ST STEPHEN COLLEGE

UNIVERSITY OF DELHI

COURSE :- BSC (PROG) COMPUTER SCIENCE

SEMESTER :- 3RD

YEAR :- 2025-2026

PAPER NAME :- DESIGN AND ANALYSIS OF ALGORITHMS

UNIQUE PAPER CODE :- 2342572302

PRACTICAL FILE

ROLL NUMBER :- 24080582015

NAME :- VISHAL KUMAR JHA

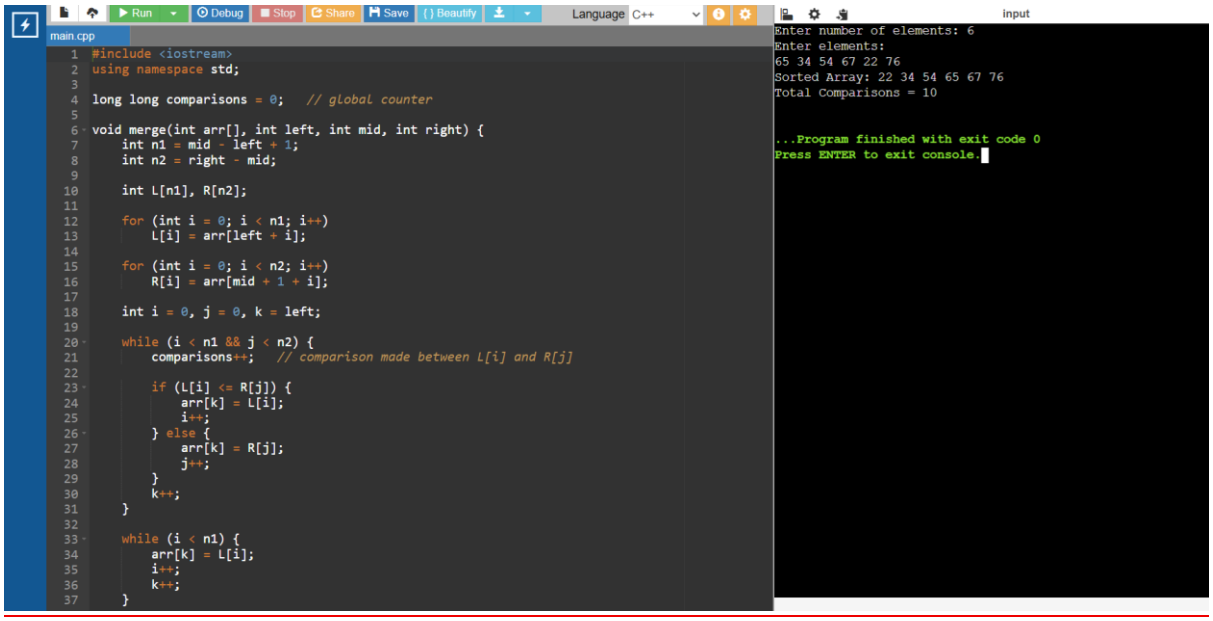
Practical 1 :- Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).

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

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    int comparisons = 0;

    // Insertion Sort
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0) {
            comparisons++;          // comparison made
            if (arr[j] > key) {
                arr[j + 1] = arr[j]; // shifting
                j--;
            } else {
                break;
            }
        }
        arr[j + 1] = key;
    }
```

```
Enter number of elements: 7
Enter elements:
7 6 5 4 3 2 1
Sorted Array: 1 2 3 4 5 6 7
Total Comparisons = 21


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

```cpp
    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    cout << "\nTotal Comparisons = " << comparisons << endl;

    return 0;
}
```

Practical 2 :- Write a program to sort the elements of an array using Merge Sort (The program should report the number of comparisons).



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

long long comparisons = 0;   // global counter

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];

    for (int i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        comparisons++;   // comparison made between L[i] and R[j]

        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
```

```
Enter number of elements: 6
Enter elements:
65 34 54 67 22 76
Sorted Array: 22 34 54 65 67 76
Total Comparisons = 10


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

```
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    mergeSort(arr, 0, n - 1);

    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    cout << "\nTotal Comparisons = " << comparisons << endl;
```

Practical 3 :- Write a program to sort the elements of an array using Heap Sort (The program should report the number of comparisons).

