Question 1.

Given f(n)= 7n+5, Write a program to prove that f(n)= O(n) and find the n0 value. Plot a graph for f(n) and c*g(n) where c is a constant and for varying n values (10 to 30).

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

int main() {
    int c = 8; // Choose c such that f(n) <= c*g(n)
    int n0 = 1;

    cout << "n0 values where 7n + 5 <= c*n for n in range 1 to 30:\n";
    for (int n = 1; n <= 30; ++n) {
        int fn = 7 * n + 5;
        int gn = c * n;
        if (fn <= gn)
            cout << "n = " << n << ", f(n) = " << fn << ", c*g(n) = " << gn << "\n";
    }

    return 0;
}
```

plot:

```
import matplotlib.pyplot as plt

n = list(range(10, 31))
f_n = [7 * i + 5 for i in n]
c = 8
g_n = [c * i for i in n]

plt.plot(n, f_n, label='f(n) = 7n + 5')
plt.plot(n, g_n, label='c*g(n) = 8n', linestyle='--')
plt.xlabel('n')
plt.ylabel('Function value')
plt.title('f(n) vs c*g(n)')
plt.legend()
plt.grid(True)
plt.show()
```

Given a set of men and women design and implement Gale- Shapeley algorithm to determine the stable set of marriages among them. Comment on the efficiency and the time complexity of the same.

```
#include <iostream>
#include <vector>
#include <queue>
#include <string>
```

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

int main() {
    vector<string> men = {"A", "B", "C"};
    vector<string> women = {"V", "W", "X"};

    map<string, vector<string>> men_pref = {
        {"A", {"V", "W", "X"}},
        {"B", {"W", "V", "X"}},
        {"C", {"V", "W", "X"}}
    };

    map<string, vector<string>> women_pref = {
        {"V", {"A", "B", "C"}},
        {"W", {"B", "C", "A"}},
        {"X", {"C", "A", "B"}}
    };

    map<string, string> engaged_to;
    map<string, bool> is_engaged;
    map<string, int> men_next;

    queue<string> free_men;
    for (auto m : men) {
        free_men.push(m);
        men_next[m] = 0;
    }

    auto prefers = [&](const string& woman, const string& new_man, const string&
current_man) {
        for (auto& m : women_pref[woman]) {
            if (m == new_man) return true;
            if (m == current_man) return false;
        }
        return false;
    };

    while (!free_men.empty()) {
        string m = free_men.front();
        free_men.pop();

        string w = men_pref[m][men_next[m]];
        men_next[m]++;

        if (!is_engaged[w]) {
            engaged_to[w] = m;
            is_engaged[w] = true;
        } else {
            string m1 = engaged_to[w];
            if (prefers(w, m, m1)) {
                engaged_to[w] = m;
                free_men.push(m1);
            } else {
                free_men.push(m);
            }
        }
    }

    cout << "Stable Marriages:\n";
    for (auto& pair : engaged_to)
        cout << pair.second << " - " << pair.first << "\n";

    return 0;
}
```

Question 2
Given f(n)= 3n2+4n+3, Write a program to prove that f(n)= Ω(n) and find the n0 value. Plot a graph for f(n) and c*g(n) where c is a constant and for varying n values (10 to 30).

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

int main() {
    int c = 3;  // Constant such that f(n) >= c * g(n)
    int n0 = 1;

    cout << "Verifying f(n) = 3n^2 + 4n + 3 >= c*n for n = 1 to 30:\n";
    for (int n = 1; n <= 30; ++n) {
        int fn = 3 * n * n + 4 * n + 3;
        int gn = c * n;
        if (fn >= gn)
            cout << "n = " << n << ", f(n) = " << fn << ", c*g(n) = " << gn << "\n";
    }

    return 0;
}
```

```
import matplotlib.pyplot as plt

n = list(range(10, 31))
f_n = [3 * i**2 + 4 * i + 3 for i in n]
c = 3
g_n = [c * i for i in n]

