1.Transportation

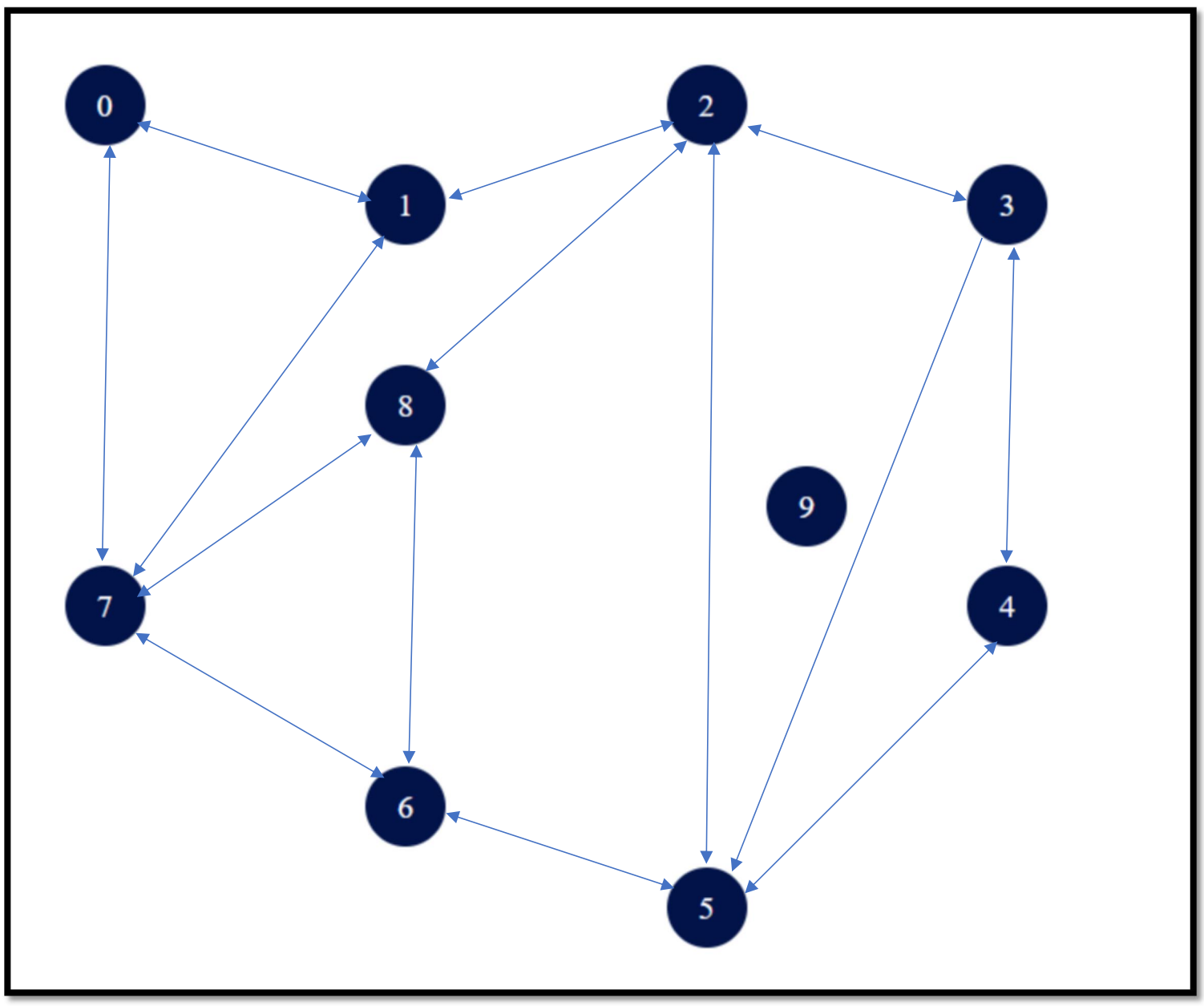
I .Sub-task: Find the shortest path for buses to take across the city to reduce emissions and increase efficiency.

SDG Goal: 11 (Sustainable Cities and Communities)
Target: 11.2
Indicator: 11.2.1

Description:
Optimizing public transportation routes in the city to reduce travel time and vehicle emissions. This will help make public transport more efficient, encouraging more people to use it, leading to a reduction in traffic congestion and air pollution.

