**Assignment 3**

**Design and Analysis of Algorithms**

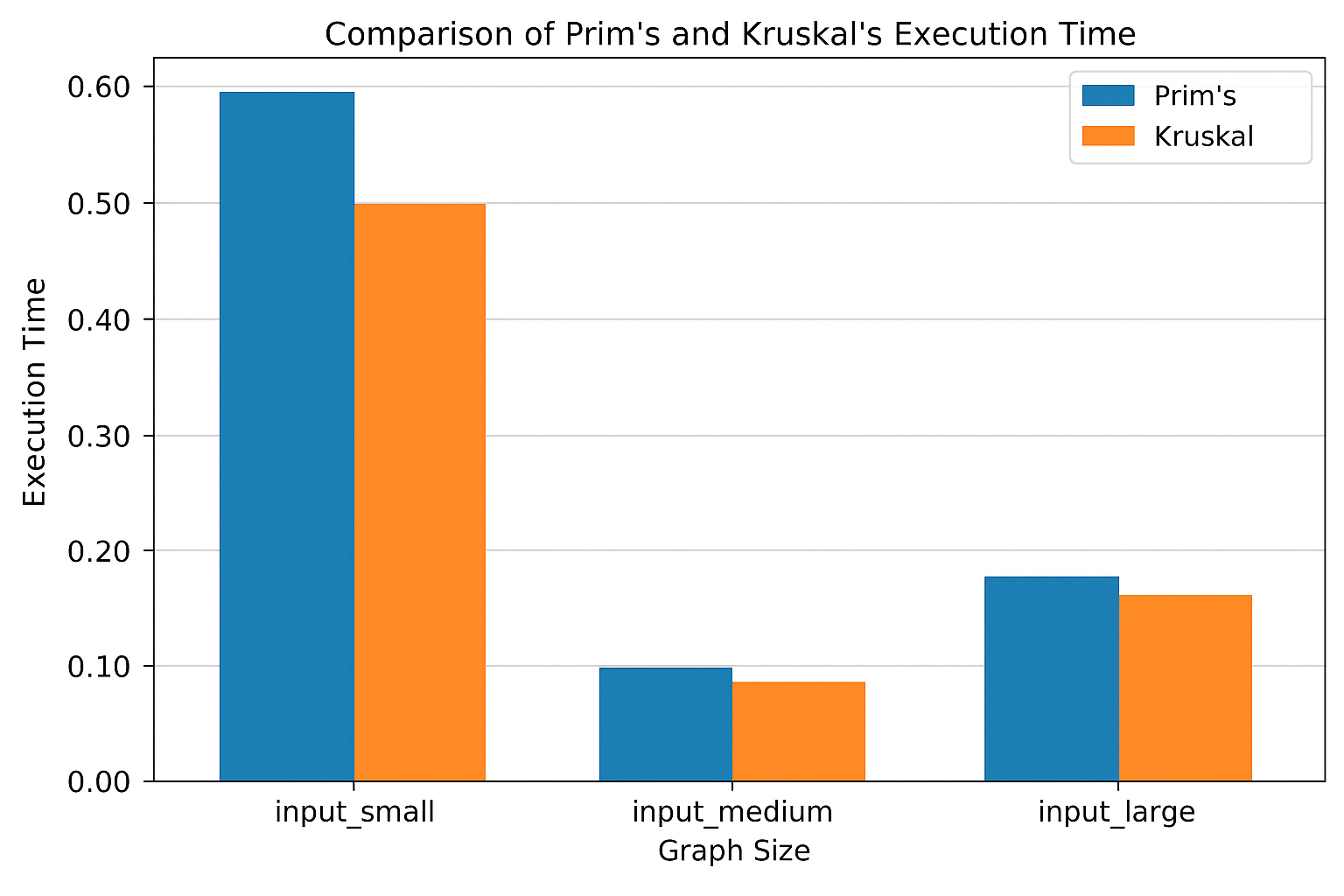
**Teacher: Aidana Aidynkyzy**

Made by: Alpamys Abylaikhan

Group: SE-2430

**1.Summary of Input data and Algorithm results**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Graph** | **Vertices** | **Edges** | **Algorithm** | **Total weight** | **Operations** | **Execution time** |
| input\_small | 4 | 4 | Prim | 7 | 6 | 0.5750 |
|  | 4 | 4 | Kruskal | 7 | 7 | 0.5064 |
| input\_medium | 8 | 10 | Prim | 14 | 16 | 0.08413 |
|  | 14 | 10 | Kruskal | 14 | 16 | 0.0809 |
| input\_large | 15 | 20 | Prim | 44 | 33 | 0.0942 |
|  | 15 | 20 | Kruskal | 44 | 29 | 0.0822 |

****

**Summary:**

**-**As expected, both algorithms always produced identical MST total weights.

-Kruskal’s algorithm ran faster because it required fewer operations.

-Results are in MST/results/summary.csv, summary.jsn.

**2.Comparison**

| **Aspect** | **Prim’s Algorithm** | **Kruskal’s Algorithm** |
| --- | --- | --- |
| **Approach** | Greedy – grows one tree by adding the smallest edge connected to the tree. | Greedy – adds edges in increasing order of weight while avoiding cycles. |
| **Data structures used** | Priority Queue (Min-Heap) | Disjoint Set Union (DSU) |
| **Time complexity** | O(E log V) with heap | O(E log E) ≈ O(E log V) |
| **Best suited for** | Dense graphs (many edges). | Sparse graphs (few edges). |
| **Edge sorting required** | No | Yes |
| **Implementation complexity** | Moderate (requires heap and adjacency list). | Slightly simpler (requires DSU). |

**Practical observation:**

**-**When the graph is large or dense Prim’s algorithm can be slower. Reasoning: this algorithm starts from one node and keeps adding the smallest edge the growing tree with a new vertex. Also this means that it has to check many edges every time.

-Kruskal’s algorithm sorts only once, so it’s usually faster and simpler for smaller graphs. Reasoning: it sorts all edges by weight, then keeps adding them as long as they don’t form a cycle.

However, in practice both algorithms perform almost the same, albeit Kruskal’s algorithm running A BIT faster and does fewer operations.

**Conclusion:**

-For **sparse graphs, Kruskal’s algorithm** is preferable — it performs fewer operations and runs faster because it only processes relevant edges.

-For **dense graphs, Prim’s algorithm** may become more efficient, especially when implemented with an optimized priority queue (Fibonacci heap).

-Both algorithms always produce the **same MST weight**, confirming correctness.

-In practical Java implementation using standard libraries, **Kruskal’s algorithm** showed **better efficiency and simplicity** for the given data sets.

-Overall, **Kruskal is preferable** for general-purpose MST tasks unless the graph is extremely dense.