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

long long comparisons = 0;   // global comparison counter

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    // Compare parent with left child
    if (left < n) {
        comparisons++;   // comparison made
        if (arr[left] > arr[largest]) {
            largest = left;
        }
    }

    // Compare parent with right child
    if (right < n) {
        comparisons++;   // comparison made
        if (arr[right] > arr[largest]) {
            largest = right;
        }
    }

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
```

```
Enter number of elements: 7
Enter elements:
98 43 65 22 12 56 89
Sorted Array: 12 22 43 56 65 89 98
Total Comparisons = 16


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

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp

```cpp
}

void heapSort(int arr[], int n) {
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    // Extract elements one by one
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    heapSort(arr, n);

    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    cout << "\nTotal Comparisons = " << comparisons << endl;

    return 0;
}
```

Practical 4 :- Write a program to sort the elements of an array using Quick Sort (The program should report the number of comparisons).

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp

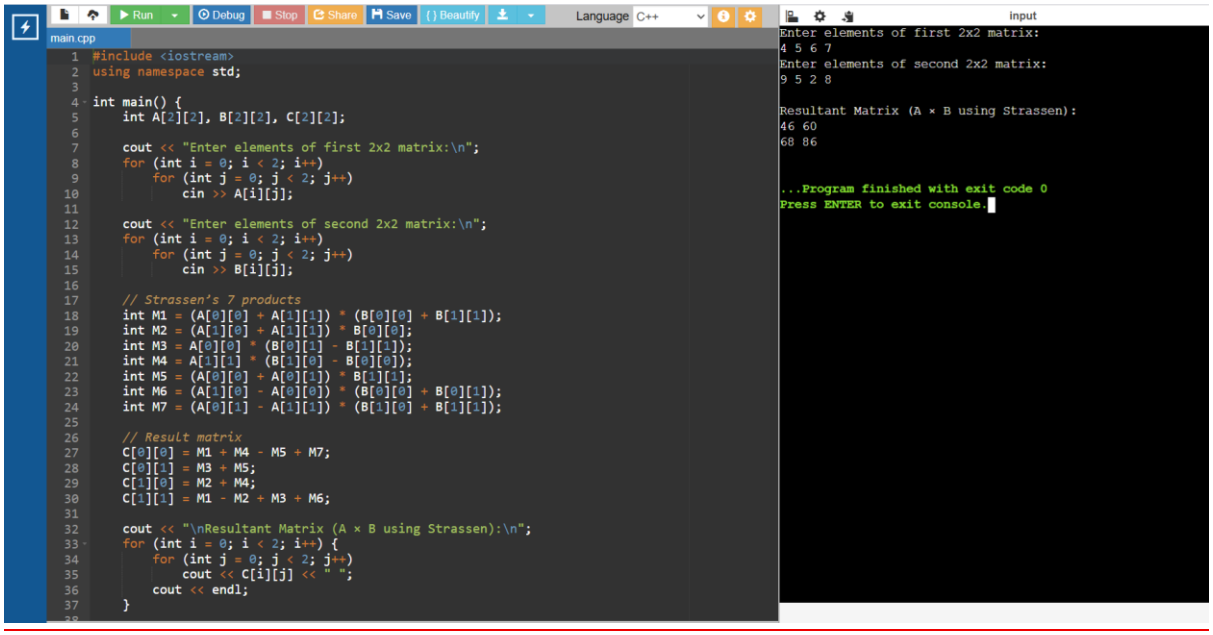
```cpp
#include <iostream>
using namespace std;

long long comparisons = 0;   // global comparison counter

int partitionArray(int arr[], int low, int high) {
    int pivot = arr[high];   // choose last element as pivot
    int i = low - 1;

    for (int j = low; j < high; j++) {
        comparisons++;   // comparison made between arr[j] and pivot

        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i + 1], arr[high]);
    return i + 1;  // pivot index after partition
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partitionArray(arr, low, high);

        quickSort(arr, low, pi - 1);   // left part
        quickSort(arr, pi + 1, high);  // right part
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
```