Input Data:

Represented the city as a graph, where each bus station is a node and the routes between them are weighted edges (distance or time).



Bus routes with distances between stations as edges

From/To	0	1	2	3	4	5	6	7	8	9
0	0	4	INF	INF	INF	INF	INF	8	INF	INF
1	4	0	8	INF	INF	INF	INF	11	INF	INF
2	INF	8	0	7	INF	4	INF	INF	2	INF
3	INF	INF	7	0	9	14	INF	INF	INF	INF
4	INF	INF	INF	9	0	10	INF	INF	INF	INF
5	INF	INF	4	14	10	0	2	INF	INF	INF
6	INF	INF	INF	INF	INF	2	0	1	6	INF
7	8	11	INF	INF	INF	INF	1	0	7	INF
8	INF	INF	2	INF	INF	INF	6	7	0	INF
9	INF	INF	INF	INF	INF	INF	INF	INF	INF	0

Algorithm Used:

Dijkstra's Algorithm for shortest path

Heap data structure for efficient priority queue operation

Code:

1.Dijekstra's code

```
 #include < iostream >
using namespace std;
int v = 5;
int m[10][10] = {{0,1,1,0,0}, {1,0,0,1,1},
{1,0,0,0,1}, {0,1,0,0,0}, {0,1,1,0,0}};
int visited[10];
void dfs(int m[10][10], int v, int source) {
visited[source] = 1;
for (int i = 0; i < v; i++) {
if (m[source][i] == 1 && visited[i] == 0) {
cout << i << "    ";
dfs(m, v, i);
}
}
}
int main() {
int source;
for (int i = 0; i < v; i++)
visited[i] = 0;
```

```
cout << "Enter the source vertex: ";
cin >> source;

cout << "The DFS Traversal is...
";
cout << source << "  ";
dfs(m, v, source);

return 0;
}
```

2.Heap

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

class MaxHeap {
private:
vector heap;

void HeapifyUp(int i) {
    while (i > 1 && heap[i / 2] < heap[i]) {
        swap(heap[i], heap[i / 2]);
        i = i / 2;
    }
}

void HeapifyDown(int i) {
    int n = heap.size() - 1;
    int v = heap[i];
    bool isHeap = false;
```

```
while (!isHeap && 2 * i <= n) {
    int j = 2 * i;

    if (j < n && heap[j] < heap[j + 1])
        j++;

    if (v >= heap[j])
        isHeap = true;
    else {
        heap[i] = heap[j];
            i = j;
        }
    }
    heap[i] = v;
}

public:
MaxHeap() {
    heap.push_back(-1); // Placeholder to simplify index calculations
}

void insert(int value) {
    heap.push_back(value);
    HeapifyUp(heap.size() - 1);
}

void deleteMax() {
    if (heap.size() > 1) {
        cout << "Deleted root: " << heap[1] << endl;
        heap[1] = heap.back();
        heap.pop_back();
```

```
        if (heap.size() > 1) {
            HeapifyDown(1);
        }
    } else {
        cout << "Heap is empty!" << endl;
    }
}

void printHeap() {
    if (heap.size() > 1) {
        cout << "Heap elements: ";
        for (size_t i = 1; i < heap.size(); i++) {
            cout << heap[i] << " ";
        }
        cout << endl;
    } else {
        cout << "Heap is empty!" << endl;
    }
}

void printRoot() {
    if (heap.size() > 1) {
        cout << "Root element (max): " << heap[1] << endl;
    } else {
        cout << "Heap is empty!" << endl;
    }
}
};

int main() {
MaxHeap maxHeap;
```

```
int choice, value;

do {
    cout << "Menu:";
    cout << "1. Insert";
    cout << "2. Delete Root";
    cout << "3. Print Heap";
    cout << "4. Print Root";
    cout << "5. Exit";
    cout << "Enter your choice: ";
    cin >> choice;

switch (choice) {
    case 1:
        cout << "Enter value to insert: ";
        cin >> value;
        maxHeap.insert(value);
        break;
    case 2:
        maxHeap.deleteMax();
        break;
    case 3:
        maxHeap.printHeap();
        break;
      case 4:
        maxHeap.printRoot();
        break;
      case 5:
        cout << "Exit
";
        break;
```

