1)

```
#include <iostream>

#include <climits>

#define V 6

const char* sprinklers[V] = {"A", "B", "C", "D", "E", "F"};

int minKey(int key[], bool mstSet[]) {

  int m = INT_MAX, idx = -1;

  for (int v = 0; v < V; v++) if (!mstSet[v] && key[v] < m) m = key[v], idx = v;

  return idx;

}

void primMST(int g[V][V], int start) {

  int parent[V], key[V]; bool mstSet[V];

  for (int i = 0; i < V; i++) key[i] = INT_MAX, mstSet[i] = 0;

  key[start] = 0, parent[start] = -1;

  for (int c = 0; c < V - 1; c++) {

    int u = minKey(key, mstSet); mstSet[u] = 1;

    for (int v = 0; v < V; v++)

      if (g[u][v] && !mstSet[v] && g[u][v] < key[v]) parent[v] = u, key[v] = g[u][v];

  }

  int total = 0;

  std::cout << "Edge\tLength\n";

  for (int i = 0; i < V; i++)

    if (parent[i] != -1)

      std::cout << sprinklers[parent[i]] << "-" << sprinklers[i] << "\t" << g[i][parent[i]] << "\n",
total += g[i][parent[i]];
```

```
    std::cout << "Total piping needed = " << total << " meters\n";
}
int main() {
    int g[V][V] = {
        //A B C D E F
        {0,5,2,6,0,0}, // A
        {5,0,2,0,0,0}, // B
        {2,2,0,0,3,0}, // C
        {6,0,0,0,3,7}, // D
        {0,0,3,3,0,8}, // E
        {0,0,0,7,8,0}  // F
    };
    primMST(g, 5); // Start from F (index 5)
    return 0;
}
```

1 Modified)

```
#include <iostream>
#include <climits>

#define V 9

const char* sprinklers[V] = {"A", "B", "C", "D", "E", "F", "G", "H", "I"};

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, index = -1;
    for (int v = 0; v < V; v++)
        if (!mstSet[v] && key[v] < min)
            min = key[v], index = v;

    return index;
}

void primMST(int g[V][V], int start) {
    int parent[V], key[V];
    bool mstSet[V] = {false};

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX;

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

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;

        for (int v = 0; v < V; v++)
            if (g[u][v] && !mstSet[v] && g[u][v] < key[v])
                parent[v] = u, key[v] = g[u][v];
    }
    int total = 0;

    std::cout << "Edge\tLength\n";

    for (int i = 0; i < V; i++) {

        if (parent[i] != -1) {

            std::cout << sprinklers[parent[i]] << "-" << sprinklers[i] << "\t" << g[i][parent[i]] << "\n";

            total += g[i][parent[i]];

        }

    }

    std::cout << "Total piping needed = " << total << " meters\n";

}


int main() {

    int g[V][V] = {

        {0, 4, 0, 0, 0, 0, 0, 8, 0},

        {4, 0, 8, 0, 0, 0, 0, 11, 0},

        {0, 8, 0, 7, 0, 4, 0, 0, 2},

        {0, 0, 7, 0, 9, 14, 0, 0, 0},

        {0, 0, 0, 9, 0, 10, 0, 0, 0},

        {0, 0, 4, 14, 10, 0, 2, 0, 0},

        {0, 0, 0, 0, 0, 2, 0, 1, 6},

        {8, 11, 0, 0, 0, 0, 1, 0, 7},

        {0, 0, 2, 0, 0, 0, 6, 7, 0}

    };


    primMST(g, 5); // Start from vertex F (index 5)
```

```
  return 0;

}
```

2)

```
#include

#include

#define V 6 // Number of locations

const char* cities[V] = {"A", "B", "C", "D", "E", "F"};

int minKey(int key[], bool mstSet[]) { int min = INT_MAX, min_index = -1; for (int v = 0; v < V;
v++) if (!mstSet[v] && key[v] < min) min = key[v], min_index = v; return min_index; }

void primMST(int graph[V][V], int start) { int parent[V], key[V]; bool mstSet[V] = {false};

for (int i = 0; i < V; i++)
  key[i] = INT_MAX;

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

for (int count = 0; count < V - 1; count++) {
  int u = minKey(key, mstSet);
  mstSet[u] = true;

  for (int v = 0; v < V; v++)
    if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])
      parent[v] = u, key[v] = graph[u][v];
}

int total = 0;
std::cout << "Edge\tCost\n";
for (int i = 0; i < V; i++)
  if (parent[i] != -1) {
    std::cout << cities[parent[i]] << "-" << cities[i] << "\t" << graph[i][parent[i]] << "\n";
    total += graph[i][parent[i]];
  }
std::cout << "Minimum driving route cost = " << total << " km\n";

}

int main() { int graph[V][V] = { {0, 4, 2, 0, 0, 0}, {4, 0, 1, 0, 5, 0}, {2, 1, 0, 3, 8, 0}, {0, 0, 3, 0, 0,
7}, {0, 5, 8, 0, 0, 9}, {0, 0, 0, 7, 9, 0} };

int start;
std::cout << "Enter start location (0-5): ";
std::cin >> start;

primMST(graph, start);
return 0;

}
```