input

```
Enter number of elements: 8
Enter elements:
43 34 12 56 34 89 432 9
Sorted Array: 9 12 34 34 43 56 89 432
Total Comparisons = 19


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

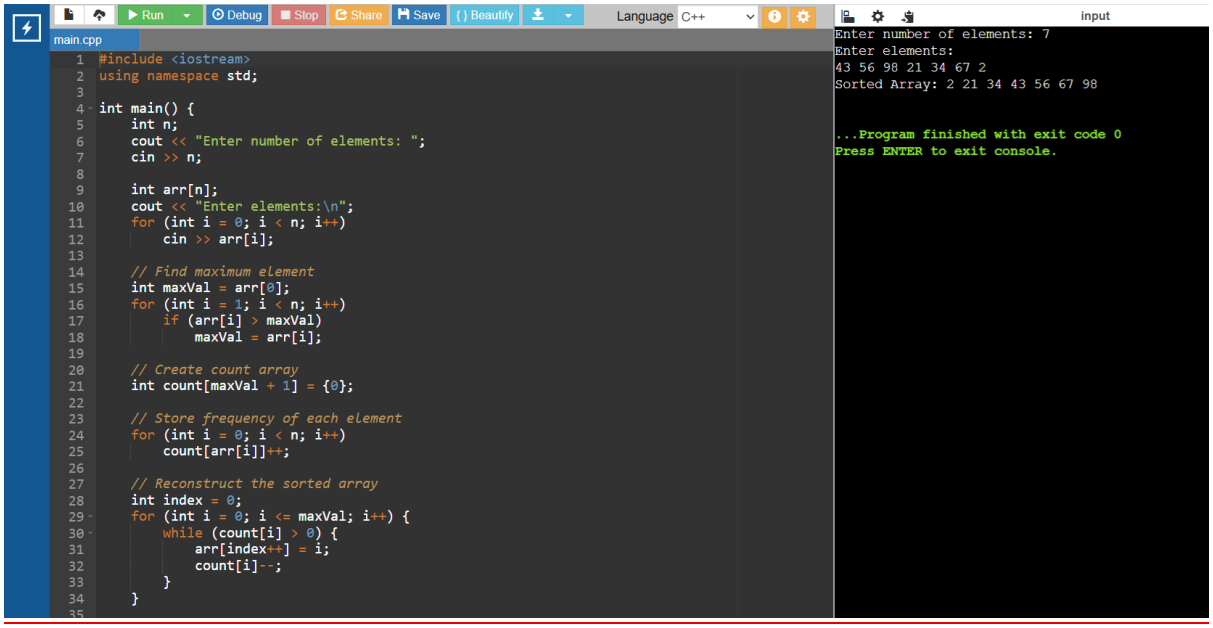
```cpp
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    quickSort(arr, 0, n - 1);

    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    cout << "\nTotal Comparisons = " << comparisons << endl;

    return 0;
}
```

Practical 5 :- Write a program to multiply two matrices using the Strassen's algorithm for matrix multiplication.




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

int main() {
    int A[2][2], B[2][2], C[2][2];

    cout << "Enter elements of first 2x2 matrix:\n";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            cin >> A[i][j];

    cout << "Enter elements of second 2x2 matrix:\n";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            cin >> B[i][j];

    // Strassen's 7 products
    int M1 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
    int M2 = (A[1][0] + A[1][1]) * B[0][0];
    int M3 = A[0][0] * (B[0][1] - B[1][1]);
    int M4 = A[1][1] * (B[1][0] - B[0][0]);
    int M5 = (A[0][0] + A[0][1]) * B[1][1];
    int M6 = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1]);
    int M7 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);

    // Result matrix
    C[0][0] = M1 + M4 - M5 + M7;
    C[0][1] = M3 + M5;
    C[1][0] = M2 + M4;
    C[1][1] = M1 - M2 + M3 + M6;

    cout << "\nResultant Matrix (A × B using Strassen):\n";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++)
            cout << C[i][j] << " ";
        cout << endl;
    }
```

Practical 6 :- Write a program to sort the elements of an array using Count Sort



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

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    // Find maximum element
    int maxVal = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > maxVal)
            maxVal = arr[i];

    // Create count array
    int count[maxVal + 1] = {0};

    // Store frequency of each element
    for (int i = 0; i < n; i++)
        count[arr[i]]++;

    // Reconstruct the sorted array
    int index = 0;
    for (int i = 0; i <= maxVal; i++) {
        while (count[i] > 0) {
            arr[index++] = i;
            count[i]--;
        }
    }
```

```
Enter number of elements: 7
Enter elements:
43 56 98 21 34 67 2
Sorted Array: 2 21 34 43 56 67 98