```
        default:
            cout << "Invalid choice! Please try again.
";
    }
} while (choice != 5);

return 0;
}
```

Output:

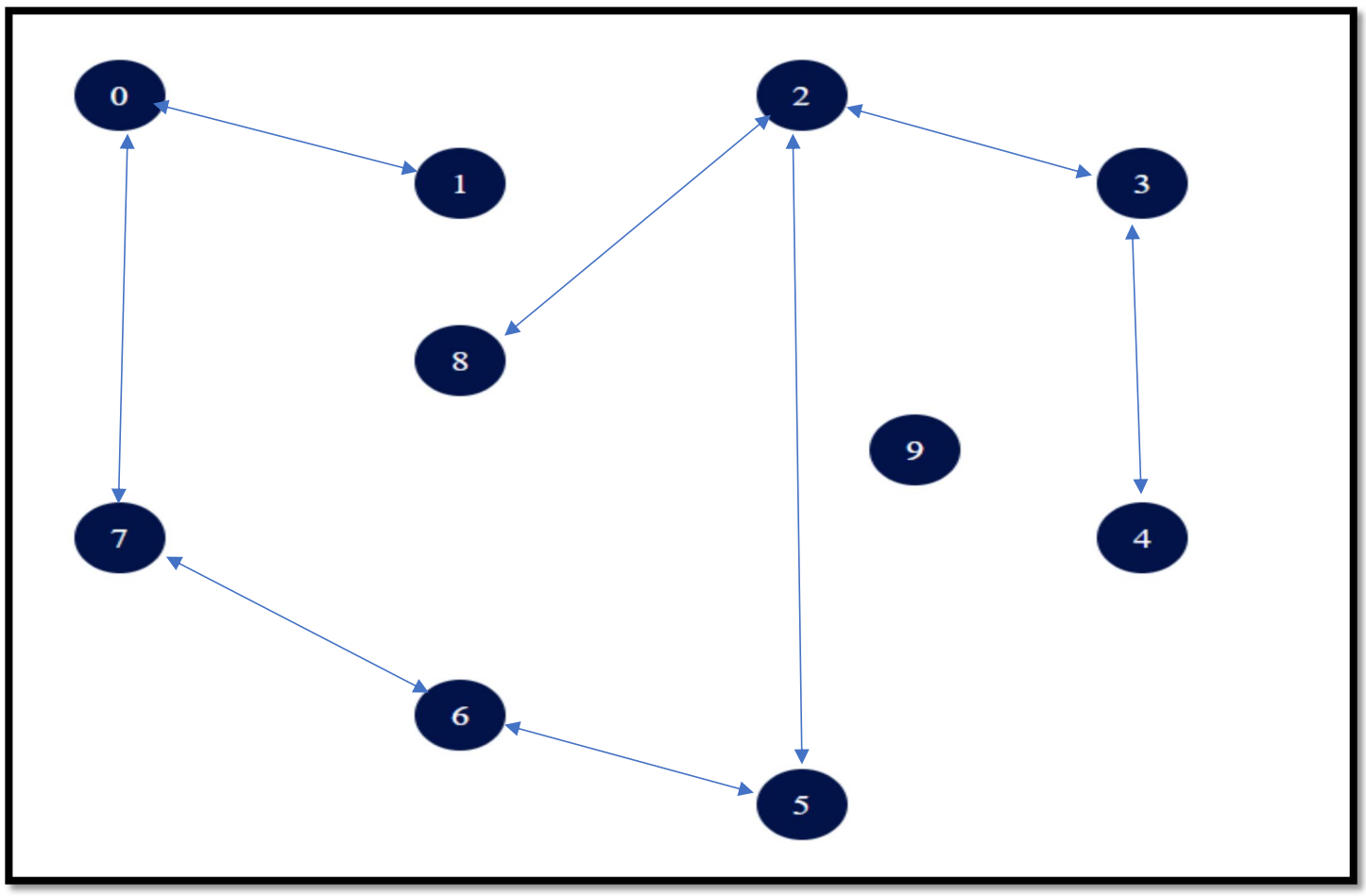
Shortest path from the starting bus station to the destination

Node	Distance
0	0
1	4
2	12
3	19
4	21
5	11
6	13
7	8
8	14
9	INF

Graph Representation:

From the algorithm, the edges contributing to the shortest path tree:

1. 0 -> 1 (4)
2. 0 -> 7 (8)
3. 7 -> 6 (1)
4. 6 -> 5 (2)
5. 5 -> 2 (4)
6. 2 -> 8 (2)
7. 2 -> 3 (7)
8. 3 -> 4 (9)



II. Public Transport Fleet Optimization

Sub-task: Sort bus or train schedules to minimize wait times for passengers.
SDG Goal: 11 (Sustainable Cities and Communities)
Target: 11.2
Indicator: 11.2.1
Description:
Arrange public transport schedules efficiently to reduce passenger wait times.

Input Data:

- Timings of public transport services.

Vehicle ID	Departure Time
201	9:15 AM
202	7:30 AM
203	11:00 AM
204	10:45 AM
205	8:05 AM
206	6:50 AM
207	8:30 AM
208	7:45 AM
209	9:25 AM
210	9:10 AM
211	8:00 AM
212	7:55 AM
213	8:50 AM
214	6:40 AM
215	10:00 AM
216	7:00 AM
217	9:40 AM
218	8:40 AM
219	6:30 AM
220	9:50 AM

Algorithm Used:

- **Insertion Sort** for sorting schedules.

  Code:

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


void insertElement(int arr[], int &size, int element, int position) {
    for (int i = size; i >= position; i--) {
        arr[i] = arr[i - 1];
    }
    arr[position - 1] = element;
    size++;
}

int main() {
    int arr[10] = {1, 2, 3, 4, 5};
    int size = 5;
    int element = 6;
    int position = 3;
```

```
insertElement(arr, size, element, position);

cout << "Updated array: ";
for (int i = 0; i < size; i++) {
    cout << arr[i] << " ";
}
cout << endl;

return 0;
}
```

