Marwadi University	Marwadi University Faculty of Technology Department of Information and Communication Technology	
Subject: DAA (01CT0512)	**AIM: Prim's Approach**	
Experiment No: 23	**Date: 10/10/2023**	**Enrolment No: 92100133020**

Prim's Approach:

Prim's algorithm finds the minimum spanning tree for a connected, undirected graph. It starts from an arbitrary node and grows the spanning tree by adding the smallest edge that connects a vertex in the tree to a vertex outside the tree.

Algorithm:

1. Initialize a set **MST** to store the minimum spanning tree, starting with an arbitrary node.
2. Initialize a priority queue **pq** with all edges of the graph.
3. While **MST** does not include all nodes, remove the smallest edge from **pq**. If it connects a node in **MST** to a node outside, add it to **MST**.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <functional>
using namespace std;

class Graph {
  int V;
  vector<pair<int, int>> *adj;

public:
  Graph(int V) {
    this->V = V;
    adj = new vector<pair<int, int>>[V];
  }

  void addEdge(int u, int v, int weight) {
    adj[u].push_back({v, weight});
    adj[v].push_back({u, weight});
  }

  void primMST() {
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    vector<int> key(V, INT_MAX);
    vector<int> parent(V, -1);
    vector<bool> inMST(V, false);
    int src = 0;
    pq.push({0, src});
```

```
        key[src] = 0;

        while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();
            inMST[u] = true;

            for (auto& neighbor : adj[u]) {
                int v = neighbor.first;
                int weight = neighbor.second;

                if (!inMST[v] && weight < key[v]) {
                    key[v] = weight;
                    pq.push({key[v], v});
                    parent[v] = u;
                }
            }
        }

        cout << "Edges in Minimum Spanning Tree:\n";
        for (int i = 1; i < V; i++) {
            cout << "Edge: " << parent[i] << " - " << i << " Weight: " << key[i] << "\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.primMST();
    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:____