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

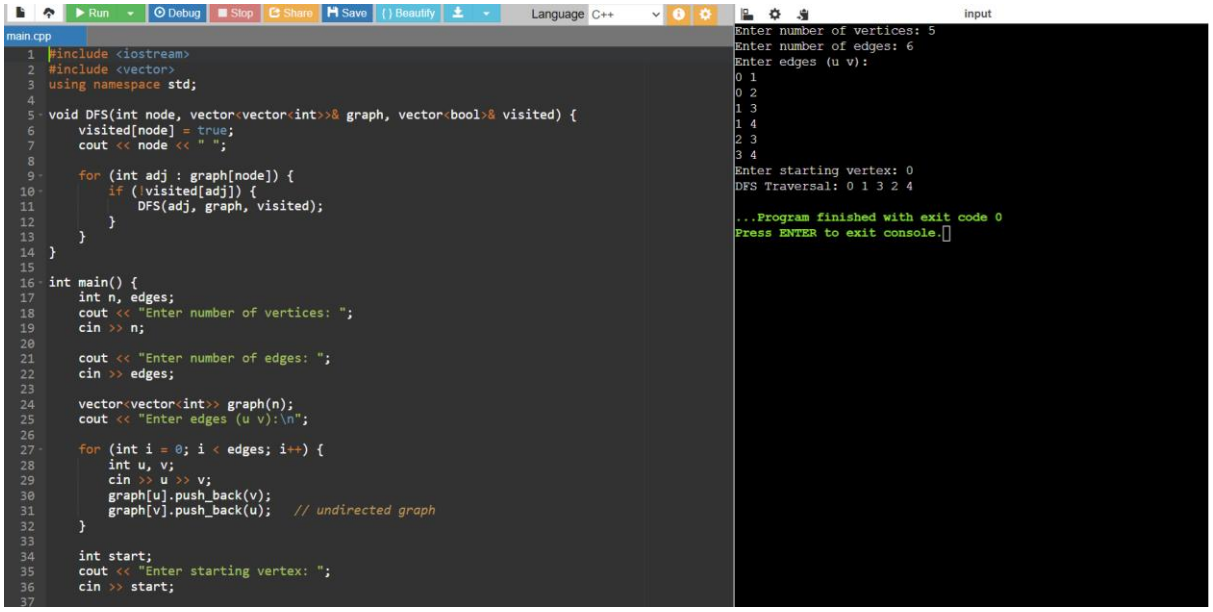
```cpp
    // Output sorted array
    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    cout << endl;
    return 0;
}
```

Practical 7 :- Display the data stored in a given graph using the Breadth-First Search algorithm

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

void BFS(int start, vector<vector<int>>& graph, int n) {
    vector<bool> visited(n, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    cout << "BFS Traversal: ";

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int adj : graph[node]) {
            if (!visited[adj]) {
                visited[adj] = true;
                q.push(adj);
            }
        }
    }
}

int main() {
    int n, edges;
    cout << "Enter number of vertices: ";
    cin >> n;

    cout << "Enter number of edges: ";
    cin >> edges;

    vector<vector<int>> graph(n);
```

```
Enter number of vertices: 5
Enter number of edges: 6
Enter edges (u v):
0 1
0 2
1 3
1 4
2 3
3 4
Enter starting vertex: 0
BFS Traversal: 0 1 2 3 4

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

Practical 8 :- Display the data stored in a given graph using the Depth-First Search algorithm.



```cpp
#include <iostream>
#include <vector>
using namespace std;

void DFS(int node, vector<vector<int>>& graph, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int adj : graph[node]) {
        if (!visited[adj]) {
            DFS(adj, graph, visited);
        }
    }
}

int main() {
    int n, edges;
    cout << "Enter number of vertices: ";
    cin >> n;

    cout << "Enter number of edges: ";
    cin >> edges;

    vector<vector<int>> graph(n);
    cout << "Enter edges (u v):\n";

    for (int i = 0; i < edges; i++) {
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);   // undirected graph
    }

    int start;
    cout << "Enter starting vertex: ";
    cin >> start;
```

```
Enter number of vertices: 5
Enter number of edges: 6
Enter edges (u v):
0 1
0 2
1 3
1 4
2 3
3 4
Enter starting vertex: 0
DFS Traversal: 0 1 3 2 4

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