Output:

Vehicle ID	Departure Time
219	**6:30 AM**
214	6:40 AM
206	6:50 AM
216	7:00 AM
202	7:30 AM
208	7:45 AM
212	7:55 AM
211	8:00 AM
205	8:05 AM
207	8:30 AM
213	8:50 AM
218	8:40 AM
201	9:15 AM
210	9:10 AM
209	9:25 AM
204	10:45 AM
215	10:00 AM
203	11:00 AM
220	9:50 AM
217	9:40 AM

3.Carpooling System for Sustainable Transportation

Sub-task: Match carpooling partners based on proximity and similar routes to reduce the number of vehicles on the road.

SDG Goal: 11 (Sustainable Cities and Communities)
Target: 11.6
Indicator: 11.6.1

Description:
The carpooling system aims to reduce the number of cars on the road, decrease

traffic congestion, and minimize carbon emissions. By matching users with similar routes and nearby locations, the system promotes sustainable urban mobility.

Input Data:

User ID	Starting Location	Destination
U1	Location A	Location D
U2	Location B	Location E
U3	Location A	Location D
U4	Location C	Location F
U5	Location B	Location E

Algorithm Used:

Union-Find (Disjoint Set Union, DSU) for clustering users with similar travel routes

Code:

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

class UnionFind {
private:
    std::vector<int> parent;
    std::vector<int> rank;

public:
    UnionFind(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
    }

    int find(int x) {
```

```
    if (parent[x] != x) {
        parent[x] = find(parent[x]);
    }
    return parent[x];
}

void unionSets(int x, int y) {
    int rootX = find(x);
    int rootY = find(y);

    if (rootX != rootY) {
        if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
        } else if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        } else {
            parent[rootY] = rootX;
            rank[rootX]++;
        }
    }
}

    bool isConnected(int x, int y) {
        return find(x) == find(y);
    }
};

void clusterUsers(const std::vector<std::pair<int, int>>& similarRoutes, int
userCount) {
    UnionFind uf(userCount);
```

