1. **How did you represent the social network? Why did you choose this representation?**

I represented the social network with a graph. Every node in the graph represents a user in the social network and every edge represents a friendship between two users (the two nodes it's connecting). I chose this representation because there are many different ways to traverse graphs to find out different types of information. I also used a disjoint set structure to represent the components of mutual friends. The disjoint set is represented as a tree. Each node in the disjoint set points to its parent in the tree. The root stores the size of the disjoint set and it is represented as a negative number to indicate it is root. Every time two users become friends, we union the corresponding disjoint set structures. When we do the union, we pick the root of the tree with smaller weight and make that become the child of the root of the larger tree. We also collapse this tree using a path compression algorithm any time we run the "find the root" of the tree. The root of the tree is basically representative of the disjoint set. The reason we use the disjoint sets is we can quickly find out whether two users are friends. If they don't belong to the same disjoint set, there is no relation between them. When there are billions of users in a social network, we will have a lot of disjoint sets and it is important to be able quickly find out whether there are no friendships. Recently, I learned the Disjoint-Set Union algorithm in my CS 170 algorithms class and was able to immediately relate to this use case.

2. **What algorithm did you use to compute the shortest chain of friends? What alternatives did you consider? Why did you choose this algorithm over the alternatives?**

To find the shortest chain of friends, I used breadth first search to find the shortest path between two user nodes in the graph. In order to expedite the algorithm, I used the disjointSet structure to initially check if the two users were connected. If two users aren't connected on the disjointSet structure (if they don't share the same root) we immediately know that the users don't have any mutual friends and therefore, there exists no chain of friends between the two users. When we run the "find root" algorithm for each node, we also do path compression, which effectively collapses the tree. This ensures that the next time we run "find root", it will run even faster. This prevents us from doing unnecessarily long breadth first search traversals only to output that no path exists.

Alternatively, I was considering representing each node in the disjoint set as a cluster of users that are all mutual friends. However, I realized it becomes unnecessarily complicated when two users from different clusters become friends, as you would have to collapse the clusters together. I also

considered using Dijkstra's algorithm to find the shortest path, where each edge weight would represent the number of mutual friends that a user has. However, I realized this would come with a lot of other computations that would only increase the runtime. I decided to use breadth first search because the edges don't have any weights attached to them and this traversal naturally finds the shortestPath between two vertices.

3. **Please enumerate the test cases you considered and explain their relevance.**

I created different types of tests to ensure this Social Network works on different edge cases. I created a test framework that allows us to create tests. It allows you to add users, add friendship, test friendship and also test the shortest path.

The first test (Test1.txt) adds a large number of users (around 550) with multiple disjoint subsets and then tests the shortest path in one of the disjoint subsets. It also connects the subsets and tests the shortest path. The second test (Test2.txt) creates two disjoint sets of users with friendships and then connects the sets with a new friendship and tests the shortest path in the newly connected set. This test also tests bad input and makes sure the test framework and the code handles that appropriately. The bad input prevents you from forming a single set. The third test (Test3.txt) is a variation of the second test where there is no bad input and through a sequence of add friend actions, we create a single set. The fourth test (Test4.txt) creates 5 individual users who have no friends and tests if they are friends with each other as well as the shortest paths between these individual components, which both return false. This test also tests if a user is friends with itself and the shortest path from a user to itself. The fifth test (Test5.txt) creates a "cycle" of friendship to make sure the algorithm still finds the shortest path by walking in both directions.