plt.plot(n, f_n, label='f(n) = 3n² + 4n + 3')
plt.plot(n, g_n, label='c*g(n) = 3n', linestyle='--')
plt.xlabel('n')
plt.ylabel('Function value')
plt.title('f(n) vs c·g(n) for Ω(n)')
plt.legend()
plt.grid(True)
plt.show()
```

A GPS navigation system needs an approach to discover the reachable areas in a given geographical region from a given source area. Design and implement an algorithm to find which nodes can be reached from a given source node for the following graph. Comment on the efficiency and the time complexity of the same

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

void bfs(int source, const vector<vector<int>>& adj, vector<bool>& visited) {
    queue<int> q;
    q.push(source);
    visited[source] = true;

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

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

int main() {
    int V = 6; // Number of nodes
    vector<vector<int>> adj(V);

    // Example Graph
    adj[0] = {1, 2};
    adj[1] = {0, 3};
    adj[2] = {0, 4};
    adj[3] = {1, 5};
    adj[4] = {2};
    adj[5] = {3};

    int source = 0;
    vector<bool> visited(V, false);

    cout << "Reachable nodes from source " << source << ":\n";
    bfs(source, adj, visited);

    return 0;
}
```

Question 3:

a)Given f(n)= $7n^2+7n+5$, Write a program to prove that f(n)= $\theta(n^2)$ and find the n_0 value. Plot a graph for f(n), c_1*g(n)and c_2*g(n) where c_1, $c_2$is a constant and for varying n values (10 to 30).

b)Design and implement merge sort algorithm that takes random number input and displays the execution time required. State the design strategy used and time complexity of the same.

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

int main() {
    int c1 = 7, c2 = 9;  // Reasonable bounding constants
    cout << "Verifying c1*g(n) <= f(n) <= c2*g(n) for n from 10 to 30:\n";

    for (int n = 10; n <= 30; ++n) {
        int fn = 7 * n * n + 7 * n + 5;
        int lower = c1 * n * n;
        int upper = c2 * n * n;
        cout << "n = " << n << ", f(n) = " << fn
             << ", c1*g(n) = " << lower << ", c2*g(n) = " << upper;
        if (fn >= lower && fn <= upper)
            cout << " ✅\n";
        else
            cout << " ❌\n";
    }

    return 0;
}
```

```
import matplotlib.pyplot as plt

n = list(range(10, 31))
f_n = [7 * i**2 + 7 * i + 5 for i in n]
c1 = 7
c2 = 9
g_n_c1 = [c1 * i**2 for i in n]
g_n_c2 = [c2 * i**2 for i in n]

plt.plot(n, f_n, label='f(n) = 7n² + 7n + 5')
plt.plot(n, g_n_c1, label='c1·g(n) = 7n²', linestyle='--')
plt.plot(n, g_n_c2, label='c2·g(n) = 9n²', linestyle='--')
plt.xlabel('n')
plt.ylabel('Function value')
plt.title('f(n) vs c1·g(n) and c2·g(n)')
plt.legend()
plt.grid(True)
plt.show()
```

b)

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <chrono>
using namespace std;
using namespace std::chrono;

// Merge function
void merge(vector<int>& arr, int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; ++i) L[i] = arr[l + i];
    for (int i = 0; i < n2; ++i) R[i] = arr[m + 1 + i];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2)
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

// Merge Sort function
void mergeSort(vector<int>& arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}

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

    vector<int> arr(n);
    srand(time(0));
    for (int i = 0; i < n; ++i)
        arr[i] = rand() % 1000;

    auto start = high_resolution_clock::now();
    mergeSort(arr, 0, n - 1);
    auto stop = high_resolution_clock::now();

    auto duration = duration_cast<microseconds>(stop - start);
    cout << "Execution time: " << duration.count() << " microseconds\n";

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

    return 0;
}
```

Question 4:

a)Given f(n)= 4n+3, Write a program to prove that f(n)= O(n) and find the n0 value. Plot a graph for f(n) and c*g(n) where c is a constant and for varying n values (10 to 30).

b)Three users in an online music portal listen to a playlist of 8 songs that are numbered from 1 to 8 in a random order. Each user needs to be recommended to another user playlist's order that has minimum number of inversions. Design and implement an algorithm to determine the number of inversions. State the design strategy used and time complexity of the same.

