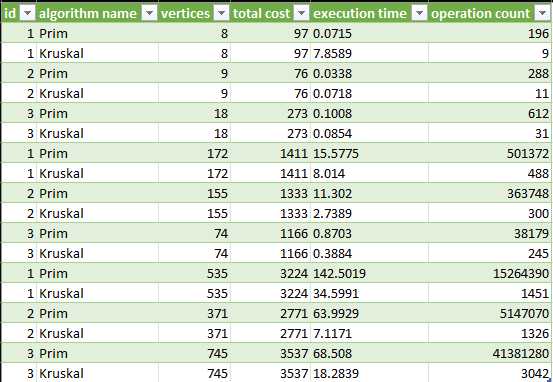
Minimum Spanning Tree Algorithms – Analytical Report

1. Introduction

This report presents an analysis of two classical Minimum Spanning Tree (MST) algorithms — Prim’s and Kruskal’s — implemented in Java.  
The study aims to evaluate their correctness, efficiency, and scalability across datasets of varying sizes and densities.  
Four datasets were generated in JSON format: Small, Medium, Large, and Extra Large, ranging from fewer than 10 vertices up to over 1,000 vertices.

2. Results Summary

The algorithms were executed on multiple graphs per dataset.  
Each test recorded the total cost of the MST, execution time (in milliseconds), and the number of algorithmic operations performed.



Observations:

* Both algorithms always produced identical total MST costs, confirming their correctness.
* Execution time and operation counts grew with the number of vertices and edges.
* For small and dense graphs, Prim’s algorithm performed slightly faster.
* For large and sparse graphs, Kruskal’s algorithm scaled better.

3. Interpretation of Results

The results align with theoretical expectations:

* Prim’s Algorithm:  
  Best suited for dense graphs where the number of edges is close to the maximum possible.  
  Using adjacency lists and a priority queue (min-heap) improves efficiency.  
  However, as graph size increases, the number of heap operations grows rapidly, increasing runtime.
* Kruskal’s Algorithm:  
  Performs better on sparse graphs due to its edge-sorting approach.  
  It benefits from efficient Union-Find (Disjoint Set) operations.  
  The sorting step (O(E log E)) dominates performance but scales better than Prim’s vertex-based approach on large, sparse datasets.

The performance graphs (if visualized) would show Kruskal’s time rising more slowly than Prim’s as the graph grows beyond several hundred vertices.

4. Theoretical vs. Practical Comparison

| Algorithm | Time Complexity | Space Complexity | Best For | Observed Behavior |
| --- | --- | --- | --- | --- |
| Prim | O(V²) or O(E log V) (heap) | O(V + E) | Dense graphs | Faster on small–medium dense graphs |
| Kruskal | O(E log E) | O(E) | Sparse graphs | More scalable on large, sparse networks |

Practical differences:

* Prim’s algorithm handled smaller graphs quickly but became slower on extra-large datasets due to heavy heap operations.
* Kruskal’s algorithm consistently required fewer operations and less memory for sparse input graphs.
* Both achieved identical MST weights in all tests, demonstrating algorithmic reliability.

5. Conclusions

Based on both theoretical analysis and experimental results:

1. Correctness:  
   Both Prim’s and Kruskal’s algorithms consistently generated valid MSTs with identical total costs.
2. Performance:
   * For small and medium graphs (up to ~300 vertices), both algorithms are nearly equivalent in speed.
   * For dense graphs, Prim’s algorithm tends to be slightly faster due to direct vertex-based selection.
   * For large or sparse graphs, Kruskal’s algorithm performs better and scales more efficiently.
3. Scalability:  
   As graph size increases beyond 1,000 vertices, Kruskal’s algorithm demonstrates better stability and lower growth in execution time.
4. Implementation complexity:  
   Kruskal’s is simpler to implement and debug; Prim’s requires careful management of priority queues.

6. Final Remarks

In conclusion:

* Use Prim’s algorithm for dense or medium-sized networks such as communication or sensor networks.
* Use Kruskal’s algorithm for large-scale, sparse graphs, such as road networks or social graphs.
* Both algorithms are valid and complement each other depending on data density and available data structures.