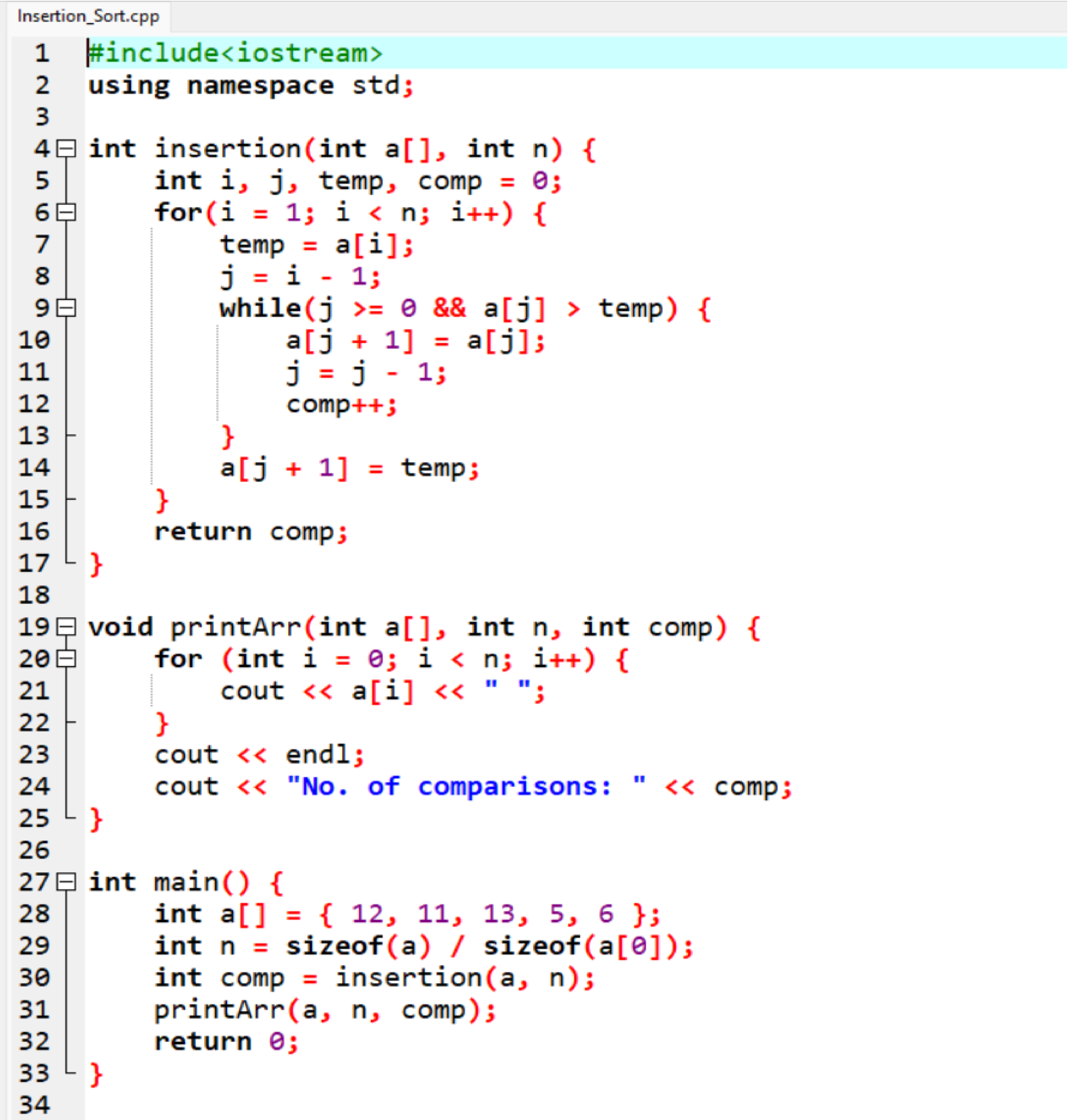
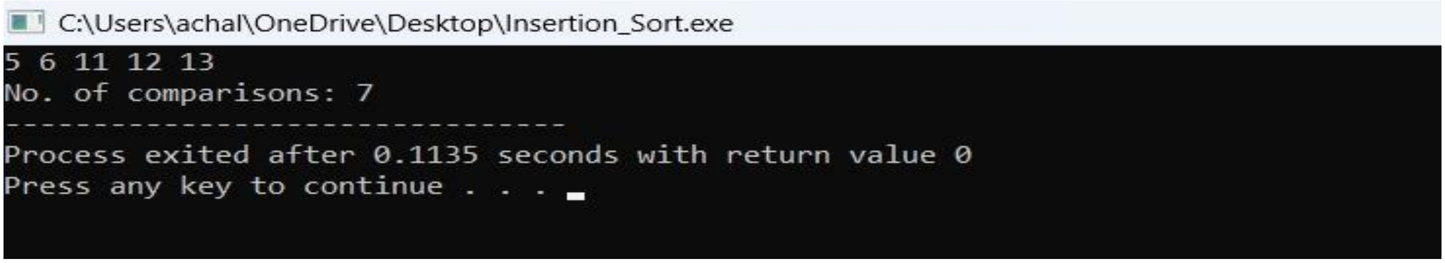
1) Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).

PROGRAM:



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

int insertion(int a[], int n) {
    int i, j, temp, comp = 0;
    for(i = 1; i < n; i++) {
        temp = a[i];
        j = i - 1;
        while(j >= 0 && a[j] > temp) {
            a[j + 1] = a[j];
            j = j - 1;
            comp++;
        }
        a[j + 1] = temp;
    }
    return comp;
}

void printArr(int a[], int n, int comp) {
    for (int i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    cout << "No. of comparisons: " << comp;
}

int main() {
    int a[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(a) / sizeof(a[0]);
    int comp = insertion(a, n);
    printArr(a, n, comp);
    return 0;
}

```

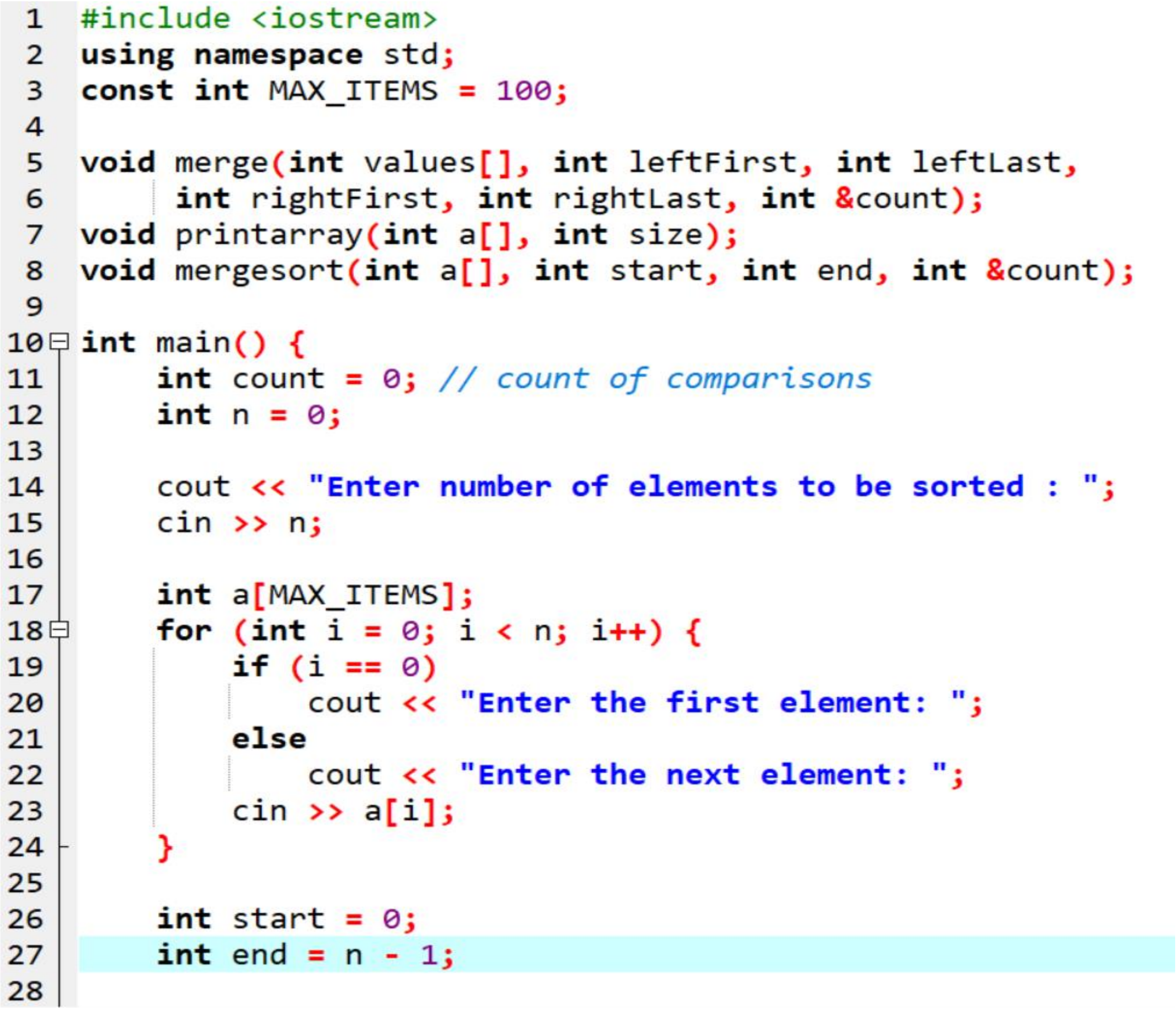
OUTPUT:



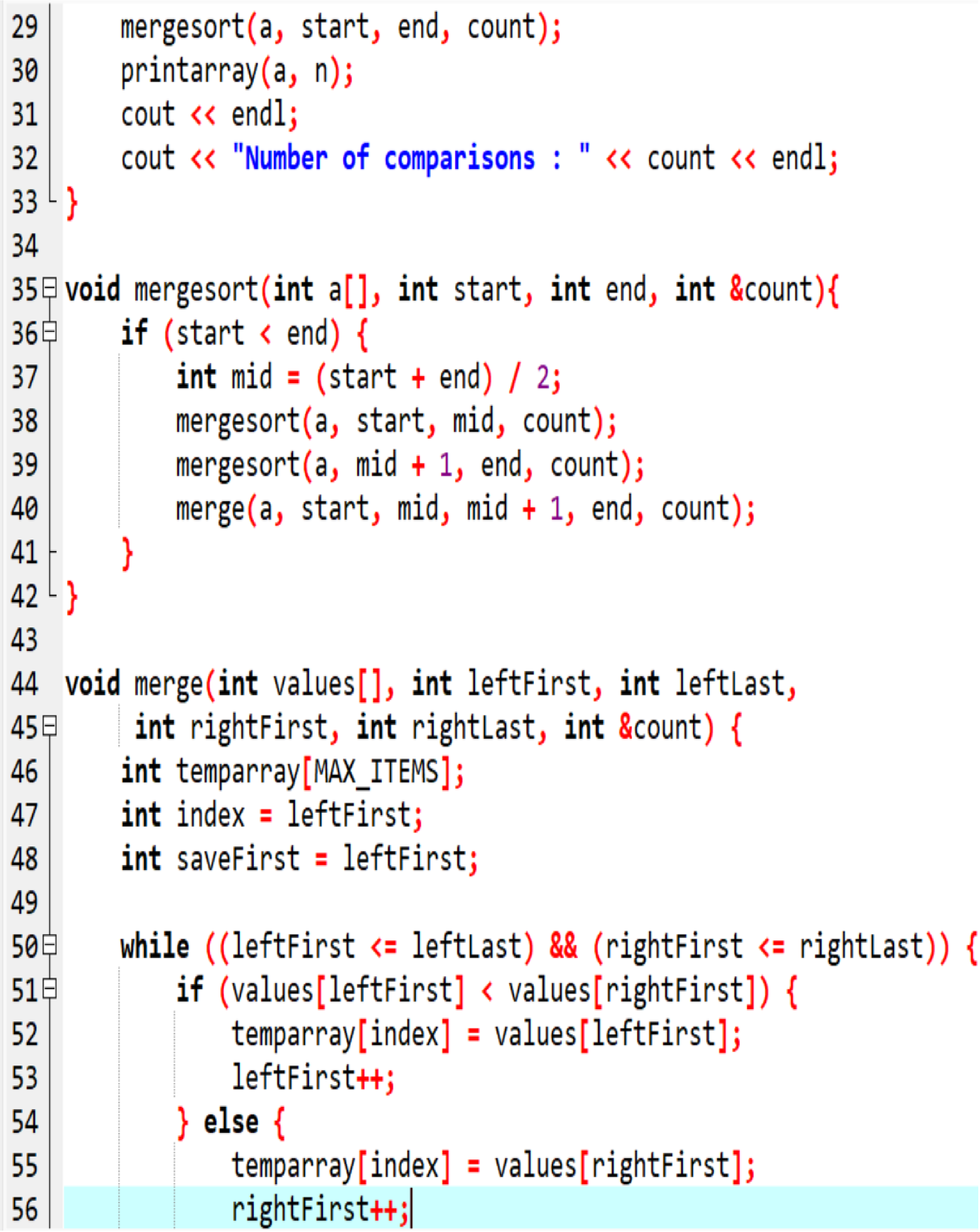
```
5 6 11 12 13
No. of comparisons: 7
--------------------------------
Process exited after 0.1135 seconds with return value 0
Press any key to continue . . .
```

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

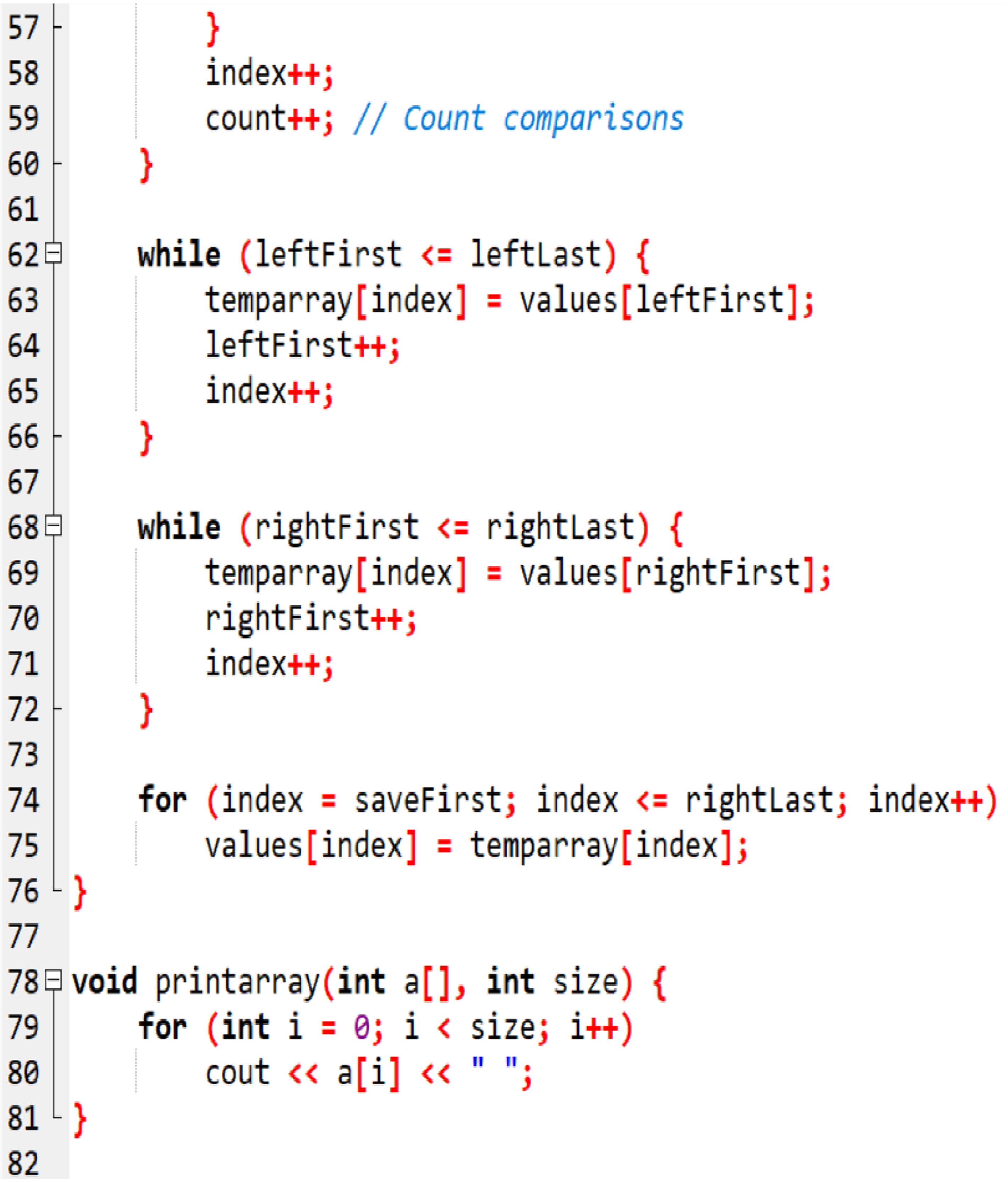
PROGRAM:



```cpp
#include <iostream>
using namespace std;
const int MAX_ITEMS = 100;

void merge(int values[], int leftFirst, int leftLast,
     int rightFirst, int rightLast, int &count);
void printarray(int a[], int size);
void mergesort(int a[], int start, int end, int &count);

int main() {
    int count = 0; // count of comparisons
    int n = 0;

    cout << "Enter number of elements to be sorted : ";
    cin >> n;

    int a[MAX_ITEMS];
    for (int i = 0; i < n; i++) {
        if (i == 0)
            cout << "Enter the first element: ";
        else
            cout << "Enter the next element: ";
        cin >> a[i];
    }

    int start = 0;
    int end = n - 1;

```

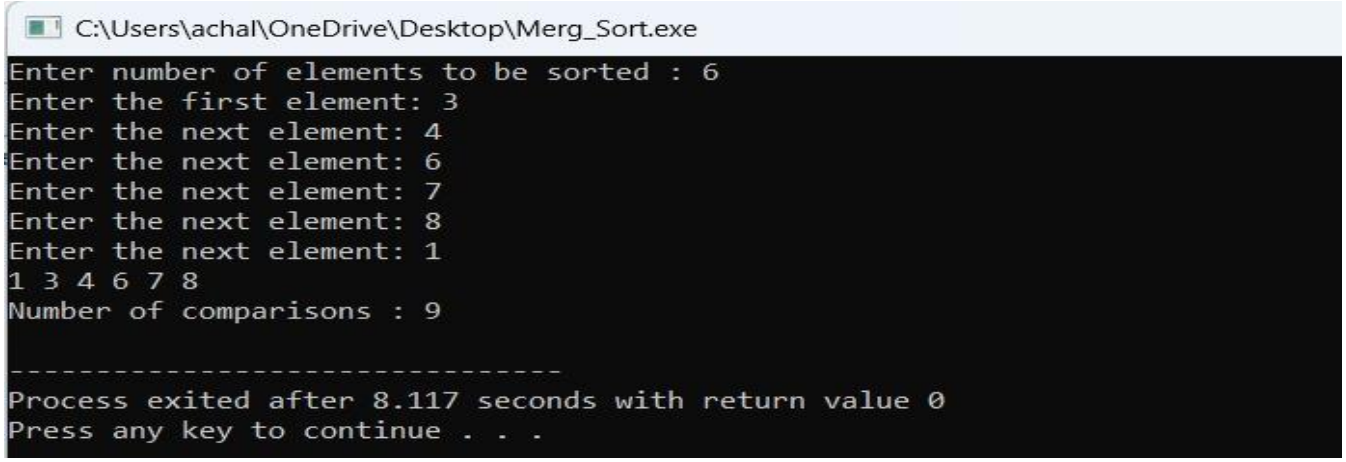


```
    mergesort(a, start, end, count);
    printarray(a, n);
    cout << endl;
    cout << "Number of comparisons : " << count << endl;
}

void mergesort(int a[], int start, int end, int &count){
    if (start < end) {
        int mid = (start + end) / 2;
        mergesort(a, start, mid, count);
        mergesort(a, mid + 1, end, count);
        merge(a, start, mid, mid + 1, end, count);
    }
}

void merge(int values[], int leftFirst, int leftLast,
     int rightFirst, int rightLast, int &count) {
    int temparray[MAX_ITEMS];
    int index = leftFirst;
    int saveFirst = leftFirst;

    while ((leftFirst <= leftLast) && (rightFirst <= rightLast)) {
        if (values[leftFirst] < values[rightFirst]) {
            temparray[index] = values[leftFirst];
            leftFirst++;
        } else {
            temparray[index] = values[rightFirst];
            rightFirst++;
```



```
        }
        index++;
        count++; // Count comparisons
    }

    while (leftFirst <= leftLast) {
        temparray[index] = values[leftFirst];
        leftFirst++;
        index++;
    }

    while (rightFirst <= rightLast) {
        temparray[index] = values[rightFirst];
        rightFirst++;
        index++;
    }

    for (index = saveFirst; index <= rightLast; index++)
        values[index] = temparray[index];
}

void printarray(int a[], int size) {
    for (int i = 0; i < size; i++)
        cout << a[i] << " ";
}

```

OUTPUT:

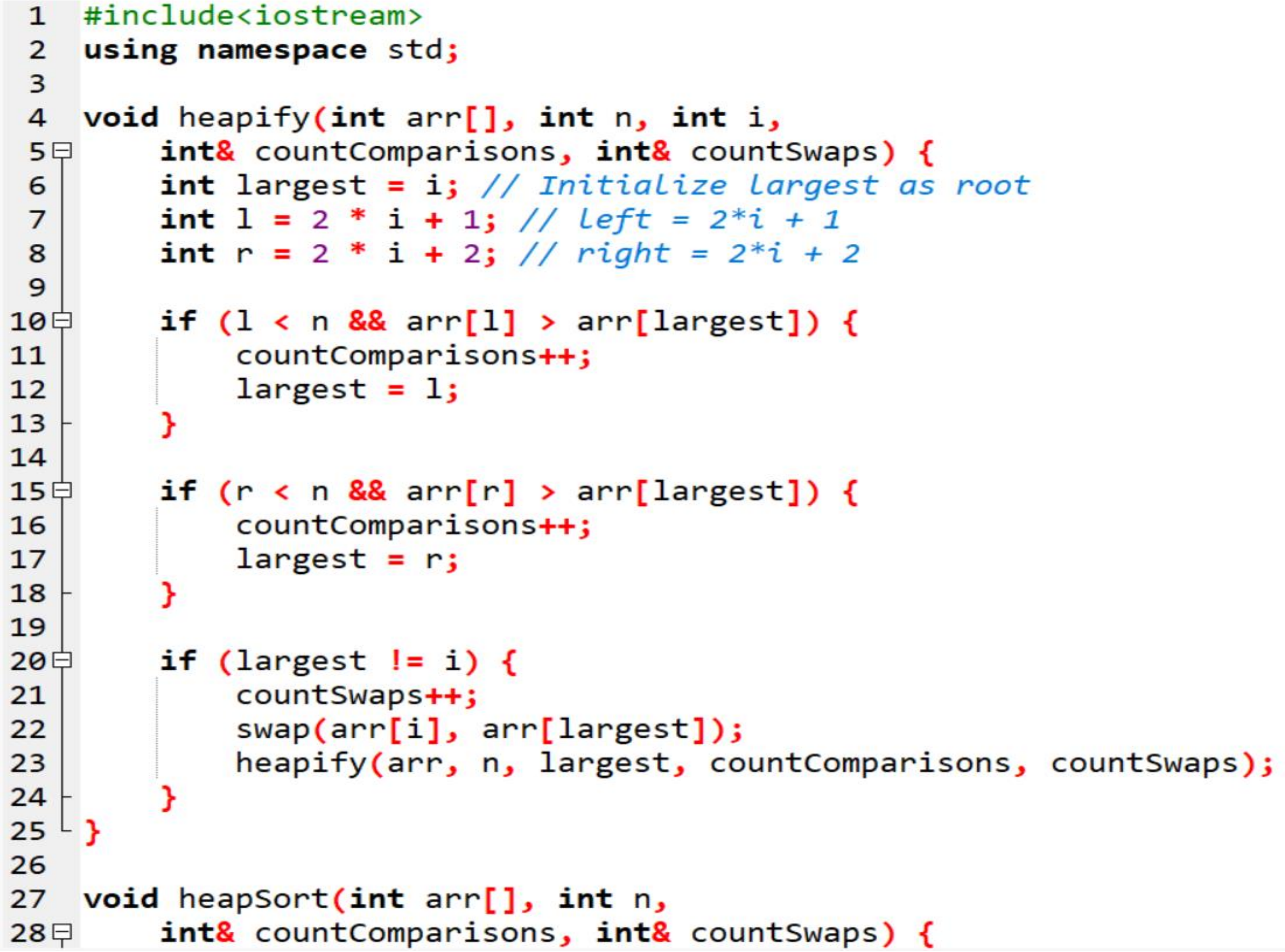


C:\Users\achal\OneDrive\Desktop\Merg_Sort.exe

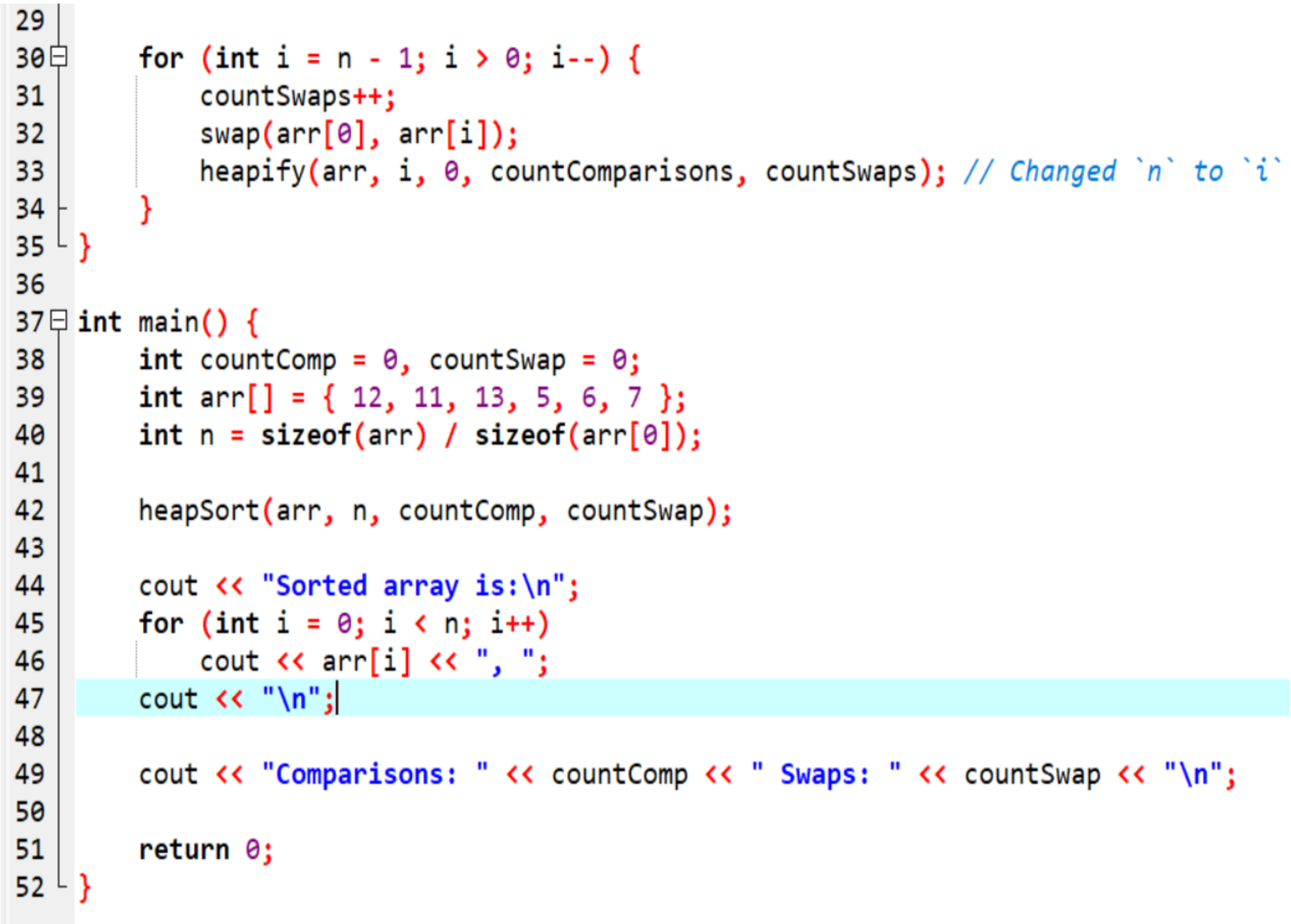
```
Enter number of elements to be sorted : 6
Enter the first element: 3
Enter the next element: 4
Enter the next element: 6
Enter the next element: 7
Enter the next element: 8
Enter the next element: 1
1 3 4 6 7 8
Number of comparisons : 9

