2.4  b. \( S \rightarrow OR0 \mid 1R1 \mid \varepsilon \)  
\( R \rightarrow OR \mid 1R \mid \varepsilon \)  

c. \( S \rightarrow 0 \mid 1 \mid 00S \mid 01S \mid 10S \mid 11S \)  
e. \( S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon \)  
f. \( S \rightarrow S \)

For each variable \( A \) in the CFG, characterize the string derived by \( A \), i.e., \( A \Rightarrow^* x \) iff \( x \) is of the form \( \ldots \)

Problems 3 and 4 are given in problems 3.

2.9  A CFG \( G \) that generates \( A \) is given as follows:  
\[ G = (V, \Sigma, R, S), V = \{S, E_{ab}, E_{bc}, C, A\}, \text{ and } \Sigma = \{a, b, c\}. \]  
The rules are:

\[
\begin{align*}
S & \rightarrow E_{ab}C \mid AE_{bc} \\
E_{ab} & \rightarrow aE_{ab}b \mid \varepsilon \\
E_{bc} & \rightarrow bE_{bc}c \mid \varepsilon \\
C & \rightarrow Cc \mid \varepsilon \\
A & \rightarrow Aa \mid \varepsilon 
\end{align*}
\]

Initially substituting \( E_{ab} \) for \( S \) generates any string with an equal number of \( a \)'s and \( b \)'s followed by any number of \( c \)'s. Initially substituting \( E_{bc} \) for \( S \) generates any string with an equal number of \( b \)'s and \( c \)'s prepended by any number of \( a \)'s.

2.22  We construct a PDA \( P \) that recognizes \( C \). First it nondeterministically branches to check either of two cases: that \( x \) and \( y \) differ in length or that they have the same length but differ in some position. Handling the first case is straightforward. To handle the second case, it operates by guessing corresponding positions on which the strings \( x \) and \( y \) differ, as follows. It reads the input at the same time as it pushes some symbols, say \( 1s \), onto the stack. At some point it nondeterministically guesses a position in \( x \) and it records the symbol it is currently reading there in its finite memory and skips to the \( \\# \). Then it pops the stack while reading symbols from the input until the stack is empty and checks that the symbol it is now currently reading is different from the symbol it had recorded. If so, it accepts.

Here is a more detailed description of \( P \)'s algorithm. If something goes wrong, for example, popping when the stack is empty, or getting to the end of the input prematurely, \( P \) rejects on that branch of the computation.

1. Nondeterministically jump to either 2 or 4.
2. Read and push these symbols until read \( \# \). Reject if \( \# \) never found.
3. Read and pop symbols until the end of the tape. Reject if another \( \# \) is read or if the stack empties at the same time the end of the input is reached. Otherwise accept.
4. Read next input symbol and push 1 onto stack.
5. Nondeterministically jump to either 4 or 6.
6. Record the current input symbol \( a \) in the finite control.
7. Read input symbols until \( \# \) is read.
8. Read the next symbol and pop the stack.
9. If stack is empty, go to 10, otherwise go to 8.
10. Accept if the current input symbol isn't \( a \). Otherwise reject.
Problem 3.

(a) Suppose that $K$ were regular. Then the language $\overline{K \cap a^*a^*}$ would be regular. What is $K \cap a^*a^*$?

$$K \cap a^*a^* = \{ x \ y \ | \ x, y \in a^*, \text{ and } x \text{ is a permutation of } y \}$$

$$= \{ a^n \ a^n \ | \ n \geq 0 \}$$

Applying the Pumping Lemma for regular languages to $K \cap a^*a^*$ will show the non-regularity of $\overline{K \cap a^*a^*}$.

(b) Notice that for all $x, y \in \Sigma^*$, $x$ is a permutation of $y$ if and only if $\#_a(x) = \#_a(y)$ and $\#_b(x) = \#_b(y)$.

Thus, $x \ y \in K$ if and only if $\#_a(x) \neq \#_a(y)$ or $\#_b(x) \neq \#_b(y)$.

