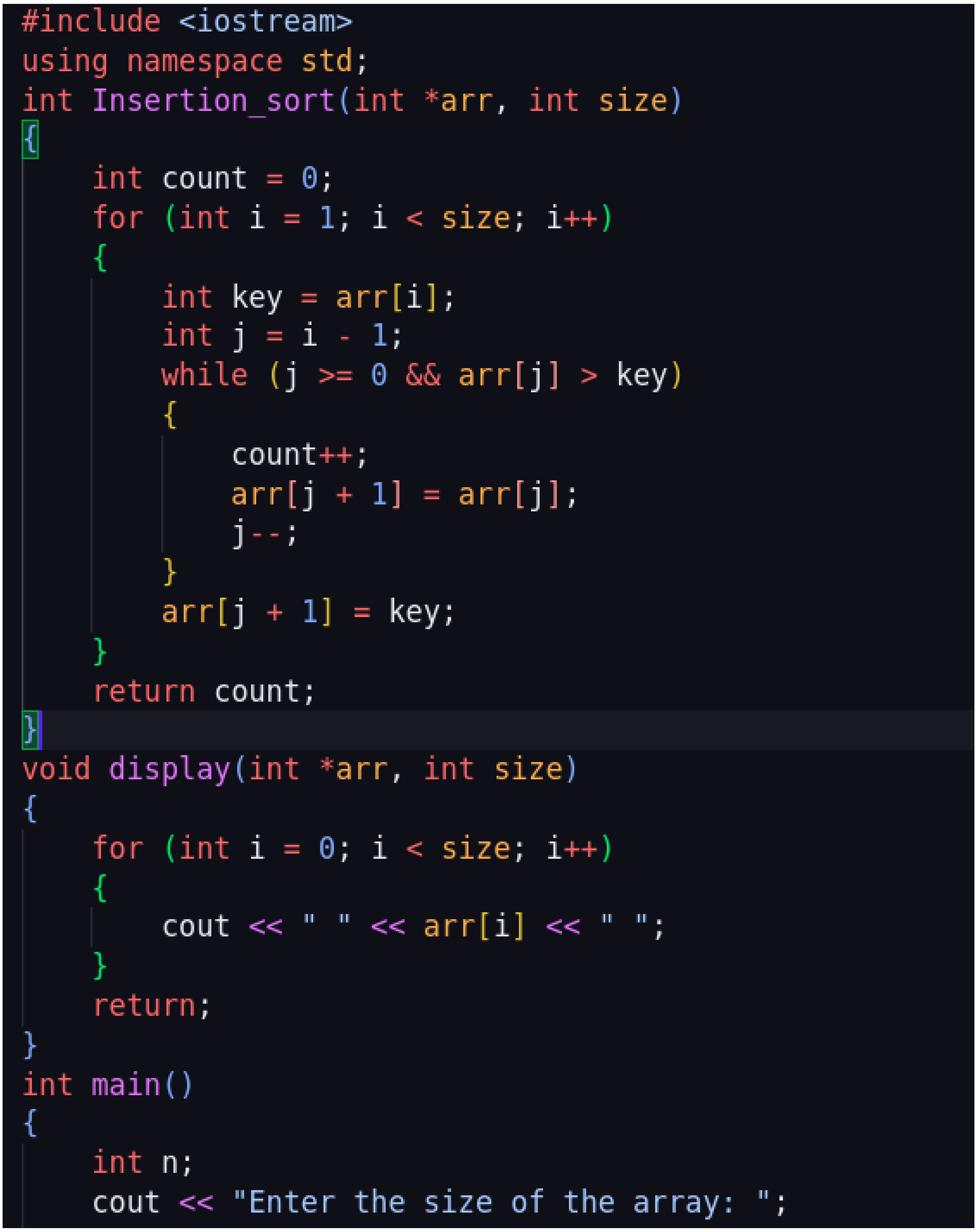
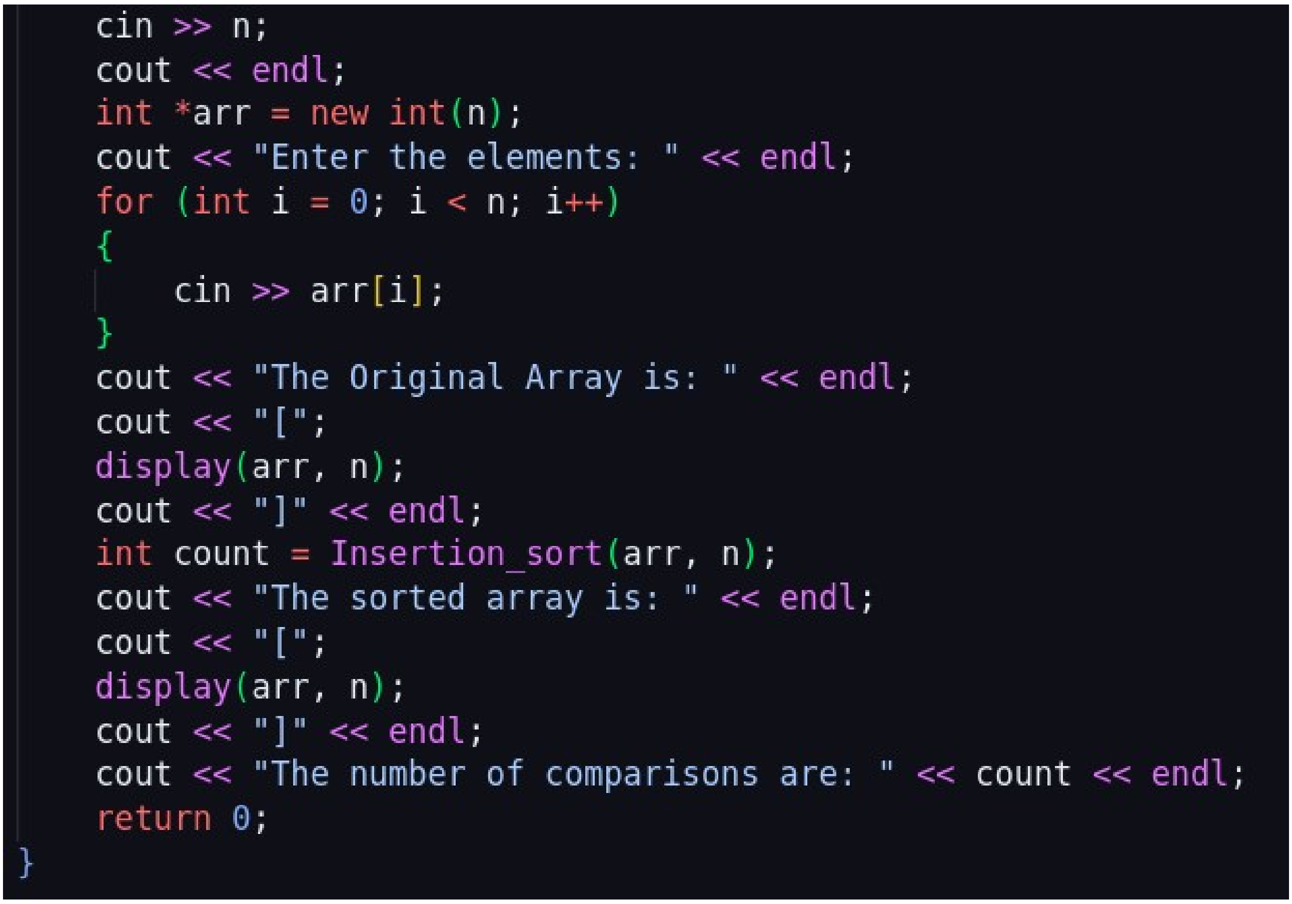
Design and Analysis of Algorithms Practicals

Ques1) Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).

Code:

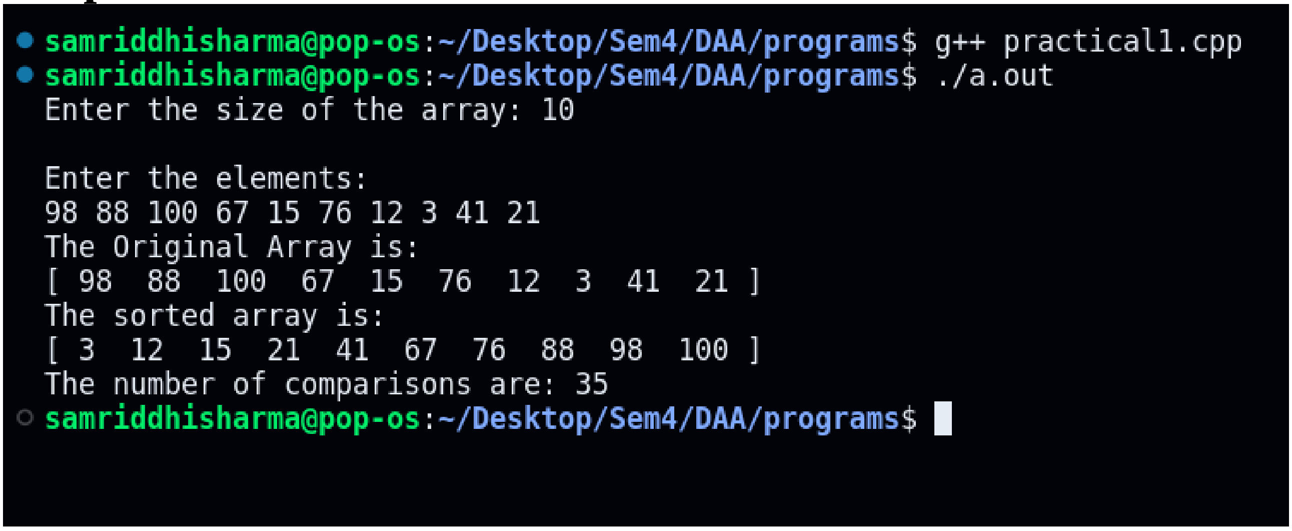


```cpp
#include <iostream>
using namespace std;
int Insertion_sort(int *arr, int size)
{
    int count = 0;
    for (int i = 1; i < size; i++)
    {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            count++;
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
    return count;
}
void display(int *arr, int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << " " << arr[i] << " ";
    }
    return;
}
int main()
{
    int n;
    cout << "Enter the size of the array: ";
```



```
    cin >> n;
    cout << endl;
    int *arr = new int(n);
    cout << "Enter the elements: " << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    cout << "The Original Array is: " << endl;
    cout << "[";
    display(arr, n);
    cout << "]" << endl;
    int count = Insertion_sort(arr, n);
    cout << "The sorted array is: " << endl;
    cout << "[";
    display(arr, n);
    cout << "]" << endl;
    cout << "The number of comparisons are: " << count << endl;
    return 0;
}
```

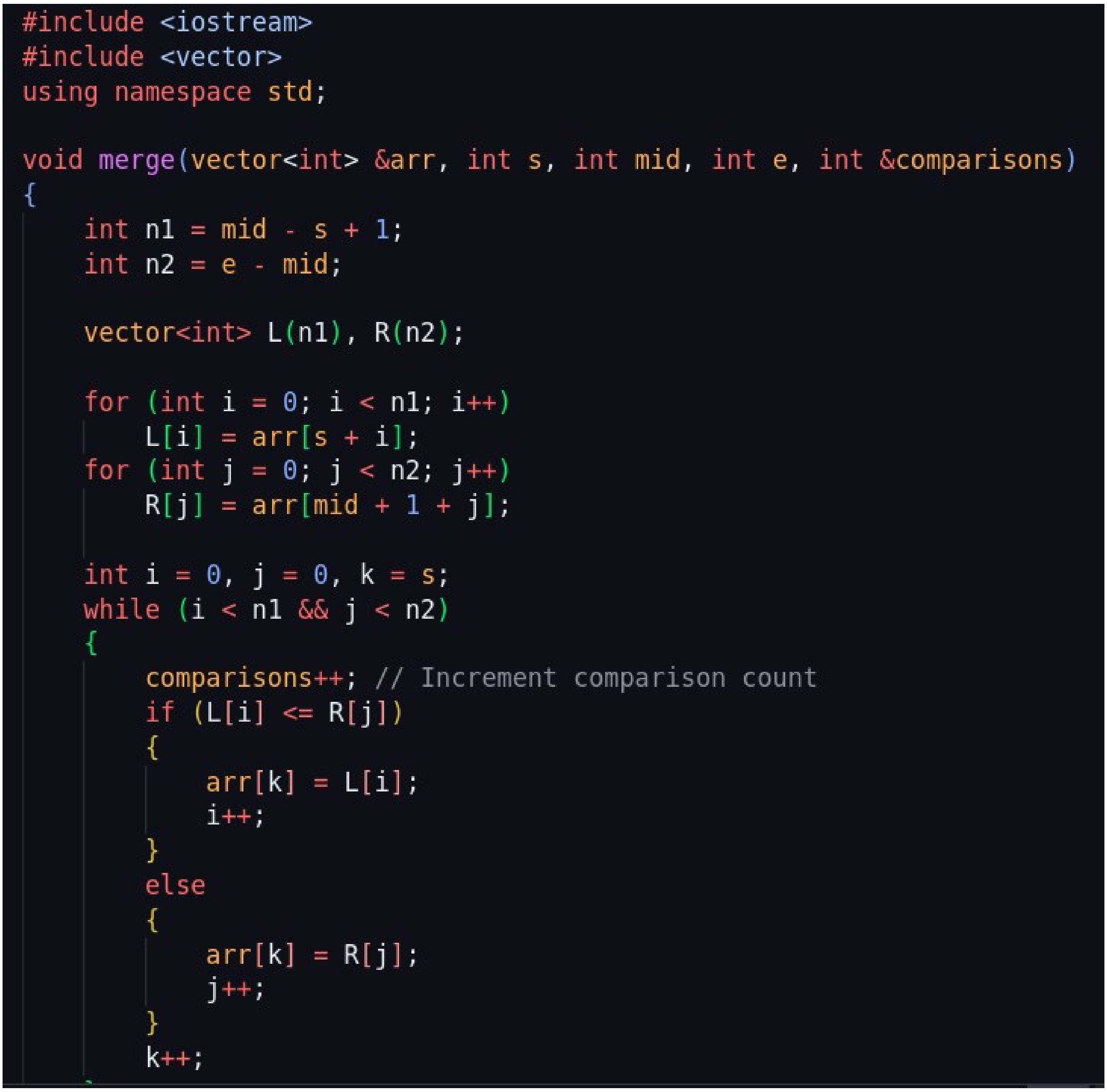
Output:



