Technical Question 3:

This question posed quite a challenge for me. For this question, I chose to rely on Kruskal’s Minimum Spanning Tree (MST) algorithm. The algorithm takes any graph object along with the number of vertices and returns the MST.

First, we create an empty graph with the list’s length as the number of vertices (since each item in the list represents a vertex). We then loop through each list item and its associated edges. We check the naming of the list item key. Since our example list is using letter characters, I need to convert these to their lowercase equivalents and then as integers with A as 0 and Z being 26 (we need to do this for our later sort method). So, in checking the key and first edge value for strings and converting if necessary, we finally call the graph.add\_edge, thus constructing the edge from the given list item.

Once the loop is complete, we call the KruskalMST method providing it the graph we just constructed and the number of vertices.

I needed to modify the if else statements in lines 86 – 92 because the original algorithm returned print statements and I needed to output the result in a similar format to the input. While cycling through each edge, we check that the parents of each node are not equal. If they are not equal, we check the result list for the given key. If it exists, we append the v and w values. If it does not exist, we simply create the key.

Finally, we return this result list.

The question3 method (before calling the Kruskal algorithm) has a time efficiency of O(E) since we need to build the initial graph and by doing so will do this E times. Space efficiency is O(E) as well since we need to create an array of size E. In the algorithm, we begin by sorting the graph which takes O(Elog(E)) time and O(E) space. Next, looping through each node to create subsets of V length will take O(V-1) or simply O(V) time and space efficiency.

Next, the find and union method will take O(log(V -1)) which can be simplified to be O(log(V)) time. Space constraint is O(V). However, the creation of the result array adds an addition O(V) space so that ends up being O(2V) space. The final time efficiency is O(E) + O(Elog(E)). We can simplify this down to O(Elog(E)).