PRACTICAL NO : 05

Aim: Write a program to solve Fractional Knapsack problem.

Algorithm:

Begin

Take an array of structure Item

Declare value, weight, knapsack weight and density

Calculate density=value/weight for each item

Sorting the items array on the order of decreasing density

We add values from the top of the array to total value until the bag is full, i.e; total value <= W

End

CODE :

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

typedef struct {        int v;          int w;          float d;    } Item;

void input(Item items[],int sizeOfItems) {
  cout <<"\n Enter total "<< sizeOfItems <<" item's values and weight \n"<<endl;
  for(int i = 0; i < sizeOfItems; i++) {
```

```
    cout << " Enter "<< i+1 << " V ";
    cin >> items[i].v;
    cout << " Enter "<< i+1 << " W ";
    cin >> items[i].w;
    cout<<endl;    }
}

void display(Item items[], int sizeOfItems) {        int i;
  cout << " values: ";
  for(i = 0; i < sizeOfItems; i++) {          cout << items[i].v << "\t";        }
  cout << endl << " weight: ";
  for (i = 0; i < sizeOfItems; i++) {         cout << items[i].w << "\t";        }
  cout << endl;
}

bool compare(Item i1, Item i2) {      return (i1.d > i2.d);         }

float knapsack(Item items[], int sizeOfItems, int W) {
  int i, j;
  float totalValue = 0, totalWeight = 0;
  for (i = 0; i < sizeOfItems; i++) {    items[i].d = (float)items[i].v / items[i].w;      }
  sort(items, items+sizeOfItems, compare);
  cout << " values : ";
  for(i = 0; i < sizeOfItems; i++) {        cout << items[i].v << "\t";       }
  cout << endl << " weights: ";
```

```
  for (i = 0; i < sizeOfItems; i++) {        cout << items[i].w << "\t";        }
  cout << endl << " ratio  : ";
  for (i = 0; i < sizeOfItems; i++) {       cout << items[i].d << "\t";          }
 cout << endl;

  for(i=0; i<sizeOfItems; i++) {
    if(totalWeight + items[i].w<= W) {
      totalValue += items[i].v ;
      totalWeight += items[i].w;
    } else {
      int wt = W-totalWeight;
      totalValue += (wt * items[i].d);
      totalWeight += wt;
      break;          }
  }
  cout << "\n Total weight in bag " << totalWeight<<endl;
  return totalValue;
}

int main() {
  int W,n;
  cout<<"\n Please provide the input size for Knapsack problem : ";
  cin>>n;
  Item items[n];
  input(items,n);
```

```
  cout << "\n Entered data \n";
  display(items,n);
  cout<< "\n Enter Knapsack weight : ";
  cin >> W;
  float mxVal = knapsack(items,n, W);
  cout <<"\n Max value for "<< W <<" weight is "<< mxVal;
}
```

OUTPUT :

```
input

Please provide the input size for Knapsack problem : 3

Enter total 3 item's values and weight

Enter 1 V 20
Enter 1 W 30

Enter 2 V 50
Enter 2 W 60

Enter 3 V 80
Enter 3 W 90


Entered data
values: 20      50      80
weight: 30      60      90

Enter Knapsack weight : 100
values : 80     50      20
weights: 90     60      30
ratio  : 0.888889       0.833333        0.666667

Total weight in bag 100

Max value for 100 weight is 88.3333

...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL NO : 06

Aim: Implementation and Time analysis of kruskal's Minimum spanning tree algo.

Algorithm:

MST_KRUSKAL(G,W)

A=FI

For each vertex v in V[G]

do Make-Set(v)

Sort the edges of E in no decreasing

Order by weight W

do if Find-set(n) not equal to Find-set(V)

then A = AU{(u,v)}

Union(W,V)

Return A

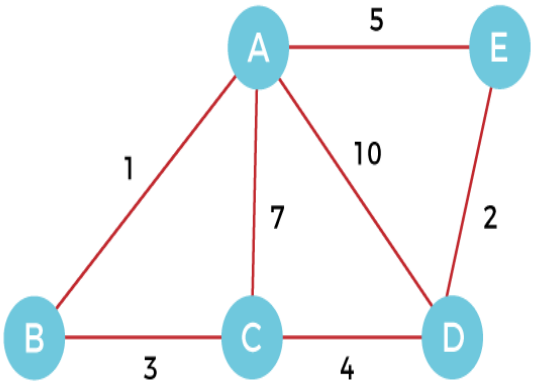
Code:

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

const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];
```

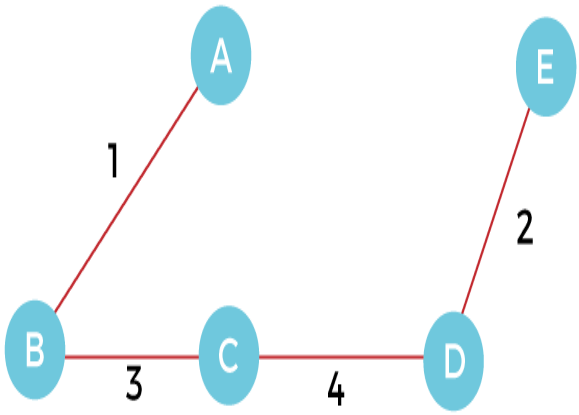
```
void init() {    for(int i = 0;i < MAX;++i){    id[i] = i;       }    }


int root(int x)    {
                              while(id[x] != x)    {          id[x] = id[id[x]];
                                                                 x = id[x];                       }
   return x;            }


void union1(int x, int y) {
   int p = root(x);
   int q = root(y);
   id[p] = id[q];      }


long long kruskal(pair<long long, pair<int, int> > p[]) {
   int x, y;
   long long cost, minimumCost = 0;
   for(int i = 0;i < edges;++i)   {
      x = p[i].second.first;
      y = p[i].second.second;
      cost = p[i].first;


      if(root(x) != root(y)) {       minimumCost += cost;
                                          union1(x, y);           }
   }
   return minimumCost;
}
```

