Kruskal's Approach:

Kruskal's algorithm finds the minimum spanning tree for a connected, undirected graph. It grows the spanning tree by adding the smallest edge that doesn't form a cycle.

Algorithm:

1. Initialize a set **MST** to store the minimum spanning tree.
2. Sort all edges of the graph in ascending order of weights.
3. Iterate through sorted edges. If adding the edge to **MST** doesn't form a cycle, add it to **MST**.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Graph {
   int V;
   vector<pair<int, pair<int, int>>> edges;

public:
   Graph(int V) {
      this->V = V;
   }

   void addEdge(int u, int v, int weight) {
      edges.push_back({weight, {u, v}});
   }

   int findParent(vector<int>& parent, int u) {
      if (parent[u] != u)
         parent[u] = findParent(parent, parent[u]);
      return parent[u];
   }

   void unionSets(vector<int>& parent, vector<int>& rank, int u, int v) {
      int rootU = findParent(parent, u);
      int rootV = findParent(parent, v);
      if (rank[rootU] < rank[rootV])
         parent[rootU] = rootV;
      else if (rank[rootU] > rank[rootV])
         parent[rootV] = rootU;
```

```
        else {
            parent[rootU] = rootV;
            rank[rootV]++;
        }
    }

    void kruskalMST() {
        vector<int> parent(V);
        vector<int> rank(V, 0);
        for (int i = 0; i < V; i++)
            parent[i] = i;

        vector<pair<int, pair<int, int>>> result;
        sort(edges.begin(), edges.end());

        for (auto edge : edges) {
            int weight = edge.first;
            int u = edge.second.first;
            int v = edge.second.second;

            int rootU = findParent(parent, u);
            int rootV = findParent(parent, v);

            if (rootU != rootV) {
                result.push_back({weight, {u, v}});
                unionSets(parent, rank, rootU, rootV);
            }
        }

        cout << "Edges in Minimum Spanning Tree:\n";
        for (auto edge : result) {
            cout << "Edge: " << edge.second.first << " - " << edge.second.second << " Weight: " << edge.first << "\n";
        }
    }
};

int main() {
    Graph g(5);
    g.addEdge(0, 1, 2);
    g.addEdge(0, 3, 6);
    g.addEdge(1, 2, 3);
    g.addEdge(1, 3, 8);
    g.addEdge(1, 4, 5);
    g.addEdge(2, 4, 7);
    g.addEdge(3, 4, 9);
    g.kruskalMST();
    return 0;
}
```

Output:

```
Edges in Minimum Spanning Tree:
Edge: 0 - 1 Weight: 2
Edge: 1 - 2 Weight: 3
Edge: 0 - 3 Weight: 6
Edge: 1 - 4 Weight: 5
```

Space complexity: _____

Justification:_____
_____

Time complexity:

Best case time complexity: _____

Justification:_____
_____

Worst case time complexity: _____

Justification:_____