```
for (const auto& route : similarRoutes) {
    uf.unionSets(route.first, route.second);
}

std::unordered_map<int, std::vector<int>> clusters;
for (int i = 0; i < userCount; ++i) {
    int root = uf.find(i);
    clusters[root].push_back(i);
}
    std::cout << "User Clusters:\n";
    for (const auto& cluster : clusters) {
        std::cout << "Cluster with root " << cluster.first << ": ";
        for (int user : cluster.second) {
            std::cout << user << " ";
        }
        std::cout << "\n";
    }
}

int main() {
    int userCount = 6;

    std::vector<std::pair<int, int>> similarRoutes = {
        {0, 1}, {1, 2}, {3, 4}, {4, 5}
    };

    clusterUsers(similarRoutes, userCount);

    return 0;
}
```

Output:

Carpool Group	Users	Starting Point	Destination	Route
Group 1	U1, U3	Location A	Location D	Shortest Route
Group 2	U2, U5	Location B	Location E	Shortest Route
Group 3	U4	Location C	Location F	Shortest Route

Tourism

1.Promoting Eco-Friendly Transport Options for Tourists

Sub-task: Recommend eco-friendly transport options for tourists based on available routes and distances. **SDG Goal:** 11 (Sustainable Cities and Communities)

Target: 11.2

Indicator: 11.2.1

Description:

The goal is to promote sustainable transport options like electric buses, bikes, and walking routes for tourists, thereby reducing the carbon footprint associated with tourism activities.

Input Data:

Spot A	Spot B	Distance (km)	Mode
Spot 1	Spot 2	5	Electric Bus
Spot 1	Spot 3	3	Bike
Spot 2	Spot 3	4	Walking Path
Spot 3	Spot 4	2	Electric Bus
Spot 2	Spot 4	6	Bike

Algorithm Used:

Prim's Algorithm for finding the minimum spanning tree

Code:

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

struct Edge {
    int from, to, weight;
    string mode;
```

```
};

void findEcoFriendlyRoutes(int n, vector<Edge> edges) {
    vector<vector<pair<int, pair<int, string>>>> graph(n + 1);

    for (const auto& edge : edges) {
        graph[edge.from].push_back({edge.to, {edge.weight, edge.mode}});
        graph[edge.to].push_back({edge.from, {edge.weight, edge.mode}});
    }

    vector<bool> visited(n + 1, false);
    priority_queue<pair<int, pair<int, string>>, vector<pair<int, pair<int,
string>>>, greater<>> pq;
    pq.push({0, {1, ""}});
    int totalCost = 0;

    cout << "Optimized Routes:\n";
    while (!pq.empty()) {
        auto [cost, nodeInfo] = pq.top();
        int node = nodeInfo.first;
        string mode = nodeInfo.second;
        pq.pop();

        if (visited[node]) continue;
        visited[node] = true;
        if (!mode.empty()) {
            cout << "Spot " << node << ": Mode - " << mode << ", Distance - " << cost
<< " km\n";
            totalCost += cost;
        }

        for (const auto& [neighbor, details] : graph[node]) {
```

```
            int weight = details.first;
            string transportMode = details.second;
            if (!visited[neighbor]) {
                pq.push({weight, {neighbor, transportMode}});
            }
        }
    }
    cout << "Total Distance: " << totalCost << " km\n";
}

int main() {
    int n = 4;
    vector<Edge> edges = {
        {1, 2, 5, "Electric Bus"},
        {1, 3, 3, "Bike"},
        {2, 3, 4, "Walking Path"},
        {3, 4, 2, "Electric Bus"},
        {2, 4, 6, "Bike"}
    };

    findEcoFriendlyRoutes(n, edges);
    return 0;
}
```

Output:

Route	Transport Mode	Distance (km)
Spot 1 → Spot 3	Bike	3
Spot 3 → Spot 4	Electric Bus	2
Spot 1 → Spot 2	Electric Bus	5

2. Travel Package Customization

Sub-task: Sort travel packages by price to suit diverse customer budgets.

SDG Goal: 11 (Sustainable Cities and Communities)

Target: 11.4

Indicator: 11.4.1

Description: Sort travel packages in ascending order to help customers choose budget-friendly options while promoting sustainable travel practices.

Input Data:

- Travel package details with pricing.