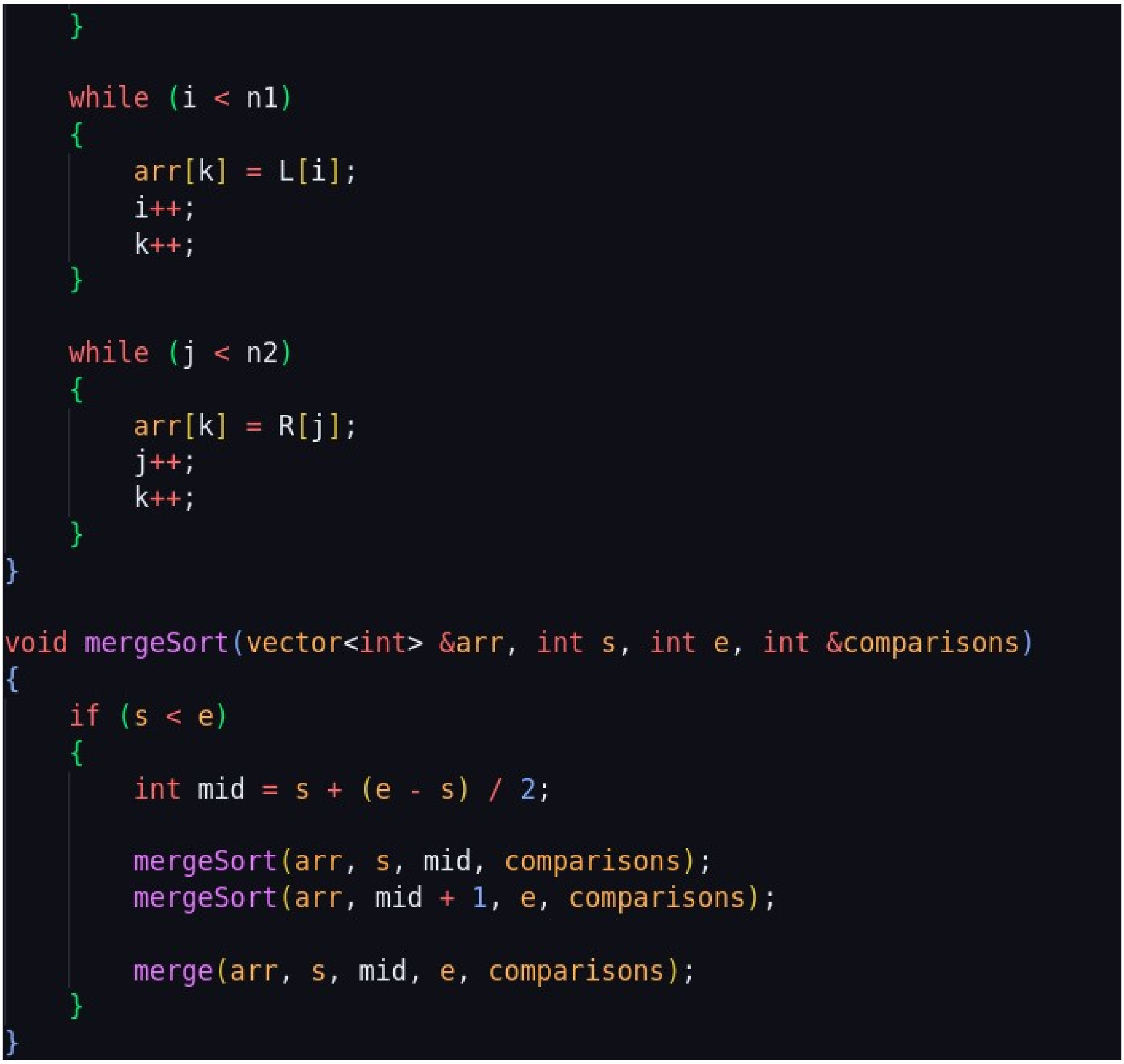
```
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ practical1.cpp
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
  Enter the size of the array: 10

  Enter the elements:
  98 88 100 67 15 76 12 3 41 21
  The Original Array is:
  [ 98   88   100   67   15   76   12   3   41   21 ]
  The sorted array is:
  [ 3   12   15   21   41   67   76   88   98   100 ]
  The number of comparisons are: 35
○ samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

Ques2) Write a program to sort the elements of an array using Merge Sort (The program should report the number of comparisons).

Code:

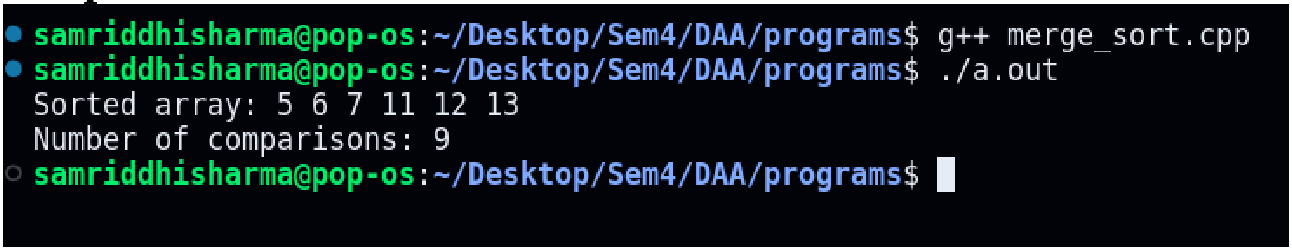


```
#include <iostream>
#include <vector>
using namespace std;

void merge(vector<int> &arr, int s, int mid, int e, int &comparisons)
{
    int n1 = mid - s + 1;
    int n2 = e - mid;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)
        L[i] = arr[s + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = s;
    while (i < n1 && j < n2)
    {
        comparisons++; // Increment comparison count
        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++;
    }

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

void mergeSort(vector<int> &arr, int s, int e, int &comparisons)
{
    if (s < e)
    {
        int mid = s + (e - s) / 2;

        mergeSort(arr, s, mid, comparisons);
        mergeSort(arr, mid + 1, e, comparisons);

        merge(arr, s, mid, e, comparisons);
    }
}
```

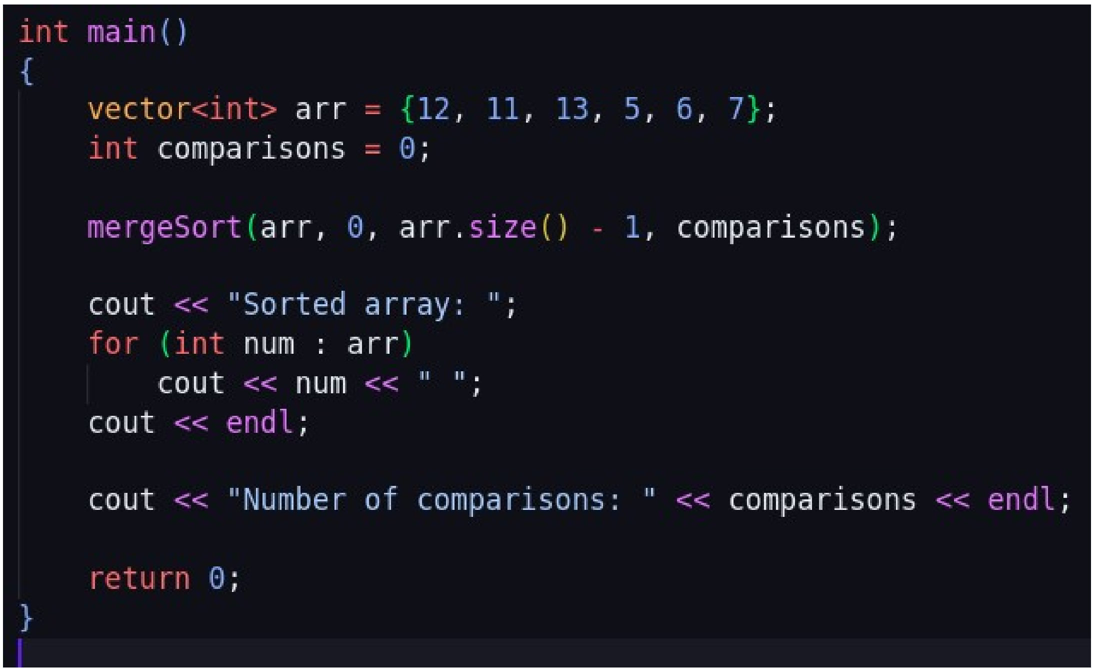
Output:



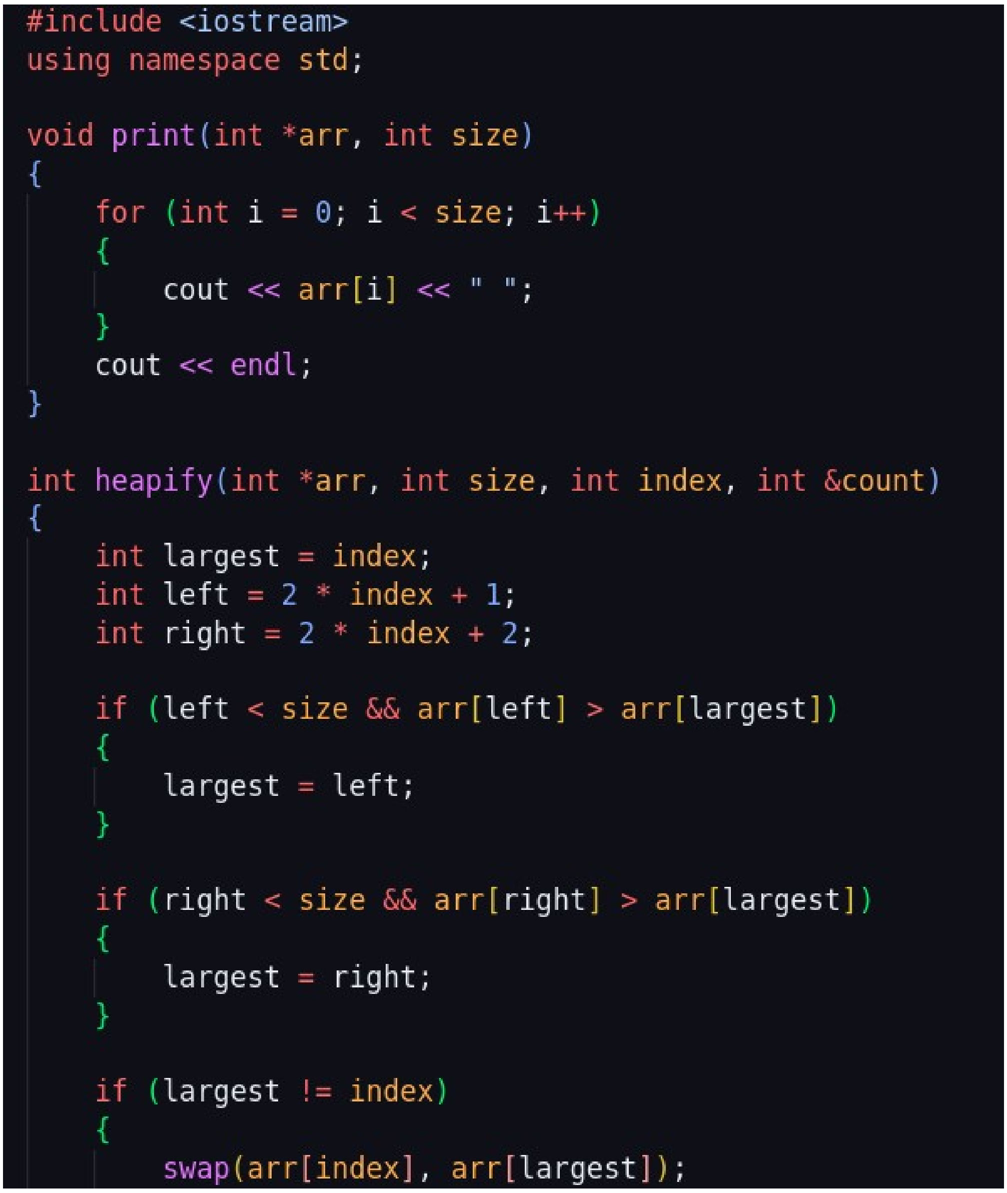
```
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ merge_sort.cpp
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
Sorted array: 5 6 7 11 12 13
Number of comparisons: 9
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

Ques3) Write a program to sort the elements of an array using Heap Sort (The program should report the number of comparisons).

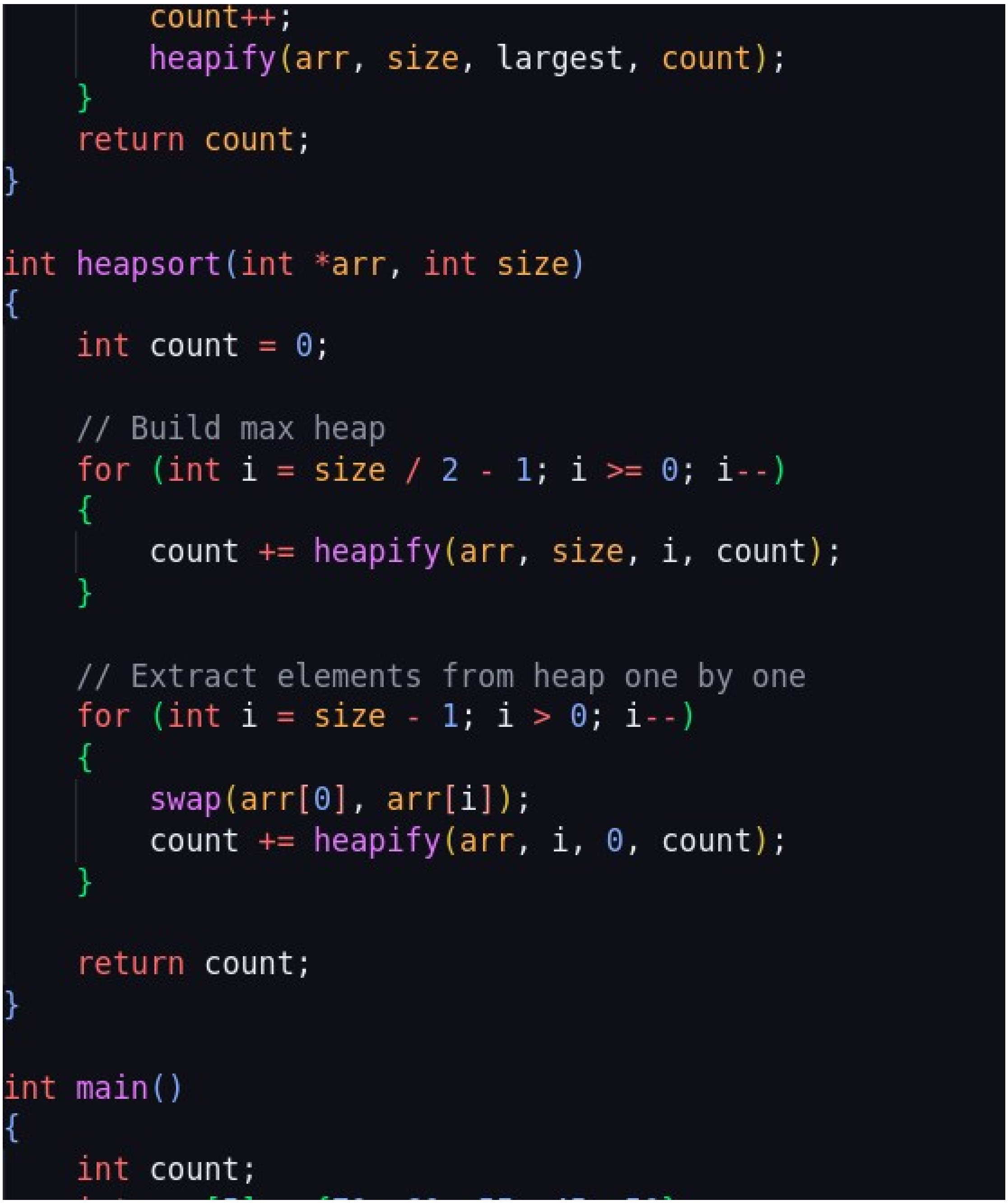
Code:



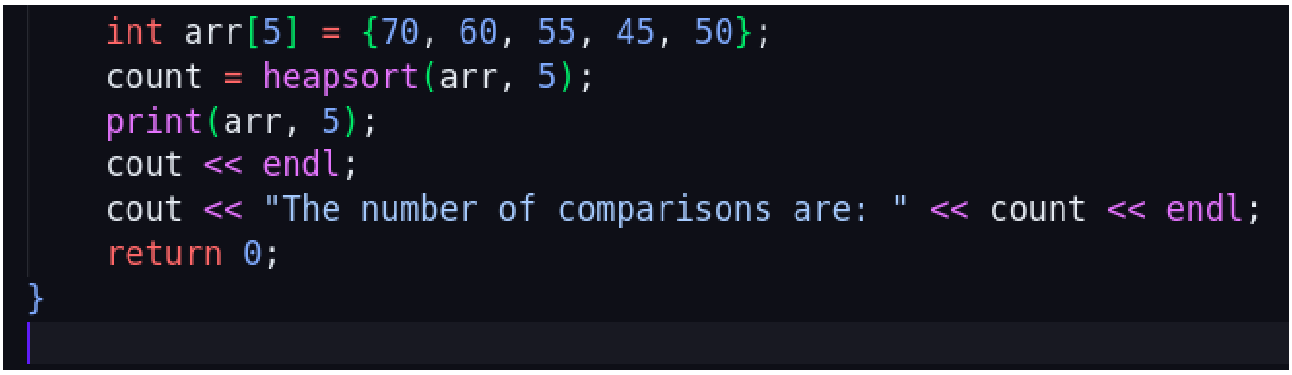
```cpp
int main()
{
    vector<int> arr = {12, 11, 13, 5, 6, 7};
    int comparisons = 0;

    mergeSort(arr, 0, arr.size() - 1, comparisons);

    cout << "Sorted array: ";
    for (int num : arr)
        cout << num << " ";
    cout << endl;

    cout << "Number of comparisons: " << comparisons << endl;

    return 0;
}
```



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

void print(int *arr, int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int heapify(int *arr, int size, int index, int &count)
{
    int largest = index;
    int left = 2 * index + 1;
    int right = 2 * index + 2;

    if (left < size && arr[left] > arr[largest])
    {
        largest = left;
    }

    if (right < size && arr[right] > arr[largest])
    {
        largest = right;
    }

    if (largest != index)
    {
        swap(arr[index], arr[largest]);
```

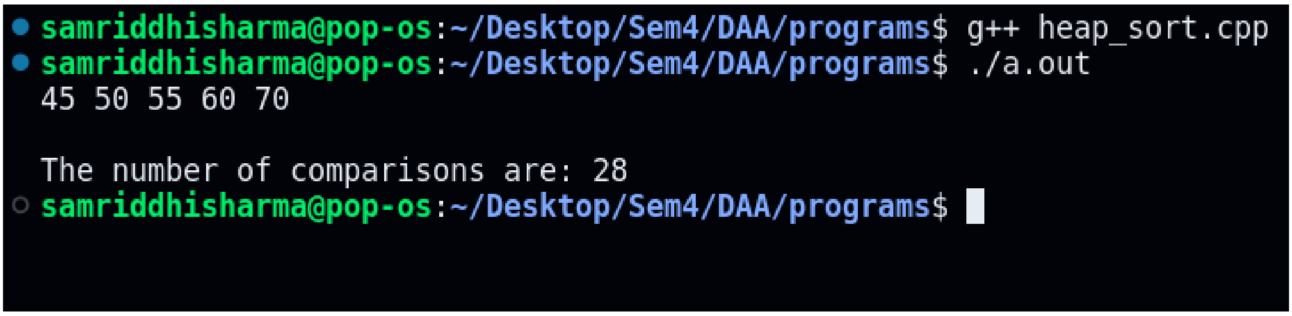


```
        count++;
        heapify(arr, size, largest, count);
    }
    return count;
}

int heapsort(int *arr, int size)
{
    int count = 0;

    // Build max heap
    for (int i = size / 2 - 1; i >= 0; i--)
    {
        count += heapify(arr, size, i, count);
    }

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

    return count;
}

int main()
{
    int count;
```



```
    int arr[5] = {70, 60, 55, 45, 50};
    count = heapsort(arr, 5);
    print(arr, 5);
    cout << endl;
    cout << "The number of comparisons are: " << count << endl;
    return 0;
}
```

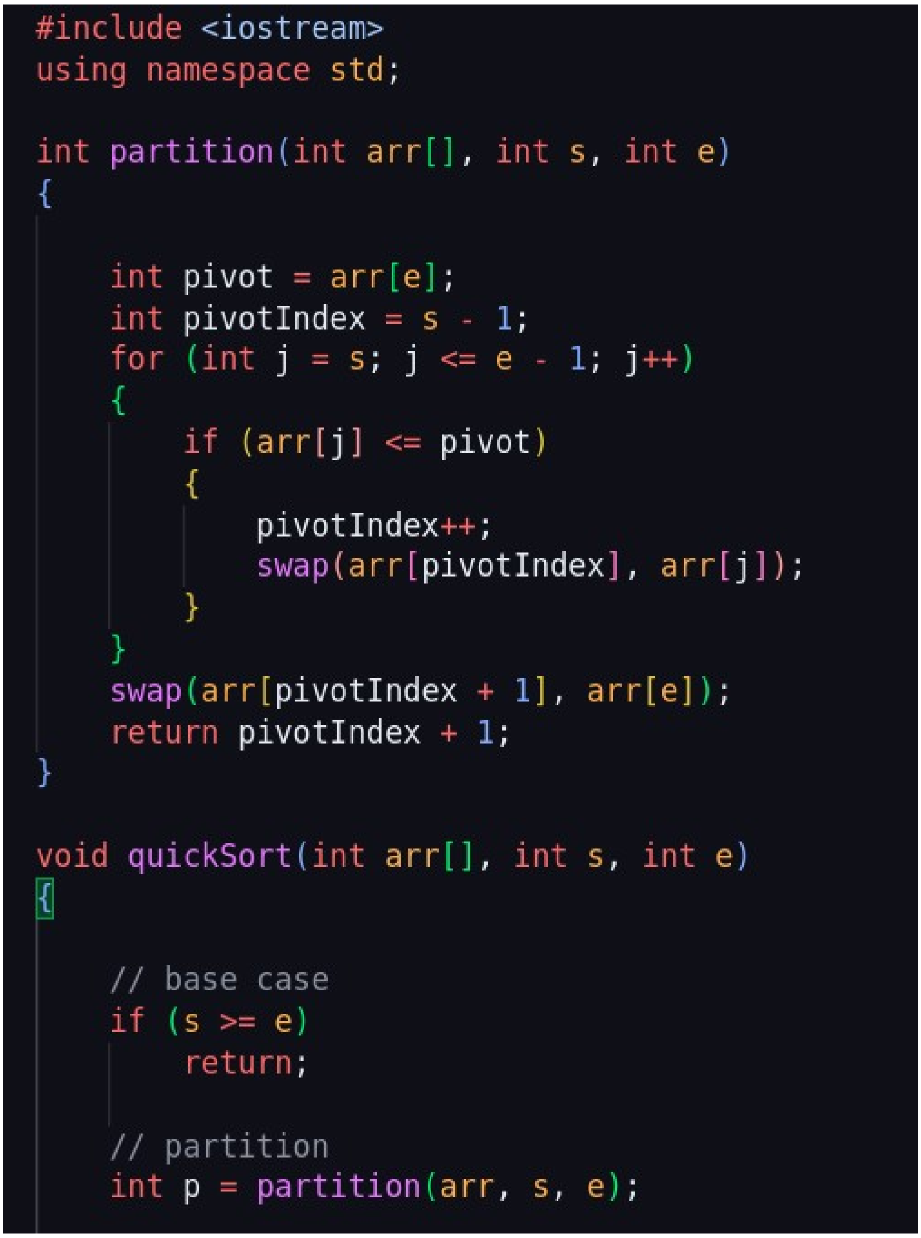
Output:



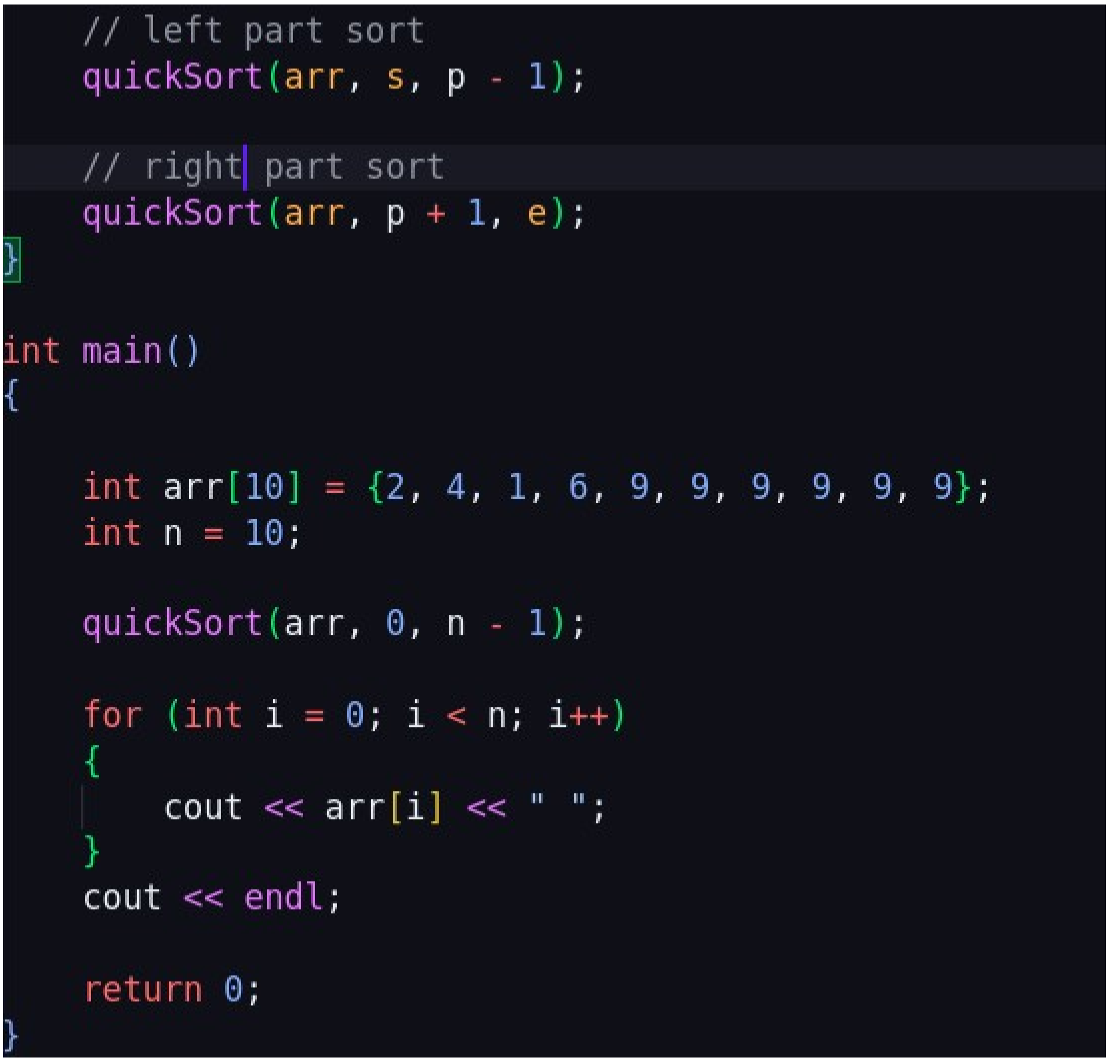
```
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ heap_sort.cpp
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
  45 50 55 60 70

  The number of comparisons are: 28
○ samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

Ques4) Write a program to sort the elements of an array using Quick Sort.

Code:

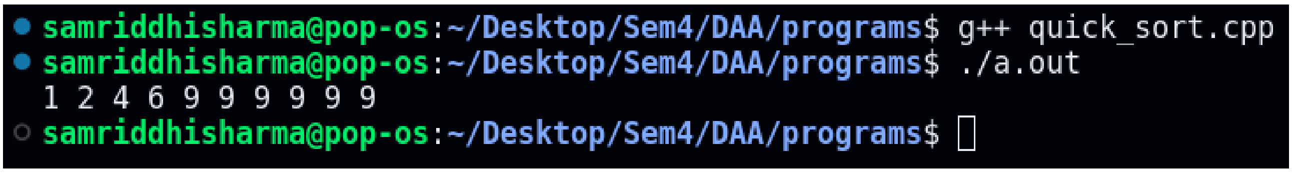


```
#include <iostream>
using namespace std;

int partition(int arr[], int s, int e)
{

    int pivot = arr[e];
    int pivotIndex = s - 1;
    for (int j = s; j <= e - 1; j++)
    {
        if (arr[j] <= pivot)
        {
            pivotIndex++;
            swap(arr[pivotIndex], arr[j]);
        }
    }
    swap(arr[pivotIndex + 1], arr[e]);
    return pivotIndex + 1;
}

void quickSort(int arr[], int s, int e)
{

    // base case
    if (s >= e)
        return;

    // partition
    int p = partition(arr, s, e);
```



```
    // left part sort
    quickSort(arr, s, p - 1);

    // right part sort
    quickSort(arr, p + 1, e);
}

int main()
{

    int arr[10] = {2, 4, 1, 6, 9, 9, 9, 9, 9, 9};
    int n = 10;

    quickSort(arr, 0, n - 1);

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

    return 0;
}
```

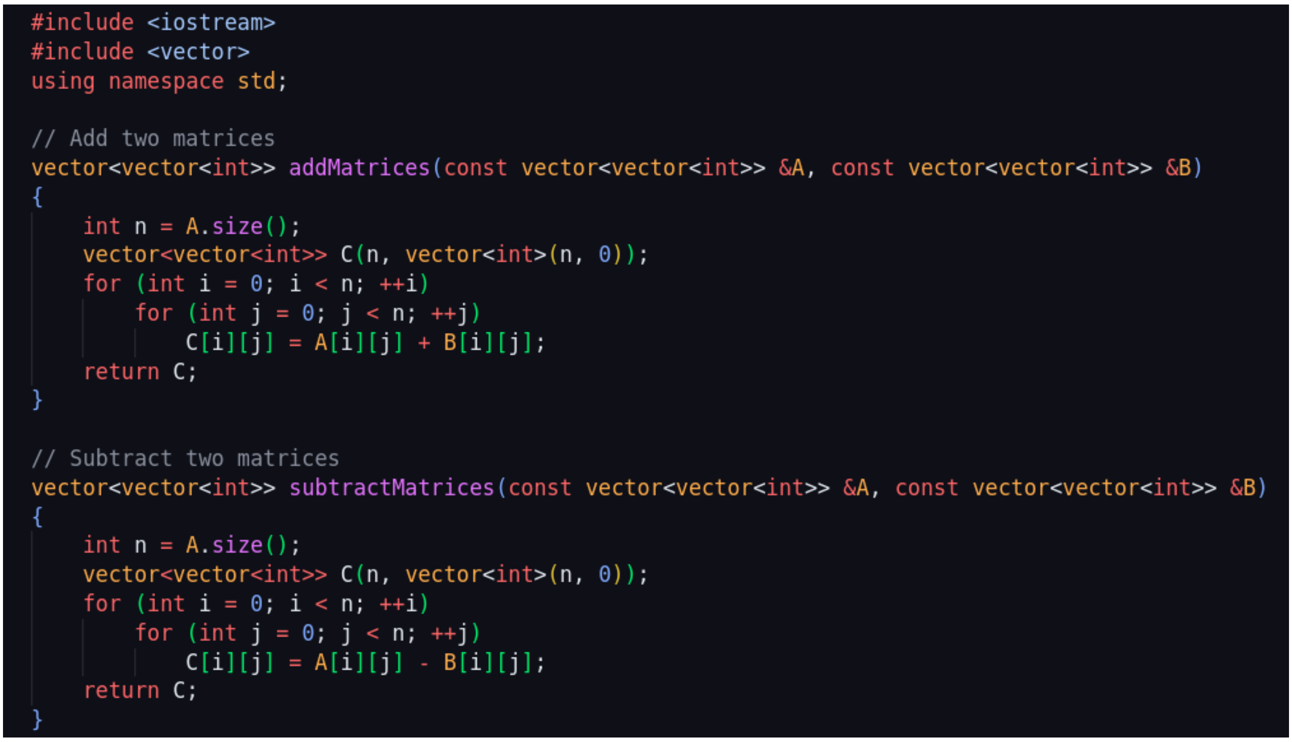
Output:



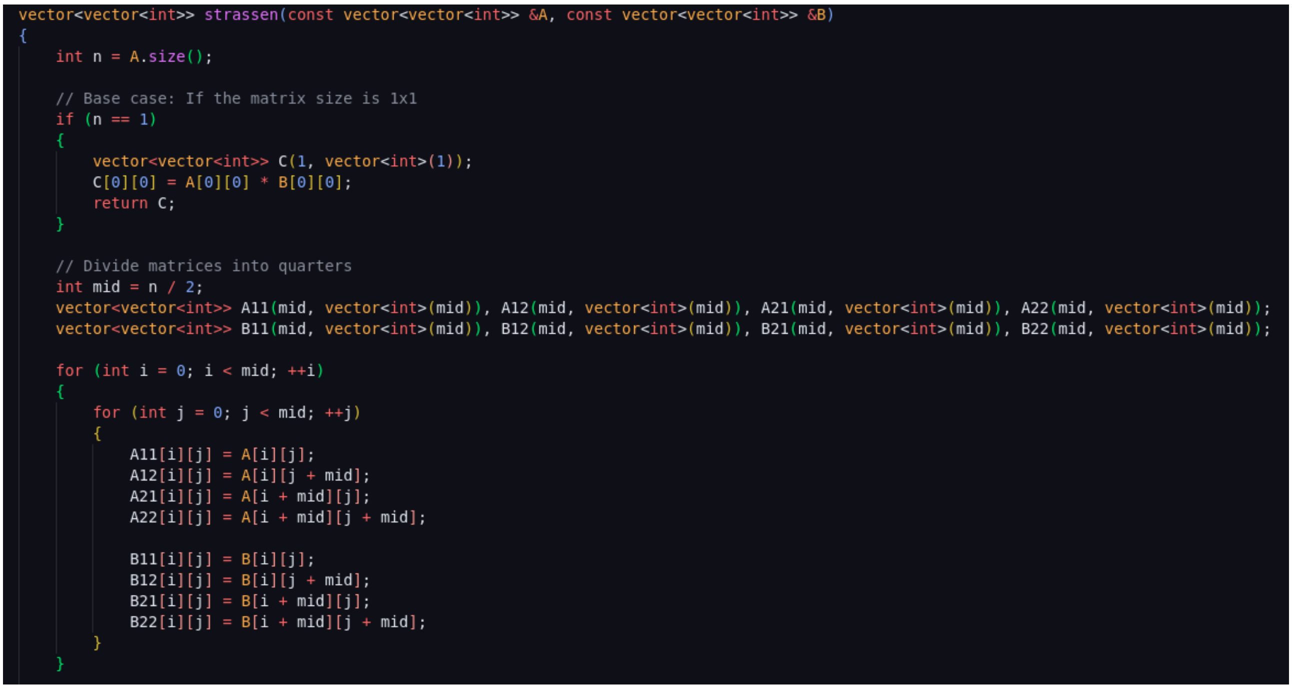
```
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ quick_sort.cpp
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
  1 2 4 6 9 9 9 9 9 9
○ samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

Ques5) Write a program to multiply two matrices using the Strassen's algorithm for matrix multiplication.

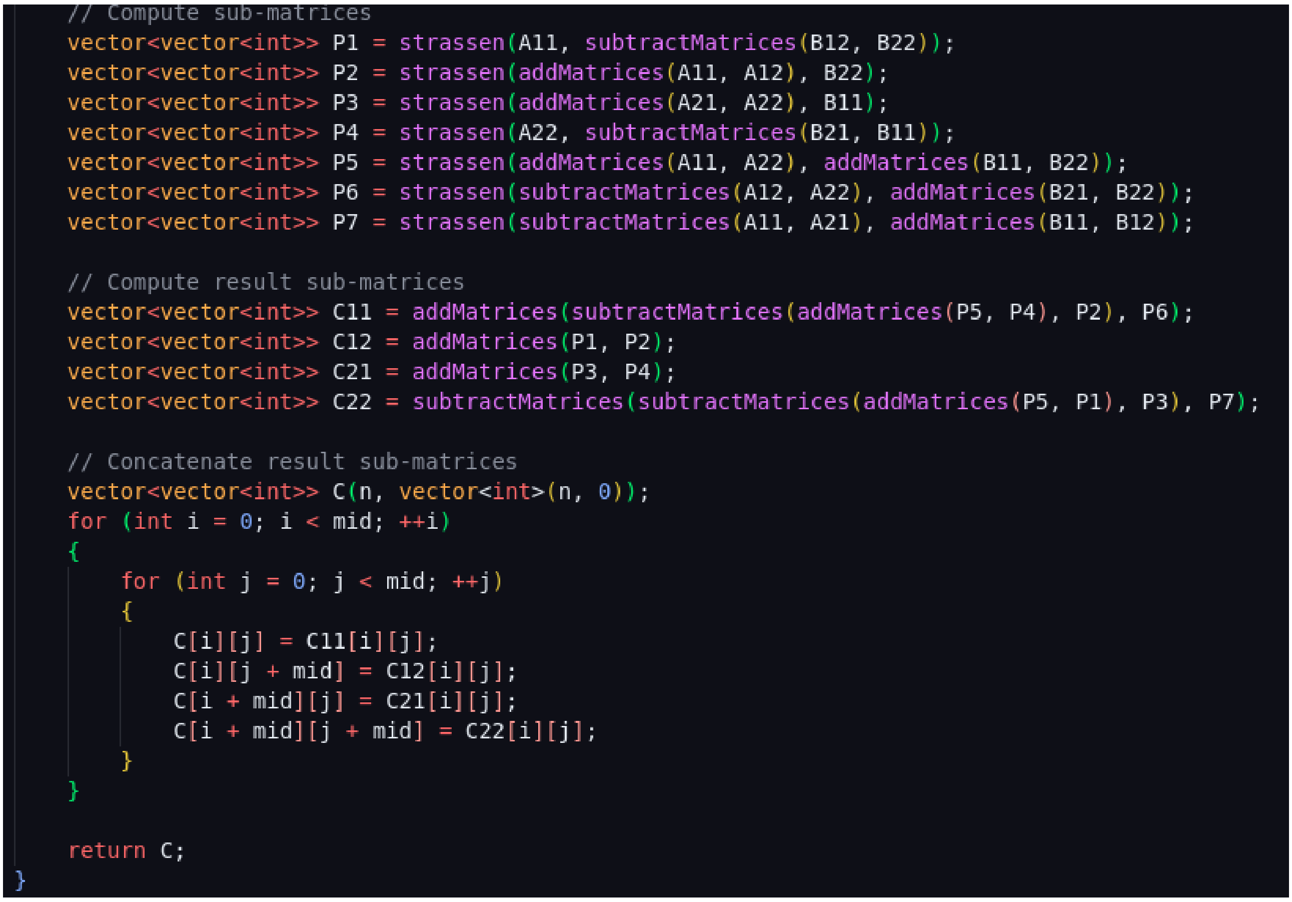
Code:



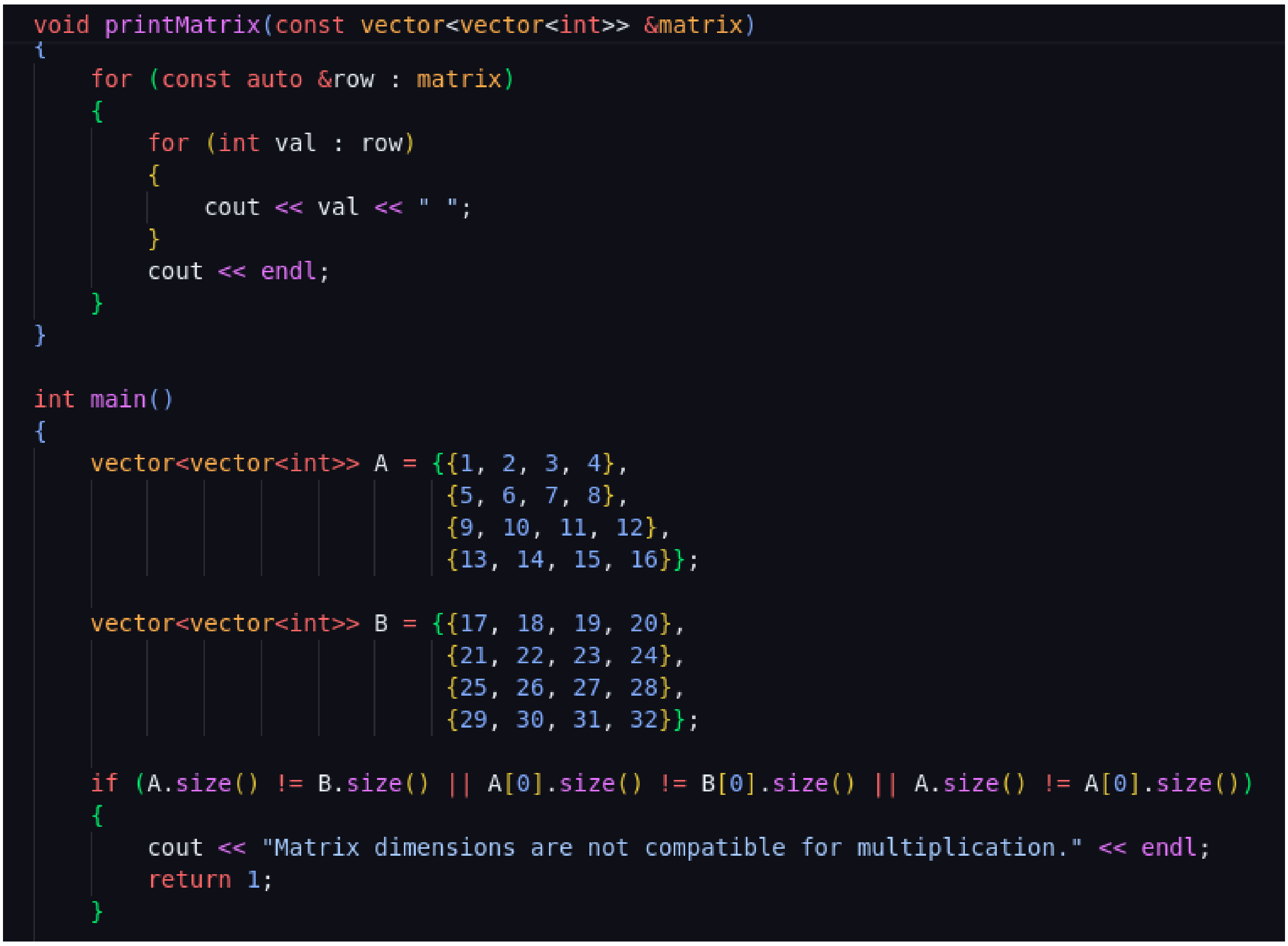
```
#include <iostream>
#include <vector>
using namespace std;

// Add two matrices
vector<vector<int>> addMatrices(const vector<vector<int>> &A, const vector<vector<int>> &B)
{
    int n = A.size();
    vector<vector<int>> C(n, vector<int>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}

// Subtract two matrices
vector<vector<int>> subtractMatrices(const vector<vector<int>> &A, const vector<vector<int>> &B)
{
    int n = A.size();
    vector<vector<int>> C(n, vector<int>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            C[i][j] = A[i][j] - B[i][j];
    return C;
}
```



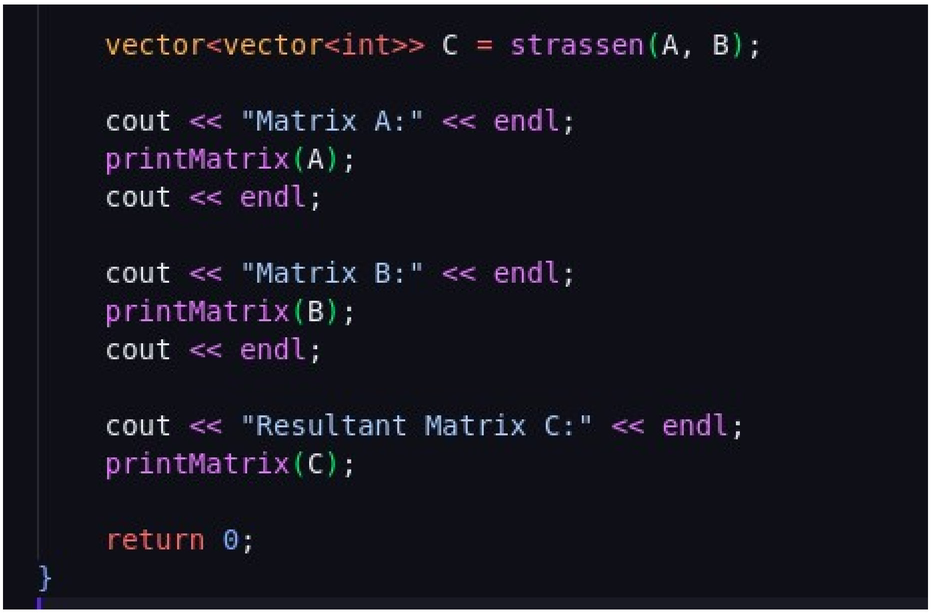
```
vector<vector<int>> strassen(const vector<vector<int>> &A, const vector<vector<int>> &B)
{
    int n = A.size();

    // Base case: If the matrix size is 1x1
    if (n == 1)
    {
        vector<vector<int>> C(1, vector<int>(1));
        C[0][0] = A[0][0] * B[0][0];
        return C;
    }

    // Divide matrices into quarters
    int mid = n / 2;
    vector<vector<int>> A11(mid, vector<int>(mid)), A12(mid, vector<int>(mid)), A21(mid, vector<int>(mid)), A22(mid, vector<int>(mid));
    vector<vector<int>> B11(mid, vector<int>(mid)), B12(mid, vector<int>(mid)), B21(mid, vector<int>(mid)), B22(mid, vector<int>(mid));

    for (int i = 0; i < mid; ++i)
    {
        for (int j = 0; j < mid; ++j)
        {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + mid];
            A21[i][j] = A[i + mid][j];
            A22[i][j] = A[i + mid][j + mid];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + mid];
            B21[i][j] = B[i + mid][j];
            B22[i][j] = B[i + mid][j + mid];
        }
    }
```



```
    // Compute sub-matrices
    vector<vector<int>> P1 = strassen(A11, subtractMatrices(B12, B22));
    vector<vector<int>> P2 = strassen(addMatrices(A11, A12), B22);
    vector<vector<int>> P3 = strassen(addMatrices(A21, A22), B11);
    vector<vector<int>> P4 = strassen(A22, subtractMatrices(B21, B11));
    vector<vector<int>> P5 = strassen(addMatrices(A11, A22), addMatrices(B11, B22));
    vector<vector<int>> P6 = strassen(subtractMatrices(A12, A22), addMatrices(B21, B22));
    vector<vector<int>> P7 = strassen(subtractMatrices(A11, A21), addMatrices(B11, B12));

    // Compute result sub-matrices
    vector<vector<int>> C11 = addMatrices(subtractMatrices(addMatrices(P5, P4), P2), P6);
    vector<vector<int>> C12 = addMatrices(P1, P2);
    vector<vector<int>> C21 = addMatrices(P3, P4);
    vector<vector<int>> C22 = subtractMatrices(subtractMatrices(addMatrices(P5, P1), P3), P7);

