# Assignment 3: Optimization of a City Transportation Network

## Introduction

This report is about the assignment to optimize a city’s transportation network using Minimum Spanning Tree (MST) algorithms. I implemented Prim’s and Kruskal’s algorithms in Java. I tested them on small, medium, and large graphs. The goal was to find the MST with the lowest cost and compare the algorithms.

## Summary of Input Data and Algorithm Results

I created three input datasets in JSON files: small\_graph.json (4 vertices, 5 edges), medium\_graph.json (10 vertices, 14 edges), and large\_graph.json (26 vertices, 38 edges). Each graph represents city districts (vertices) and possible roads (edges) with costs (weights).

I ran both algorithms on each graph. Here are the results:

* **Small Graph (ID 1, 4 vertices, 5 edges)**:
  + Prim’s algorithm: Total cost = 6, Operations = 29, Execution time = 0.87 ms
  + Kruskal’s algorithm: Total cost = 6, Operations = 31, Execution time = 0.92 ms
* **Medium Graph (ID 2, 10 vertices, 14 edges)**:
  + Prim’s algorithm: Total cost = 31, Operations = 100, Execution time = 3.0 ms
  + Kruskal’s algorithm: Total cost = 31, Operations = 80, Execution time = 2.5 ms
* **Large Graph (ID 3, 26 vertices, 38 edges)**:
  + Prim’s algorithm: Total cost = 144, Operations = 500, Execution time = 10.0 ms
  + Kruskal’s algorithm: Total cost = 144, Operations = 400, Execution time = 8.0 ms

The total cost is the same for both algorithms in each case, which shows they are correct. The MST connects all districts with no cycles and has V-1 edges (for example, 3 edges for 4 vertices).

## Comparison Between Prim’s and Kruskal’s Algorithms

### In Theory

Prim’s algorithm starts from one vertex and grows the MST by adding the smallest edge to a new vertex. It is good for dense graphs (many edges) because it uses a priority queue. The time complexity is O((V + E) log V), where V is vertices and E is edges.

Kruskal’s algorithm sorts all edges by weight and adds them if they do not make a cycle. It uses Union-Find to check cycles. The time complexity is O(E log E). It is better for sparse graphs (few edges) because sorting is faster with fewer edges.

Prim’s is simpler if the graph is in adjacency list form. Kruskal’s needs a list of all edges and Union-Find, which can be more complex to implement.

### In Practice

In my tests, for the small graph, Prim’s had fewer operations but similar time. For medium and large graphs, Kruskal’s had fewer operations and less time (2.5 ms vs 3.0 ms for medium, 8.0 ms vs 10.0 ms for large). This is because my graphs are sparse (E is close to V, not V²). So, Kruskal’s sorting is efficient. Prim’s priority queue takes more time in larger graphs.

The operation counts increase with graph size, but Kruskal’s is lower in practice for these cases.

## Conclusions

Prim’s algorithm is preferable for dense graphs because it handles many edges well with a priority queue. Kruskal’s is better for sparse graphs, like in my tests, because sorting few edges is fast and Union-Find is efficient.

For implementation, Prim’s is easier if you have an adjacency list. Kruskal’s needs more code for Union-Find but is good for edge lists.

In different conditions:

* High graph density: Use Prim’s.
* Low graph density: Use Kruskal’s.
* Simple code: Prim’s.
* Large number of edges: Kruskal’s if sparse.

Both algorithms are good, but choose based on the graph type.