Problem 1. Consider the CFG $G$ defined by the following productions:

$$S \rightarrow aS \mid Sb \mid a \mid b$$

(a) Prove by induction on the string length that no string in $L = L(G)$ has $ba$ as a substring.

Proof (induction): Let $P_n$ be the statement that no string $x \in L(G)$, $|x| = n$ has the substring $ba$.

Base Case: $n = 1$. The only strings of length 1 in $L$ are $a$ and $b$, neither of which has $ba$ as a substring. Therefore, $P_n$ holds for $n = 1$.

Assume $P_n$ holds true for some $n \geq 1$. We want to show that $P_{n+1}$ follows.

Take some string $w \in L$ of length $n$. Then $S \Rightarrow^* w$. To produce a string of length $n + 1$, we must follow the productions $S \rightarrow aS \rightarrow^* aw$ or $S \rightarrow Sb \rightarrow^* wb$. By the inductive assumption, $w$ has no substring $ba$. By prepending $a$ or appending $b$ we cannot create the substring $ba$. Hence $P_{n+1}$ holds true.

Therefore, $P_n$ holds true for all $n \geq 1$, and so no strings in $L$ have the substring $ba$. □

(b) $L(G) = a^*bb^* + aa^*b^*$, that is, the language of $a$’s followed by $b$’s, with at least one $a$ or one $b$.

Some notes:

• The induction in part (a) could be done as either an induction on the length of the strings in $L(G)$, or as an induction on the length of the sentential

  - in my humble opinion, it was easier to use the proof (as above) using the sentences (the strings in $L(G)$), rather than messing too much with the sentential

• A different approach was to prove a stronger statement: any sentence derived from $S$

  is of the form $a^*b^*$; and then show that this implies there is no string with substring $ba$

• A common error was to write that $L(G) = a^*b^*$. Note that this includes $\epsilon$ (the empty string), which simply is not true! The shortest strings in $L(G)$ are $a$ or $b$. 
Problem 2

(a) Let \( L = \{ w \in (a,b)^* : \#a(w) = \#b(w) \} \). Then the following is a grammar for \( L \):

\[
S \rightarrow aSb \mid bSaS \mid \varepsilon
\]

The idea behind this grammar is that if \( w \) is a string in \( L \), then \( w \) is either \( \varepsilon \), begins with \( a \), or begins with \( b \). If it begins with \( a \), we know there has to be a \( b \) somewhere further down the string. In particular, there must be a \( b \) somewhere in the string such that the substring between the starting \( a \) and this \( b \) is another string of \( L \). In other words, the substring between the \( a \) and the \( b \) has balanced numbers of \( a \)'s and \( b \)'s. Here's why: We can keep a tally of the imbalance between \( a \)'s and \( b \)'s in the string. Every time we see an \( a \), we add 1 to the tally. Every time we see a \( b \), we subtract 1. We know that this tally has to eventually be 0 at the end of the string for \( w \) to be in \( L \). If \( w \) starts with an \( a \), this tally starts out positive. At some point, it must return to 0. The first time that happens, we must have just decremented the tally upon seeing a \( b \). This is the \( b \) we want. The substring up to this \( b \) has balanced \( a \)'s and \( b \)'s. Since the rest of the string must preserve the balance between \( a \)'s and \( b \)'s, we know that it has to be itself another member of \( L \). This is the intuition behind the \( aSbS \) production.

Similarly, if \( w \) starts with a \( b \), we can take care of it with the \( bSaS \) production.

(b) Let \( L = \{ \text{rev}(b(n)) \$ b(n + 1) : n \geq 1 \} \). Then the following is a grammar for \( L \):

\[
S \rightarrow 1S0 \mid 0A1 \mid B0
\]
\[
A \rightarrow 0A0 \mid 1A1 \mid B
\]
\[
B \rightarrow 1\$1
\]

The intuition is that we create each string from the left and right ends and move toward the middle. That is, we write \( b(n) \) and \( b(n + 1) \) starting from their least significant bits (which will effectively reverse \( b(n) \) while keeping \( b(n + 1) \) in order). If the least significant bit of \( b(n) \) is 1, then the corresponding bit in \( b(n + 1) \) is 0. The “carry” from this will affect the following bits, until we see a 0 in \( b(n) \). This is captured by the \( S \rightarrow 1S0 \) production, where we are doing addition by 1, remembering that the carry bit will affect the next bit in the numbers. When we finally get to a 0 in \( b(n) \), we can set the corresponding bit in \( b(n + 1) \) to 1, and stop worrying about the carry. This is done with the \( S \rightarrow 0A1 \) production. From there, \( A \) simply produces the same bits on both sides. We stop when we reach the most significant bit, and use a \( A \rightarrow B \) production. Since leading zeros are not allowed, there must be 1’s around the \$ sign, thus \( B \rightarrow 1\$1 \).

