For this project I made a Kruskal’s sorting algorithm. I chose this algorithm because it breaks it down into subproblems until it gets a complete minimum spanning tree. Whereas prims operate like trying to draw a straight line until you get the best answer. Kruskal’s is still greedy however it creates smaller greedy problems.

Step 0: Edge and set objects.

This program uses two objects in its core. The first is the edge object which has 3 ints, the integer representing the source of the edge, the integer representing the destination of the edge, and the integer representing the weight/degree of the edge. It also contained a compareTo method which compares the weight of the edge. This is important because the edge class implements javas Comparable interface.

The other object is the *set* object which is to represent the path of a subset until it is used in the primary MST. This is done by making an array of them with their parent representing the vertex of origin and the rank representing order.

Step 1: Sorting a lists of edges

The complete list of edges is the Edgelist array. It is created and filled with edge objects when a new graph is created. Then the source,destination, and weight add to all those edges. After that the Arrays.sort() method is used in conjunction with the compareTo method of the edges. The Arrays.sort() method is a built-in merge sort that is a part of the Comparable interface.

Step 2: Adding to the MST in order of weight and checking for loops

A set called subsets is created and filled with basic entries which will be modified later in order to avoid creating a loop. Then until the MST spans all vertices the program will do the following process. It will get the next edge in the sorted list and increment the index number. Then it will get the roots of the destination and source for this edge. If the roots are not the same then that edge is added to the MST, if they are the same it's ignored.

This is continued until the index of the MST equals the amount of vertices which assumes that means that the MST contains an edge to connect all vertices.

My approximation of the time complexity should be O(E log E/V). It is either or because it is based on the larger amount. This is due to the central loop having to check both all of the vertices and all the edges doing for each edge. Usually it will be O(E log E) because usually most graphs will have more edges than vertices.In my 100 vertices test I found that it matches what described with a little margin for error because I’m casting the nanotime into integers.