    // Concatenate result sub-matrices
    vector<vector<int>> C(n, vector<int>(n, 0));
    for (int i = 0; i < mid; ++i)
    {
        for (int j = 0; j < mid; ++j)
        {
            C[i][j] = C11[i][j];
            C[i][j + mid] = C12[i][j];
            C[i + mid][j] = C21[i][j];
            C[i + mid][j + mid] = C22[i][j];
        }
    }

    return C;
}
```

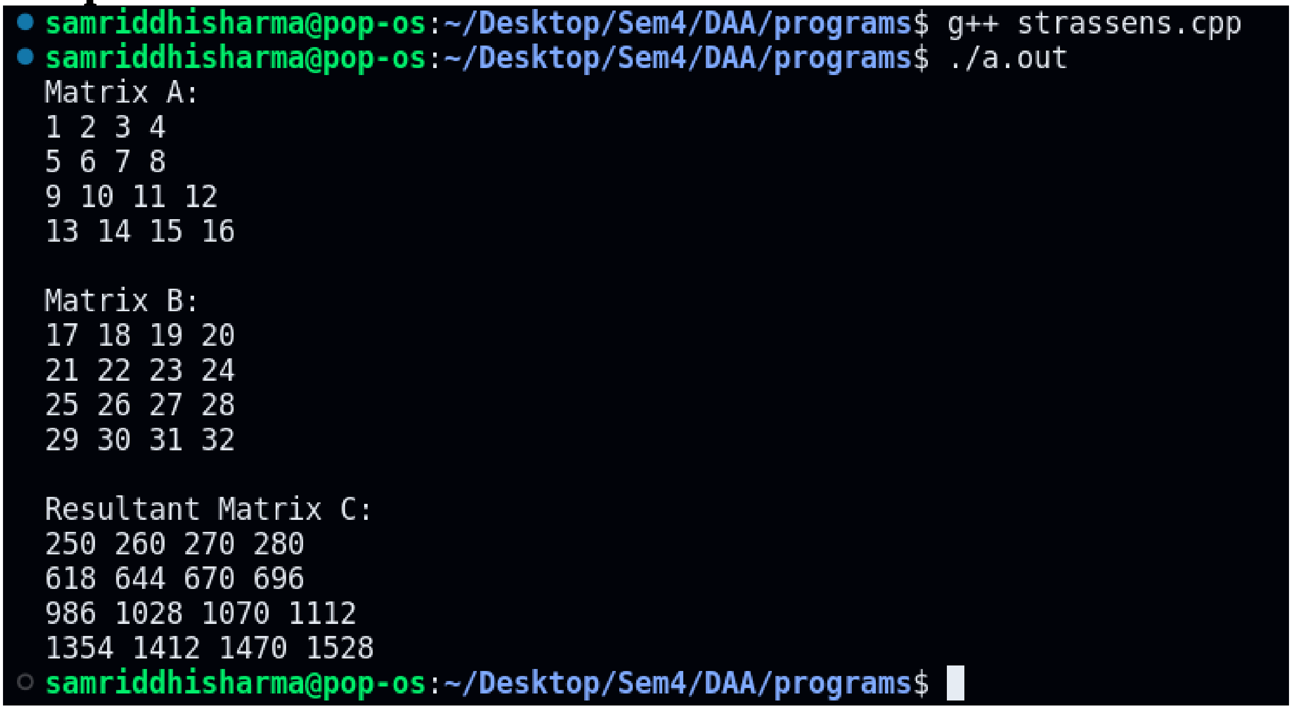


```
void printMatrix(const vector<vector<int>> &matrix)
{
    for (const auto &row : matrix)
    {
        for (int val : row)
        {
            cout << val << " ";
        }
        cout << endl;
    }
}

int main()
{
    vector<vector<int>> A = {{1, 2, 3, 4},
                             {5, 6, 7, 8},
                             {9, 10, 11, 12},
                             {13, 14, 15, 16}};

    vector<vector<int>> B = {{17, 18, 19, 20},
                             {21, 22, 23, 24},
                             {25, 26, 27, 28},
                             {29, 30, 31, 32}};

    if (A.size() != B.size() || A[0].size() != B[0].size() || A.size() != A[0].size())
    {
        cout << "Matrix dimensions are not compatible for multiplication." << endl;
        return 1;
    }
```



```
    vector<vector<int>> C = strassen(A, B);

    cout << "Matrix A:" << endl;
    printMatrix(A);
    cout << endl;

    cout << "Matrix B:" << endl;
    printMatrix(B);
    cout << endl;

    cout << "Resultant Matrix C:" << endl;
    printMatrix(C);

    return 0;
}
```

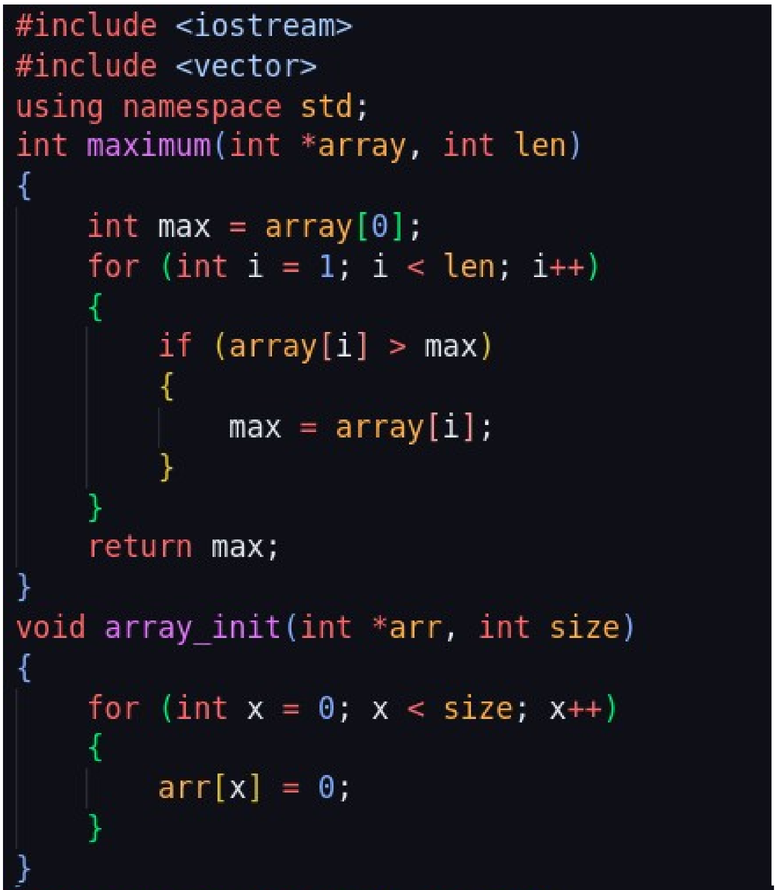
Output:



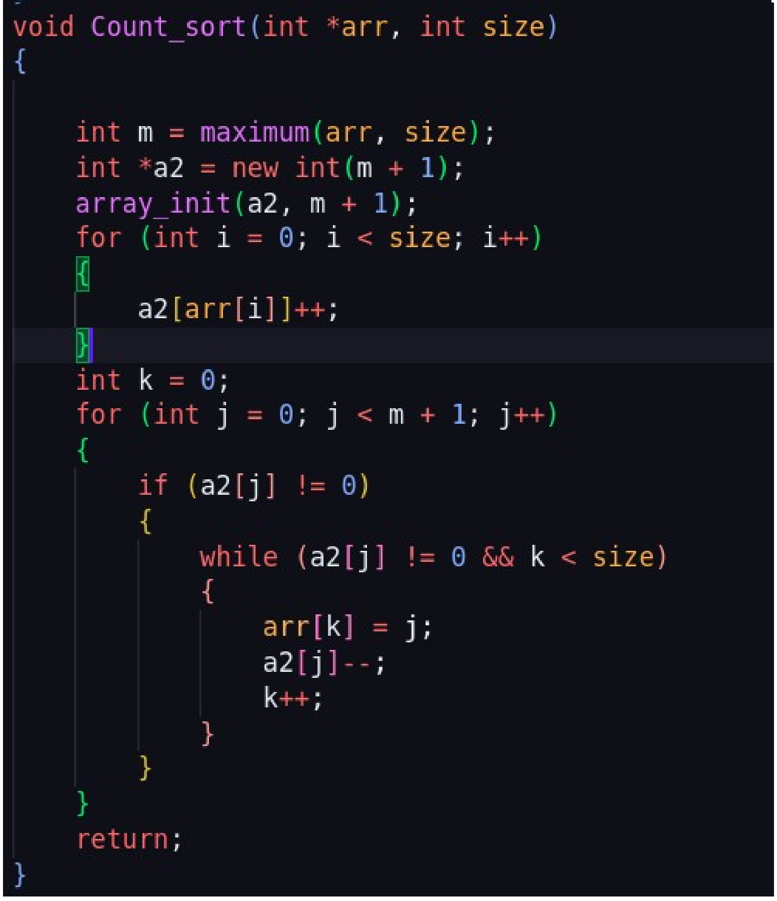
```
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ strassens.cpp
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
  Matrix A:
  1 2 3 4
  5 6 7 8
  9 10 11 12
  13 14 15 16

  Matrix B:
  17 18 19 20
  21 22 23 24
  25 26 27 28
  29 30 31 32

  Resultant Matrix C:
  250 260 270 280
  618 644 670 696
  986 1028 1070 1112
  1354 1412 1470 1528
○ samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

Ques6) Write a program to sort the elements of an array using Count Sort.

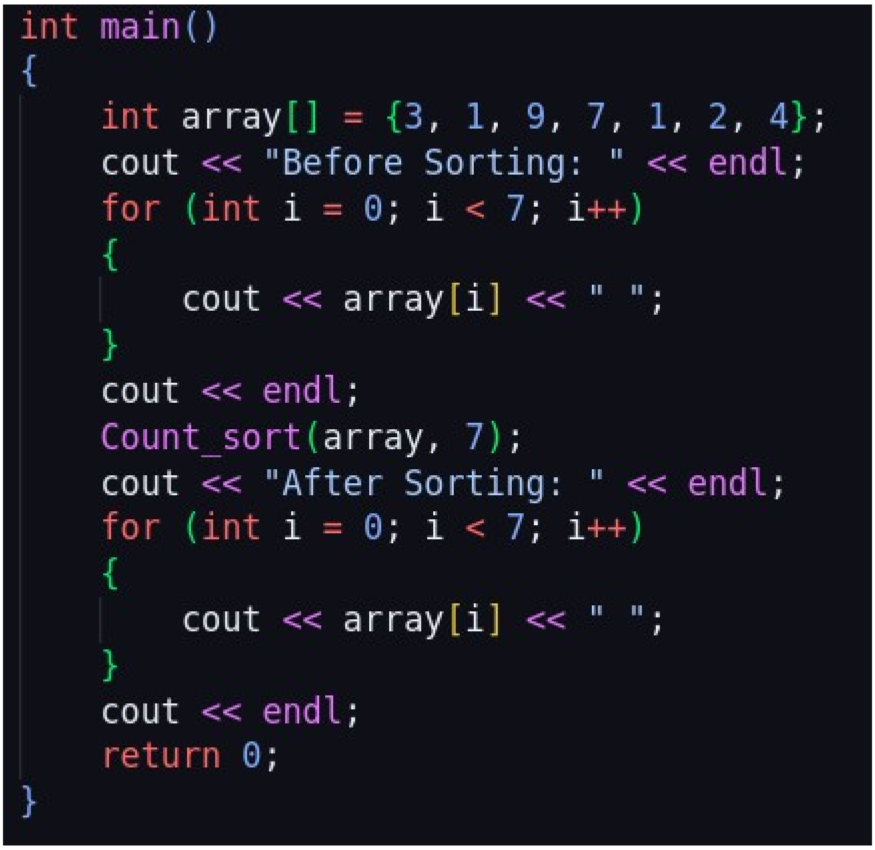
Code:



```cpp
#include <iostream>
#include <vector>
using namespace std;
int maximum(int *array, int len)
{
    int max = array[0];
    for (int i = 1; i < len; i++)
    {
        if (array[i] > max)
        {
            max = array[i];
        }
    }
    return max;
}
void array_init(int *arr, int size)
{
    for (int x = 0; x < size; x++)
    {
        arr[x] = 0;
    }
}
```

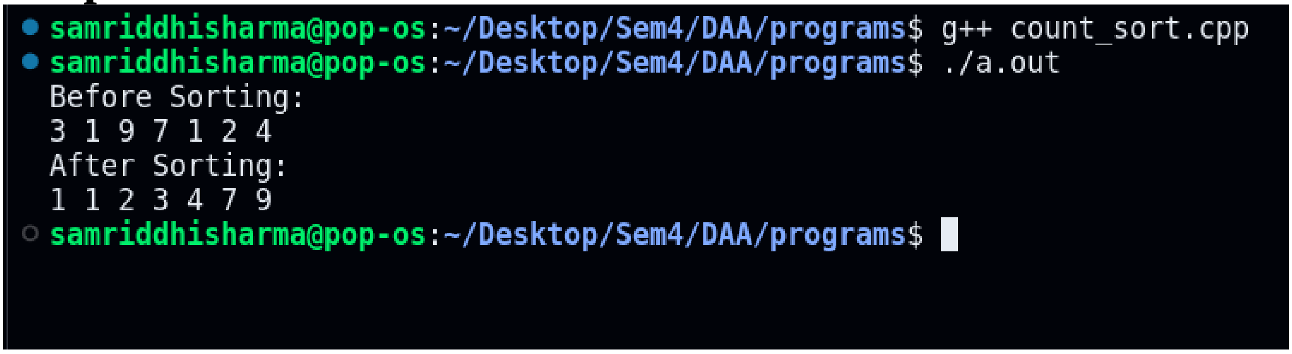


```cpp
void Count_sort(int *arr, int size)
{

    int m = maximum(arr, size);
    int *a2 = new int(m + 1);
    array_init(a2, m + 1);
    for (int i = 0; i < size; i++)
    {
        a2[arr[i]]++;
    }
    int k = 0;
    for (int j = 0; j < m + 1; j++)
    {
        if (a2[j] != 0)
        {
            while (a2[j] != 0 && k < size)
            {
                arr[k] = j;
                a2[j]--;
                k++;
            }
        }
    }
    return;
}
```



```
int main()
{
    int array[] = {3, 1, 9, 7, 1, 2, 4};
    cout << "Before Sorting: " << endl;
    for (int i = 0; i < 7; i++)
    {
        cout << array[i] << " ";
    }
    cout << endl;
    Count_sort(array, 7);
    cout << "After Sorting: " << endl;
    for (int i = 0; i < 7; i++)
    {
        cout << array[i] << " ";
    }
    cout << endl;
    return 0;
}
```