3) /* Fractional Knapsack

Pseudocode

```
function fracKnapsack(items, n, capacity):
    sort items in descending order of profit/weight ratio
    totalProfit = 0
    for i = 0 to n-1:
        if capacity >= items[i].weight:
            capacity -= items[i].weight
            totalProfit += items[i].profit
            print item fully included
        else:
            fraction = capacity / items[i].weight
            totalProfit += items[i].profit * fraction
        print item fractionally included
        break
  print totalProfit


  Complexity Analysis

Sorting: O(n log n) (for sorting items by profit/weight ratio)
Loop: O(n) (each item is considered at most once)
Total Time Complexity: O(n log n)
Space Complexity: O(1) (ignoring input storage; only a few variables used)
This is efficient and optimal for the fractional knapsack problemComplexity Analysis

*/

#include <bits/stdc++.h>

using namespace std;

struct item
{
  int weight;
  int profit;
};

bool cmp(item a, item b)
```

```
{
   return (double)a.profit / a.weight > (double)b.profit / b.weight;
}

void fracKnapsack(item items[], int n, int capacity)
{
   double totalProfit = 0.0;
   sort(items, items + n, cmp);
   cout << "Selected items(profit,weight):" << endl;
   for (int i = 0; i < n; i++)
   {
      if (capacity >= items[i].weight)
      {
         capacity -= items[i].weight;
         totalProfit += items[i].profit;

         cout << "(" << items[i].profit << "," << items[i].weight << ") - fully included" << endl;
      }
      else
      {
         double fraction = (double)capacity / items[i].weight;
         totalProfit += items[i].profit * fraction;
         cout << "(" << items[i].profit << ", " << items[i].weight << ") - " << fraction * 100 << "%
included" << endl;
         break;
      }
```

```
    }
    cout << "Maximum profit: " << totalProfit << endl;
}

int main()
{
    item items[] = {
        {1, 10}, {2, 15}, {3, 25}, {2, 12}};
    int n = sizeof(items) / sizeof(items[0]);
    int capacity = 5;

    clock_t start = clock();
    fracKnapsack(items, n, capacity);
    clock_t end = clock();

    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC * 1000;
    cout << "Time taken: " << time_taken << " ms" << endl;

    return 0;
}


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

int knapsack(int W, int wt[], int val[], int n)
```

```
{
    int dp[n + 1][W + 1];
    for (int i = 0; i <= n; i++)
    {
        dp[i][0] = 0;
    }
    for (int j = 0; j <= W; j++)
    {
        dp[0][j] = 0;
    }

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= W; j++)
        {
            if (wt[i - 1] > j)
            {
                dp[i][j] = dp[i - 1][j];
            }
            else
            {
                dp[i][j] = max(val[i - 1] + dp[i - 1][j - wt[i - 1]], dp[i - 1][j]);
            }
        }
    }
    return dp[n][W];
}


int main()
{
   int val[] = {10, 20, 50, 60};
   int wt[] = {2, 3, 4, 5};
   int W = 8;
   int n = 4;
   clock_t start = clock();
   cout << "Maximum value in knapsack:" << knapsack(W, wt, val, n) << endl;
   clock_t end = clock();

   double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC * 1000;
   cout << "Time Taken: " << time_taken << "ms" << endl;
   return 0;
}
```

Modification

add

```
int val[] = {10, 20, 50, 60, 25}; // Added 25
int wt[]  = {2,  3,  4,  5,  3};  // Added 3
int n = 5; // Updated number of items
```

Duplicate

```
int val[] = {10, 20, 50, 60, 25, 50}; // Duplicated 50
int wt[]  = {2,  3,  4,  5,  3,  4};  // Duplicated 4
int n = 6; // Updated number of items
```

Both

```
int val[] = {10, 20, 50, 60, 25, 50}; // Added 25 and duplicated 50

int wt[]  = {2,  3,  4,  5,  3,  4};  // Added 3 and duplicated 4

int W = 8;

int n = 6;
```