Package ID	Package Name	Price (INR)	Duration (Days)	Destination
101	Goa Beach Getaway	15,000	5	Goa
102	Himachal Adventure	25,000	7	Himachal Pradesh
103	Kerala Backwaters	30,000	6	Kerala
104	Rajasthan Royalty	35,000	8	Rajasthan
105	Andaman Islands	50,000	7	Andaman and Nicobar
106	Leh Ladakh Trek	40,000	10	Ladakh
107	Ooty Hill Station	10,000	4	Ooty
108	Kashmir Paradise	28,000	5	Kashmir
109	Kerala Beaches	20,000	5	Kerala
110	Sikkim Exploration	22,000	6	Sikkim
111	Dubai Luxury Escape	80,000	6	Dubai
112	Maldives Seaside Retreat	75,000	7	Maldives
113	Paris City Break	90,000	5	Paris
114	London Exploration	85,000	6	London
115	Tokyo Adventure	95,000	8	Tokyo
116	New York Experience	1,00,000	7	New York
117	Bali Beach Holiday	60,000	6	Bali
118	Switzerland Alps Escape	70,000	7	Switzerland
119	Australia Wildlife Tour	65,000	7	Australia
120	Egypt Pyramids Journey	55,000	6	Egypt

Algorithm Used:

- **Selection Sort** for sorting packages by price.

  Code :

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

void selectionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
```

```
        }
        swap(arr[i], arr[minIndex]);
    }
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int size = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, size);
        cout << "Sorted array: ";
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;

        return 0;
    }
```

Output:

Package ID	Package Name	Price (INR)	Duration (Days)	Destination
107	Ooty Hill Station	10,000	4	Ooty
101	Goa Beach Getaway	15,000	5	Goa
109	Kerala Beaches	20,000	5	Kerala
110	Sikkim Exploration	22,000	6	Sikkim
102	Himachal Adventure	25,000	7	Himachal Pradesh
108	Kashmir Paradise	28,000	5	Kashmir
103	Kerala Backwaters	30,000	6	Kerala
104	Rajasthan Royalty	35,000	8	Rajasthan
106	Leh Ladakh Trek	40,000	10	Ladakh
105	Andaman Islands	50,000	7	Andaman and Nicobar
120	Egypt Pyramids Journey	55,000	6	Egypt
117	Bali Beach Holiday	60,000	6	Bali
119	Australia Wildlife Tour	65,000	7	Australia
118	Switzerland Alps Escape	70,000	7	Switzerland
112	Maldives Seaside Retreat	75,000	7	Maldives
111	Dubai Luxury Escape	80,000	6	Dubai
114	London Exploration	85,000	6	London
115	Tokyo Adventure	95,000	8	Tokyo
116	New York Experience	1,00,000	7	New York
113	Paris City Break	90,000	5	Paris

3. Tourist Attraction Ranking

Sub-task: Rank tourist destinations based on popularity and reviews.
SDG Goal: 11 (Sustainable Cities and Communities)

Target: 11.4

Indicator: 11.4.1
Description:
Sort and rank destinations based on user reviews to recommend the best tourist spots.

Input Data:

- Ratings and reviews for tourist attractions.

Attraction ID	Attraction Name	Rating (1-10)	Number of Reviews	Accessibility Score (1-10)	Environmental Impact (1-10)
1	Central Park	9.5	200	8	7
2	Eiffel Tower	9.7	500	9	6
3	Great Barrier Reef	9.3	300	7	10
4	Taj Mahal	9.2	150	8	8
5	Grand Canyon	9.6	250	9	9
6	Mount Fuji	8.9	100	7	9
7	Machu Picchu	9.8	400	9	8
8	Statue of Liberty	8.5	180	8	6
9	Colosseum	9	220	8	7
10	Niagara Falls	9.4	350	9	9
11	Acropolis of Athens	9.3	450	8	7
12	Kyoto Imperial Palace	8.8	130	7	8
13	Mount Everest	9.9	600	10	10
14	Alhambra	9.6	350	8	8
15	Sydney Opera House	9.2	290	9	6
16	Pyramids of Giza	9.1	170	7	9
17	Victoria Falls	9.4	210	8	8
18	Mont Saint-Michel	9	180	8	7
19	Machu Picchu (2nd entry)	9.8	550	9	8
20	Yellowstone National Park	9.7	400	9	10

Algorithm Used:

- **Bubble Sort** for sorting attraction ratings..

Code:

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

void bubble_sort(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    bubble_sort(arr, n);

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