```cpp
    int start;
    cout << "Enter starting vertex: ";
    cin >> start;

    vector<bool> visited(n, false);

    cout << "DFS Traversal: ";
    DFS(start, graph, visited);

    return 0;
}
```

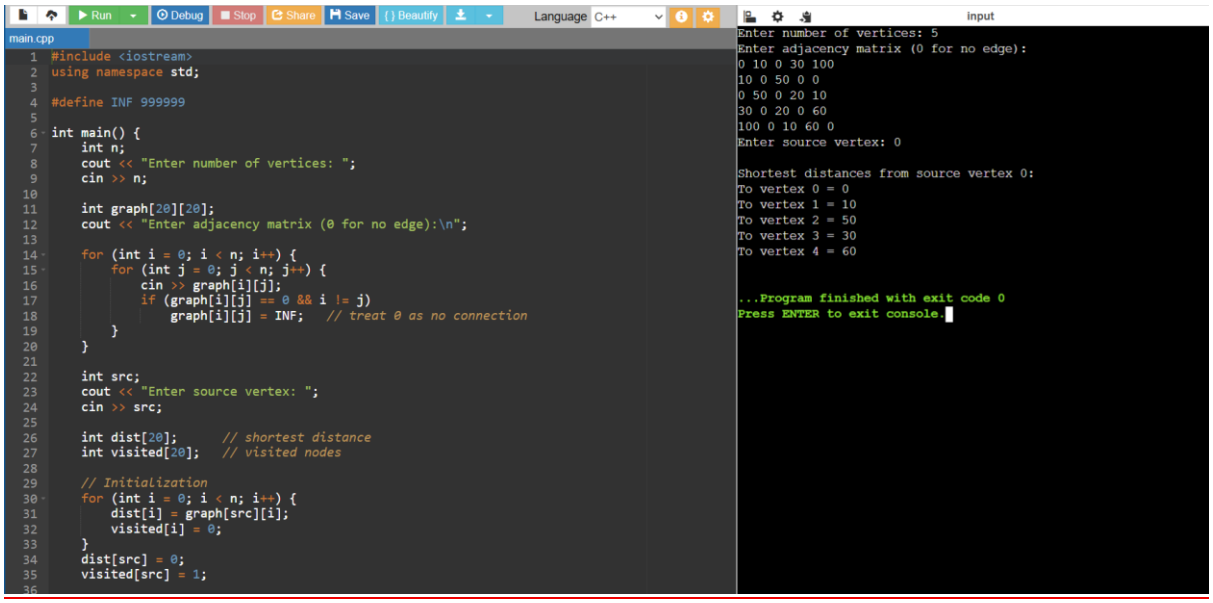
Practical 9 :- Write a program to determine a minimum spanning tree of a graph using the Prim's algorithm.

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

#define INF 999999

int main() {
    int n;
    cout << "Enter number of vertices: ";
    cin >> n;

    int graph[20][20];
    cout << "Enter adjacency matrix (0 for no edge):\n";

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
            if (graph[i][j] == 0 && i != j)
                graph[i][j] = INF;  // treat 0 as no edge
        }
    }

    int visited[20] = {0};
    visited[0] = 1;  // start from vertex 0

    int edges = 0;
    int minCost = 0;

    cout << "\nEdges in Minimum Spanning Tree:\n";

    while (edges < n - 1) {
        int u = -1, v = -1, min = INF;

        // Find minimum edge
        for (int i = 0; i < n; i++) {
            if (visited[i]) {
                for (int j = 0; j < n; j++) {
                    if (!visited[j] && graph[i][j] < min) {
```

```
Enter number of vertices: 5
Enter adjacency matrix (0 for no edge):
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

Edges in Minimum Spanning Tree:
0 - 1 : 2
1 - 2 : 3
1 - 4 : 5
0 - 3 : 6

Minimum Cost = 16


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

```cpp
    while (edges < n - 1) {
        int u = -1, v = -1, min = INF;

        // Find minimum edge
        for (int i = 0; i < n; i++) {
            if (visited[i]) {
                for (int j = 0; j < n; j++) {
                    if (!visited[j] && graph[i][j] < min) {
                        min = graph[i][j];
                        u = i;
                        v = j;
                    }
                }
            }
        }

        cout << u << " - " << v << " : " << graph[u][v] << endl;
        minCost += graph[u][v];
        visited[v] = 1;
        edges++;
    }

    cout << "\nMinimum Cost = " << minCost << endl;

