Analytical Report

1. Summary of Input Data and Algorithm Results

This section summarizes the performance metrics (execution time and operation count) for both Prim's and Kruskal's algorithms across the generated datasets.

Table 1: Comparative Results of MST Algorithms

Graph ID	V (Vertices)	E (Edges)	MST Total Cost	Algorithm	Execution Time (ms)	Operation Count
small_dense_4V	4	5	0.6000	Prim's	0	18
			0.6000	Kruskal's	0	17
medium_sparse_15V	15	25	4.5300	Prim's	0	88
			4.5300	Kruskal's	0	101
large_dense_30V	30	60	9.0400	Prim's	0	200
			9.0400	Kruskal's	0	194

Key Findings:

- **Correctness:** The Total MST Cost is identical for both Prim's and Kruskal's algorithms across all three graphs, confirming the correctness of the implementations.

- **Time:** All execution times are recorded as 0 ms, which is typical for such small datasets when running on modern hardware. This makes the **Operation Count** the crucial metric for analyzing efficiency.

---

2. Comparison of Prim's and Kruskal's Algorithms

A. Theoretical Efficiency

The efficiency of MST algorithms depends on the relationship between the number of vertices (V) and the number of edges (E).

Algorithm	Primary Data Structure	Asymptotic Complexity	Optimal For
Prim's Algorithm	Indexed Min-Priority Queue	$O(E \log V)$	**Dense** graphs (where E is close to V^2)
Kruskal's Algorithm	Sorting and Union-Find	$O(E \log E)$	**Sparse** graphs (where E is close to V)

- For **sparse** graphs, $E \approx V$, so $O(E \log E)$ is similar to $O(E \log V)$.
- For **dense** graphs, $E \approx V^2$. Since $\log V$ is much smaller than $\log E$ (which is close to $2 \log V$), Prim's algorithm becomes significantly faster than Kruskal's.

B. Performance in Practice (Operation Count Analysis)

We analyze the Operation Count to see how the implementations behave:

- **Small/Medium (Sparse-like) Graphs:**
  - For **medium_sparse_15V** ($V=15, E=25$), Kruskal's algorithm performed **101** operations while Prim's performed **88** operations.
  - **Observation:** The instrumented Prim's implementation was slightly more efficient here. This suggests that the overhead of Kruskal's initial edge sorting and Union-Find operations outweighs the simplicity of its main loop for this specific small, sparse graph.
- **Large (Dense-like) Graph:**
  - For **large_dense_30V** ($V=30, E=60$), Prim's algorithm performed **200** operations, while Kruskal's performed **194** operations.
  - **Observation:** Kruskal's implementation again showed a slightly lower operation count, which contradicts the theoretical prediction for a *truly* dense graph. However, for $V=30$, a truly dense graph would have $E \approx 435$. Since $E=60$ is still relatively small, the graph behaves more like a

sparse graph, allowing Kruskal to remain highly competitive due to the efficiency of modern sorting and the fast $\alpha(V)$ complexity of the Union-Find structure.

---

3. Conclusions on Preferable Algorithm

The choice between Prim's and Kruskal's algorithms depends on the graph structure and implementation complexity.

Condition	**Preferred Algorithm**	**Reason**
Graph Density (Theoretical)	**Prim's**	**$O(E \log V)$** is faster when E is very large (dense networks), such as a fully interconnected metropolitan area.
Graph Density (Theoretical)	**Kruskal's**	**$O(E \log E)$** is optimal when E is small (sparse networks), such as a small regional network with few roads.
Implementation Complexity	**Kruskal's**	Requires sorting and a Disjoint Set (Union-Find) structure. The core loop logic is simple.
Implementation Complexity	**Prim's**	Requires a more complex Indexed Min-Priority Queue (like the IndexMinPQ used), which is harder to implement correctly.

Final Recommendation:

For large, truly dense graphs (where the number of edges is high), Prim's algorithm is theoretically preferred for better performance. For all other cases (sparse or moderately sized graphs), Kruskal's algorithm is often favored due to its cleaner separation of concerns (sorting edges first, then processing) and slightly simpler implementation logic than managing an indexed priority queue.

---

4. References

1. Assignment 3: Optimization of a City Transportation Network (Minimum Spanning Tree).