--------------------------------
Process exited after 8.117 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

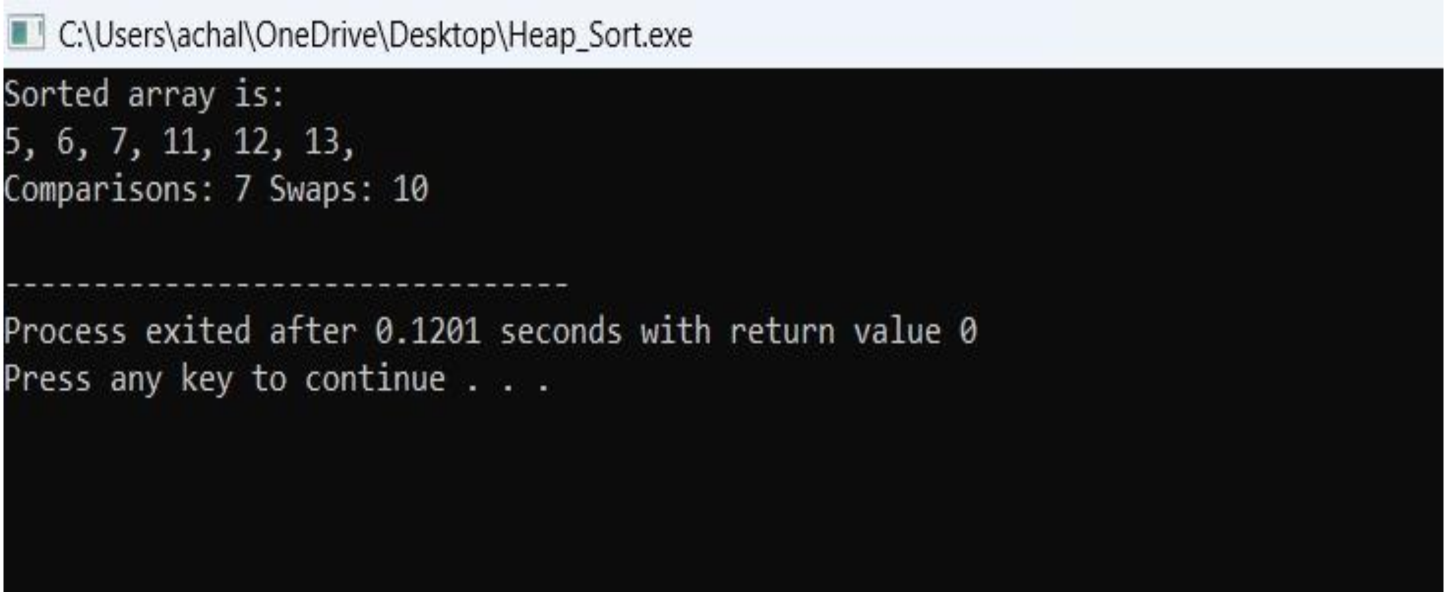


```
#include<iostream>
using namespace std;

void heapify(int arr[], int n, int i,
    int& countComparisons, int& countSwaps) {
    int largest = i; // Initialize largest as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    if (l < n && arr[l] > arr[largest]) {
        countComparisons++;
        largest = l;
    }

    if (r < n && arr[r] > arr[largest]) {
        countComparisons++;
        largest = r;
    }

    if (largest != i) {
        countSwaps++;
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest, countComparisons, countSwaps);
    }
}

void heapSort(int arr[], int n,
    int& countComparisons, int& countSwaps) {
```



```cpp

    for (int i = n - 1; i > 0; i--) {
        countSwaps++;
        swap(arr[0], arr[i]);
        heapify(arr, i, 0, countComparisons, countSwaps); // Changed `n` to `i`
    }
}

int main() {
    int countComp = 0, countSwap = 0;
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n, countComp, countSwap);

    cout << "Sorted array is:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << ", ";
    cout << "\n";

    cout << "Comparisons: " << countComp << " Swaps: " << countSwap << "\n";

    return 0;
}
```

OUTPUT:

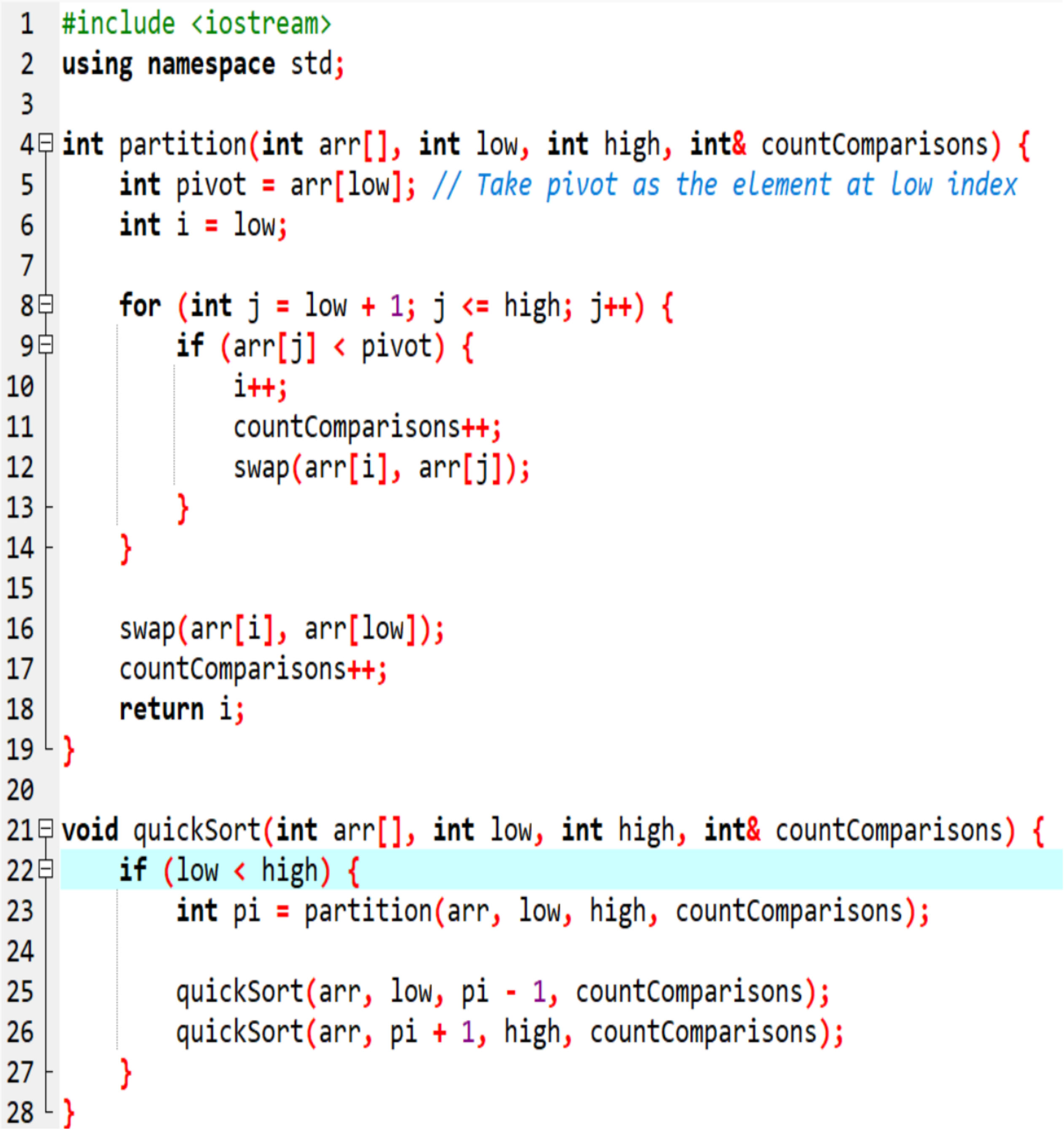


C:\Users\achal\OneDrive\Desktop\Heap_Sort.exe

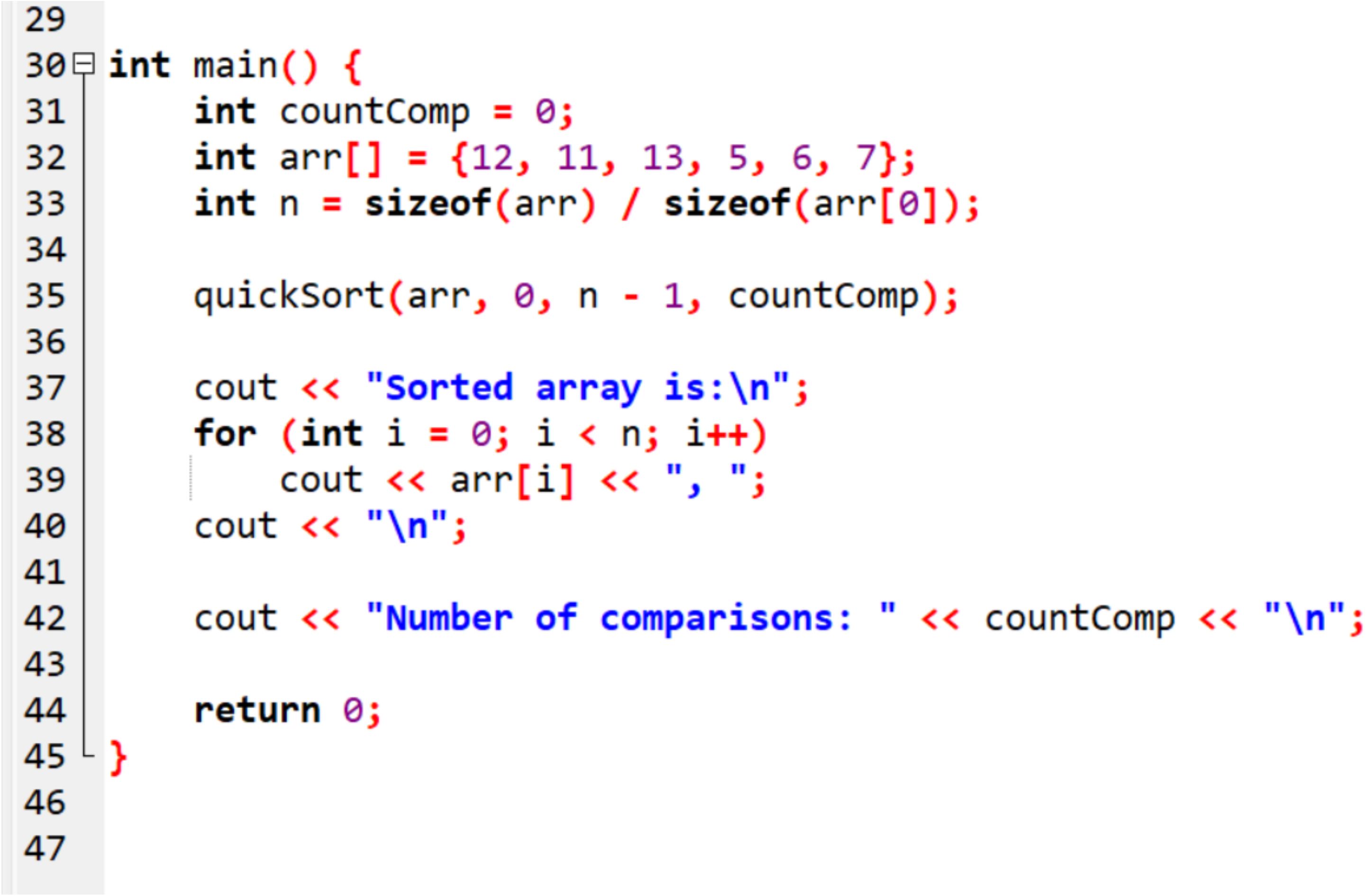
```
Sorted array is:
5, 6, 7, 11, 12, 13,
Comparisons: 7 Swaps: 10


--------------------------------
Process exited after 0.1201 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

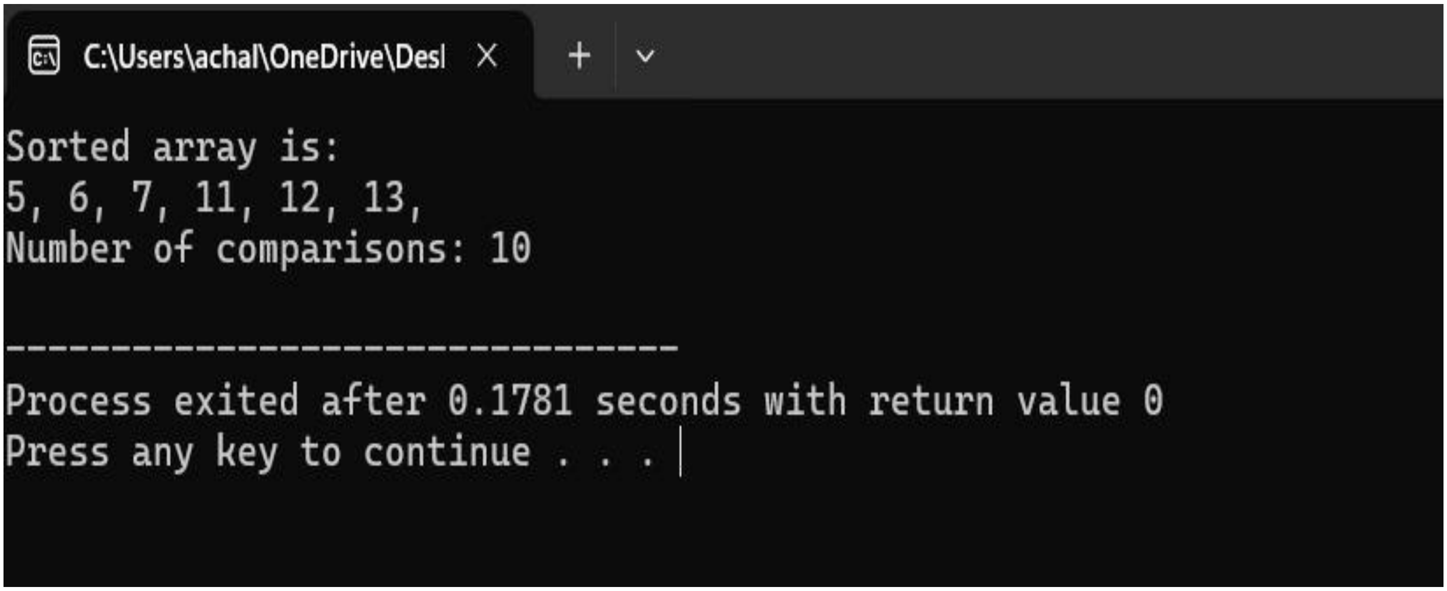


