7.2. So, the way that the insertion sort works is that all elements are simply sorted one by one by checking which numbers are larger than others. If all values are the same, then no elements’ positions actually need to be changed, which means that the sort time needs to be O(n).

7.17a. For merge sort, the average runtime for sorted inputs is O(nlog(n)).

7.33. Consider a tree with a given height of O(log(n)). This tree’s traversal is required for any given sorting algorithm, and thus, any given algorithm takes the height to complete. This means that any comparison-based algorithm takes O(log(n)) time to complete.

9.1. s, G, D, H, A, B, E, I, F, C, t is the final answer for the topological ordering.

9.5b. I just used my eyes for this one. If it’s unweighted, then that means that the weight of all the arrows is equal to one.

B-A = 3

B-C = 1

B-D = 2

B-E = 1

B-F = 2

B-G = 1

9.25. So, there are three different types of edges: back, forward, and cross edges. If the cross edge is labeled (v, w), then the node v most likely has the larger preorder and postorder value than w. If this is true, then it’s a cross edge. Otherwise, if v has the larger preorder value, then the easiest way to check is to keep the numbers on a stack, which will easily tell you whether or not it’s a forward or back edge.

9.5a.

B-A = 6 (B-G-E-D-A)

B-C = 2

B-D = 4 (B-G-E-D)

B-E = 2 (B-G-E)

B-F = 3 (B-G-E-F)

B-G = 1

9.15a.

Prim’s algorithm:

Starting off with one node, Prim’s algorithm continues by adding an unconnected node to the graph and then finds the smallest weight out of the given node edges.

Kruskal’s algorithm starts off by creating a forest of individual node trees. Each tree can be two connected nodes with the shortest path between them. Sort them by weight and then build a tree adding by the least weighted to the most weighted.

TREE REPRESENTATIONS:Letter

Description automatically generatedText, letter

Description automatically generatedA picture containing text, receipt

Description automatically generated

9.39.

Start off by coloring every vertex that’s null in the graph and adding it to a queue. Then we dequeue the starting Node. Then we move on to every node adjacent to the starting node. If the node hasn’t been colored, we color it. If we colored the start node red, we color the next node blue, otherwise we color it red. We continue this until all the nodes are colored. Finally, if the node is already colored then we color the node the same color as the starting node. An extra data structure is used because we need a linked list to represent the adjacency matrix.