Write a CFG for \((0 + 1)^* - \{101001000...10^n1|n \geq 1\}\)

First note that there are two main reasons that a string might not be in the
language, \(L = \{101001000...10^n1|n \geq 1\}\).

1. It is not of the form \(1(0^*1)^*\), or does not have atleast two blocks.

2. It has two adjacent blocks of ones such that the number of ones in
the second block is not one more than the number of ones in the first
block.

We can write CFGs for both of these. For the first:

\[
X \rightarrow 1X|0X|\epsilon \\
Z \rightarrow 0Z|\epsilon \\
S_1 \rightarrow 0X|X11X|X0|1|1Z1
\]

Notice how we can fail to satisfy condition one: if we start or end with a
zero, or if we have two adjacent ones without any seperating zeros we will
fail. Finally, if we have zero or one block, we will not be caught by any of
the previous expressions, but will still be invalid.

Now for the second condition, considering seeing two adjacent blocks, such
that there are \(i\) zeros in the first block:

\[
S_2 \rightarrow X1G1X|X1L1X \\
G \rightarrow GE00 \quad \text{greater than } i+1 \text{ zeros in the second block} \\
GE \rightarrow 0GE0|GE0|1 \\
L \rightarrow 0LE0 \quad \text{less than } i+1 \text{ zeros in second block} \\
LE \rightarrow 0LE0|0LE|1
\]

What we observe is that a string will fail condition two if, for these two ad-
jacent blocks the first with a zeros and the second with \(b\), \(b \neq i + 1\). There
are only two ways that this can fail, \(b < i + 1\), or \(b > i + 1\). The condition
\(L\) captures the first case, and the condition \(G\) captures the second case.

Finally, the production that ties all of this together:

\[
S \rightarrow S_1|S_2
\]
2. **L as the intersection of two CFL’s:**

S1 generates the language $a^ib^j\epsilon^k$.

S1 -> XY
X -> aXb | epsilon
Y -> cYd | epsilon

S2 generates the language $a^*b^*c^*d^*$.

S2 -> AZD
A -> aA | epsilon
D -> dD | epsilon
Z -> bZc | epsilon

L = S1 intersect S2

**Show that the complement of L is a context-free language:**

The idea here is to generate an error so that there is more of one of the letters than the others. I will show how to create a difference between the number of a’s and b’s. Differences between other letters are similar.

S -> AXY | XBY
X -> aXb | epsilon
Y -> cYd | epsilon
A -> aA | epsilon
B -> bB | epsilon

All strings not of the form $a^*b^*c^*d^*$ are also in the CFL.
Exercise 6.2.1: Design a PDA to accept each of the following languages. You may accept either by final state or by empty stack, whichever is more convenient.

c) The set of all strings of 0’s and 1’s with an equal number of 0’s and 1’s.

We shall accept by empty stack. Symbol $X$ will be used to count the 0’s on the input. Symbol $Y$ will be used to count the 1’s on the input. In state $q$, the start state, where we have seen no 1’s, we add an $X$ to the stack for each 0 seen. On the other hand, if in this state we have seen no 0’s, we add a $N$ to the stack for each 1 seen. The first $X$ or $N$ replaces $Z_0$, the start symbol. When we see a 1, we pop the stack, one $X$ for each input 1. When we see a 0, we pop the stack, one $N$ for each input 0. Formally, the PDA is $((q),\{0,1\},\{X,N,Z_0\},\delta,q,Z_0)$. The rules:

1. $\delta(q,0,Z_0) = \{(q,X)\}$ //These 2 rules initially replace start symbol
2. $\delta(q,1,Z_0) = \{(q,N)\}$
3. $\delta(q,0,\varepsilon) = \{(q,X)\}$ //These 2 rules put an element onto the stack when it’s empty
4. $\delta(q,1,\varepsilon) = \{(q,N)\}$
5. $\delta(q,0,X) = \{(q,XX)\}$ //These 2 rules put on more symbols of the same kind
6. $\delta(q,1,N) = \{(q,NN)\}$
7. $\delta(q,0,N) = \{(q,\varepsilon)\}$ //These 2 rules erase the opposite symbol
8. $\delta(q,1,X) = \{(q,\varepsilon)\}$
Question 4

Write a CFG for the set of all strings of balanced parentheses. 
\( S \) is the start state:

\[
S \rightarrow (S) \mid S(SS)S \mid \epsilon
\]

Question 5

Give a decision procedure for determining if a regular set is finite. 
We can do this in one of two ways:

- We can convert the regular set into a DFA and check to see if a string of length \( n \) to \( 2n \) appears, where \( n \) is the number of states in the DFA; or

- We can convert the regular set into a regular expression and see if there is a closure on a non-empty string (for example, \( 0^* \) is in, \( \epsilon^* \) is not).