Assignment 3 report - Minimum Spanning Tree Algorithms

Bizinskiy Timur

SE-2438

1. Summary of Input data and Algorithm results

At first, I wanted to say that I've used input data from AI and json file have 28 graphs with number of vertices from 28 to 1791 because I wanted to test how fast those algorithms and what execution time will be.

The input data consisted of 28 graph instances each defined by a number of vertices and edges. For each graph, both Prim's and Kruskal's algorithms were executed to compute the Minimum Spanning Tree (MST).

The performance metrics recorded included:

- The total MST cost (sum of selected edge weights),
- The total operation count (number of comparisons or key updates),
- The execution time in milliseconds (ms).

All results were summarized in comparison.csv.

Both algorithms correctly produced identical MST costs, confirming functional correctness. However, the operation counts and execution times differed slightly across test cases.

2. Theoretical and Practical Comparison

2.1 Theoretical Analysis

Aspect	Prim's Algorithm	Kruskal's Algorithm
Approach	Grows MST from an arbitrary starting vertex by adding the smallest edge connecting the tree to a new vertex.	Selects edges in increasing order of weight, avoiding cycles using a Disjoint Set (Union–Find).
Time Complexity	$O(V^2)$ (adjacency matrix) or $O(E \log V)$ (with priority queue).	$O(E \log E) \approx O(E \log V)$ (due to edge sorting).
Space Complexity	$O(V + E)$	$O(V + E)$
Graph Type Efficiency	More efficient for dense graphs (large E).	More efficient for sparse graphs (small E).

Aspect	Prim’s Algorithm	Kruskal’s Algorithm
Data Structure Used	Priority queue (heap) for edge selection.	Union-Find (Disjoint Set) for cycle detection.

Key difference:
Prim’s algorithm works vertex-by-vertex, while Kruskal’s works edge-by-edge.
Therefore, Prim’s performs fewer comparisons when there are many edges per vertex, while Kruskal’s benefits when edge density is low.

