Project 2

Prathik Ranganatha Gowda

1 Problem Statement

We are required to analyze Kruskal's algorithm to find the minimum spanning tree.

2 Theoretical Analysis

The time complexity for Kruskal's algorithm is O(mlogn).
Where,
n = number of vertices
m = number of edges (which I have taken to be (n*(n-1))/2)
Analysis:
The algorithm can be divided into 3 parts:

1. Sort edges by weight:
   This step will need O(mlogm) time to sort the edges by weight. As m cannot be greater than n^2 as a result log(m) is bounded by 2(logn). Therefore, this step takes O(mlogn) time.
2. Find operation:
   We perform two find operations inside the while loop to determine whether adding the edge would result in a cycle. We perform this operation only once per edge, which would take O(mlogn) times.
3. Union operation:
   When we apply the union operation to the edge addition, it takes O(nlogn) time to complete this step because we add n-1 edges.
4. Therefore, the time complexity of Kruskal's algorithm to find the minimum spanning tree is O(mlogn + nlogn). This can be expressed as **O(mlogn)** time because m is greater than n.

3 Experimental Analysis

3.1 Program Listing

```
1 usage
def MinimumSpanningTree(self): #this methos finds the minimum spanning tree for the graph
    start_time = time.time_ns() #starts the timer
    result = [] #to store the edges of the minimum spanning tree
    parent = [] #empty list to store the parent of each node in the graph
    rank = [] #empty list to store the rank of each node in the graph
    i, e = 0, 0 #i variable is used to iterate over the edges of the graph, e variable used to keep track of number of edges in the minimum spanning tree
    self.edges = sorted(self.edges, key=lambda item: item[2])  #sorts the edges in ascending order of their weights
    for node in range(self.n):
        parent.append(node) #initialize the parent array
        rank.append(0) #initialize the rank array
    while e < self.n - 1: #iterates over the edges in sorted order until the edges in the minimum spanning tree is one less than the number of nodes given as input
        source, destination, weight = self.edges[i]
        i += 1
        x = self.find(parent, source) #finds the root of the tree containing the source node
        y = self.find(parent, destination) #finds the root of the tree containing the destination node
        # checks if the source and destination are in the same tree, if they are not then merges the trees together and adds the edge to the minimum spanning tree
        if x != y:
            e = e + 1
            result.append([source, destination, weight]) #adding the edge to the minimum spanning tree
            self.union(parent, rank, x, y) #merges the trees together
    stop_time = time.time_ns() #stops the timer
```

The above code snippet represents Kruskal's algorithm for finding the minimum spanning tree for a graph with n vertices and m edges which is split into 3 parts. Firstly, which includes sorting of edges in ascending order, Secondly, inside the while loop two find operations to check whether adding the edge will result in a cycle. Lastly, the union operation merges the trees and adds the edge to the minimum spanning tree.

The input n values used are: 50, 100, 150, 200, 300, 500, 750, 1000.

The corresponding m values are: 1225, 4950, 11175, 19900, 44850, 124750, 280875, 499500.

3.2 Data Normalization Notes

1. The theoretical values have been normalized by multiplying them with a constant: 25.11296658.
2. We can arrive at this constant by using the formula,
   Normalization constant = Avg. of experimental values/Avg. of theoretical values.
3. The values have been changed to log scale to base 10 to get straight lines.

3.3 Output Numerical Data

n	m	Experimental value in ns	Adjusted experimental value (with log)	Theoretical Value	Scaling Constant	Theoretical Value * Scaling constant	Adjusted theoretical value (with log)
50	1225	984200	5.993083361	6913.723832		173624.1155	5.239610046
100	4950	1988100	6.298438225	32887.08814		825892.3454	5.916923441
150	11175	3224800	6.508502785	80782.04887		2028676.893	6.307212883
200	19900	3989800	6.600951126	152112.7382		3820002.11	6.582063603
300	44850	7978800	6.901937579	369062.5183		9268254.687	6.966997959
500	124750	34457600	7.537285025	1118481.59		28088390.78	7.448526859
750	280875	62859900	7.798373686	2682566.003		67367190.39	7.828448435
1000	499500	121098900	8.083140198	4977909.25		125010068.6	8.096944994
		236582100		9420714.96	25.11296658		

3.4 Graph

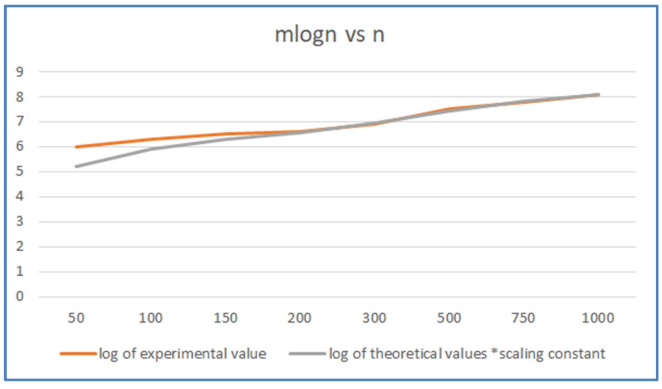

Fig: Graph for mlogn vs n values.

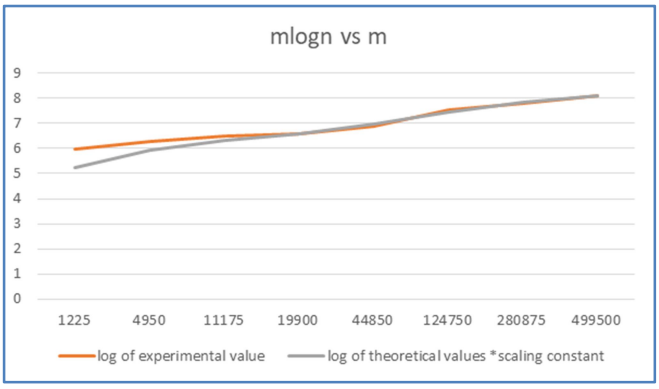

Fig: Graph for mlogn vs m values.

3.5 Graph Observations

In the line graphs above we can observe almost similar graph lines of the adjusted experimental value and theoretical value converging. This helps to confirm my asymptotic analysis that Kruskal's algorithm takes O(mlogn) time.

4 Conclusions

After comparing the experimental and theoretical results of the graph we will be able to conclude that the time complexity of the algorithm is O(mlogn).