CS577
Homework #5, Problem #2
Sreeram Danda

#2a)

Consider the graph G = (V, E, W) such that there are three edges and vertices. Let w_1 be the weight between vertices 1 and 2. Let w_2 be the weight between vertices 2 and 3. Let w_3 be the weight between vertices 3 and 1. Now consider the spanning tree built using edges 1-2 and 2-3. This spanning tree has a width of max{w_1, w_2}. Now consider another spanning tree: 2-3, 3-1. This spanning tree has a width of max{w_2, w_3}. Assuming that w_2 is the largest weight, then both spanning trees have the same most expensive edge and therefore are both minimum width spanning trees of graph G. Now, if we continue to add vertices and edges, we can note that given any two spanning trees that contain the same edge that is locally the most expensive edge in the spanning tree, they can both be considered minimum width spanning trees.

#2b)

A minimum spanning tree has the property that there are no cycles that exist within the tree. Since we are given *s* spanning trees the only way to disqualify the trees from being spanning trees is adding a cycle. Given the group of trees, connecting one spanning tree to another spanning tree with one edge will still result in a valid spanning tree. This is because no prior path exists between any two vertices in the two spanning trees. Therefore, adding a single edge between any two points within the spanning tree will not add a cycle. Then, we can consider the combined tree as one spanning tree and repeat the procedure for the rest of the spanning trees. The number of required edges would then be the number of trees minus one so, *s* - 1.

#2c)

```
min_width_spanning_tree (G, weight_threshold)
  |   sorted_edges = minHeap(all edges e ∈ G)
  |   tree = ∅
  |   for(all vertices v ∈ G)
  |     |   make-set(v)
  |   for(i = 0...weight_threshold; i++)
  |     |   min_edge = sorted_edges.findMin()
  |     |   while(min_edge <= i)
  |     |     |   tree = tree ∪ (min_edge.u, min_edge.v)
  |     |     |   union(u,v)
  |     |     |   sorted_edges.removeMin()
  |     |     |   min_edge = sorted_edges.findMin()
  |     |     |   if(tree is connected using BST)
  |     |     |     |   return tree;
```

Time Complexity: Building the minimum heap takes O(|E|). Making an empty tree is O(1). Calling make-set on all vertices of G takes O(|V|) time. The next loop is called as many times as there are

distinct edge weights so at worst case |E| times. Calling findMin() takes O(log|V|) time and it is called 'D' number of times where D represents the number of edges with the same weight. The two union calls are run in O(1) time and are called D times so it takes O(D) time to run them all. Similarly to findMin(), removeMin() takes O(D log(|V|). Using BST to check if the tree is connected takes O(|V|+|E|). Therefore the running time of the algorithm can be expressed as:

$$O(|E|) + O(1) + O(|V|) + O(|E|) * O(D \log(|V|) + D + D \log(|V|) + |E|+|V|)$$
$$= O(|V|) + O(|E|* 2 *D \log(|V|) + D + |E|+|V|)$$
$$= O(|E|*D \log(|V|)$$

If *m = O(n)*, the best upper bound would be O(1). This is because each vertex would only have two connections with no cycles. This can be depicted as a line with vertices at different points along the line. The minimum width spanning tree would be the minimum spanning tree which is the initial graph itself.

Program Correctness: The program begins by ordering all edges by weight in a minimum heap. Then the program creates a set for each vertex. This represents the tree structure where the threshold is 0. Each vertex is its own set. Then, the algorithm increments the threshold by one each time. For each iteration of the loop, the algorithm finds the minimum weighted edge in the min heap and checks if it is within the threshold. If so the algorithm correctly adds the vertices to the tree and connects them. Then the algorithm checks if the tree is connected. If it is connected a minimum width spanning tree is made. This is true because the tree cannot be a spanning tree if it is connected and it must be minimum width because if it was possible to make the spanning tree with a lesser maximum width it would have already made that connection beforehand. If the algorithm does not return a tree, then there are vertices which cannot be connected using the given weight threshold.