```cpp
#include <iostream>
using namespace std;

int partition(int arr[], int low, int high, int& countComparisons) {
    int pivot = arr[low]; // Take pivot as the element at low index
    int i = low;

    for (int j = low + 1; j <= high; j++) {
        if (arr[j] < pivot) {
            i++;
            countComparisons++;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i], arr[low]);
    countComparisons++;
    return i;
}

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

        quickSort(arr, low, pi - 1, countComparisons);
        quickSort(arr, pi + 1, high, countComparisons);
    }
}
```



```cpp

int main() {
    int countComp = 0;
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1, countComp);

    cout << "Sorted array is:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << ", ";
    cout << "\n";

    cout << "Number of comparisons: " << countComp << "\n";

    return 0;
}


```

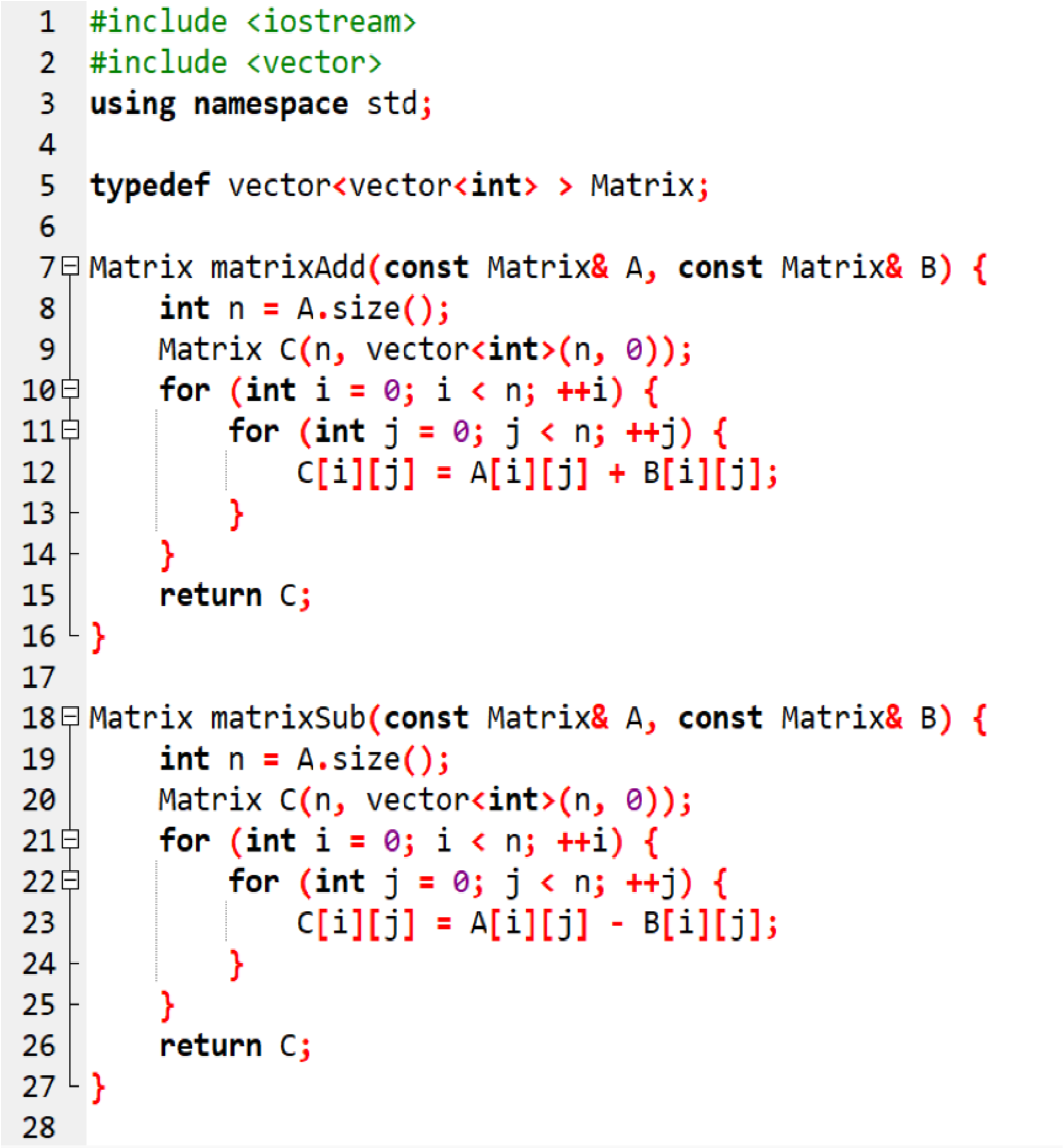
OUTPUT:



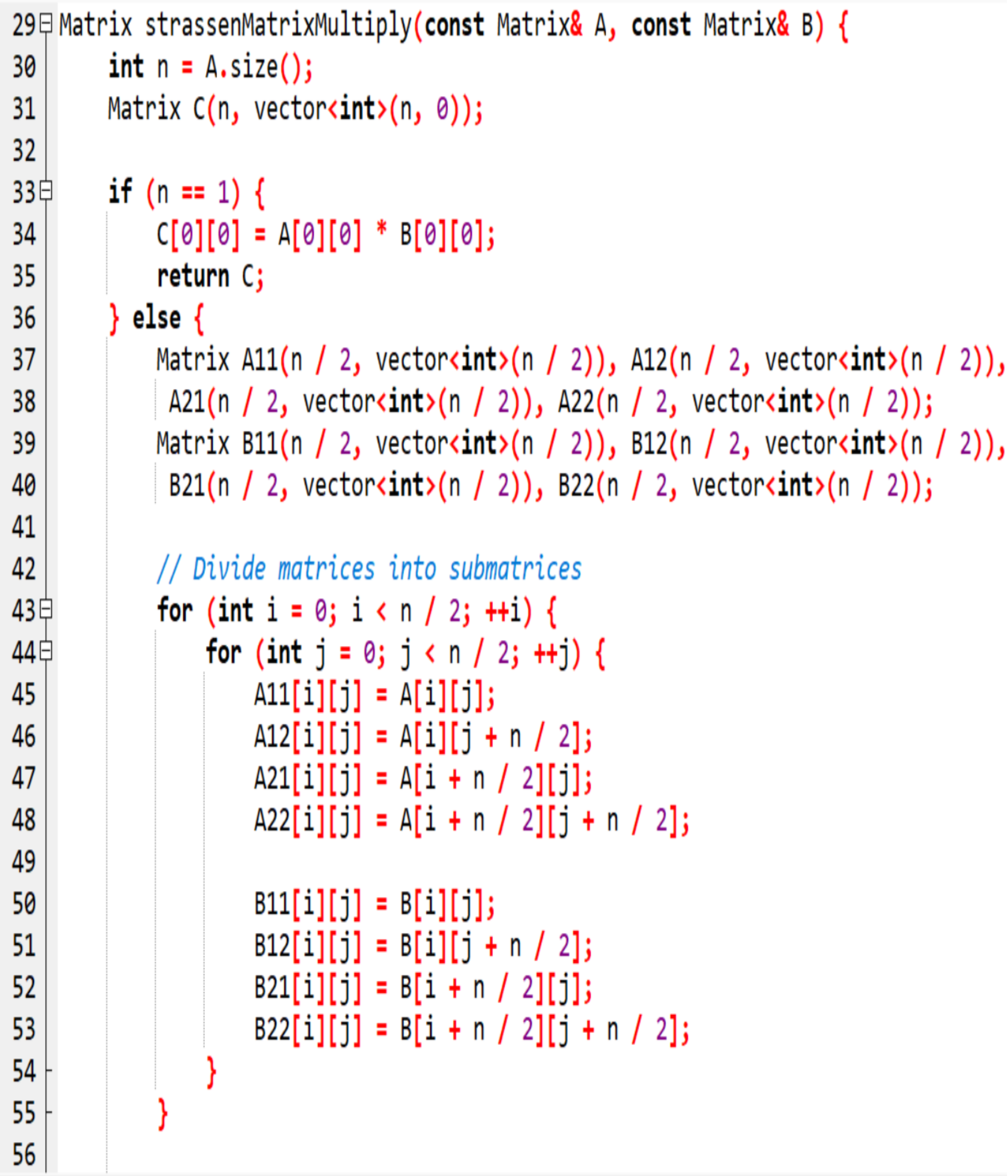
```
Sorted array is:
5, 6, 7, 11, 12, 13,
Number of comparisons: 10

--------------------------------
Process exited after 0.1781 seconds with return value 0
Press any key to continue . . .
```

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

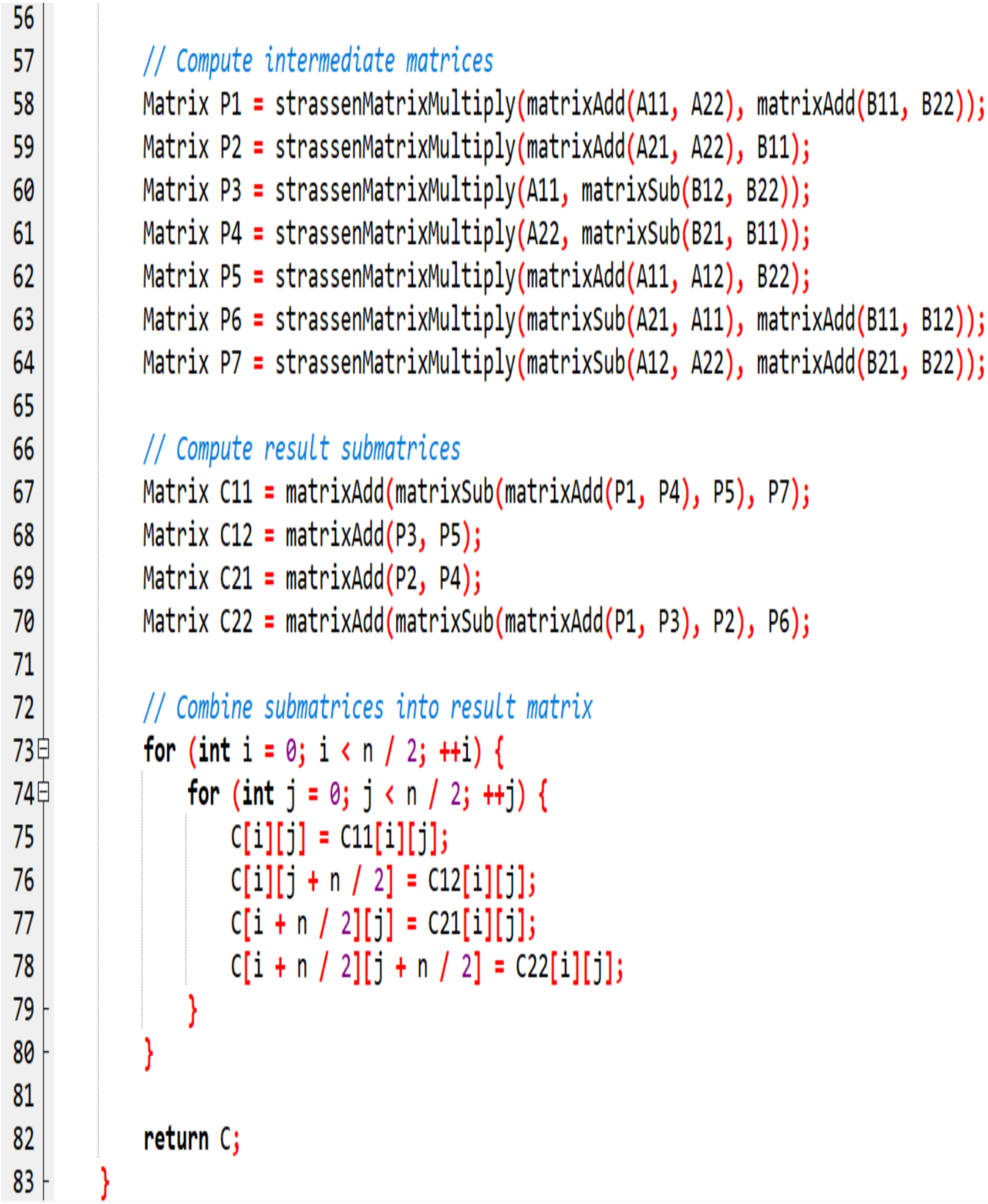
PROGRAM:



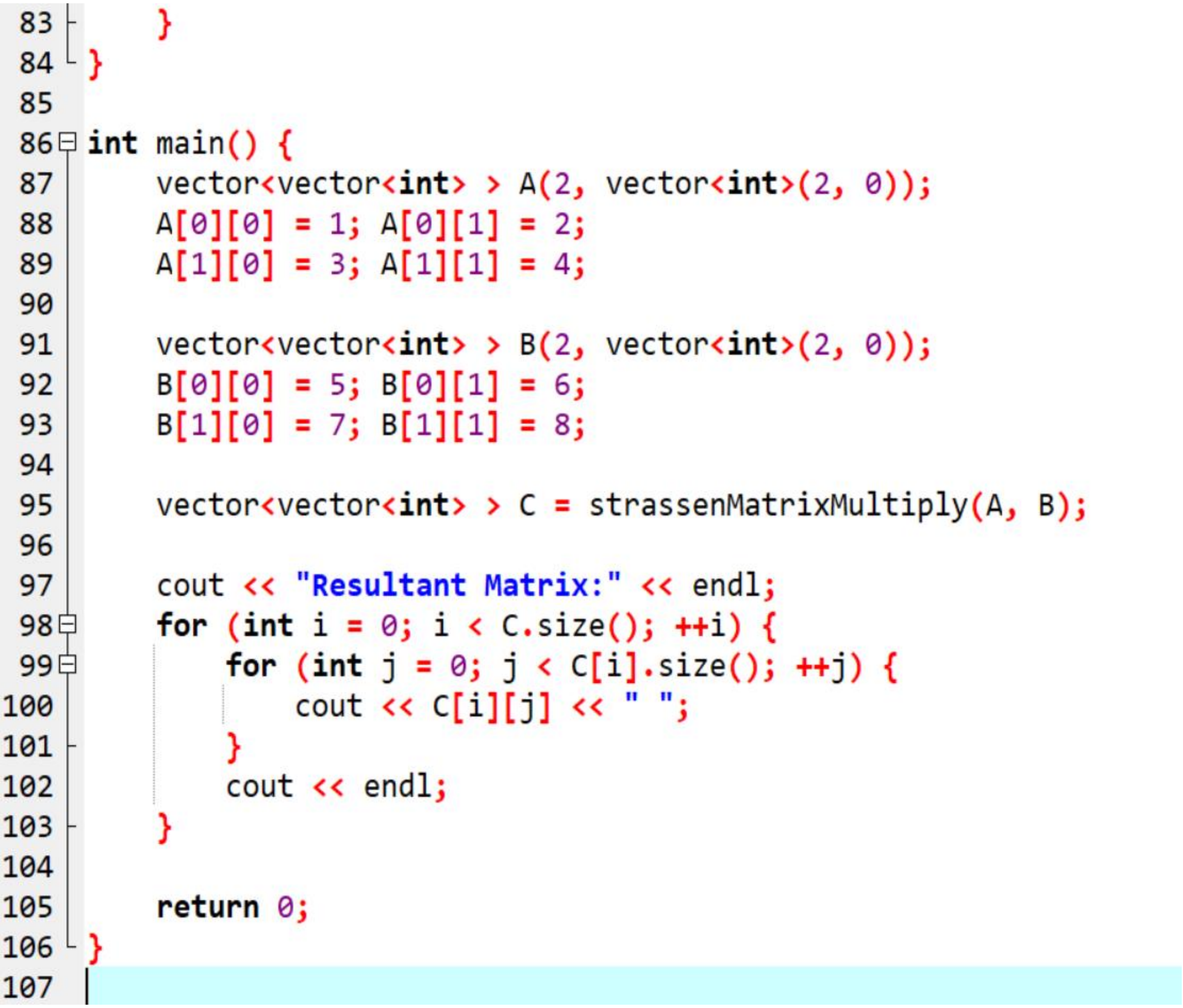
```
#include <iostream>
#include <vector>
using namespace std;

typedef vector<vector<int> > Matrix;