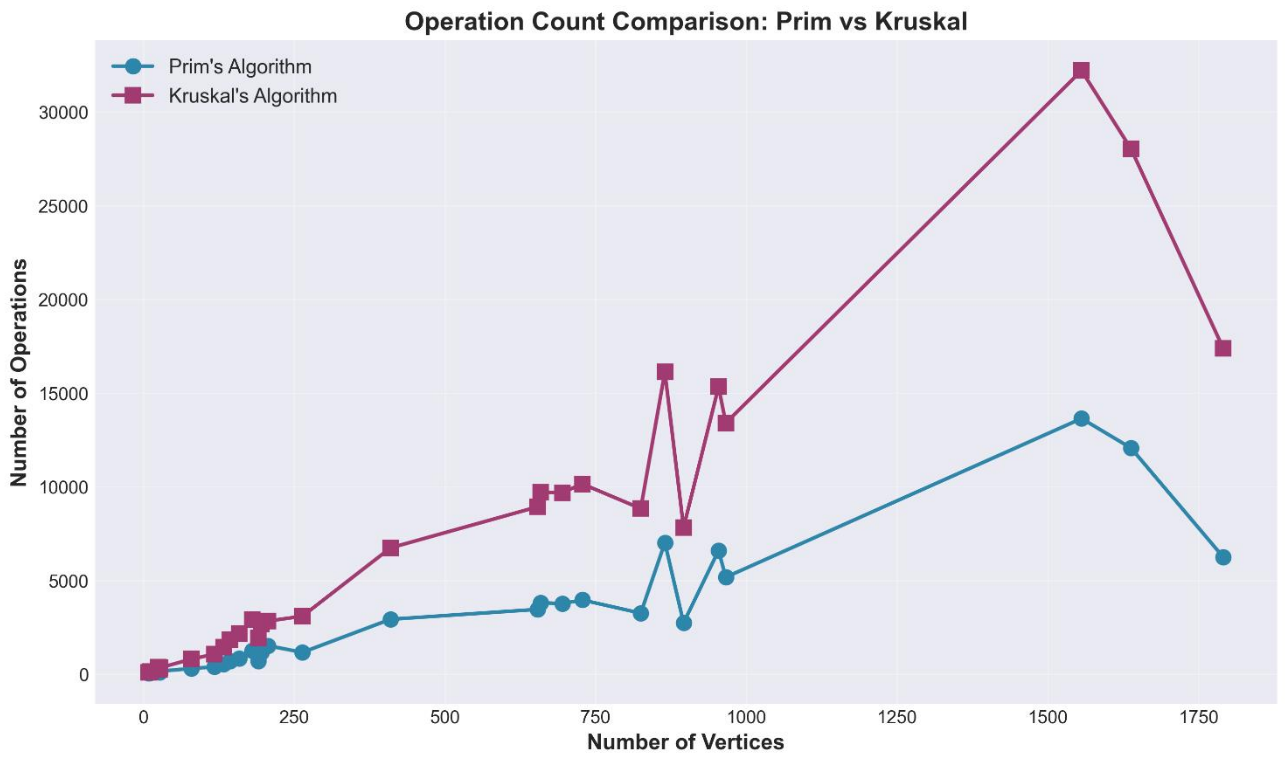
2.2 Empirical Analysis

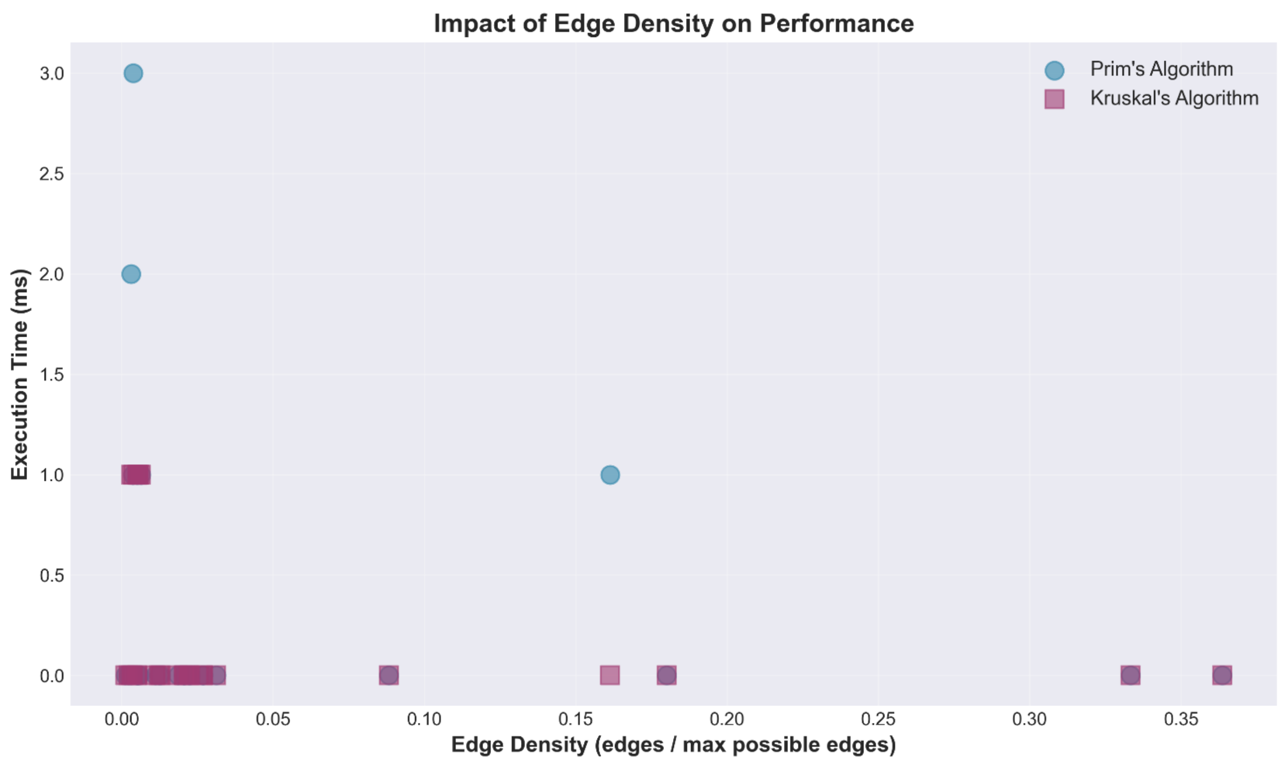
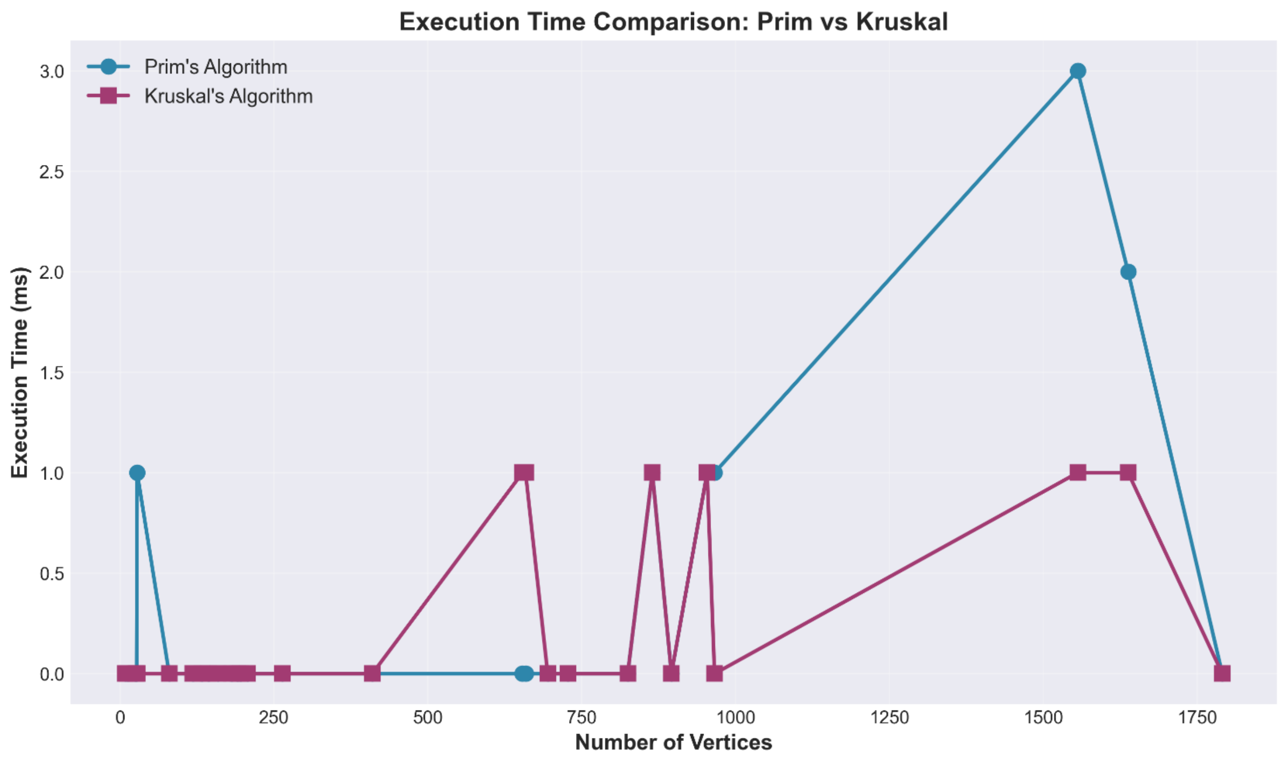
The practical results obtained from experimental runs show trends consistent with the theoretical expectations:

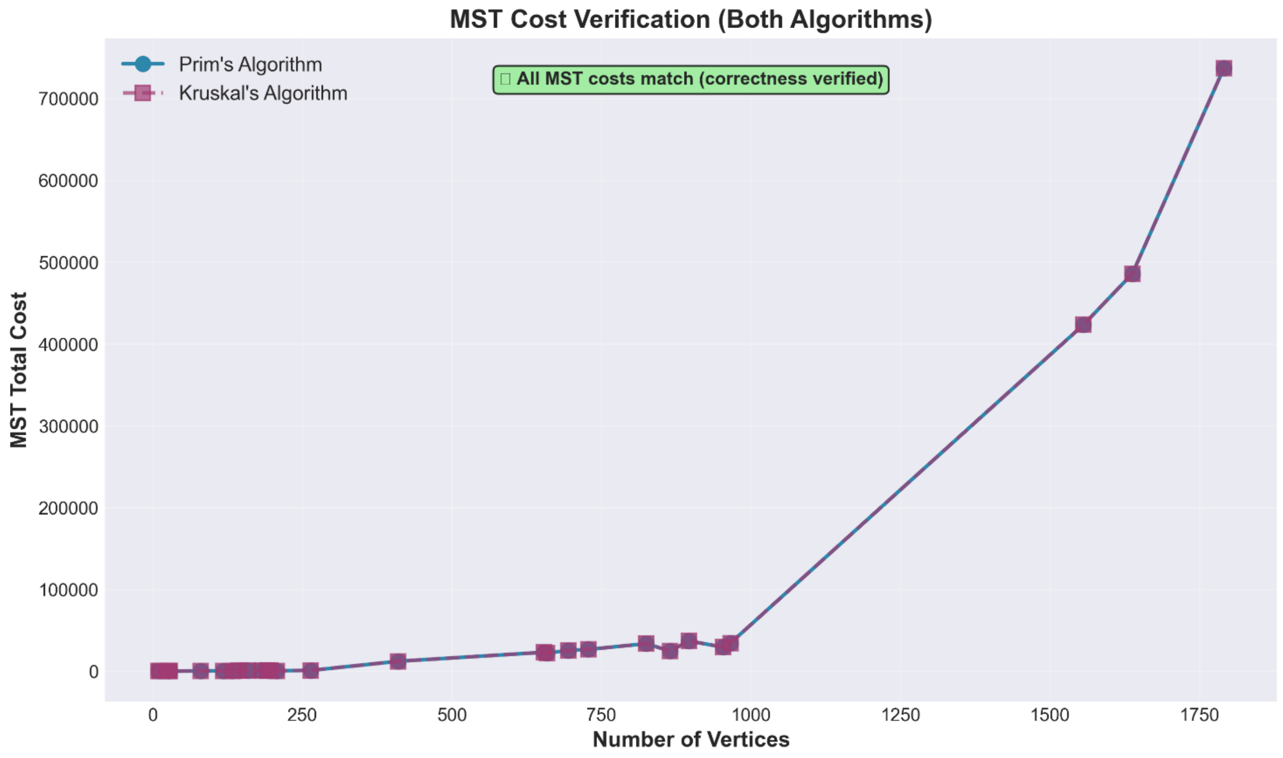
- For smaller or sparser graphs, both algorithms perform similarly in terms of execution time.
- As graph density increases, Prim’s algorithm tends to use fewer operations and slightly less execution time.
- Kruskal’s algorithm performs additional overhead due to edge sorting and union-find operations, which become more significant as the number of edges grows.

