of tape cells can be marked in this process.
2. Position the tape head on the first tape at the left hand end.
3. While the symbol on the first tape under the tape head is not a blank:
   a. Make the second tape entirely blank. (Start at the left hand end, and move right writing blanks until the first blank is read.)
b. Copy all symbols from the first tape starting at the current position of the tape head up to and including the marked symbol onto the second tape or blank if the end of the input is reached (starting at the left hand end of the second tape), removing the mark on the last symbol. Leave the tape head on the first tape positioned on the cell right after the last (marked) symbol copied.

c. Run $M$ on the contents of the second tape. If it rejects, then $M'$ rejects (that branch terminates) and stops. If it runs forever, then that branch of the computation runs forever. If it accepts, continue from (3).

4. Accept.”

$M'$ will only accept if if there is a branch of the nondeterminism that accepts and hence it exits the loop. That means it is possible to partition the input string into substrings that are all in $L$. Otherwise all branches either reject or run forever. In either case $M'$ will not accept. Thus, $M'$ recognizes $L^*$. 
TM to decide Cliques

Solution:

(a) Describe a way to encode an adjacency matrix $A$ and a subset $W$ as a Turing machine input.

Given a graph $G = (V, E)$, with a finite ordered set $V = \{v_1, v_2, \ldots, v_n\}$ of vertices, and a finite set $E$ of edges, where each edge is a pair $\{x, y\} \subseteq V$. We will encode a subset $W$ of $V$ as the string $< W > = w_1 w_2 \ldots w_n \in \{0, 1\}^*$, where $w_i = 1$ if $v_i \in W$ and $w_i = 0$ otherwise. The adjacency matrix for $G$ is a $|V| \times |V|$ matrix $A = (a_{ij})$ with entries from $\{0, 1\}$, where $a_{ij}$ is 1 if $\{v_i, v_j\} \in E$ and is 0 otherwise. Since $A$ is a symmetric matrix, we only need to encode the upper triangular matrix. Also since to detect cliques we do not check if an edge is a loop, we do not need to encode the diagonal. Thus, we will encode $A$ as the string $< A > = #a_{12}a_{13}\ldots a_{1n}##a_{23}\ldots a_{2n}##\ldots##a_{(n-1)n}$. That is after the first # records the edges from $v_1$, in between the next two #s the edges from $v_2$ not including the edges already captured by $v_1$ and so on. Finally we encode $< W, A > = < W > < A >$.

(b) Using this encoding, give a high level description of a deterministic Turing machine that decides whether $W$ is a clique in $A$.

$M =$ “on input $< W, A >$:
1. Mark the current symbol with a special tape symbol $\$$(This is the current vertex we are considering) and move to the right. If a 0 was replaced (in that case, we do not care about edges from this vertex) go to (2). If a 1 was replaced (in this case we have to be sure that there is an edge from all vertices we have included in our clique before) go to (5).
2. Move right until either a # is seen or a $\$\#$ is seen. If a $\$\#$ is seen first accept (We’ve processed all vertices and the edges were found as required by a clique). If a # is seen continue from (3). IF a # is seen continue from (4).
3. Move right until a 0 or 1 is seen. Replace that symbol with a X moving to the right (We do not care if there is an edge between these two vertices since the latter vertex is not included in the clique). Go to (2).
4. Replace the # with a X and move to the right replacing every symbol with an X until a # or $\$\#$ is under the tape (We do not need to have edges from the current vertex to others since it is not in the clique). Go to (8).
5. Move right until either a # is seen or a $\$\#$ is seen. If a $\$\#$ is seen first accept (We’ve processed all vertices and the edges were found as required by a clique). If a # is seen continue from (6). IF a # is seen continue from (7).
6. Move right until a 0 or 1 is seen. IF a 1 is seen (There is an edge between these two vertices in the clique), replace it with a X moving to the right and go to (5). IF a 0 is seen (There is not an edge between two vertices that were supposed to be in the clique), reject.
7. Replace the \# with a \(\dot{\#}\) and move to the left (We will need to make sure that all vertices we have yet to see that are in our clique will have an edge from this vertex. That is what the dot signifies). Go to (8).