Matrix matrixAdd(const Matrix& A, const Matrix& B) {
    int n = A.size();
    Matrix 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;
}

Matrix matrixSub(const Matrix& A, const Matrix& B) {
    int n = A.size();
    Matrix 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;
}

```



```cpp
Matrix strassenMatrixMultiply(const Matrix& A, const Matrix& B) {
    int n = A.size();
    Matrix C(n, vector<int>(n, 0));

    if (n == 1) {
        C[0][0] = A[0][0] * B[0][0];
        return C;
    } else {
        Matrix A11(n / 2, vector<int>(n / 2)), A12(n / 2, vector<int>(n / 2)),
         A21(n / 2, vector<int>(n / 2)), A22(n / 2, vector<int>(n / 2));
        Matrix B11(n / 2, vector<int>(n / 2)), B12(n / 2, vector<int>(n / 2)),
         B21(n / 2, vector<int>(n / 2)), B22(n / 2, vector<int>(n / 2));

        // Divide matrices into submatrices
        for (int i = 0; i < n / 2; ++i) {
            for (int j = 0; j < n / 2; ++j) {
                A11[i][j] = A[i][j];
                A12[i][j] = A[i][j + n / 2];
                A21[i][j] = A[i + n / 2][j];
                A22[i][j] = A[i + n / 2][j + n / 2];

                B11[i][j] = B[i][j];
                B12[i][j] = B[i][j + n / 2];
                B21[i][j] = B[i + n / 2][j];
                B22[i][j] = B[i + n / 2][j + n / 2];
            }
        }

```

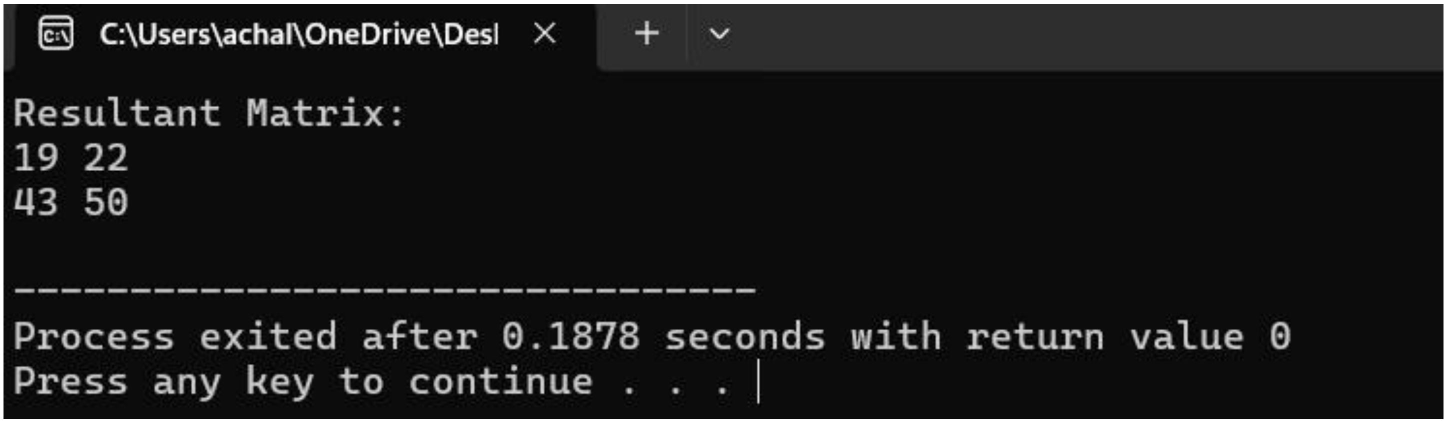


```

    // Compute intermediate matrices
    Matrix P1 = strassenMatrixMultiply(matrixAdd(A11, A22), matrixAdd(B11, B22));
    Matrix P2 = strassenMatrixMultiply(matrixAdd(A21, A22), B11);
    Matrix P3 = strassenMatrixMultiply(A11, matrixSub(B12, B22));
    Matrix P4 = strassenMatrixMultiply(A22, matrixSub(B21, B11));
    Matrix P5 = strassenMatrixMultiply(matrixAdd(A11, A12), B22);
    Matrix P6 = strassenMatrixMultiply(matrixSub(A21, A11), matrixAdd(B11, B12));
    Matrix P7 = strassenMatrixMultiply(matrixSub(A12, A22), matrixAdd(B21, B22));

    // Compute result submatrices
    Matrix C11 = matrixAdd(matrixSub(matrixAdd(P1, P4), P5), P7);
    Matrix C12 = matrixAdd(P3, P5);
    Matrix C21 = matrixAdd(P2, P4);
    Matrix C22 = matrixAdd(matrixSub(matrixAdd(P1, P3), P2), P6);

    // Combine submatrices into result matrix
    for (int i = 0; i < n / 2; ++i) {
        for (int j = 0; j < n / 2; ++j) {
            C[i][j] = C11[i][j];
            C[i][j + n / 2] = C12[i][j];
            C[i + n / 2][j] = C21[i][j];
            C[i + n / 2][j + n / 2] = C22[i][j];
        }
    }

    return C;
}
```



```cpp
    }
}

int main() {
    vector<vector<int> > A(2, vector<int>(2, 0));
    A[0][0] = 1; A[0][1] = 2;
    A[1][0] = 3; A[1][1] = 4;

    vector<vector<int> > B(2, vector<int>(2, 0));
    B[0][0] = 5; B[0][1] = 6;
    B[1][0] = 7; B[1][1] = 8;

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

    cout << "Resultant Matrix:" << endl;
    for (int i = 0; i < C.size(); ++i) {
        for (int j = 0; j < C[i].size(); ++j) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}

```

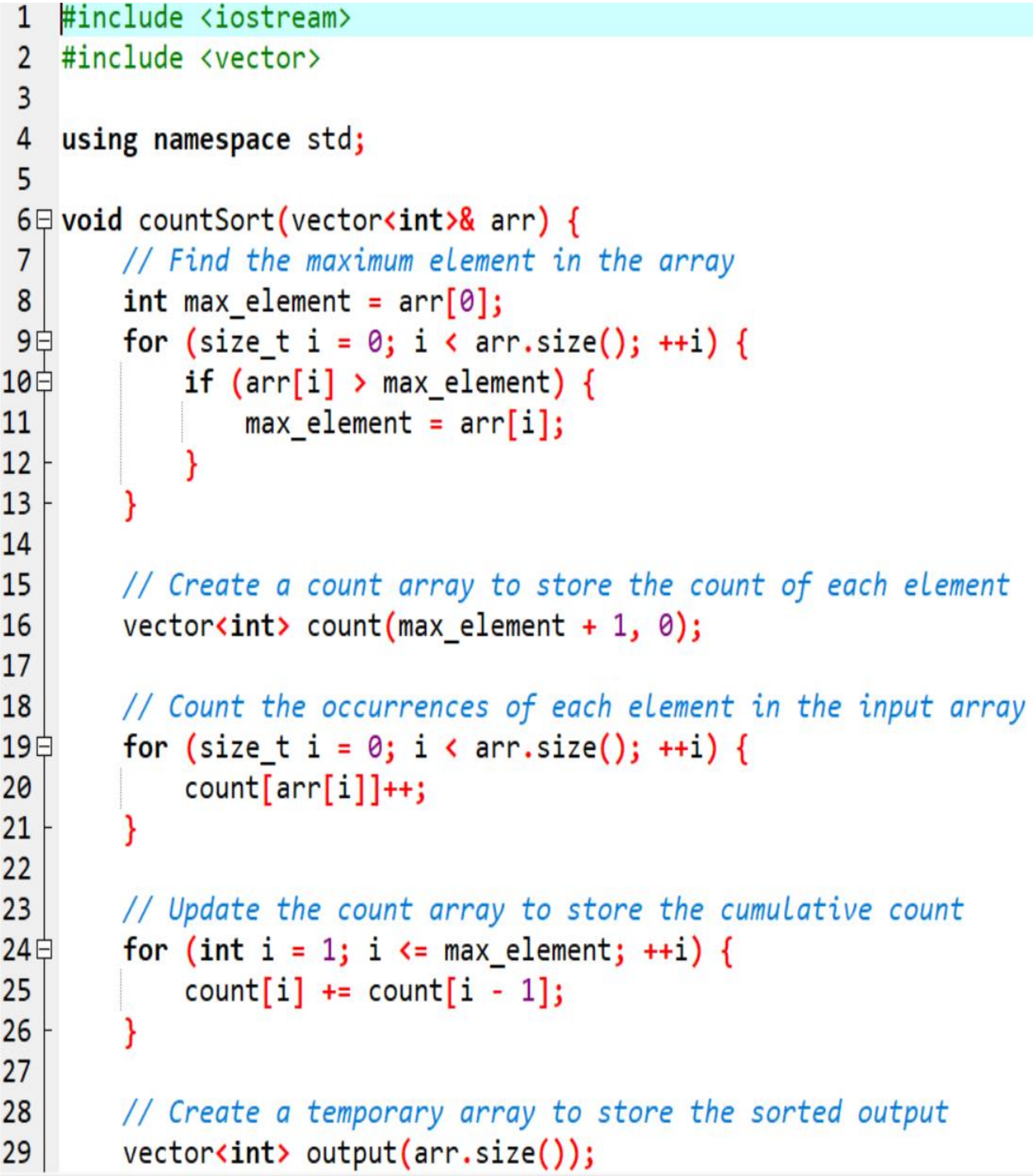
OUTPUT:



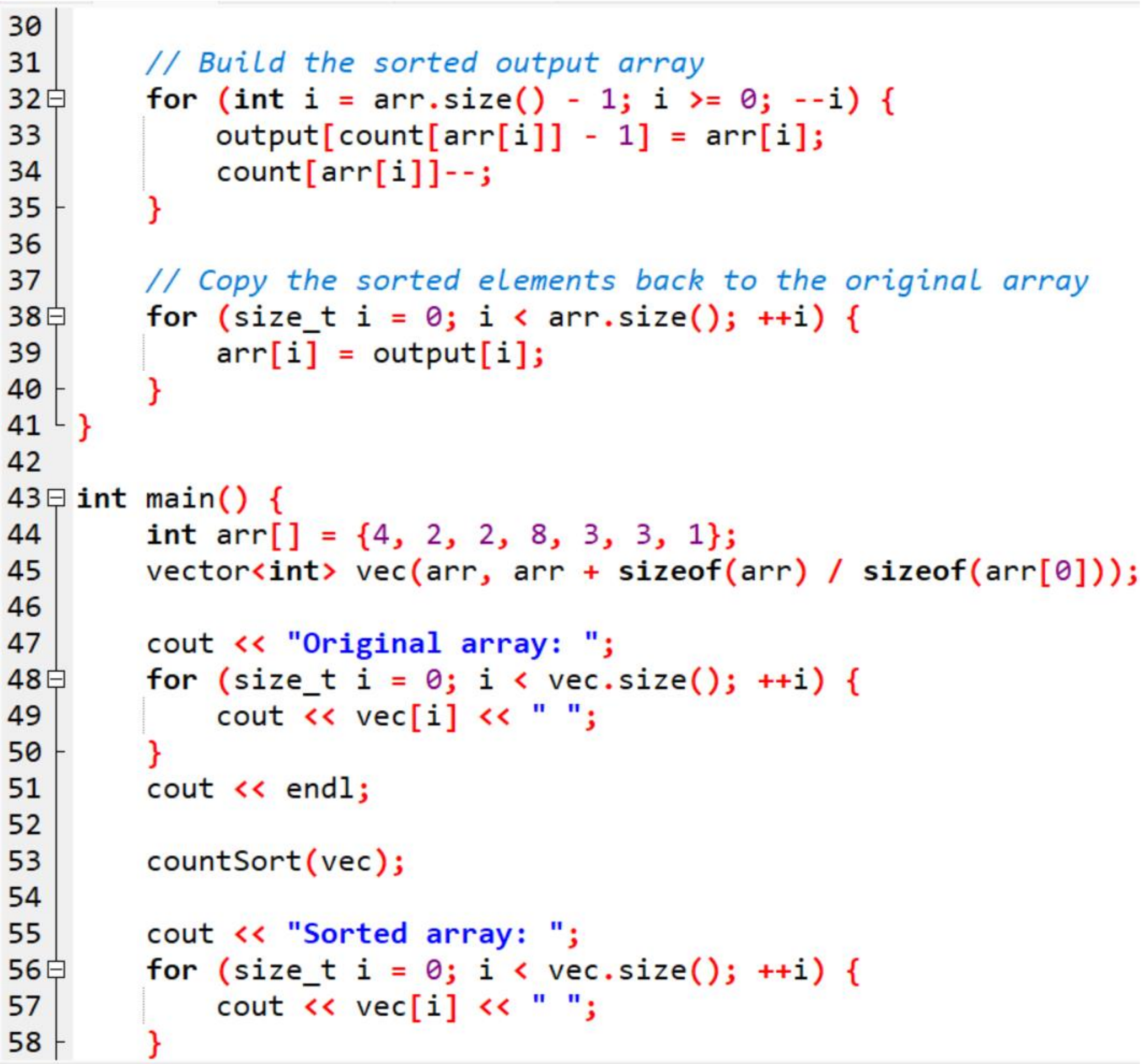
```
Resultant Matrix:
19 22
43 50