    return 0;
}
```

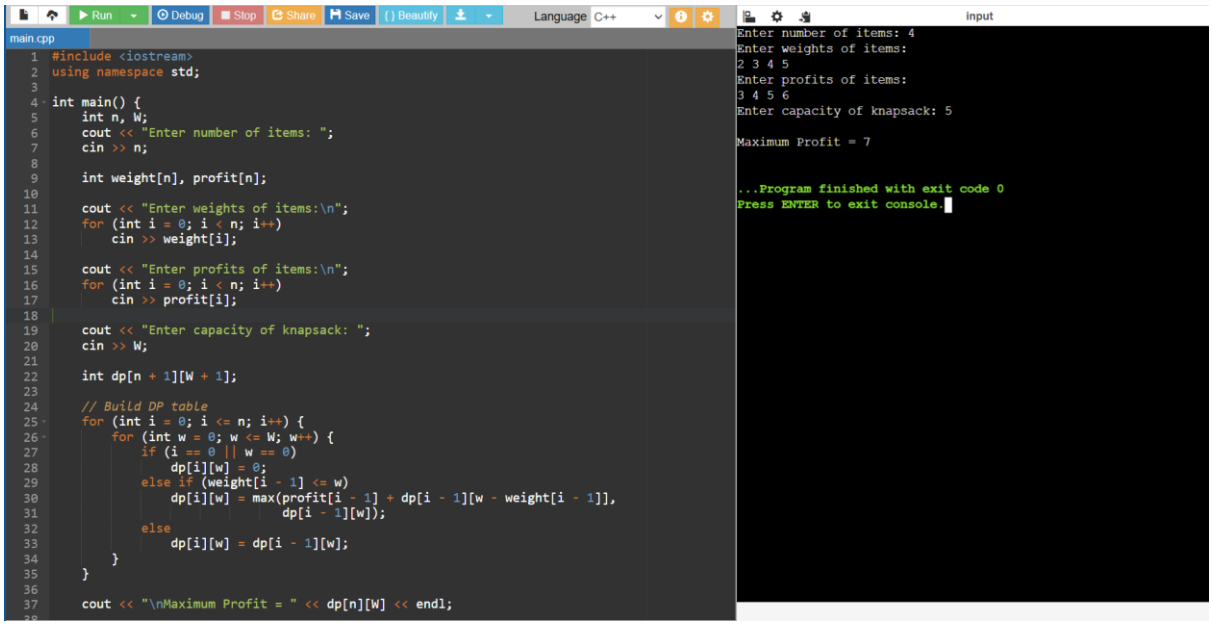
Practical 10 :- Write a program to determine the shortest path from a given node s to the other nodes of a graph using the Dijkstra's algorithm.




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

#define INF 999999

int main() {
    int n;
    cout << "Enter number of vertices: ";
    cin >> n;

    int graph[20][20];
    cout << "Enter adjacency matrix (0 for no edge):\n";

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
            if (graph[i][j] == 0 && i != j)
                graph[i][j] = INF;   // treat 0 as no connection
        }
    }

    int src;
    cout << "Enter source vertex: ";
    cin >> src;

    int dist[20];      // shortest distance
    int visited[20];   // visited nodes

    // Initialization
    for (int i = 0; i < n; i++) {
        dist[i] = graph[src][i];
        visited[i] = 0;
    }
    dist[src] = 0;
    visited[src] = 1;
```

```
    // Dijkstra's main loop
    for (int count = 1; count < n; count++) {
        int minDist = INF, u = -1;

        // Pick unvisited vertex with smallest distance
        for (int i = 0; i < n; i++) {
            if (!visited[i] && dist[i] < minDist) {
                minDist = dist[i];
                u = i;
            }
        }

        visited[u] = 1;

        // Update distances
        for (int v = 0; v < n; v++) {
            if (!visited[v] && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Output
    cout << "\nShortest distances from source vertex " << src << ":\n";
    for (int i = 0; i < n; i++) {
        cout << "To vertex " << i << " = " << dist[i] << endl;
    }

    return 0;
}
```

Practical 11 :- Write a program to solve the 0-1 knapsack problem using Dynamic Programming.




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

int main() {
    int n, W;
    cout << "Enter number of items: ";
    cin >> n;

    int weight[n], profit[n];

    cout << "Enter weights of items:\n";
    for (int i = 0; i < n; i++)
        cin >> weight[i];

    cout << "Enter profits of items:\n";
    for (int i = 0; i < n; i++)
        cin >> profit[i];

    cout << "Enter capacity of knapsack: ";
    cin >> W;

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

    // Build DP table
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weight[i - 1] <= w)
                dp[i][w] = max(profit[i - 1] + dp[i - 1][w - weight[i - 1]],
                               dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    cout << "\nMaximum Profit = " << dp[n][W] << endl;
```

```
    cout << "\nMaximum Profit = " << dp[n][W] << endl;

    return 0;
}
```