### 1. ****Algorithm (Prim's Minimum Spanning Tree Algorithm)****

* **Step 1:** Initialize a key[] array for all vertices as infinity (INT\_MAX). Set the first vertex's key value to 0.
* **Step 2:** Maintain a parent[] array to store the Minimum Spanning Tree (MST) structure.
* **Step 3:** Repeat the following for V-1 times, where V is the number of vertices:
  + Find the minimum key value from the set of vertices not yet included in the MST.
  + Add this vertex to the MST set.
  + Update the key[] and parent[] arrays for all adjacent vertices of the newly added vertex if the edge weight is less than their current key value.
* **Step 4:** After constructing the MST, print the edges and their weights.

### 2. ****Complexity Analysis****

#### Time Complexity:

* **Finding the minimum key vertex** (via minKey): This operation takes O(V)O(V)O(V) time since it linearly scans the key array.
* **Updating the key and parent arrays** for all vertices adjacent to the selected vertex takes O(V)O(V)O(V) in each iteration.
* This process repeats for V−1V-1V−1 vertices.
* **Total time complexity:** O(V2)O(V^2)O(V2).

In the worst case, for a fully connected graph with VVV vertices:

* **Time complexity:** O(V2)O(V^2)O(V2), since for each vertex, we check all its neighbors.

#### Space Complexity:

* Three arrays are used: key[], parent[], and mstSet[]. Each requires space proportional to the number of vertices.
* **Total space complexity:** O(V)O(V)O(V).

### 3. ****Recursive Mathematical Equations****

Let T(V)T(V)T(V) represent the time taken by the algorithm for a graph with VVV vertices.

At each iteration, you:

1. Find the minimum key vertex, which takes O(V)O(V)O(V).
2. Update the keys and parents for the remaining vertices, which takes O(V)O(V)O(V).

Thus, for VVV vertices, the recursive relation for the time complexity is:

T(V)=O(V)+T(V−1)T(V) = O(V) + T(V-1)T(V)=O(V)+T(V−1)

Expanding this recurrence relation:

T(V)=O(V)+O(V−1)+O(V−2)+⋯+O(1)=O(∑i=1Vi)=O(V2)T(V) = O(V) + O(V-1) + O(V-2) + \dots + O(1) = O\left( \sum\_{i=1}^{V} i \right) = O(V^2)T(V)=O(V)+O(V−1)+O(V−2)+⋯+O(1)=O(i=1∑V​i)=O(V2)

### 4. ****Discussions****

**Greedy Approach**: Prim's algorithm uses a greedy approach, always choosing the edge with the minimum weight from the set of vertices not yet included in the MST.

**Graph Representation**: This implementation assumes the graph is represented as an adjacency matrix. If we use an adjacency list with a priority queue (like a binary heap or Fibonacci heap), the time complexity can be reduced to O(Elog⁡V)O(E \log V)O(ElogV), where EEE is the number of edges and VVV is the number of vertices.

**Comparison with Kruskal's Algorithm**: While Prim’s algorithm works better for dense graphs (due to its adjacency matrix representation), Kruskal’s algorithm is more efficient for sparse graphs as it only considers the edges and can be optimized with union-find data structures.

**Edge Weights**: The algorithm assumes the graph is connected and all edge weights are non-negative. If the graph is disconnected, the algorithm will only generate the MST for the connected component that includes the starting vertex.