1. Describe how you would write a function to find the intersection of two sets. Your solution should NOT use the built-in intersection method.
2. Describe how you would write a function to find the union of two sets. Your solution should NOT use the built-in union method.

Q1:

In the beginning, there would be 2 sets of integers in 2 separate lists. We can use 2 separate loops to iterate through the lists. When iterating the lists, add the integers into a dictionary as a key. Use if-else conditions to check if the dictionary already has that integer as a key, if so, increase the value associated with that key. After iterating both lists, use another loop to iterate the dictionary, and check if the value associated with each key is greater than 1. If the value is greater than 1, this means that the value of that key has appeared more than once from the 2 lists. An intersection means that an integer exists in both lists. Thus, we would want to print that integer, to display the intersection of the 2 lists.

The performance is O(n), given the sample sizes of both lists are the same. Although there are 3 loops, the coefficient is dropped, and this leaves us with O(n).

In general, I would consider these 3 cases. First, 2 lists with only 1 intersected value; Second, 2 lists with more than 1 intersected value; Last, 2 lists without any intersected values. I might also consider 2 identical lists.

Q2:

The overall approach for a union set would be similar to that of an intersection set. But when 2 sample lists are iterated, and added the values into a dictionary. Since a union means that the value exists in either one or both of the lists, we just need a loop to iterate through the dictionary and print all the keys of the dictionary. The values do not matter. The values were used to detect duplicates, thus, if that value exists in both of the lists, then the value associated with that key is 2.

The performance is O(n), given the sample sizes of both lists are the same. Although there are 3 loops, the coefficient is dropped, and this leaves us with O(n).

In general, I would consider these 3 cases. First, 2 lists with no overlapped values; Second, 2 lists with 1 overlapped value; Last, 2 lists with more than 1 overlapped value. I might also consider 2 identical lists.