Now we have $K = \{ x \ y \ | \ x, y \in \{a, b\}^* \text{ and } \#_a(x) \neq \#_a(y) \}$

$$\cup \{ x \ y \ | \ x, y \in \{a, b\}^* \text{ and } \#_b(x) \neq \#_b(y) \}$$

Each disjoint is generated by a context-free grammar by considering $\neq$ as $<$ or $>$. (Similar samples were covered in lectures.)
Theorem 4. A string of the given language is of no form 
\[ w_1 \$ w_2 \$ \ldots \$ w_n \] 
for some \( n \geq 1 \) with \( w_i \in \Sigma^* \) 
for \( i = 1, 2, \ldots, n \) and \( w_i = w_j \) for some \( i, j \in \{1, 2, \ldots, n\} \).
Thus, we can consider the following disjunction:

1. \( w_1 \$ w_2 \$ \ldots \$ w_i \$ \ldots \$ w_n \) (\( i = n, j = 1 \))
   \[ w_i = w_n \]
   a string of m's
   a string of m's
   can be generated by
   can be generated by
   a CFG (in fact, regular)
   a CFG (in fact, regular)

2. \( w_1 \$ w_2 \$ \ldots \$ w_i \$ \ldots \$ w_j \$ \ldots \$ w_n \)
   Similar to above
   Can be generated
   Similar to above
   by a CFG

(\( i \neq j \), and \( w_i = w_j \))
2.35 Assume $G$ generates a string $w$ using a derivation with at least $2^b$ steps. Let $n$ be the length of $w$. By the results of Problem 2.26, $n \geq \frac{2^b}{2} \cdot 2^{b-1} > 2^{b-1}$. Consider a parse tree of $w$. The right-hand side of each rule contains at most two variables, so each node of the parse tree has at most two children. Additionally, the length of $w$ is at least $2^b$, so the parse tree of $w$ must have height at least $b + 1$ to generate a string of length at least $2^b$. Hence, the tree contains a path with at least $b + 1$ variables, and therefore some variable is repeated on that path. Using a surgery on trees argument identical to the one used in the proof of the CFL pumping lemma, we can now divide $w$ into pieces $uvxyz$ where $u^n x y^i z \in G$ for all $i \geq 0$. Therefore, $L(G)$ is infinite.

2.44 Let $M_A$ be a DFA that recognizes $A$, and $M_B$ be a DFA that recognizes $B$. We construct a PDA recognizing $A \circ B$. This PDA simulates $M_A$ on the first part of the string pushing every symbol it reads on the stack until it guesses that it has reached the middle of the input. After that it simulates $M_B$ on the remaining part of the string popping the stack for every symbol it reads. If the stack is empty at the end of the input and both $M_A$ and $M_B$ accepted, the PDA accepts. If something goes wrong, for example, popping when the stack is empty, or getting to the end of the input prematurely, the PDA rejects on that branch of the computation.

2.31 Assume $B$ is context-free and get its pumping length $p$ from the pumping lemma. Let $s = 0^{p+1}2^{2p}0^p$. Because $s \in B$, it can be split $s = uvxyz$ satisfying the conditions of the lemma. We consider several cases.

i) If both $u$ and $y$ contain only 0's (or only 1's), then $uv^2 xy^2 z$ has unequal numbers of 0s and 1s and hence won't be in $B$.

ii) If $u$ contains only 0s and $y$ contains only 1s, or vice versa, then $uv^2 xy^2 z$ isn't a palindrome and hence won't be in $B$.

iii) If both $u$ and $y$ contain both 0s and 1s, condition 3 is violated so this case cannot occur.

iv) If one of $u$ and $y$ contain both 0s and 1s, then $uv^2 xy^2 z$ isn't a palindrome and hence won't be in $B$.

Hence $s$ cannot be pumped and contradiction is reached. Therefore $B$ isn't context-free.

2.32 Assume $C$ is context-free and get its pumping length $p$ from the pumping lemma. Let $s = 1^p3^p2^p4^p$. Because $s \in C$, it can be split $s = uvxyz$ satisfying the conditions of the lemma. By condition 3, $uvy$ cannot contain both 1s and 2s, and cannot contain both 3s and 4s. Hence $uv^2 xy^2 z$ doesn't have equal number of 1s and 2s or of 3s and 4s, and therefore won't be a member of $C$, so $s$ cannot be pumped and contradiction is reached. Therefore $C$ isn't context-free.