There is one special case where the number \( n \) is of the form \( 2^k - 1 \). That is, \( b(n) \) is a string of 1’s. This is the only time when \( b(n + 1) \) will be longer than \( b(n) \). We deal with this by the \( S \rightarrow B0 \) production, where we give \( b(n + 1) \) an extra 0. (Note that if we use this production, we won’t be able to use any \( A \) productions, so \( b(n) \) is indeed a string of 1’s.)
Problem 3a. The example \( \{0^n1^n \mid n \geq 1 \} \) is enough. It is easy to see that the following grammar establishes this language to be a symmetric language, while we already know that it is not regular.

\[
S \rightarrow 0S1 | \epsilon
\]

Problem 3b. Let \( L \) be a regular language. Suppose we have a DFA for \( L \) given as \((Q, \Sigma, \delta, F, s)\) where \( Q \) is the set of states, \( \delta \) the transition function etc. Now, define the grammar to be \( G = (N, \Sigma, P, S) \). Define \( N \), the set of non-terminals of our grammar to \( \{S\} \cup Q \times Q \) i.e. of the form \((p, q)\) where \( p, q \in Q \). The terminals of our grammar are symbols of \( \Sigma \). The production rules are of the format

\[
S \rightarrow \epsilon \text{ if } s \in F
\]

\[
S \rightarrow a \text{ if } \delta(s, a) \in F
\]

\[
S \rightarrow a(p', q'b) \text{ if } \delta(s, a) = p' \land \delta(q', b) \in F.
\]

We also have the following productions

\[
(p, p) \rightarrow \epsilon
\]

\[
(p, q) \rightarrow a \text{ if } \delta(p, a) = q
\]

\[
(p, q) \rightarrow a(p', q')b \text{ if } \delta(p, a) = p' \land \delta(q', b) = q.
\]

Correctness follows by simple induction of the following claims.

\[
(p, q) \rightarrow^* w \iff \hat{\delta}(p, w) = q
\]

\[
S \rightarrow^* w \iff \exists f \in F, \hat{\delta}(s, w) = f.
\]

Problem 3c. There are many ways to prove this. Here is one. Suppose, as before, the grammar is \( G = (N, \Sigma, P, S) \). Over a single letter alphabet, the order of the letters do not matter. So we might as well consider the productions to be of the form

\[
A \rightarrow aaB
\]

\[
C \rightarrow \epsilon
\]

\[
D \rightarrow a
\]

1
where $A, B, C$ and $D$ are non-terminals. Let us first convert the rules $A \rightarrow aaB$ to rules of the form $A \rightarrow aB$ by introducing new non-terminals in $N$, and adding production rules. Now, we can define a FA $M = (Q, \Sigma, \delta, s)$ that has the set of states $Q = N \cup \{f\}$ where $f$ is a new state name that does not appear in $N$. The transition function is

$$\delta(A, a) = \{B \mid A \rightarrow aB \in P\} \cup \{f \mid A \rightarrow a \in P\}$$

The start state of the automata is $s = S$ and the final states are $F = \{f\} \cup \{S \mid S \rightarrow \epsilon \in P\}$. The proof is again by induction.

$$B \in \hat{\delta}(A, w) \text{ iff } A \rightarrow^* wB$$
$$f \in \hat{\delta}(A, w) \text{ iff } A \rightarrow^* w, w \neq \epsilon$$
$$S \in F \text{ iff } S \rightarrow w \text{ iff } \hat{\delta}(S, \epsilon) \in F,$$
Problem 4. Let \( L = \{(1^i0^j)^j | i, j \geq 0\} \). There are three different ways to create strings in \( L^c \):

1. Begin the string with a 0
2. End the string with a 1
3. Create a string with an unequal number of 1’s followed by 0’s or an unequal number of 0’s followed by 1s

More specifically, we can state (3) as follows: create a string \( 1^{x_i}0^{y_i}1^{x_{i+1}}0^{y_{i+1}} \cdots 1^{x_n}0^{y_n} \) such that:

- \( x_i \geq 1, y_i \geq 1 \) for all \( i \)
- some \( x_i \neq y_i \) or some \( y_i \neq x_{i+1} \)

Using these rules, we can then define the following production rules:

<table>
<thead>
<tr>
<th>Production</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow X0U1X</td>
<td>X1V0X</td>
</tr>
<tr>
<td>( A \rightarrow 0A</td>
<td>0 )</td>
</tr>
<tr>
<td>( B \rightarrow 1B</td>
<td>1 )</td>
</tr>
<tr>
<td>( X \rightarrow 0X</td>
<td>1X</td>
</tr>
</tbody>
</table>
| \( U \rightarrow 1U0 | 1A0 | 1B0 \) | \( 11^n00^m, \) where \( n \neq m \),
| \( V \rightarrow 0V1 | 0A1 | 0B1 \) | \( 00^n11^m, \) with \( n \neq m \) |

Then the language generated by the grammar above is \( L^c \).

**Common Errors:**

- Failing to include some of the shorter strings such as \( 110 \)
- Including a production rule of the form \( M \rightarrow \epsilon \), which ended up deriving strings from \( L \)
- Creating a production rule \( S \rightarrow XUX \), which allows the \( X \) production rules to balance the number of 1’s and 0’s.

**Some recommendations:**

- Be careful using \( \epsilon \) as a terminal—you’re often better off using a real symbol as a terminal
- You may need to treat shorter strings as special cases; try to think of all the possible short strings which your CFG may not appropriately handle