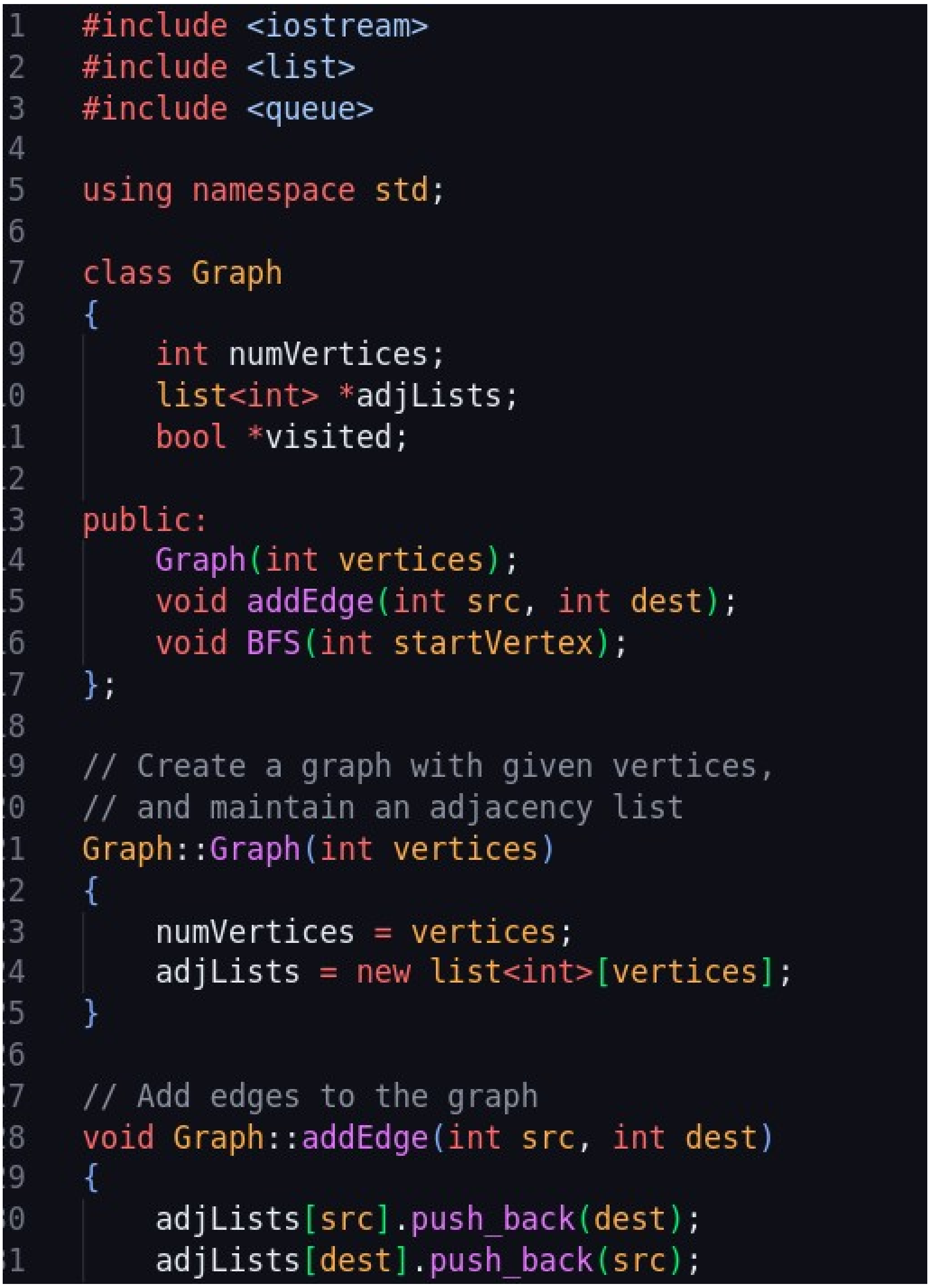
Output:



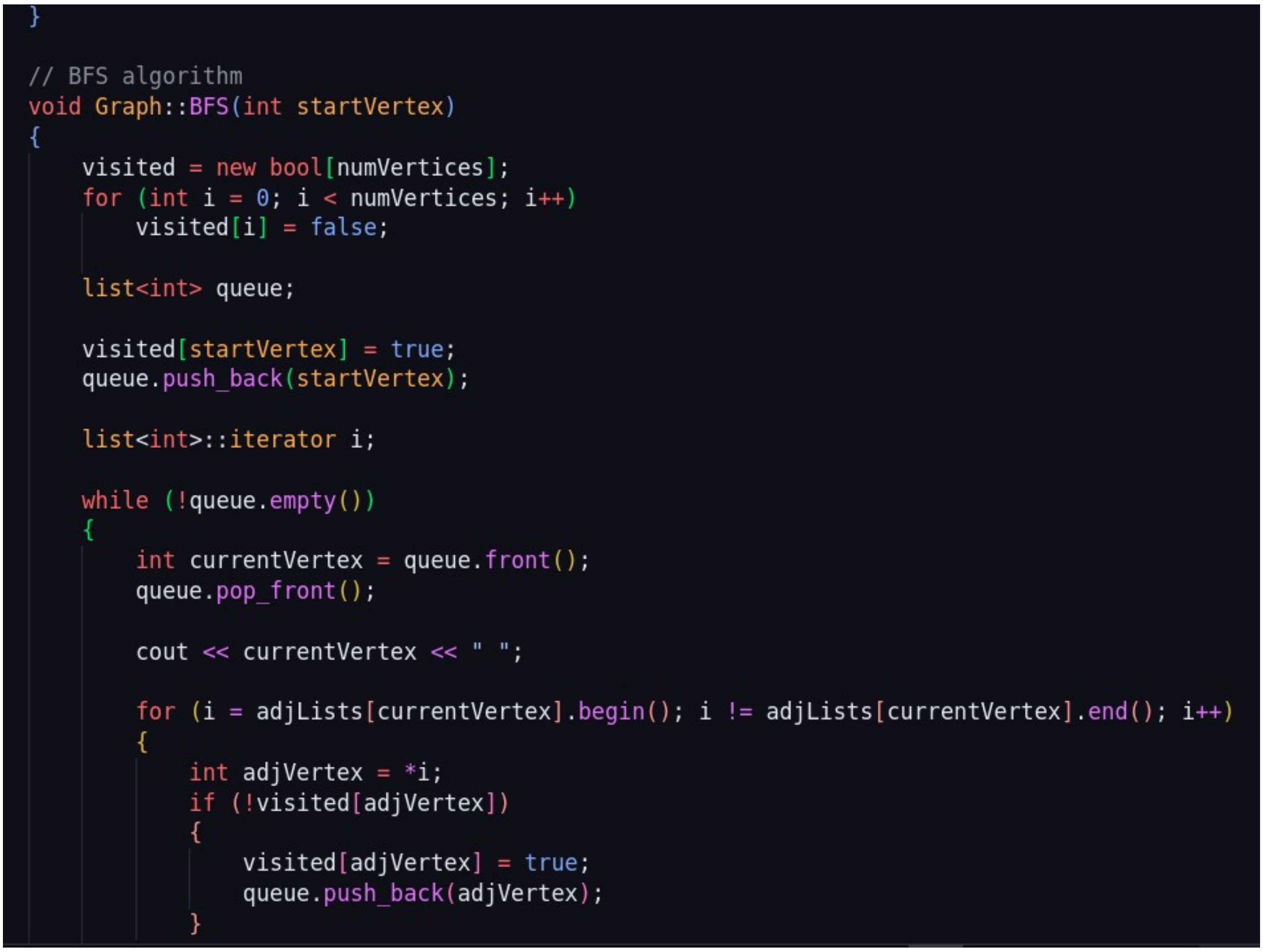
```
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ count_sort.cpp
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
  Before Sorting:
  3 1 9 7 1 2 4
  After Sorting:
  1 1 2 3 4 7 9
○ samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

Ques7) Display the data stored in a given graph using the Breadth-First Search algorithm.

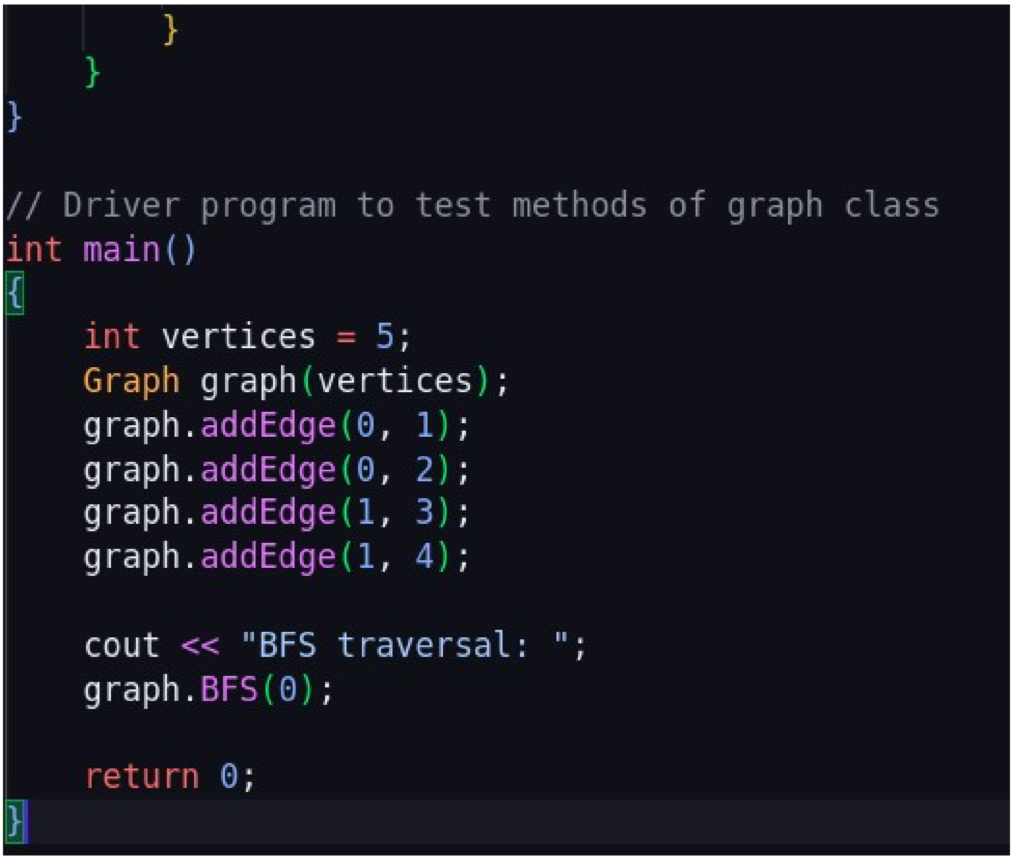
Code:



```cpp
#include <iostream>
#include <list>
#include <queue>

using namespace std;

class Graph
{
    int numVertices;
    list<int> *adjLists;
    bool *visited;

public:
    Graph(int vertices);
    void addEdge(int src, int dest);
    void BFS(int startVertex);
};

// Create a graph with given vertices,
// and maintain an adjacency list
Graph::Graph(int vertices)
{
    numVertices = vertices;
    adjLists = new list<int>[vertices];
}

// Add edges to the graph
void Graph::addEdge(int src, int dest)
{
    adjLists[src].push_back(dest);
    adjLists[dest].push_back(src);
```

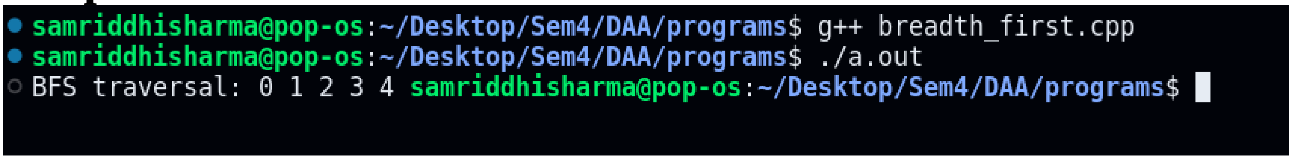


```
}

// BFS algorithm
void Graph::BFS(int startVertex)
{
    visited = new bool[numVertices];
    for (int i = 0; i < numVertices; i++)
        visited[i] = false;

    list<int> queue;

    visited[startVertex] = true;
    queue.push_back(startVertex);

    list<int>::iterator i;

    while (!queue.empty())
    {
        int currentVertex = queue.front();
        queue.pop_front();

        cout << currentVertex << " ";

        for (i = adjLists[currentVertex].begin(); i != adjLists[currentVertex].end(); i++)
        {
            int adjVertex = *i;
            if (!visited[adjVertex])
            {
                visited[adjVertex] = true;
                queue.push_back(adjVertex);
            }
```



```
        }
    }
}

// Driver program to test methods of graph class
int main()
{
    int vertices = 5;
    Graph graph(vertices);
    graph.addEdge(0, 1);
    graph.addEdge(0, 2);
    graph.addEdge(1, 3);
    graph.addEdge(1, 4);

    cout << "BFS traversal: ";
    graph.BFS(0);

