Prims algorithm:

```
#include <stdio.h>
#include <limits.h>

#define V 5 // Number of vertices in the graph
#define INF INT_MAX // Representing infinity

// Function to find the vertex with the minimum key value
int minKey(int key[], int mstSet[]) {
    int min = INF, min_index;

    for (int v = 0; v < V; v++) {
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}

// Function to implement Prim's algorithm
void primMST(int graph[V][V]) {
    int parent[V]; // Array to store the constructed MST
    int key[V];    // Key values used to pick minimum weight edge
    int mstSet[V]; // To represent the set of vertices included in MST

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++) {
        key[i] = INF;
        mstSet[i] = 0;
    }
```

```c
    // Always include the first vertex in the MST
    key[0] = 0;     // Make key 0 so that this vertex is picked first
    parent[0] = -1; // First node is always the root of the MST

    // Find the MST for the given graph
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum key vertex from the set of vertices not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST set
        mstSet[u] = 1;

        // Update key value and parent index of the adjacent vertices of the picked vertex
        for (int v = 0; v < V; v++) {
            // Update key[v] only if graph[u][v] is smaller than key[v] and v is not yet in MST
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
                key[v] = graph[u][v];
                parent[v] = u;
            }
        }
    }

    // Print the constructed MST
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++) {
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
    }
}

int main() {
    // Example graph represented as an adjacency matrix
    int graph[V][V] = {
```

```
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };

    // Function call
    primMST(graph);

    return 0;
}
```

Sample output:

```
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

kruskal algorithm:

```
#include <stdio.h>
#include <stdlib.h>

#define V 4 // Number of vertices in the graph

// Structure to represent a weighted edge
typedef struct {
    int src, dest, weight;
} Edge;

// Structure to represent a subset for Union-Find
typedef struct {
```

```
    int parent, rank;
} Subset;

// Function prototypes
int find(Subset subsets[], int i);
void unionSets(Subset subsets[], int x, int y);
int compareEdges(const void *a, const void *b);

// Function to implement Kruskal's algorithm
void kruskalMST(Edge edges[], int E) {
    Edge result[V]; // Array to store the MST
    int e = 0; // Index variable for result[]
    int i = 0; // Index variable for sorted edges

    // Step 1: Sort all edges in non-decreasing order of their weight
    qsort(edges, E, sizeof(Edge), compareEdges);

    // Allocate memory for creating V subsets
    Subset *subsets = (Subset*) malloc(V * sizeof(Subset));

    // Initialize subsets
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Step 2: Process each edge in sorted order
    while (e < V - 1 && i < E) {
        // Get the next edge
        Edge next_edge = edges[i++];

        int x = find(subsets, next_edge.src);
```

```
        int y = find(subsets, next_edge.dest);

        // If including this edge does not cause a cycle
        if (x != y) {
            result[e++] = next_edge;
            unionSets(subsets, x, y);
        }
    }

    // Print the constructed MST
    printf("Edge \tWeight\n");
    for (i = 0; i < e; ++i) {
        printf("%d - %d \t%d \n", result[i].src, result[i].dest, result[i].weight);
    }

    free(subsets);
}

// Function to find the set of an element (with path compression)
int find(Subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}

// Function to do union of two subsets (by rank)
void unionSets(Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank) {
```

```
        subsets[xroot].parent = yroot;
    } else if (subsets[xroot].rank > subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// Comparator function to sort edges by their weights
int compareEdges(const void *a, const void *b) {
    Edge *edgeA = (Edge *)a;
    Edge *edgeB = (Edge *)b;
    return edgeA->weight > edgeB->weight;
}

int main() {
    // Example graph: edges {src, dest, weight}
    Edge edges[] = {
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };

    int E = sizeof(edges) / sizeof(edges[0]);

    // Function call
    kruskalMST(edges, E);

    return 0;
```

}

Sample ouput:

Edge	Weight
2 - 3	4
0 - 3	5
0 - 1	10