--------------------------------
Process exited after 0.1878 seconds with return value 0
Press any key to continue . . .
```

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

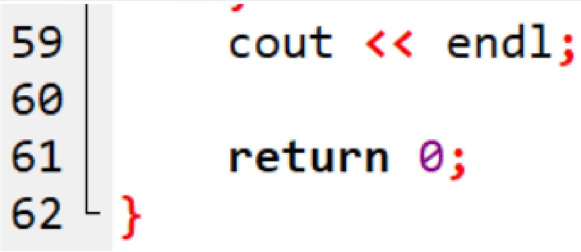
PROGRAM:



```
#include <iostream>
#include <vector>

using namespace std;

void countSort(vector<int>& arr) {
    // Find the maximum element in the array
    int max_element = arr[0];
    for (size_t i = 0; i < arr.size(); ++i) {
        if (arr[i] > max_element) {
            max_element = arr[i];
        }
    }

    // Create a count array to store the count of each element
    vector<int> count(max_element + 1, 0);

    // Count the occurrences of each element in the input array
    for (size_t i = 0; i < arr.size(); ++i) {
        count[arr[i]]++;
    }

    // Update the count array to store the cumulative count
    for (int i = 1; i <= max_element; ++i) {
        count[i] += count[i - 1];
    }

    // Create a temporary array to store the sorted output
    vector<int> output(arr.size());
```

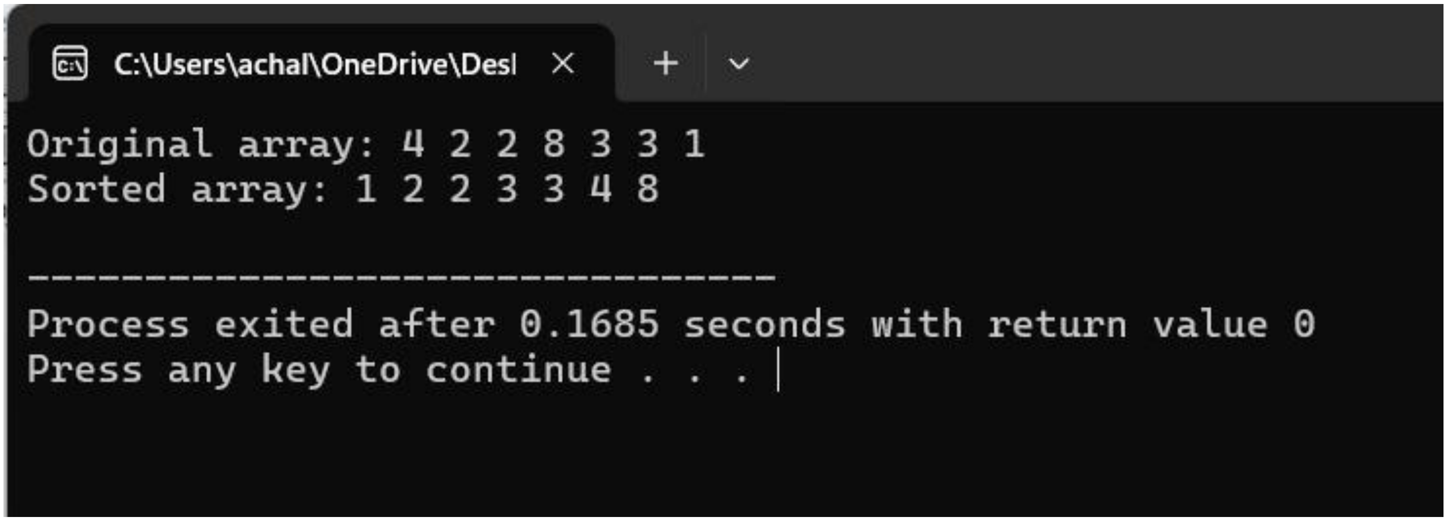


```

    // Build the sorted output array
    for (int i = arr.size() - 1; i >= 0; --i) {
        output[count[arr[i]] - 1] = arr[i];
        count[arr[i]]--;
    }

    // Copy the sorted elements back to the original array
    for (size_t i = 0; i < arr.size(); ++i) {
        arr[i] = output[i];
    }
}

int main() {
    int arr[] = {4, 2, 2, 8, 3, 3, 1};
    vector<int> vec(arr, arr + sizeof(arr) / sizeof(arr[0]));

    cout << "Original array: ";
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << " ";
    }
    cout << endl;

    countSort(vec);

    cout << "Sorted array: ";
    for (size_t i = 0; i < vec.size(); ++i) {
        cout << vec[i] << " ";
    }
```



```
    cout << endl;

    return 0;
}
```

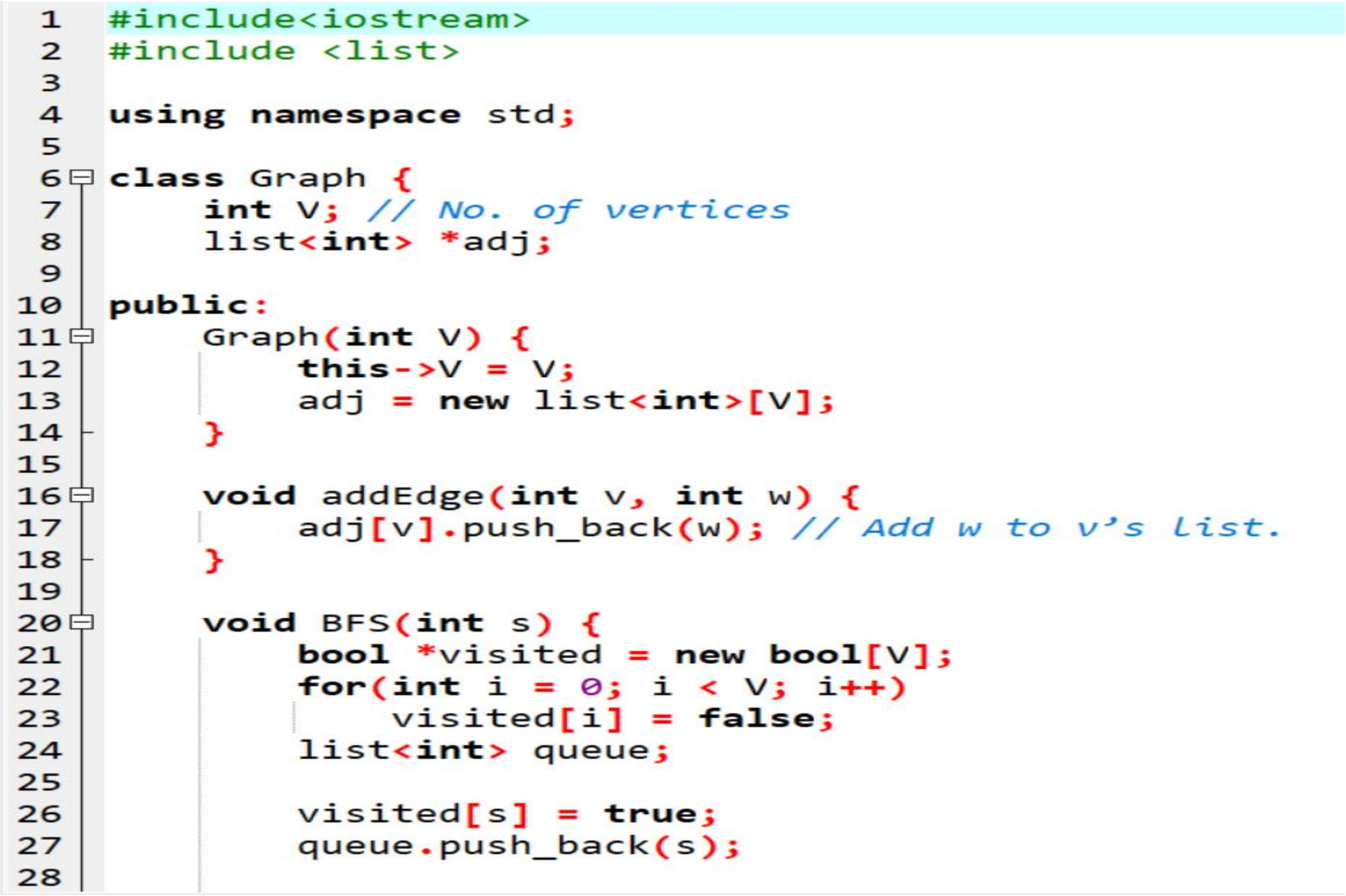
OUTPUT:



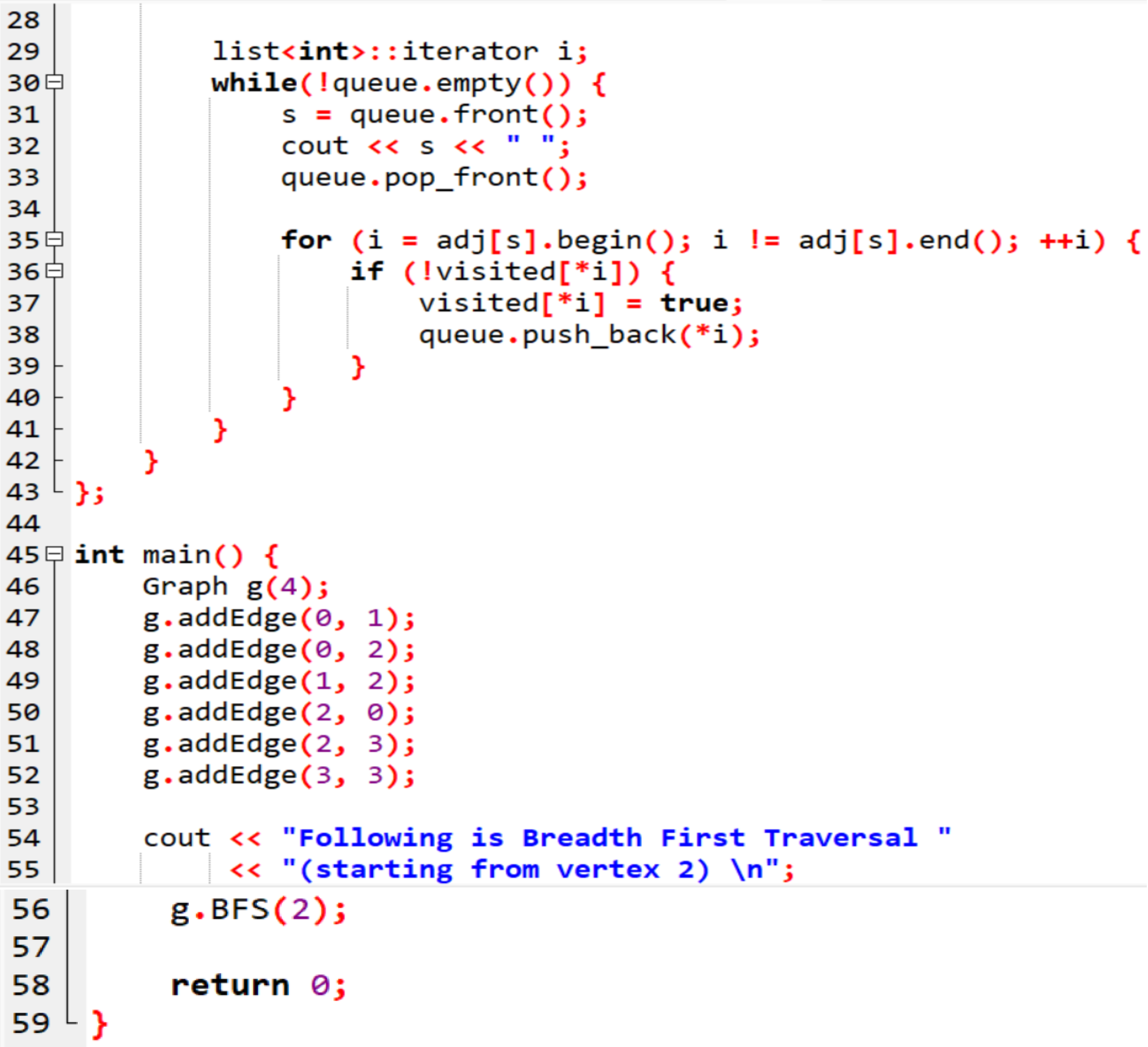
```
Original array: 4 2 2 8 3 3 1
Sorted array: 1 2 2 3 3 4 8

--------------------------------
Process exited after 0.1685 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

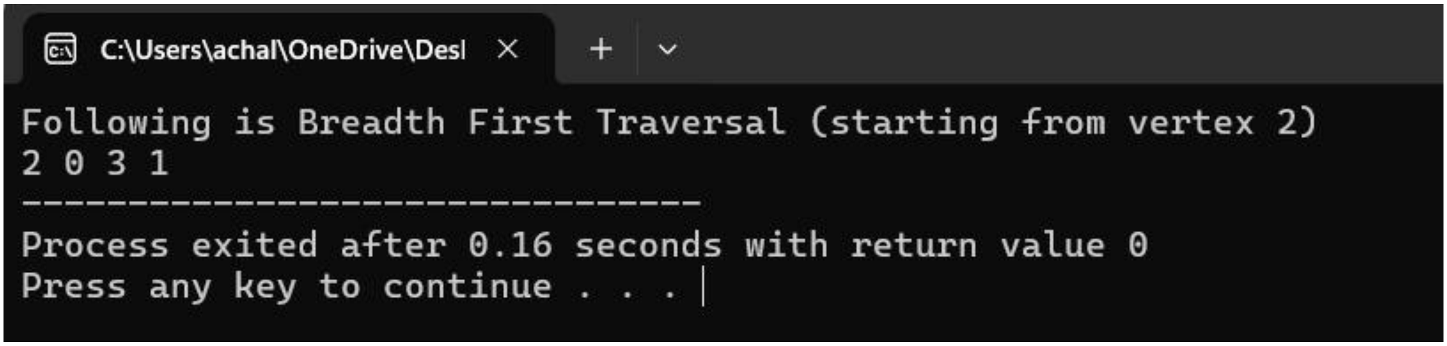


```
#include<iostream>
#include <list>

using namespace std;

class Graph {
    int V; // No. of vertices
    list<int> *adj;

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

    void addEdge(int v, int w) {
        adj[v].push_back(w); // Add w to v's list.
    }

    void BFS(int s) {
        bool *visited = new bool[V];
        for(int i = 0; i < V; i++)
            visited[i] = false;
        list<int> queue;

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