    return 0;
}
```

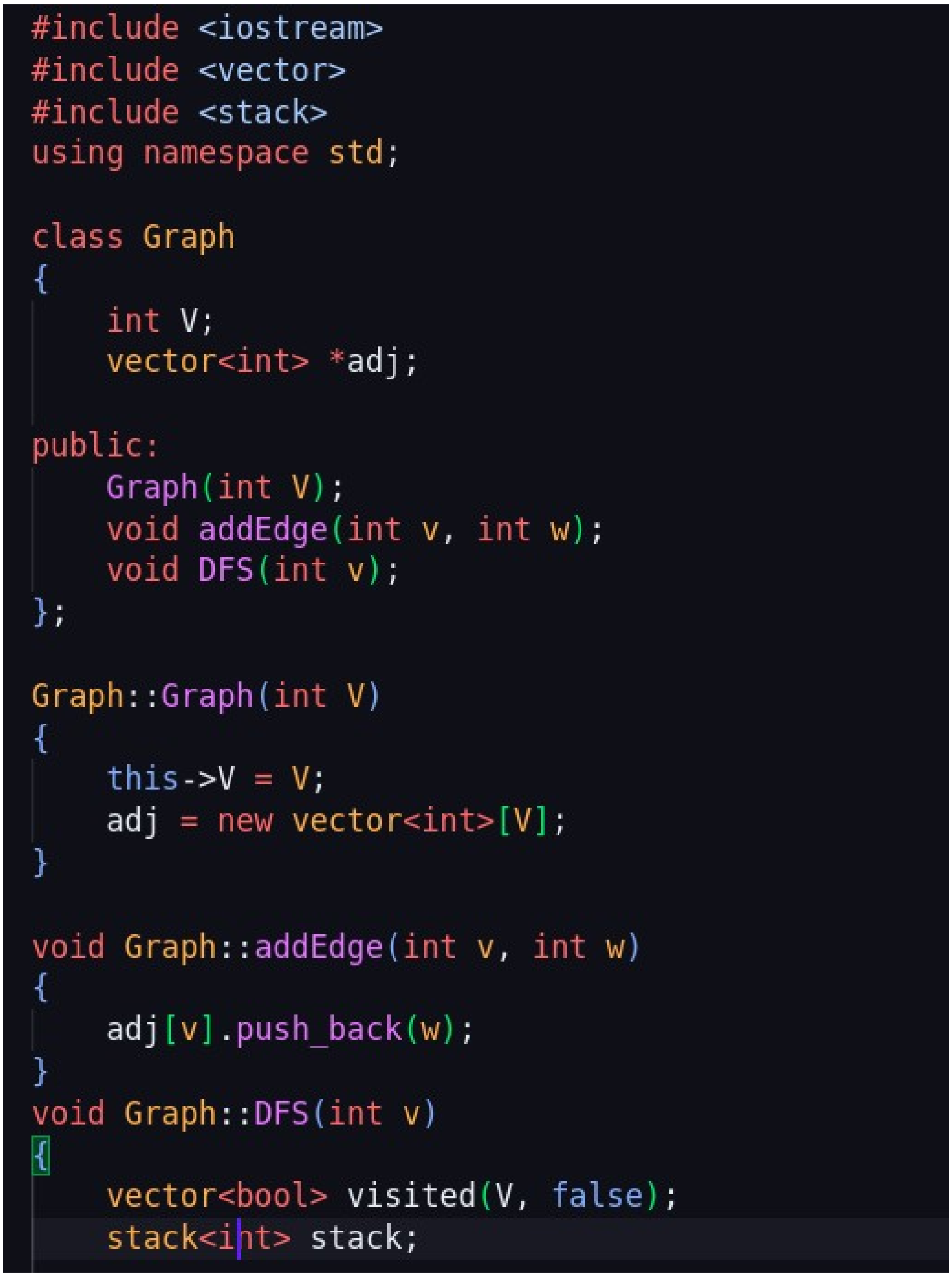
Output:



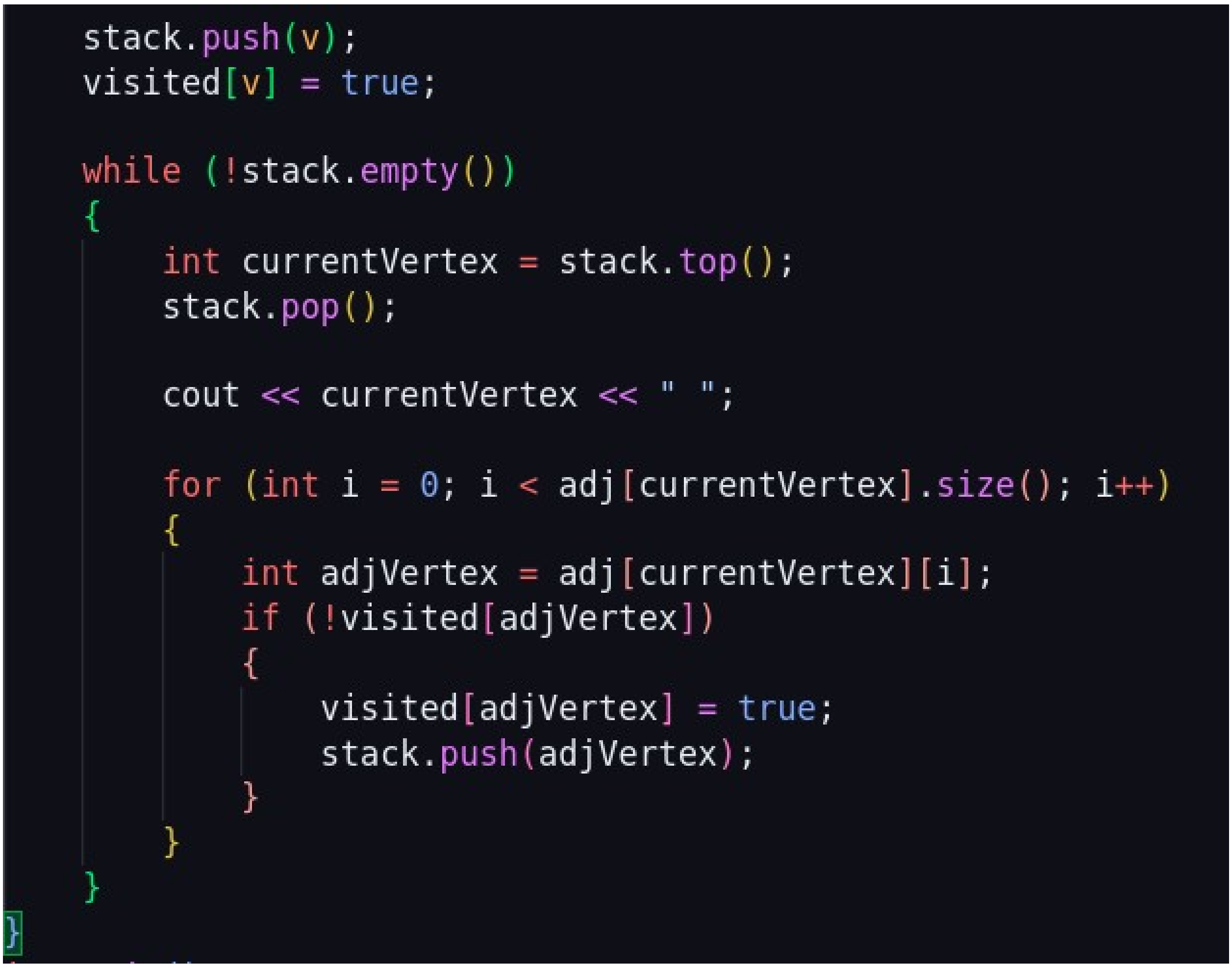
```
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ breadth_first.cpp
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
○ BFS traversal: 0 1 2 3 4 samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ █
```

Ques8) Display the data stored in a given graph using the Depth-First Search algorithm.

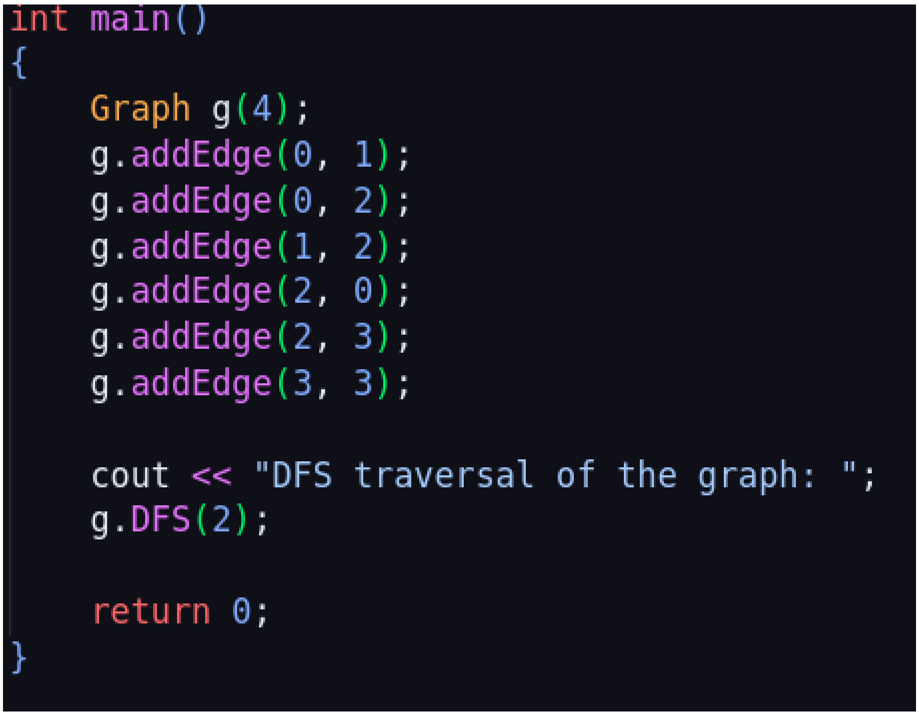
Code:



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

class Graph
{
    int V;
    vector<int> *adj;

public:
    Graph(int V);
    void addEdge(int v, int w);
    void DFS(int v);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new vector<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}
void Graph::DFS(int v)
{
    vector<bool> visited(V, false);
    stack<int> stack;
```

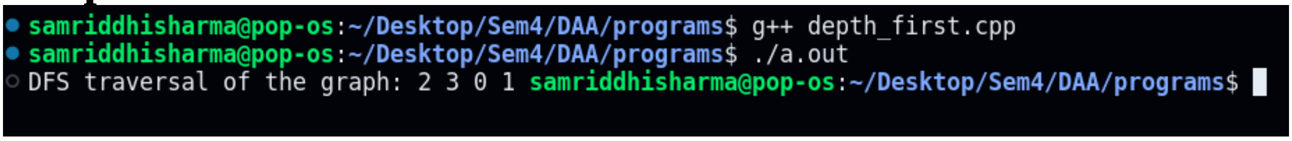


```cpp
    stack.push(v);
    visited[v] = true;

    while (!stack.empty())
    {
        int currentVertex = stack.top();
        stack.pop();

        cout << currentVertex << " ";

        for (int i = 0; i < adj[currentVertex].size(); i++)
        {
            int adjVertex = adj[currentVertex][i];
            if (!visited[adjVertex])
            {
                visited[adjVertex] = true;
                stack.push(adjVertex);
            }
        }
    }
}
```



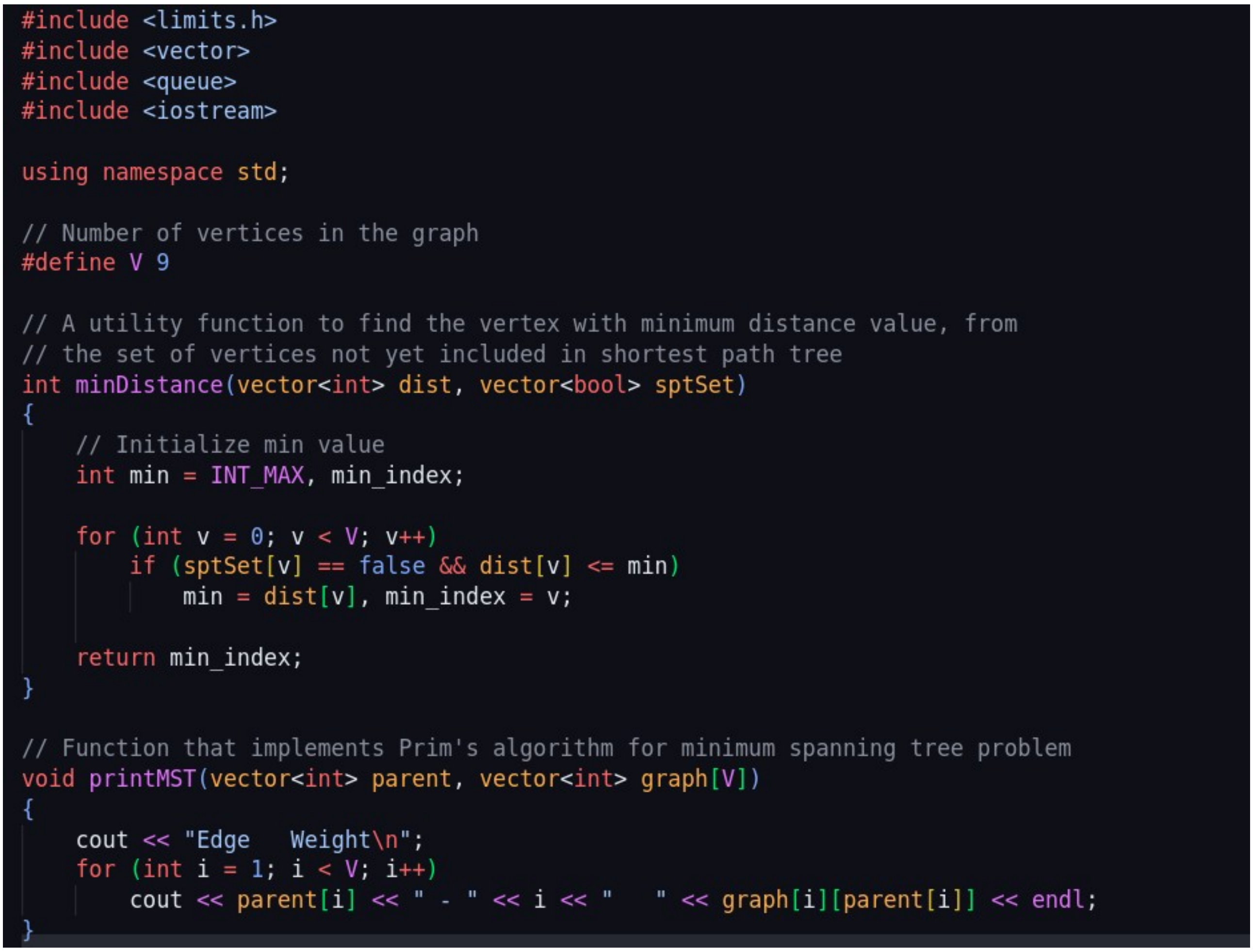
```cpp
int main()
{
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "DFS traversal of the graph: ";
    g.DFS(2);

    return 0;
}
```

Output:

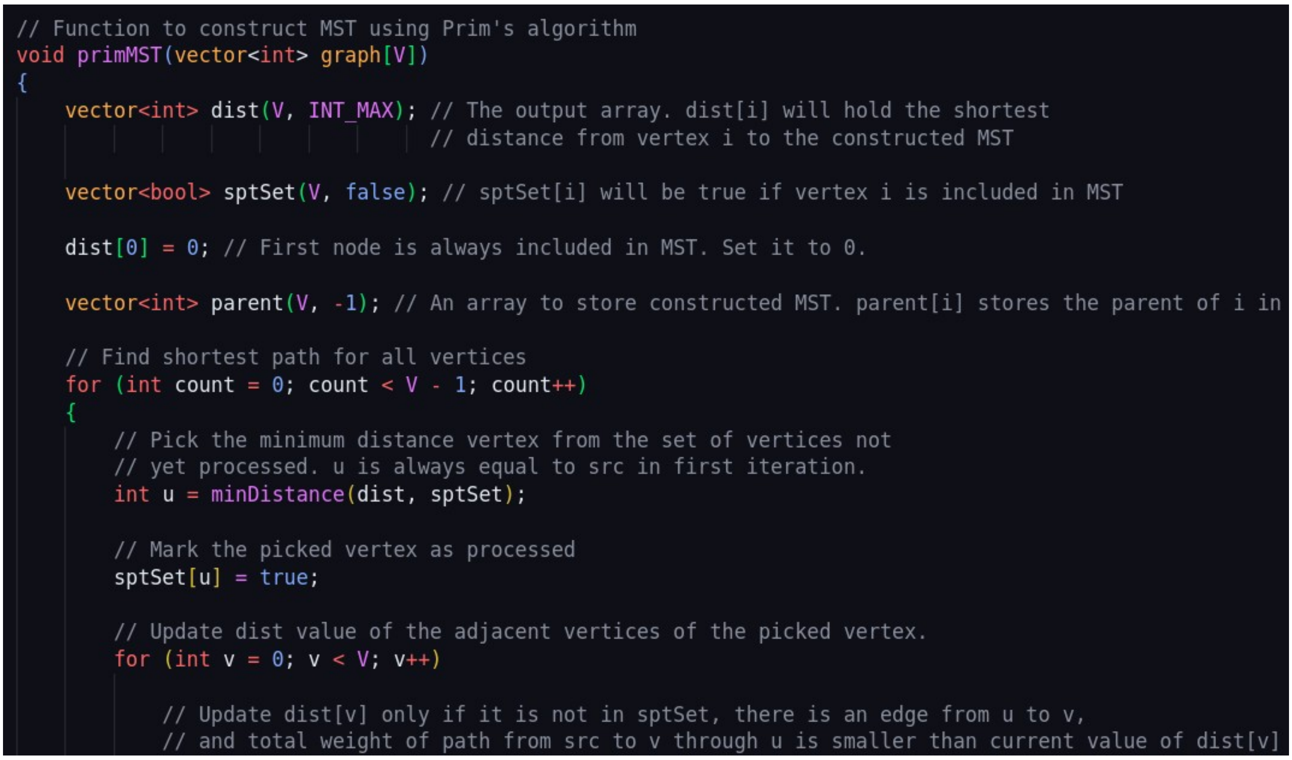


```
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ depth_first.cpp
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
DFS traversal of the graph: 2 3 0 1 samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

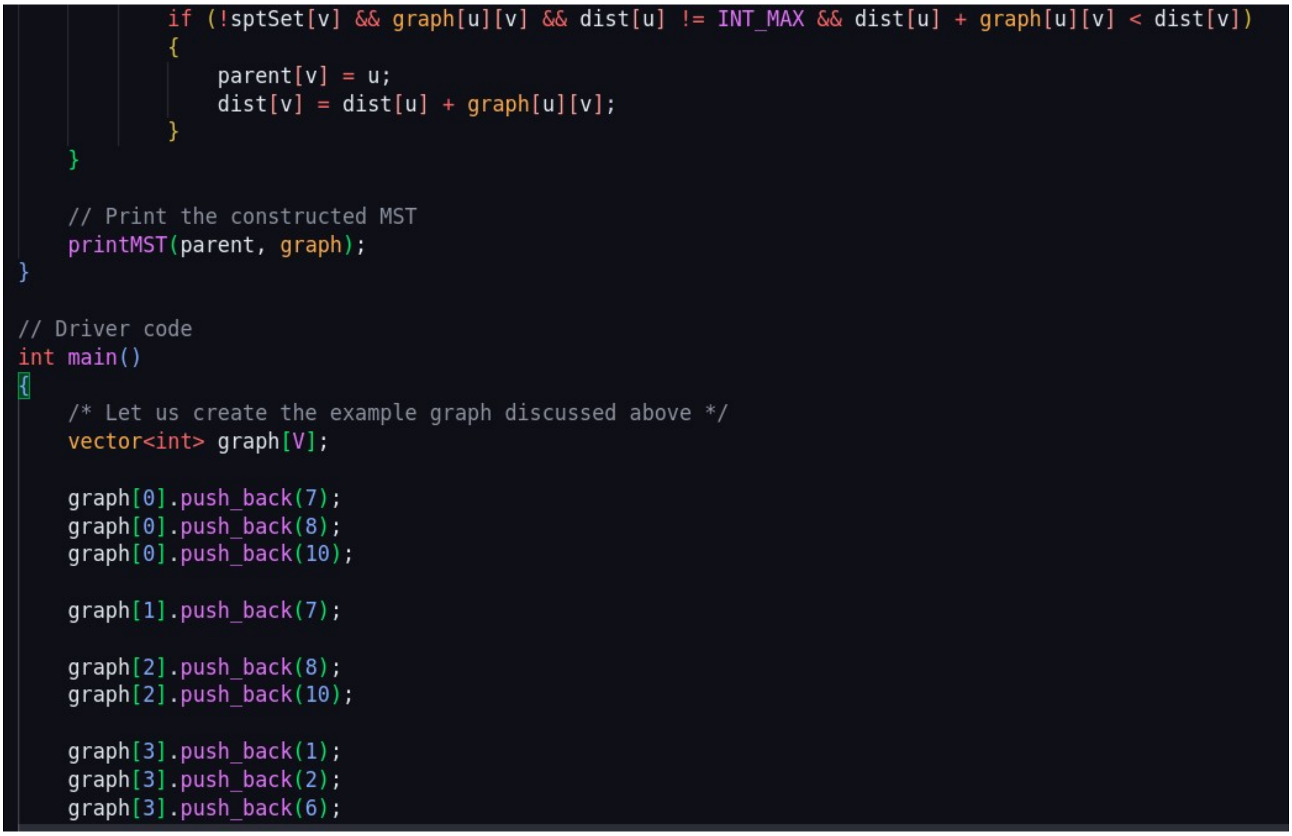
Ques9) Write a program to determine a minimum spanning tree of a graph using the Prim's algorithm.



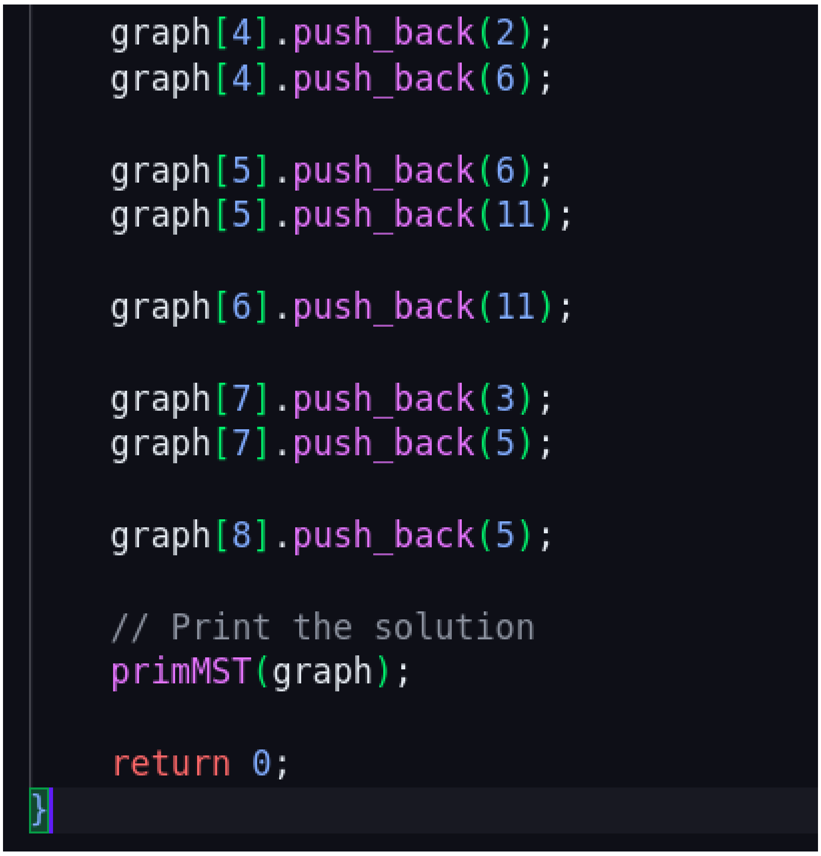
```cpp
#include <limits.h>
#include <vector>
#include <queue>
#include <iostream>