8. Move to the left until a $ is encountered. Move to the right and Go to (1).

\(M\) ensures that each vertex in the clique has edges between them, and only accepts if they do. It rejects if a vertex is found in the clique that does not have an edge to another vertex. In always either accepts or rejects after processing all the vertices of the graph, and so we that it is a decider for whether \(W\) is a clique of the graph specified by \(A\).

d) Give a formal description for your method in the (b) part.

Let \(M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)\) with
1. \(Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_a, q_r\}\),
2. \(\Sigma = \{0, 1, \#\}\),
3. \(\Gamma = \Sigma \cup \{$, \#\}, \) and
4. \(\delta\) given by the following transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>(q_1, $, R)</td>
<td>(q_1, $, R)</td>
<td>(q_r, #, R)</td>
<td>(q_r, #, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, #, R)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_1, 0, R)</td>
<td>(q_1, 1, R)</td>
<td>(q_3, X, R)</td>
<td>(q_r, $, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, #, R)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_1, X, R)</td>
<td>(q_1, X, R)</td>
<td>(q_r, #, R)</td>
<td>(q_r, #, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, #, R)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_3, X, L)</td>
<td>(q_3, X, L)</td>
<td>(q_6, #, L)</td>
<td>(q_r, $, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, #, R)</td>
</tr>
<tr>
<td>(q_4)</td>
<td>(q_4, 0, R)</td>
<td>(q_4, 1, R)</td>
<td>(q_6, #, L)</td>
<td>(q_r, $, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, X, R)</td>
<td>(q_r, #, R)</td>
</tr>
<tr>
<td>(q_5)</td>
<td>(q_r, 0, R)</td>
<td>(q_4, X, R)</td>
<td>(q_r, #, R)</td>
<td>(q_r, #, R)</td>
<td>(q_5, X, R)</td>
<td>(q_5, X, R)</td>
<td>(q_r, #, R)</td>
</tr>
<tr>
<td>(q_6)</td>
<td>(q_6, 0, L)</td>
<td>(q_6, 1, L)</td>
<td>(q_6, #, L)</td>
<td>(q_0, $, R)</td>
<td>(q_6, X, L)</td>
<td>(q_6, X, L)</td>
<td>(q_6, #, L)</td>
</tr>
</tbody>
</table>

This transition table performs as described by the informal description above. The entries for \(q_a\) and \(q_r\) are not shown because once the machine enters an accept or reject state it remains there. All of the transition to the reject state, except for the transitions from \(q_5\) with 0 on the tape, are there because the machine should never be in that state with those symbols under the tape head if the string was formatted properly.

d) Using this checking method for cliques as a ‘subroutine’, give a high level description of a deterministic Turing machine that decides whether \(A\) has a clique with at least \(k\) vertices; note that \(k\) is also to be provided on the input tape, and you must determine how.

We will encode \(k\) as \(< k > = 1^k\), that is the unary representation of \(k\). Therefore we will encode \(< k, A > = < k > < A >\). We will now define a TM \(N\) that uses \(M\) as a subroutine to decide if \(A\) has a clique of size at least \(k\).

\(N = \text{“on input } < k, A >:\)
1. If \(k\) is 0 (machine starts with a \# under the tape head, then accept (by default there is a clique of size zero). Otherwise, continue.
2. Generate the string \( w = 0^{n-1}1 \) at the end of the input tape, where \( n \) is determined by zigzagging back and forth among the \# of the input.

3. Repeat the following until \( w = 10^n \):
   a. Check if \( w \) contains \( \geq k \) 1s. This is determined by zigzagging between \( w \) and the \( k \) at the start of the input. If it does not, Go to (c). If it does continue.
   b. Simulate \( M \) on input \( w, \langle A \rangle \). If \( M \) accepts, accept. If \( M \) rejects continue.
   c. Treating \( w \) as a binary integer, increment its value by one.

4. Reject (Every possible subset was seen and there was not a clique with the right size).

Clearly, \( N \) decides if there is a clique of the right size by checking if every possible subset of \( V \) with size larger than \( k \) is a clique. If it does not find a clique after trying all the subsets, then it rejects. If it finds one while checking then it accepts. In either case it always halts. Thus, we see that \( N \) decides the language of the problem.