

```

/*
    Time complexity:  $O(E * \log(E))$ 
    Space complexity:  $O(V + E)$ 

    where E is the number of edges in the graph and
    V is the number of vertices in the graph
*/

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

class Edge {
public:
    int source;
    int dest;
    int weight;

    void printEdge() {
        cout << min(source, dest) << " " << max(source, dest) << " " << weight;
    }
};

bool compare(Edge e1, Edge e2) {
    return e1.weight < e2.weight;
}

int findParent(int v, int *parent) {
    if (parent[v] == v) {
        return v;
    }

    return findParent(parent[v], parent);
}

void printMST(Edge *input, int v, int e) {
    // Sort the input array in ascending order based on weights
    sort(input, input + e, compare);

    Edge *output = new Edge[v - 1];

    int *parent = new int[v];

    for (int i = 0; i < v; i++) {
        parent[i] = i;
    }

    int count = 0;

```

```

int i = 0;

while (count != v - 1) {
    Edge currentEdge = input[i];

    // Check if we can add the currentEdge in MST or not
    int sourceParent = findParent(currentEdge.source, parent);
    int destParent = findParent(currentEdge.dest, parent);

    if (sourceParent != destParent) {
        output[count] = currentEdge;
        count++;
        parent[sourceParent] = destParent;
    }

    i++;
}

for (int i = 0; i < v - 1; i++) {
    output[i].printEdge();
    cout << "\n";
}
}

int main() {
    int v, e;
    cin >> v >> e;
    Edge *input = new Edge[e];

    for (int i = 0; i < e; i++) {
        int s, d, w;
        cin >> s >> d >> w;
        input[i].source = s;
        input[i].dest = d;
        input[i].weight = w;
    }

    printMST(input, v, e);
}

```