4)

```
#include <bits/stdc++.h>

using namespace std;

#define V 7


const char *cities[V] = {"SF", "LA", "Denver", "Dallas", "Chicago", "NY", "Boston"};


int minDistance(int dist[], bool visited[])

{

  int min = INT_MAX, idx = -1;

  for (int v = 0; v < V; v++)

  {

    if (!visited[v] && dist[v] < min)

      min = dist[v], idx = v;

  }

  return idx;

}
```

```
void dijikstra(int graph[V][V], int src)
{
  int dist[V], parent[V];
  bool visited[V];
  for (int i = 0; i < V; i++)
    dist[i] = INT_MAX, visited[i] = false, parent[i] = -1;
  dist[src] = 0;
  for (int count = 0; count < V - 1; count++)
  {
    int u = minDistance(dist, visited);
    if (u == -1)
      break;
    visited[u] = true;
    for (int v = 0; v < V; v++)
      if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
        dist[v] = dist[u] + graph[u][v], parent[v] = u;
  }
  int target = 5;
  cout << "Shortest Time to NY: ";
  if (dist[target] == INT_MAX)
    cout << "NoPath\n";
  else
    cout << dist[target] << "s\n";
}
```

```
int main()

{

   int graph[V][V] = {

      {0, 3, 4, 5, 0, 0, 0},

      {3, 0, 7, 5, 0, 0, 0},

      {4, 7, 0, 4, 6, 0, 0},

      {5, 5, 4, 0, 5, 6, 0},

      {0, 0, 6, 5, 0, 4, 3},

      {0, 0, 0, 6, 4, 0, 2},

      {0, 0, 0, 0, 3, 2, 0}};

   dijikstra(graph, 0);

   return 0;

}
```

5)

```
#include <iostream>

#include <climits>

#define V 6

const char* sprinklers[V] = {"A", "B", "C", "D", "E", "F"};

int minKey(int key[], bool mstSet[]) {

   int m = INT_MAX, idx = -1;

   for (int v = 0; v < V; v++) if (!mstSet[v] && key[v] < m) m = key[v], idx = v;

   return idx;

}

void primMST(int g[V][V], int start) {
    int parent[V], key[V]; bool mstSet[V];
    for (int i = 0; i < V; i++) key[i] = INT_MAX, mstSet[i] = 0;
    key[start] = 0, parent[start] = -1;
    for (int c = 0; c < V - 1; c++) {
        int u = minKey(key, mstSet); mstSet[u] = 1;
        for (int v = 0; v < V; v++)
            if (g[u][v] && !mstSet[v] && g[u][v] < key[v]) parent[v] = u, key[v] = g[u][v];
    }
    int total = 0;
    std::cout << "Edge\tLength\n";
    for (int i = 0; i < V; i++)
        if (parent[i] != -1)
            std::cout << sprinklers[parent[i]] << "-" << sprinklers[i] << "\t" << g[i][parent[i]] << "\n",
total += g[i][parent[i]];
    std::cout << "Total piping needed = " << total << " meters\n";
}
int main() {
    int g[V][V] = {
        //A B C D E F
        {0,5,2,6,0,0}, // A
        {5,0,2,0,0,0}, // B
        {2,2,0,0,3,0}, // C
        {6,0,0,0,3,7}, // D
        {0,0,3,3,0,8}, // E
        {0,0,0,7,8,0}  // F
    };
    primMST(g, 5); // Start from F (index 5)

    return 0;

}



#include <bits/stdc++.h>

using namespace std;


int knapsack(int W, int wt[], int val[], int n)

{

    int dp[n + 1][W + 1];

    for (int i = 0; i <= n; i++)

    {

        dp[i][0] = 0;

    }

    for (int j = 0; j <= W; j++)

    {

        dp[0][j] = 0;

    }


    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= W; j++)

        {

            if (wt[i - 1] > j)

            {
```

```
            dp[i][j] = dp[i - 1][j];
         }
         else
         {
            dp[i][j] = max(val[i - 1] + dp[i - 1][j - wt[i - 1]], dp[i - 1][j]);
         }
      }
   }
   return dp[n][W];
}

int main()
{
   int val[] = {10, 20, 50, 60};
   int wt[] = {2, 3, 4, 5};
   int W = 8;
   int n = 4;
   clock_t start = clock();
   cout << "Maximum value in knapsack:" << knapsack(W, wt, val, n) << endl;
   clock_t end = clock();

   double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC * 1000;
   cout << "Time Taken: " << time_taken << "ms" << endl;
   return 0;
}
```