```
int main()  {
   int x, y;
   long long weight, cost, minimumCost;
   init();
   cout <<"\n Please provide the input value for NODES and EDGES for your kruskal algorithm :
";
   cin >> nodes >> edges;


   cout<<"\n Now please provide the value for X , Y and EDGES \n";
   for(int i = 0;i < edges;++i)  {
      cout<<" ";
      cin >> x >> y >> weight;
      p[i] = make_pair(weight, make_pair(x, y));
   }


   sort(p, p + edges);
   minimumCost = kruskal(p);
   cout <<"\n Minimum cost is "<< minimumCost << endl;
   return 0;
}
```

Graph :



ORIGINAL GRAPH



KRUSKAL MST GRAPH

OUTPUT :

Time Complexity : O(E logE) or O(V logV)

Space Complexity : O(log(E))

```
Input

Please provide the input value for NODES and EDGES for your kruskal algorithm : 5 7

Now please provide the value for X , Y and EDGES
1 2 1
1 3 7
1 4 10
1 5 5
2 3 3
3 4 4
4 5 2

Minimum cost is 10
```

PRACTICAL NO : 07

Aim: Implementation and Time analysis of Prims Minimum spanning tree algo.

Algorithm :

PRISM(g,w,r)

For each u in V[g]

Do key [u] = infinte

PI[r] = NIL

Key[r] = 0

Q= V[g]

While Q is not equal to FI

do u = EXTRACT-MIN[g]

For each V in adj[u]

do if V in Q & w(u,v) key[v]

Then PI = u

Key[v] = w(u,v)

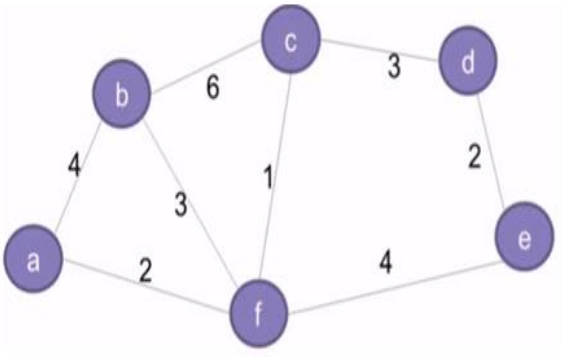
Code :

```
#include<iostream>

using namespace std;


const int V=6;

int min_Key(int key[], bool visited[]) {

   int min = 999, min_index;
```

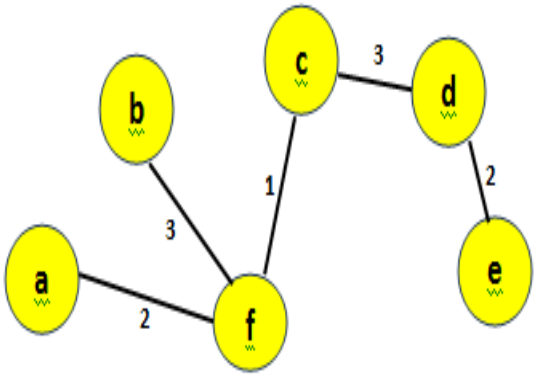
```
  for (int v = 0; v < V; v++) {
     if (visited[v] == false && key[v] < min) {
        min = key[v];
        min_index = v;       }
 }  return min_index;
}

 int print_MST(int parent[], int cost[V][V])  {
   int minCost=0;
   cout<<"\n Edge \t Weight\n";
   for (int i = 1; i< V; i++) {
          cout<<" "<<parent[i]<<" - "<<i<<" \t "<<cost[i][parent[i]]<<" \n";
  minCost+=cost[i][parent[i]];            }
       cout<<"\n Total cost is "<<minCost;           }

 void find_MST(int cost[V][V])  {
   int parent[V], key[V];
   bool visited[V];

   for (int i = 0; i< V; i++) {
     key[i] = 999;
     visited[i] = false;
      parent[i]=-1;
 }
```

```
    key[0] = 0;
    parent[0] = -1;

    for (int x = 0; x < V - 1; x++) {
        int u = min_Key(key, visited);
        visited[u] = true;
        for (int v = 0; v < V; v++) {
            if (cost[u][v]!=0 && visited[v] == false && cost[u][v] < key[v]){
                parent[v] = u;
                key[v] = cost[u][v];
            }
        }
    }
     print_MST(parent, cost);
}

int main()  {
    int cost[V][V];
    cout<<"\n Enter the vertices for a graph with 6 vetices ";
    for (int i=0;i<V;i++){
        for(int j=0;j<V;j++){       cin>>cost[i][j];        }
    } find_MST(cost);
    return 0;
}
```

GRAPH :



ORIGINAL GRAPH



PRISM MST GRAPH

OUTPUT :

```
input
Enter the vertices for a graph with 6 vetices
0 4 0 0 0 2
4 0 6 0 0 3
0 6 0 3 0 1
0 0 3 0 2 0
0 0 0 2 0 4
2 3 1 0 4 0

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

Total cost is 11

...Program finished with exit code 0
Press ENTER to exit console.
```