HW3 Report

My algorithm involved some important initial setup in order for the entire thing to work properly. Firstly, I check if the graph has none or 1 vertex and in this case, there is no mst so I simply return an empty list. Next, I first provide the final list with an initial mst by running the algorithm once starting at vertex 0. Then, the algorithm iteratively finds all msts by starting at the next vertex, then the next, until the start vertex is the last in the initial graph.

When the initial part of the algorithm calls the recursive element, it initially provides a tree with the min incoming edge of the start vertex and the queue contains all edges of the start vertex and, if the next vertex is not the greatest vertex in the graph, then the incoming edges of the next vertex are also added to the queue. In the case where the queue contains incoming edges from the start vertex and the next vertex, both vertices are considered visited, and the removal of visited edges handles all of the ineligible edge in one of the next steps. But before removing ineligible edges, I first check whether the given tree is too heavy to be an mst and then I check if it is in fact a spanning tree, and since the first check will prevent adding a non-mst, if the given tree spans the whole graph, then add it to the final list. So, with the given queue and set of visited vertices, by checking all vertices in the visited set, all incoming edges are removed that are either already in the given tree, or if that edge would create a cycle within the tree. Then, while the queue still has edges to explore, the smallest edge is polled and added to the tree, which is part of the depth-first search for the given tree.

Considering the smallest edge which was just added to the tree, the source of that edge is added to the set of visited vertices, and all incoming edges of said vertex that would not create a cycle within the given tree are added to the queue. The next step is to recursively find all possible new paths from this current tree if there are multiple edges of the same weight, but you must first make an initial recursive call to solve for the current tree as is.

After all recursive calls return a tree that is an mst, and all vertices of the graph have been visited, the next step is to remove any redundant msts that are equivalent in their set of vertices and edges. So, a simple iterative loop, cross-checks all SpanningTrees of the final list and, if any of the them are the same, one is removed. This is done, by looking at the undirected sequences of each mst, which allows for simple comparisons. The final, fixed list is then returned and it will contain all msts from the graph!