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

int main() {
    int c = 5;
    cout << "Checking if f(n) = 4n + 3 <= c·n for c = " << c << "\n";
    for (int n = 10; n <= 30; ++n) {
        int fn = 4 * n + 3;
        int gn = c * n;
        cout << "n = " << n << ", f(n) = " << fn << ", c·g(n) = " << gn;
        if (fn <= gn)
            cout << "  ✅\n";
        else
            cout << "  ❌\n";
    }

    return 0;
}
```

```
import matplotlib.pyplot as plt

n = list(range(10, 31))
f_n = [4 * i + 3 for i in n]
c = 5
g_n = [c * i for i in n]

plt.plot(n, f_n, label='f(n) = 4n + 3')
plt.plot(n, g_n, label='c*g(n) = 5n', linestyle='--')
plt.xlabel('n')
plt.ylabel('Function value')
plt.title('f(n) vs c·g(n) for O(n)')
plt.legend()
plt.grid(True)
plt.show()
```

Counting inversions:

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

// Merge function to count inversions
int mergeAndCount(vector<int>& arr, int l, int m, int r) {
    int inv = 0;
    int n1 = m - l + 1, n2 = r - m;

    vector<int> left(arr.begin() + l, arr.begin() + m + 1);
    vector<int> right(arr.begin() + m + 1, arr.begin() + r + 1);

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (left[i] <= right[j])
            arr[k++] = left[i++];
        else {
            arr[k++] = right[j++];
            inv += n1 - i; // Count inversions
        }
    }

    while (i < n1) arr[k++] = left[i++];
    while (j < n2) arr[k++] = right[j++];

    return inv;
}

// Recursive function
int countInversions(vector<int>& arr, int l, int r) {
    int inv = 0;
    if (l < r) {
        int m = l + (r - l) / 2;
        inv += countInversions(arr, l, m);
        inv += countInversions(arr, m + 1, r);
        inv += mergeAndCount(arr, l, m, r);
    }
    return inv;
}

int main() {
    vector<vector<int>> playlists = {
        {1, 3, 5, 7, 2, 4, 6, 8}, // User A
        {2, 1, 4, 3, 6, 5, 8, 7}, // User B
        {1, 2, 3, 4, 5, 6, 7, 8}  // User C
    };
    for (int i = 0; i < 3; ++i) {
        cout << "User " << char('A' + i) << "'s recommendations:\n";
        for (int j = 0; j < 3; ++j) {
            if (i == j) continue;

            vector<int> ref = playlists[j];
            // Map values of user's playlist to reference order
            vector<int> mapped(8);
            for (int k = 0; k < 8; ++k) {
                for (int m = 0; m < 8; ++m) {
                    if (playlists[i][k] == ref[m]) {
                        mapped[k] = m;
                        break;
                    }
                }
            }

            int inv = countInversions(mapped, 0, 7);
            cout << "  Compared to User " << char('A' + j) << ": " << inv << " inversions\n";
        }
    }

    return 0;
}
```

Question 5:

a)Given f(n)= 2n+3n+5, Write a program to prove that f(n)= Ω(n) and find the n0 value. Plot a graph for f(n) and c*g(n) where c is a constant and for varying n values (10 to 30).

b)In a database of numbers there is a table of unsorted numbers. The database admin now wants to sort these numbers using an approach wherein a pivot element is selected for sorting. At certain point, the first half elements are less than the pivot and right half elements are greater than the pivot. Design and implement an algorithm to solve it using random numbers and also display the execution time. State the design strategy used and time complexity of the same.

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

int main() {
    int c = 5;
    cout << "Verifying f(n) = 5n + 5 >= c·n for c = " << c << "\n";
    for (int n = 10; n <= 30; ++n) {
        int fn = 5 * n + 5;
        int gn = c * n;
        cout << "n = " << n << ", f(n) = " << fn << ", c·g(n) = " << gn;
        if (fn >= gn)
            cout << "  ✅\n";
        else
            cout << "  ❌\n";
    }

    return 0;
}
```

```
import matplotlib.pyplot as plt

n = list(range(10, 31))
f_n = [5 * i + 5 for i in n]
c = 5
g_n = [c * i for i in n]

plt.plot(n, f_n, label='f(n) = 5n + 5')
plt.plot(n, g_n, label='c·g(n) = 5n', linestyle='--')
plt.xlabel('n')
plt.ylabel('Function value')
plt.title('f(n) vs c·g(n) for Ω(n)')
plt.legend()
plt.grid(True)
plt.show()
```

Quick sort:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <chrono>
using namespace std;
using namespace std::chrono;

// Partition function
int partition(vector<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) {
        if (arr[j] < pivot) {
            ++i;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

// Quick Sort function
void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int p = partition(arr, low, high);
        quickSort(arr, low, p - 1);
        quickSort(arr, p + 1, high);
    }
}

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

    vector<int> arr(n);
    srand(time(0));
    for (int i = 0; i < n; ++i)
        arr[i] = rand() % 1000;

    auto start = high_resolution_clock::now();
    quickSort(arr, 0, n - 1);
    auto stop = high_resolution_clock::now();

    auto duration = duration_cast<microseconds>(stop - start);
    cout << "Execution time: " << duration.count() << " microseconds\n";

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

    return 0;
}
```

