**Kulmakhanbet Balaussa se-2430**

**Assignment 3 – Minimum Spanning Tree (MST) Algorithms Report**

**1. Summary of Input Data and Results**

This assignment evaluates two Minimum Spanning Tree algorithms — **Prim’s** and **Kruskal’s** —  
using multiple graph datasets of different sizes and densities.  
The graphs were read from assign\_3\_input.json, and results were automatically generated  
into output.json and results.csv.

| **Graph ID** | **Vertices** | **Edges** | **Prim Cost** | **Kruskal Cost** | **Prim Ops** | **Kruskal Ops** | **Prim Time (ms)** | **Kruskal Time (ms)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 7 | 16 | 16 | 28 | 5 | 0.089 | 3.644 |
| 2 | 10 | 13 | 32 | 32 | 117 | 9 | 0.086 | 0.047 |
| 3 | 20 | 22 | 96 | 96 | 418 | 21 | 0.191 | 0.072 |
| 4 | 6 | 8 | 17 | 17 | 40 | 5 | 0.029 | 0.022 |
| 5 | 13 | 15 | 46 | 46 | 180 | 13 | 0.123 | 0.048 |

✅ *For all graphs, both algorithms produced identical total MST costs,  
confirming algorithmic correctness.*

**2. Theoretical and Practical Comparison**

**Prim’s Algorithm**

* **Approach:** Greedy, grows MST by repeatedly adding the smallest edge connected to the tree.
* **Complexity:**
  + *Adjacency matrix:* O(V²)
  + *Adjacency list with heap:* O(E log V)
* **Advantages:**
  + Very efficient on **dense graphs**, where the number of edges E ≈ V².
  + Works naturally with adjacency matrices.
* **Disadvantages:**
  + Needs priority queue or heap for efficiency on sparse graphs.
  + Slightly more complex to implement with custom structures.

**Kruskal’s Algorithm**

* **Approach:** Sorts all edges by weight and repeatedly adds the smallest edge that doesn’t create a cycle (using Union–Find).
* **Complexity:** O(E log E) or O(E log V)
* **Advantages:**
  + Performs better on **sparse graphs** (few edges).
  + Conceptually simpler and uses disjoint set structure.
* **Disadvantages:**
  + Sorting dominates runtime; for dense graphs, it can be slower than Prim.

**Experimental Findings**

From the results:

* Both algorithms yielded identical MST costs — validating correctness.
* **Prim’s algorithm** had more internal operations but still executed quickly due to local edge selection.
* **Kruskal’s algorithm** showed faster runtime on most sparse graphs (Graphs 2, 3, 5),  
  confirming its efficiency for such cases.
* On very small or dense graphs, both performed nearly identically.

**3. Conclusions**

* **Correctness:** Both algorithms consistently produce the same MST total cost.
* **Performance:**
  + **Kruskal** is preferable for **sparse** graphs (low edge density).
  + **Prim** performs better for **dense** graphs or when using adjacency matrices.
* **Implementation complexity:**
  + Prim requires priority queue and adjacency structures.
  + Kruskal requires edge sorting and Union–Find but is simpler to understand.
* **Overall:**  
  Both algorithms are efficient; selection depends on graph structure and data representation.

**4. References**

* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009).  
  *Introduction to Algorithms (3rd ed.).* MIT Press.
* GeeksforGeeks – “Difference between Prim’s and Kruskal’s Algorithm.”
* University Lecture Notes on Minimum Spanning Trees (DAA Course).