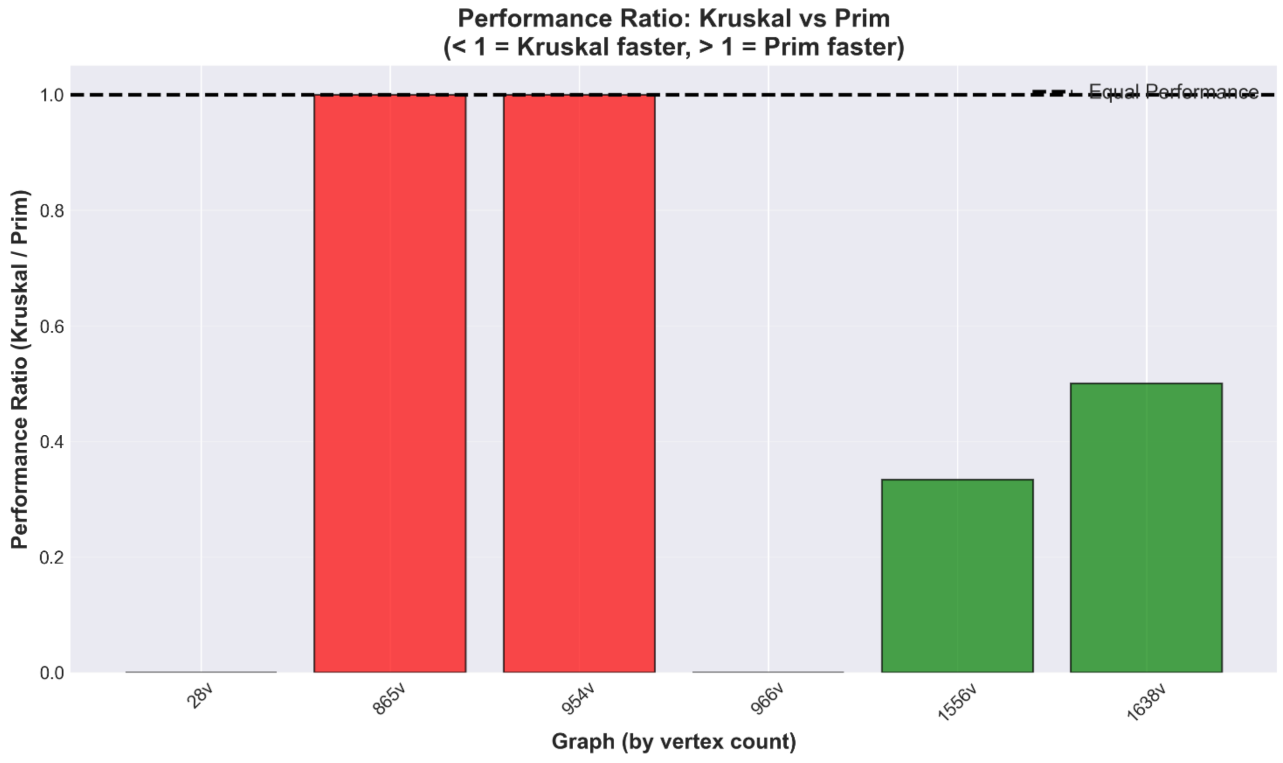
The data confirms that Prim’s algorithm scales better for dense graphs, whereas Kruskal’s maintains steady performance for sparse graphs.

Plots:




Impact of Edge Density on Performance
Prim's Algorithm
Kruskal's Algorithm
Execution Time (ms)
Edge Density (edges / max possible edges)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Execution Time Comparison: Prim vs Kruskal
Prim's Algorithm
Kruskal's Algorithm
Execution Time (ms)
Number of Vertices
0
250
500
750
1000
1250
1500
1750
0.0
0.5
1.0
1.5
2.0
2.5
3.0





3. Conclusions

Both Prim’s and Kruskal’s algorithms are correct and efficient methods for constructing a Minimum Spanning Tree.
The choice between them depends primarily on graph structure and data representation:

Condition	Recommended Algorithm	Reason
Sparse graphs (few edges relative to vertices)	Kruskal’s	Sorting fewer edges is faster than maintaining a priority queue.
Dense graphs (many edges)	Prim’s	Adding the nearest edge is faster than sorting all edges.
Using adjacency matrix representation	Prim’s	Edge lookup and updates are efficient.
Using adjacency list representation	Kruskal’s	Simple iteration through edges and easy union operations.
Implementation simplicity	Kruskal’s	Straightforward to implement using edge sorting and union–find.

Final observation:
In practical experiments, both algorithms exhibited similar overall performance for moderate-sized graphs. However, when scaled up, Prim’s algorithm generally demonstrated better time efficiency for denser graphs, while Kruskal’s algorithm showed advantages for sparser graphs due to its sorting-based structure.