Question 6:

a)Given f(n)= 8n2+3n+3, Write a program to prove that f(n)= O(n2) and find the n0 value. Plot a graph for f(n) and c*g(n) where c is a constant and for varying n values (10 to 30).

b)A truck driver is given a set of locations to be covered with their distances by a company. The company strictly orders that truck should be started from a particular location.Design and implement an algorithm that gives a greedy solution to the truck driver's problem and display the shortest path for a given source location to all other locations. State the design strategy used and time complexity of the same.

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

// Function definitions
int f(int n) {
    return 8 * n * n + 3 * n + 3;
}

int g(int n) {
    return n * n;
}
```

```
int main() {
    const int c = 9; // chosen constant
    int n0 = -1;

    // Find smallest n0 such that f(n) <= c * g(n) for all n >= n0
    for (int n = 1; n < 100; ++n) {
        bool holds = true;
        for (int k = n; k < 100; ++k) {
            if (f(k) > c * g(k)) {
                holds = false;
                break;
            }
        }
        if (holds) {
            n0 = n;
            break;
        }
    }

    cout << "f(n) <= " << c << "*n^2 for all n >= " << n0 << endl;

    // Output values to a CSV file for plotting
    ofstream fout("data.csv");
    fout << "n,f_n,cg_n\n";
    for (int n = 10; n <= 30; ++n) {
        fout << n << "," << f(n) << "," << c * g(n) << "\n";
    }
    fout.close();

    return 0;
}
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data from CSV
data = pd.read_csv("data.csv")

# Plot
plt.figure(figsize=(10, 6))
plt.plot(data['n'], data['f_n'], label='f(n) = 8n^2 + 3n + 3', marker='o')
plt.plot(data['n'], data['cg_n'], label='c * n^2 (c = 9)', linestyle='--', marker='x')

plt.title("f(n) vs c * n^2 (Proving Big-O)")
plt.xlabel("n")
plt.ylabel("Value")
plt.grid(True)
plt.legend()
plt.show()
```

Dijkistra’s algorithm:
```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

using namespace std;

// Structure to represent an edge to a neighbor with weight
struct Edge {
    int to;
    int weight;
};

using Graph = vector<vector<Edge>>;

void dijkstra(const Graph& graph, int source) {
    int n = graph.size();
    vector<int> dist(n, INT_MAX);  // Distance from source to each node
    vector<bool> visited(n, false);
    dist[source] = 0;

    // Min-heap priority queue: (distance, node)
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
    pq.push({0, source});

    while (!pq.empty()) {
        int current = pq.top().second;
        pq.pop();

        if (visited[current])
            continue;

        visited[current] = true;

        for (const Edge& edge : graph[current]) {
            int neighbor = edge.to;
            int weight = edge.weight;

            if (dist[current] + weight < dist[neighbor]) {
                dist[neighbor] = dist[current] + weight;
                pq.push({dist[neighbor], neighbor});
            }
        }
    }

    // Output the shortest distances
    cout << "Shortest distances from source node " << source << ":\n";
    for (int i = 0; i < n; ++i) {
        if (dist[i] == INT_MAX)
            cout << "Node " << i << ": Unreachable\n";
        else
```

```
            cout << "Node " << i << ": " << dist[i] << "\n";
    }
}

int main() {
    // Hardcoded graph with 5 nodes
    // Edges are bidirectional (undirected graph)
    int n = 5; // number of nodes
    Graph graph(n);

    // Adding edges: (u, v, weight)
    graph[0].push_back({1, 2});
    graph[0].push_back({2, 4});
    graph[1].push_back({0, 2});
    graph[1].push_back({2, 1});
    graph[1].push_back({3, 7});
    graph[2].push_back({0, 4});
    graph[2].push_back({1, 1});
    graph[2].push_back({4, 3});
    graph[3].push_back({1, 7});
    graph[3].push_back({4, 1});
    graph[4].push_back({2, 3});
    graph[4].push_back({3, 1});

    int source = 0; // Hardcoded source node

    dijkstra(graph, source);

    return 0;
}
```