```



```cpp

        list<int>::iterator i;
        while(!queue.empty()) {
            s = queue.front();
            cout << s << " ";
            queue.pop_front();

            for (i = adj[s].begin(); i != adj[s].end(); ++i) {
                if (!visited[*i]) {
                    visited[*i] = true;
                    queue.push_back(*i);
                }
            }
        }
    }
};

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 << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

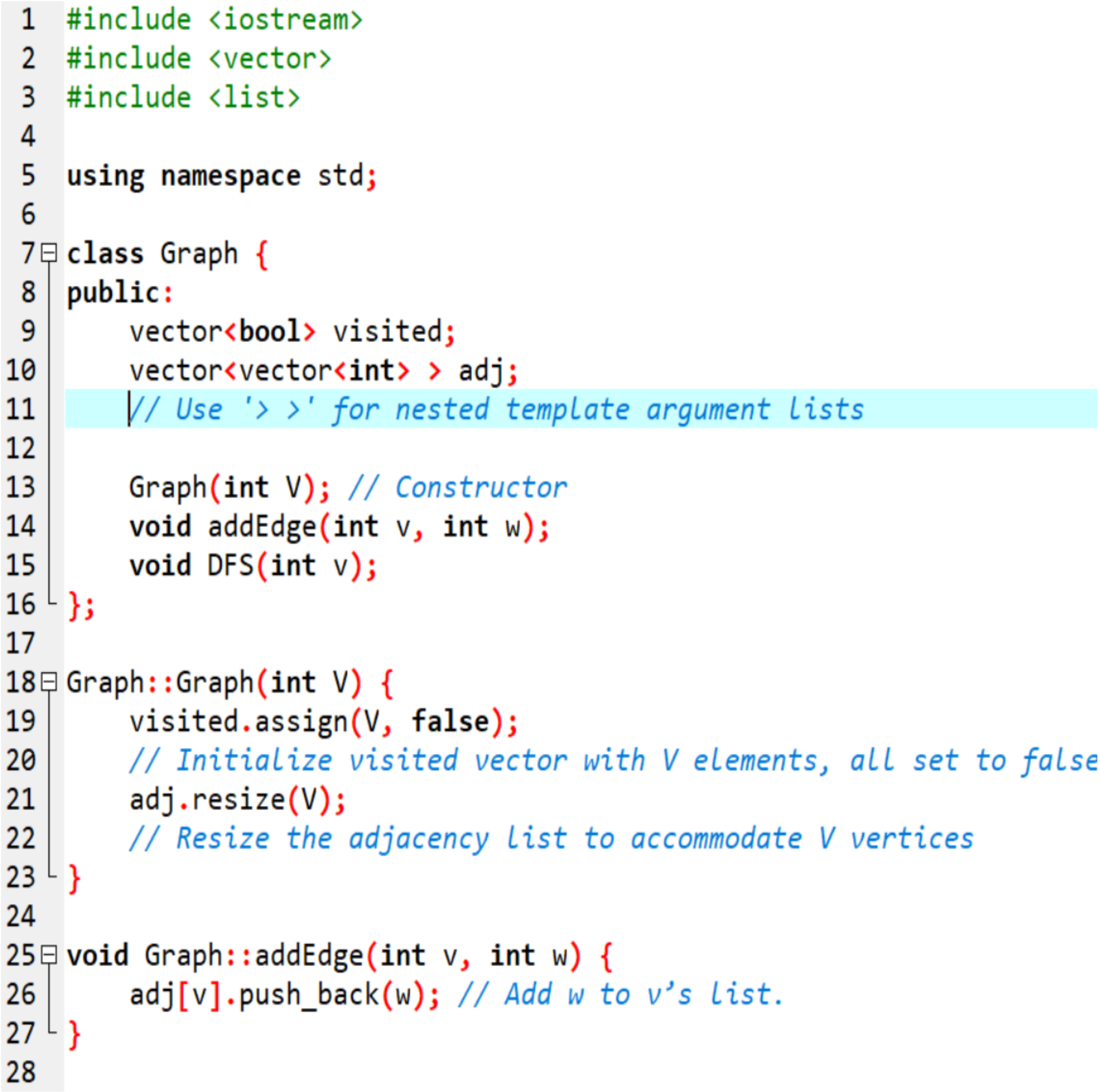
OUTPUT:



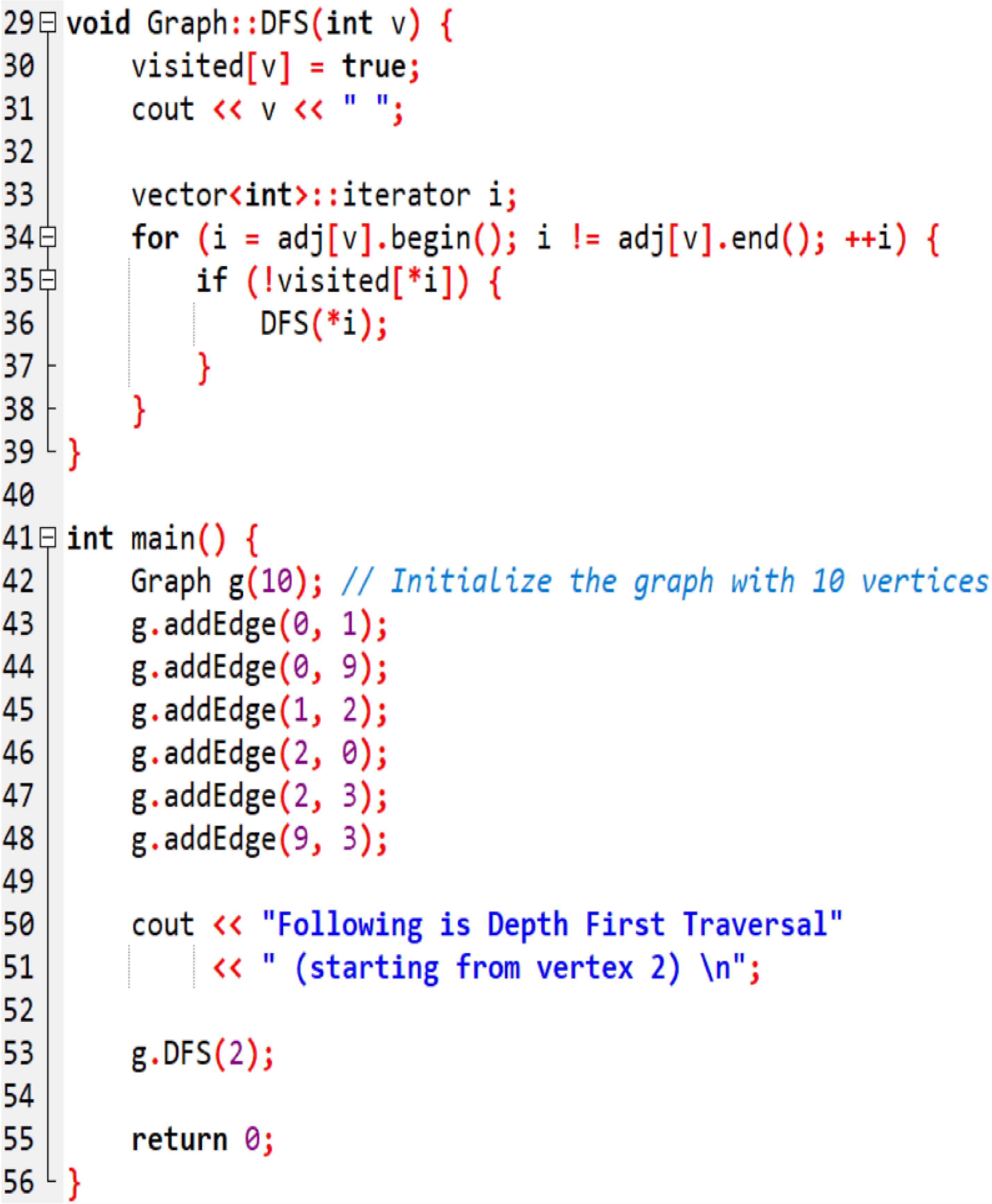
```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
--------------------------------
Process exited after 0.16 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

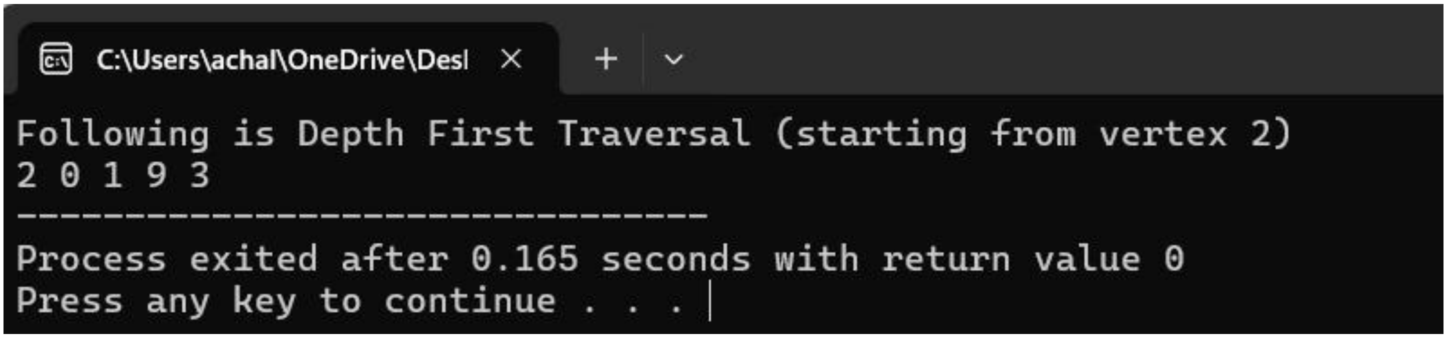


```
#include <iostream>
#include <vector>
#include <list>

using namespace std;

class Graph {
public:
    vector<bool> visited;
    vector<vector<int> > adj;
    // Use '> >' for nested template argument lists

    Graph(int V); // Constructor
    void addEdge(int v, int w);
    void DFS(int v);
};

Graph::Graph(int V) {
    visited.assign(V, false);
    // Initialize visited vector with V elements, all set to false
    adj.resize(V);
    // Resize the adjacency list to accommodate V vertices
}

void Graph::addEdge(int v, int w) {
    adj[v].push_back(w); // Add w to v’s list.
}

```



```cpp
void Graph::DFS(int v) {
    visited[v] = true;
    cout << v << " ";

    vector<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i) {
        if (!visited[*i]) {
            DFS(*i);
        }
    }
}

int main() {
    Graph g(10); // Initialize the graph with 10 vertices
    g.addEdge(0, 1);
    g.addEdge(0, 9);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(9, 3);

    cout << "Following is Depth First Traversal"
        << " (starting from vertex 2) \n";

    g.DFS(2);

    return 0;
}
```

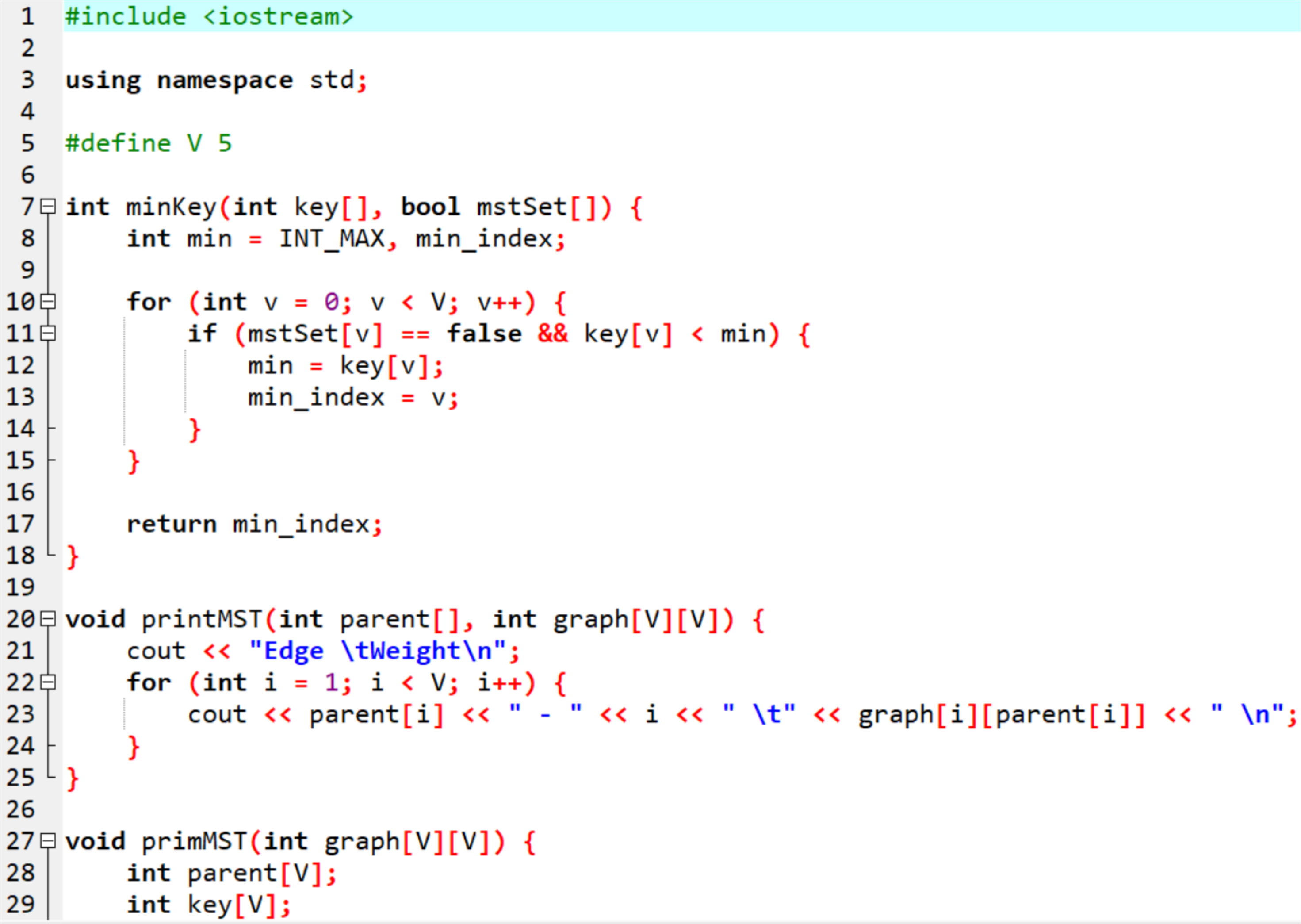
OUTPUT:



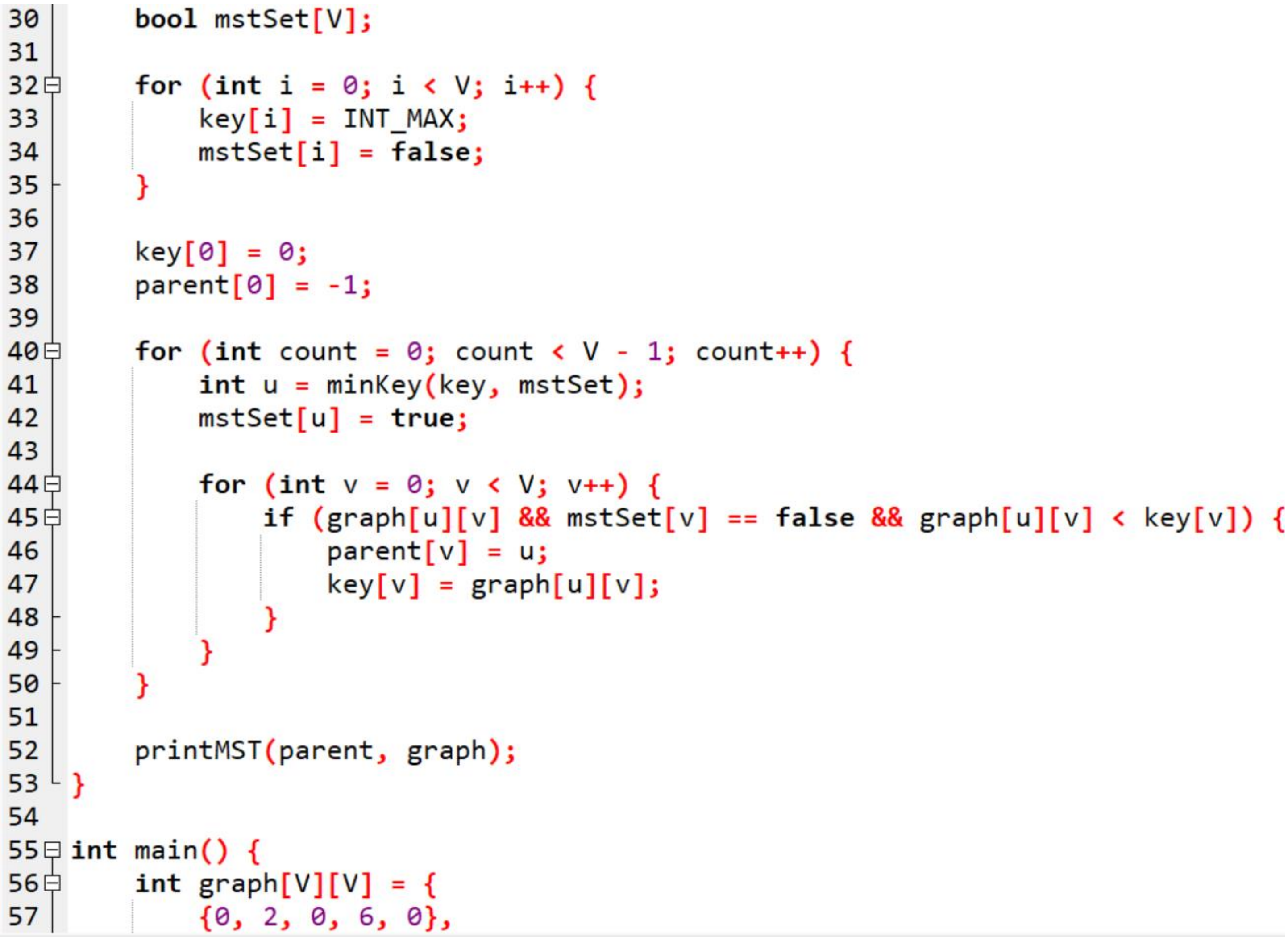
```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 9 3
--------------------------------
Process exited after 0.165 seconds with return value 0
Press any key to continue . . .
```

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

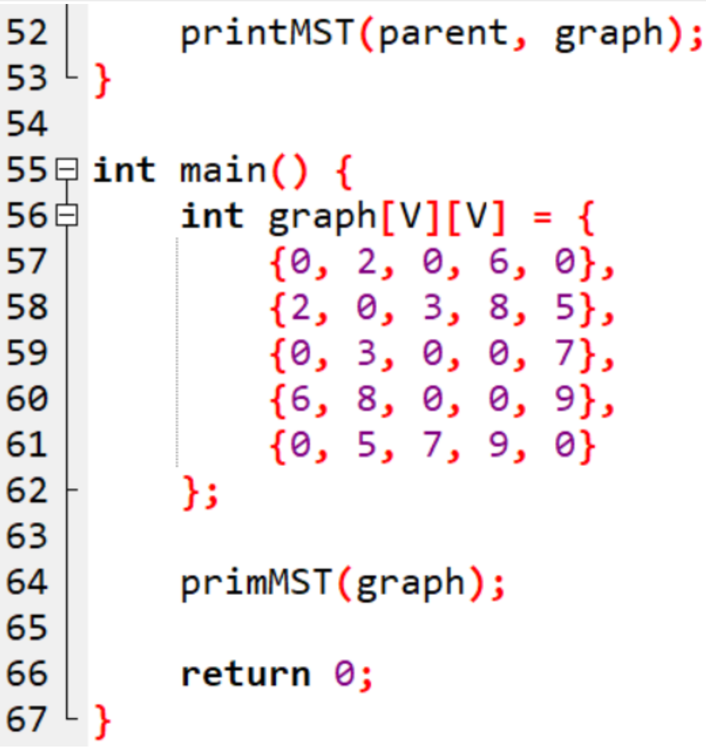
PROGRAM:



```cpp
#include <iostream>

using namespace std;

#define V 5

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, min_index;

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

    return min_index;
}

void printMST(int parent[], int graph[V][V]) {
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << " \n";
    }
}

void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
```

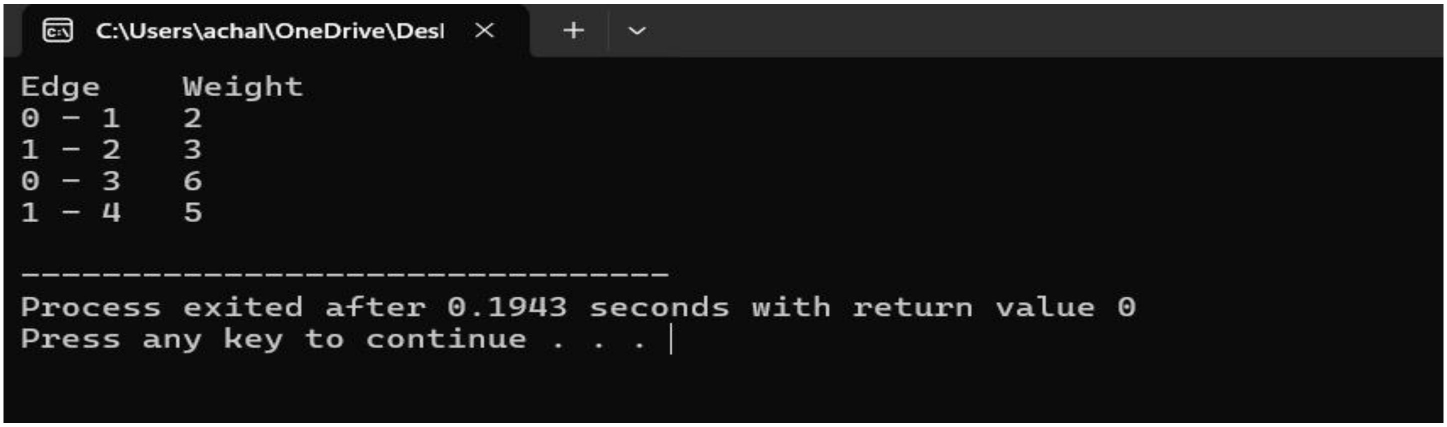


```
    bool mstSet[V];

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

    key[0] = 0;
    parent[0] = -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] == false && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
```



```
    printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {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}
    };

    primMST(graph);

    return 0;
}
```

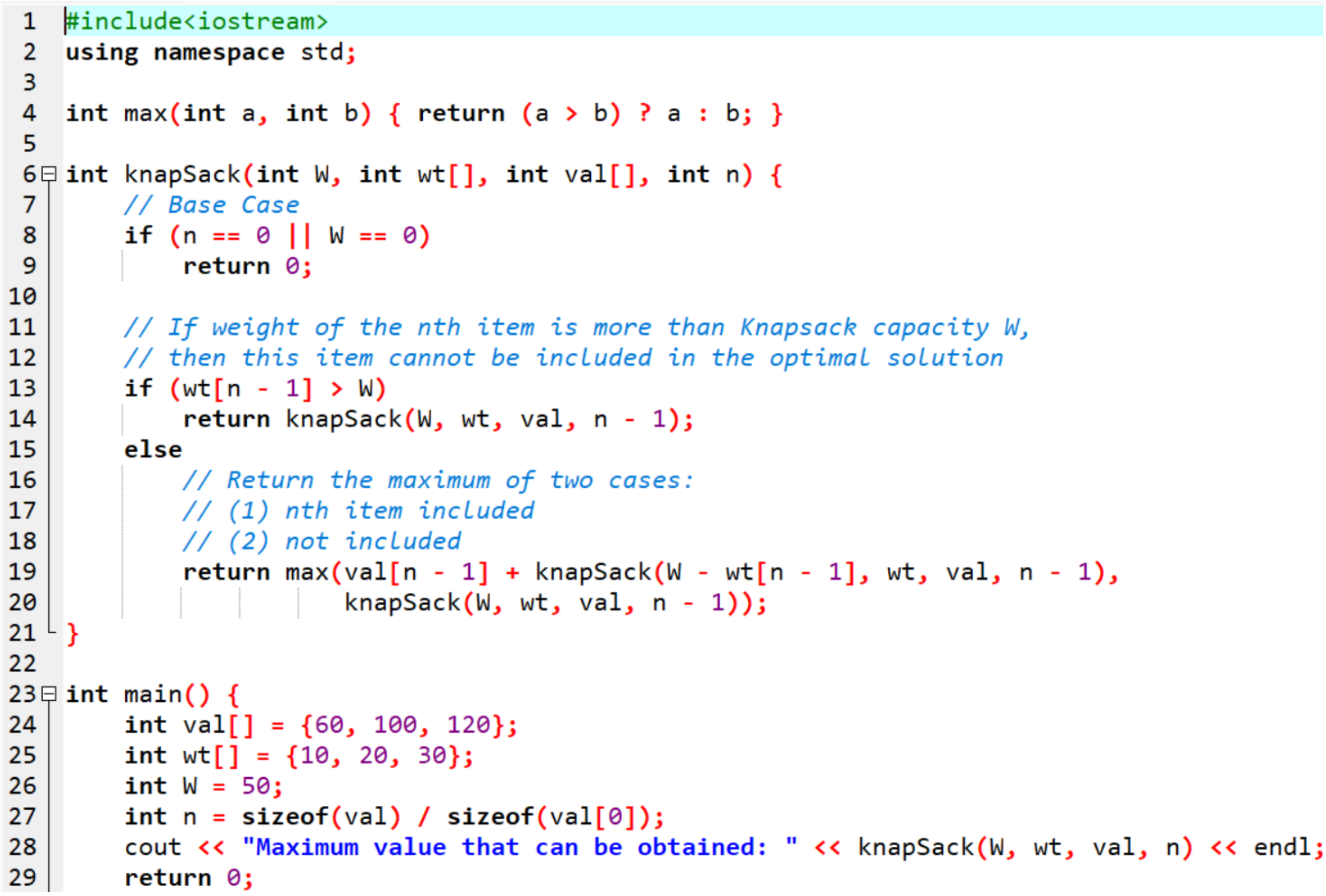
OUTPUT:



```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5

--------------------------------
Process exited after 0.1943 seconds with return value 0
Press any key to continue . . .
```

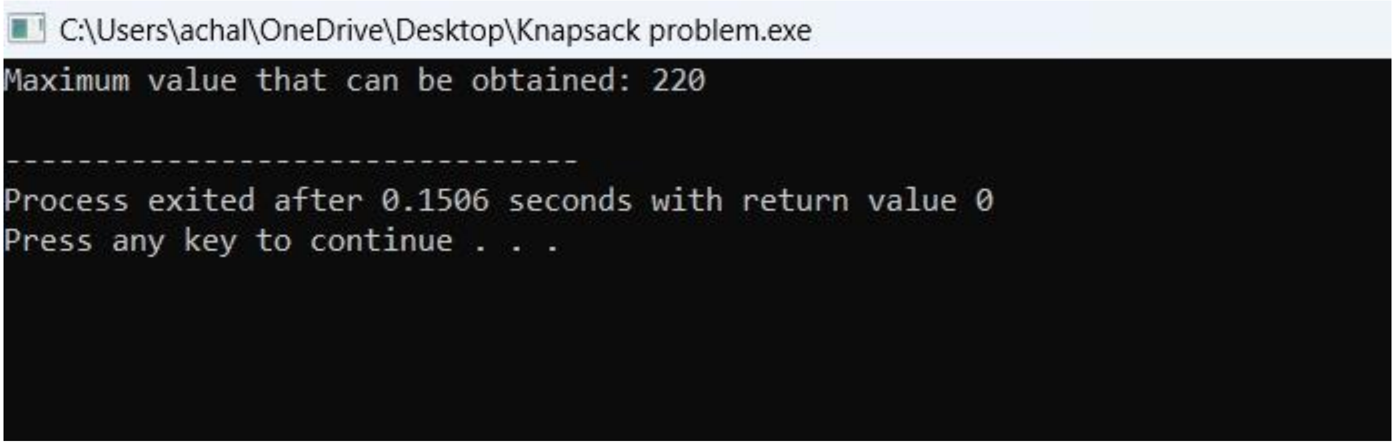
10) Write a program to solve the 0-1 knapsack problem.

PROGRAM:



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

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n) {
    // Base Case
    if (n == 0 || W == 0)
        return 0;

    // If weight of the nth item is more than Knapsack capacity W,
    // then this item cannot be included in the optimal solution
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);
    else
        // Return the maximum of two cases:
        // (1) nth item included
        // (2) not included
        return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),
                   knapSack(W, wt, val, n - 1));
}

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 obtained: " << knapSack(W, wt, val, n) << endl;
    return 0;
```

OUTPUT:



```
Maximum value that can be obtained: 220


--------------------------------
Process exited after 0.1506 seconds with return value 0
Press any key to continue . . .
```