Problem 1

Idea:
$1 + 1 = 2 \in L_1$
$11 + 1 = 2 \notin L_1$ because $11 + 1 = 12 \neq 2$

Proof:
Assume L_1 is regular for contradiction. Let p be the pumping length given by the pumping lemma. Let s be a valid equation in the form X + Y = Z, and where X, Y, and Z are non-negative integers represented in base 10, without leading zeros and each have a length of p. Since $|s| = 3p+2 \geq p$ and $s \in L_1$, the pumping lemma guarantees that s can be divided into three pieces, s = xyz. By condition 3, we have $|xy| \leq p$, so y has to be the digit of the first term. By condition 1, for each $i \geq 0$, $xy^iz \in L_1$. Let i = 2, so we have xy^2z = xyyz. Observe that by pumping up y, the first term of the equation will increase in the length of digits while the rest of the string remains the same. So if y is a non-zero string, it will no longer be a valid equation because it does not satisfy the equality due to value changes. If y was a 0, it would still not satisfy all the conditions since it would have leading zeros and that is not part of the language L_1. Therefore we arrive at a contradiction and reject our initial assumption that L_1 is regular.

Problem 2

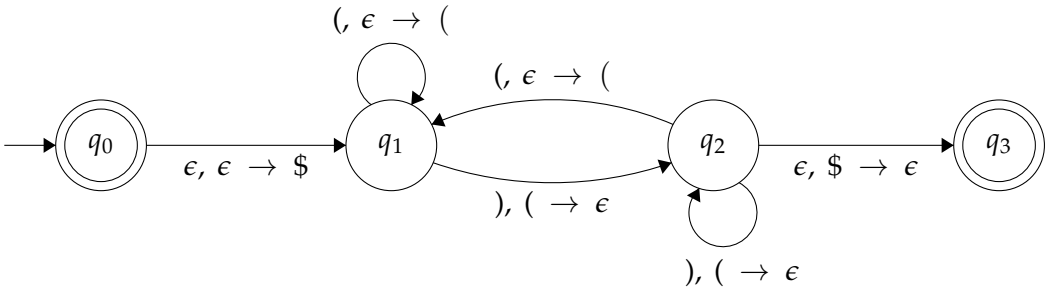
$$S \rightarrow ASBBB \mid \epsilon$$
$$A \rightarrow b \mid BC \mid abD \mid baD$$
$$B \rightarrow a \mid b$$
$$C \rightarrow ASBBB$$
$$D \rightarrow BBB$$

Derivation of abaaaaaa:
$S \rightarrow ASBBB \rightarrow ABBB \rightarrow BCBBB \rightarrow BASBBBBBB \rightarrow BABBBBBB \rightarrow BbBBBBBB \rightarrow abaaaaaa$

This grammar works by first starting at S. S will guarantee a minimum length of 4 because the minimum length of A or B is 1. When the length is 4, A can only be b, making the first quarter, which is the first letter, one b. For strings greater than length 4, you have to either substitute more S, or more A, which will maintain the string length to be a multiple of 4. A string can only end when A becomes b, so there will always be at least one b in the first quarter of the string. Or if the string is longer it can go to D. abD and baD, and D is designed so this cfg will also accept longer strings with different first quarter combinations like babababa. Since it doesn't matter what the last 3 quarters of the string is, B is placed after S, and B can be substituted with either a or b. The reason A goes into C and not S is because we do not want to have epsilon as an option, as that will violate the at least one b in the first quarter condition.

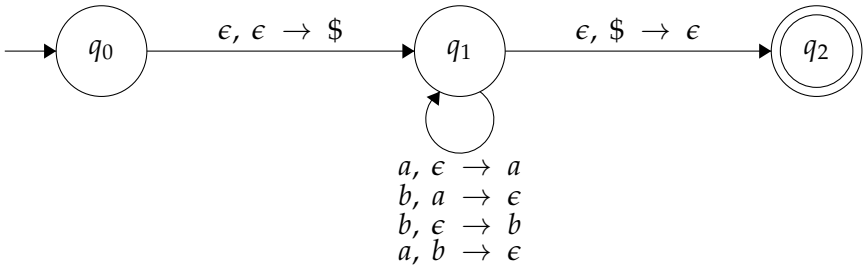
Problem 3

1. Here is the solution for part 1.



   The transition from q_0 to q_1 requires a '(' and pushes a '(' onto the stack because there must always be a '(' before a ')'. It will only move to q_2 if you can pop a '(' from the stacks, and the transitions between q_1 and q_2 allows you to have more brackets, while forcing the number of brackets to be equal. q_3 will accept the strings in the language when you reach the bottom of the stack and can pop \$. And finally q_0 is an accept state because the empty string is accepted.

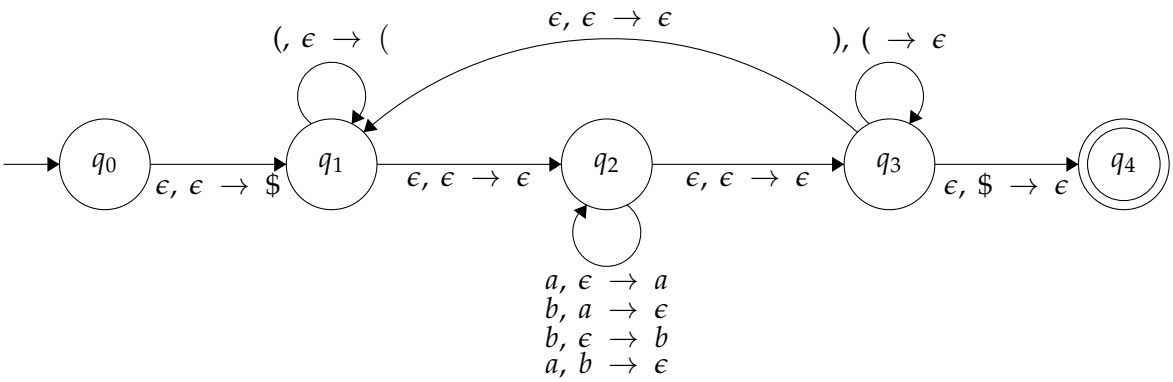
2. Here is the solution for part 2.



   Since you can only move to q_2 from q_1 after popping a \$ of the stack, this means the number of a's and b's must match. In state q_1 the loop allows you to have any number of a's and b's. However you can only have them in pairs, so for each a there is another b, and for each b there is another a. The pushing and popping of a and b off the stack ensures this, and also you can start with either a or b.

Problem 3

3. Here is the solution for part 3.



The ϵ transitions between q_1, q_2, and q_3 has non-determinism so it has ability to have brackets and letters in different orders. At the same time q_1 and q_3 forces the brackets to be properly matched, and q_2 will restrict the number of a's and b's to be the same. q_4 is the accept state and can only be reached after popping the last thing off the stack, the \$. So this accepts the empty string and strings such that the parentheses are correctly matched in the whole string and have equal numbers of as and b's.