```cpp
#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

// Union-Find Disjoint Sets Library written in OOP manner, using both path
compression and union by rank heuristics
class UnionFind {                                              // OOP style
private:
  vi p, rank, setSize;                       // remember: vi is vector<int>
  int numSets;
public:
  UnionFind(int N) {
    setSize.assign(N, 1); numSets = N; rank.assign(N, 0);
    p.assign(N, 0); for (int i = 0; i < N; i++) p[i] = i; }
  int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j) {
    if (!isSameSet(i, j)) { numSets--;
    int x = findSet(i), y = findSet(j);
    // rank is used to keep the tree short
    if (rank[x] > rank[y]) { p[y] = x; setSize[x] += setSize[y]; }
    else                   { p[x] = y; setSize[y] += setSize[x];
                             if (rank[x] == rank[y]) rank[y]++; } } }
  int numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};

vector<vii> AdjList;
vi taken;                                  // global boolean flag to avoid cycle
priority_queue<ii> pq;            // priority queue to help choose shorter edges

void process(int vtx) {    // so, we use -ve sign to reverse the sort order
  taken[vtx] = 1;
  for (int j = 0; j < (int)AdjList[vtx].size(); j++) {
    ii v = AdjList[vtx][j];
    if (!taken[v.first]) pq.push(ii(-v.second, -v.first));
} }                                // sort by (inc) weight then by (inc) id

int main() {
  int V, E, u, v, w;

  /*
  // Graph in Figure 4.10 left, format: list of weighted edges
  // This example shows another form of reading graph input
  5 7
  0 1 4
  0 2 4
  0 3 6
  0 4 6
  1 2 2
  2 3 8
  3 4 9
  */

  freopen("in_03.txt", "r", stdin);

  scanf("%d %d", &V, &E);
  // Kruskal's algorithm merged with Prim's algorithm
  AdjList.assign(V, vii());
  vector< pair<int, ii> > EdgeList;   // (weight, two vertices) of the edge
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &u, &v, &w);            // read the triple: (u, v, w)
    EdgeList.push_back(make_pair(w, ii(u, v)));                // (w, u, v)
    AdjList[u].push_back(ii(v, w));
    AdjList[v].push_back(ii(u, w));
  }
  sort(EdgeList.begin(), EdgeList.end()); // sort by edge weight O(E log E)
                      // note: pair object has built-in comparison function

  int mst_cost = 0;
  UnionFind UF(V);                     // all V are disjoint sets initially
  for (int i = 0; i < E; i++) {                      // for each edge, O(E)
    pair<int, ii> front = EdgeList[i];
    if (!UF.isSameSet(front.second.first, front.second.second)) {  // check
      mst_cost += front.first;                // add the weight of e to MST
      UF.unionSet(front.second.first, front.second.second);    // link them
  } }                       // note: the runtime cost of UFDS is very light

  // note: the number of disjoint sets must eventually be 1 for a valid MST
  printf("MST cost = %d (Kruskal's)\n", mst_cost);



// inside int main() --- assume the graph is stored in AdjList, pq is empty
  taken.assign(V, 0);                // no vertex is taken at the beginning
  process(0);   // take vertex 0 and process all edges incident to vertex 0
  mst_cost = 0;
  while (!pq.empty()) {  // repeat until V vertices (E=V-1 edges) are taken
    ii front = pq.top(); pq.pop();
    u = -front.second, w = -front.first;  // negate the id and weight again
    if (!taken[u])                 // we have not connected this vertex yet
      mst_cost += w, process(u); // take u, process all edges incident to u
  }                                        // each edge is in pq only once!
  printf("MST cost = %d (Prim's)\n", mst_cost);

  return 0;
}
```