using namespace std;

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(vector<int> dist, vector<bool> sptSet)
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// Function that implements Prim's algorithm for minimum spanning tree problem
void printMST(vector<int> parent, vector<int> graph[V])
{
    cout << "Edge   Weight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << "    " << graph[i][parent[i]] << endl;
}
```



```cpp
// Function to construct MST using Prim's algorithm
void primMST(vector<int> graph[V])
{
    vector<int> dist(V, INT_MAX); // The output array. dist[i] will hold the shortest
                                  // distance from vertex i to the constructed MST

    vector<bool> sptSet(V, false); // sptSet[i] will be true if vertex i is included in MST

    dist[0] = 0; // First node is always included in MST. Set it to 0.

    vector<int> parent(V, -1); // An array to store constructed MST. parent[i] stores the parent of i in

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++)
    {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if it is not in sptSet, there is an edge from u to v,
            // and total weight of path from src to v through u is smaller than current value of dist[v]
```

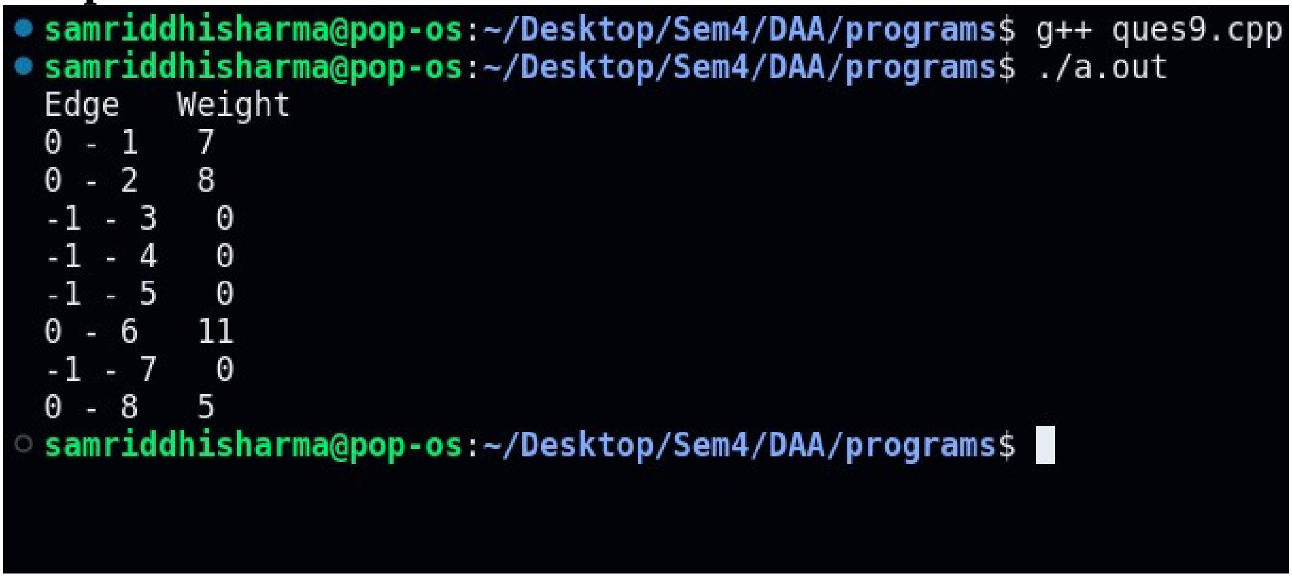


```cpp
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
            {
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
    }

    // Print the constructed MST
    printMST(parent, graph);
}

// Driver code
int main()
{
    /* Let us create the example graph discussed above */
    vector<int> graph[V];

    graph[0].push_back(7);
    graph[0].push_back(8);
    graph[0].push_back(10);

    graph[1].push_back(7);

    graph[2].push_back(8);
    graph[2].push_back(10);

    graph[3].push_back(1);
    graph[3].push_back(2);
    graph[3].push_back(6);
```



```
    graph[4].push_back(2);
    graph[4].push_back(6);

    graph[5].push_back(6);
    graph[5].push_back(11);

    graph[6].push_back(11);

    graph[7].push_back(3);
    graph[7].push_back(5);

    graph[8].push_back(5);

    // Print the solution
    primMST(graph);

    return 0;
}
```

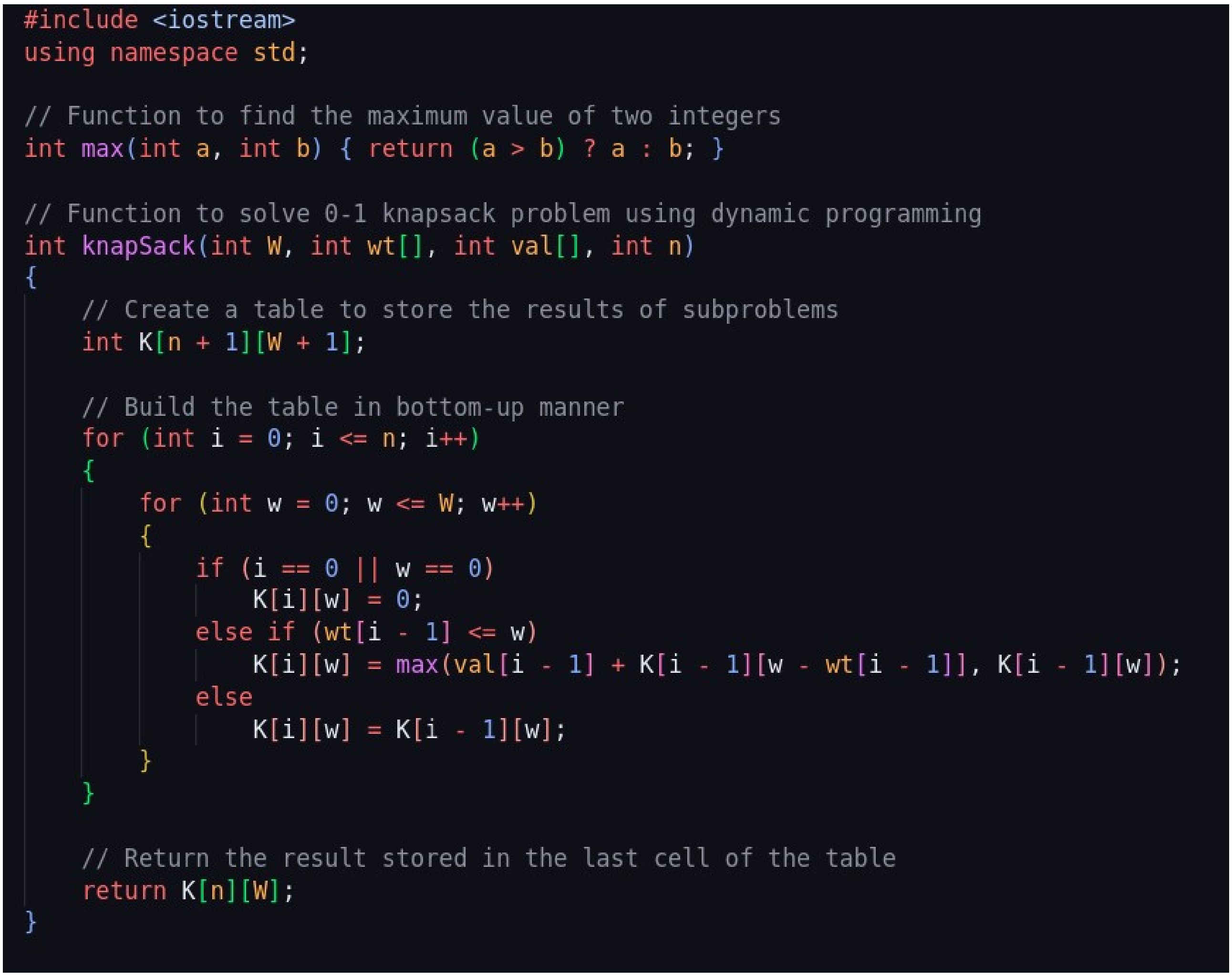
Output:



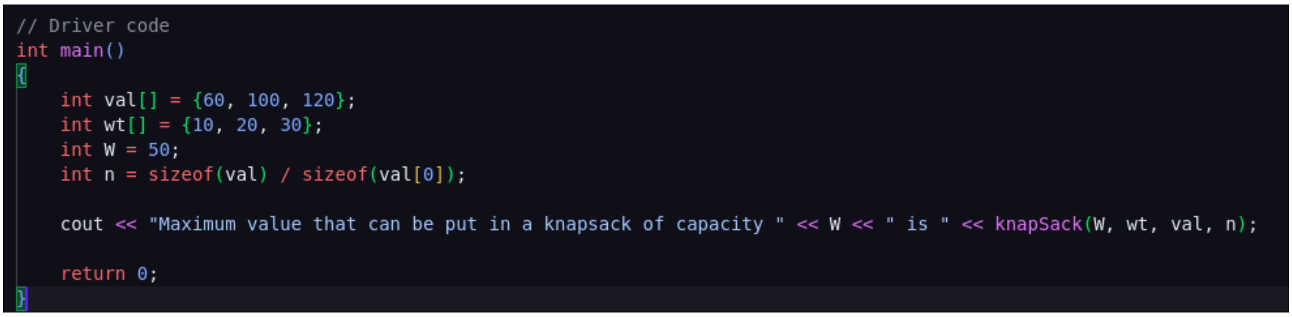
```
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ ques9.cpp
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
Edge   Weight
0 - 1   7
0 - 2   8
-1 - 3   0
-1 - 4   0
-1 - 5   0
0 - 6   11
-1 - 7   0
0 - 8   5
samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ 
```

Ques10) Write a program to solve the 0-1 knapsack problem.

Code:

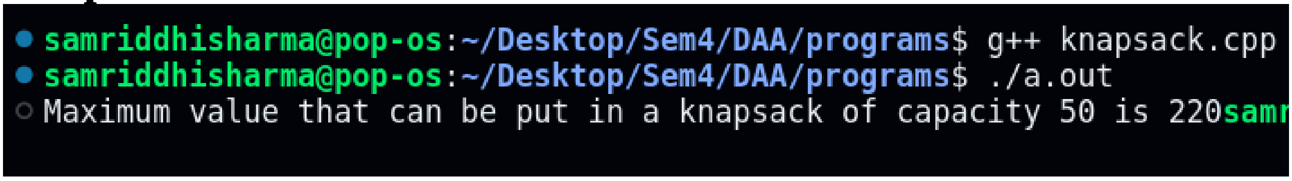


```cpp
#include <iostream>
using namespace std;

// Function to find the maximum value of two integers
int max(int a, int b) { return (a > b) ? a : b; }

// Function to solve 0-1 knapsack problem using dynamic programming
int knapSack(int W, int wt[], int val[], int n)
{
    // Create a table to store the results of subproblems
    int K[n + 1][W + 1];

    // Build the table in bottom-up manner
    for (int i = 0; i <= n; i++)
    {
        for (int w = 0; w <= W; w++)
        {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    // Return the result stored in the last cell of the table
    return K[n][W];
}
```



```cpp
// Driver code
int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);

    cout << "Maximum value that can be put in a knapsack of capacity " << W << " is " << knapSack(W, wt, val, n);

    return 0;
}
```

Output:



```
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ g++ knapsack.cpp
● samriddhisharma@pop-os:~/Desktop/Sem4/DAA/programs$ ./a.out
○ Maximum value that can be put in a knapsack of capacity 50 is 220samr
```