```
    cout << endl;


    return 0;
}
```

Output:

Attraction ID	Attraction Name	Rating (1-10)	Number of Reviews	Accessibility Score (1-10)	Environmental Impact (1-10)
13	Mount Everest	9.9	600	10	10
7	Machu Picchu	9.8	400	9	8
19	Machu Picchu (2nd entry)	9.8	550	9	8
2	Eiffel Tower	9.7	500	9	6
20	Yellowstone National Park	9.7	400	9	10
5	Grand Canyon	9.6	250	9	9
14	Alhambra	9.6	350	8	8
10	Niagara Falls	9.4	350	9	9
17	Victoria Falls	9.4	210	8	8
3	Great Barrier Reef	9.3	300	7	10
11	Acropolis of Athens	9.3	450	8	7
4	Taj Mahal	9.2	150	8	8
15	Sydney Opera House	9.2	290	9	6
9	Colosseum	9	220	8	7
18	Mont Saint-Michel	9	180	8	7
16	Pyramids of Giza	9.1	170	7	9
12	Kyoto Imperial Palace	8.8	130	7	8
6	Mount Fuji	8.9	100	7	9
8	Statue of Liberty	8.5	180	8	6

Industries

---

1. Material Sorting in Warehouses

Sub-task: Arrange raw materials in order of their delivery dates to streamline manufacturing.

SDG Goal: 11 (Sustainable Cities and Communities)

Target: 11.6

Indicator: 11.6.1

Description:

Sort warehouse inventory by delivery dates to optimize production schedules.

Input Data:

Batch ID	Material Name	Delivery Date	Supplier	Quantity
1	Steel Plates	05-01-2024	Supplier A	1000
2	Aluminum Sheets	12-03-2024	Supplier B	1500
3	Copper Wire	25-05-2024	Supplier C	2000
4	Plastic Sheets	28-02-2024	Supplier D	1200
5	Steel Pipes	19-07-2024	Supplier E	800
6	Iron Rods	07-04-2024	Supplier F	1100
7	Glass Panels	14-06-2024	Supplier G	1300
8	Rubber Rolls	15-01-2024	Supplier H	900
9	Wood Blocks	15-02-2024	Supplier I	1000
10	Plastic Bottles	05-05-2024	Supplier J	1400
11	Steel Sheets	22-08-2024	Supplier K	1500
12	Aluminum Cans	08-03-2024	Supplier L	1100
13	Copper Tubes	01-06-2024	Supplier M	900
14	Steel Wire	25-01-2024	Supplier N	1400
15	Plastic Pipes	10-09-2024	Supplier O	1100
16	Rubber Sheets	12-04-2024	Supplier P	1200
17	Glass Containers	30-07-2024	Supplier Q	1000
18	Aluminum Plates	15-10-2024	Supplier R	1300
19	Iron Sheets	05-11-2024	Supplier S	1500
20	Steel Bars	01-12-2024	Supplier T	1400

Algorithm Used:

- **Merge Sort** for efficient inventory sorting.

Code:.

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


struct d {
    int w; // Example data field, modify as needed
};
```

```
void Merge(d B[], int p, d C[], int q, d A[]) {
    int i = 0, j = 0, k = 0;

    while (i < p && j < q) {
        if (B[i].w <= C[j].w) {
            A[k++] = B[i++];
        } else {
            A[k++] = C[j++];
        }
    }
```

```
    while (i < p) {
        A[k++] = B[i++];
    }
    while (j < q) {
        A[k++] = C[j++];
    }
}

void MergeSort(d A[], int n) {
    if (n > 1) {
    int mid = n / 2;

    d B[50], C[50];
    for (int i = 0; i < mid; i++) {
        B[i] = A[i];
    }
    for (int i = mid; i < n; i++) {
        C[i - mid] = A[i];
    }

    MergeSort(B, mid);
    MergeSort(C, n - mid);
        Merge(B, mid, C, n - mid, A);
    }
}

void printArray(d A[], int n) {
    for (int i = 0; i < n; i++) {
        cout << A[i].w << " ";
    }
```

```
    cout << endl;
}

int main() {
    d A[5] = {{12}, {11}, {13}, {5}, {6}}; // Example data
    int n = sizeof(A) / sizeof(A[0]);

    cout << "Original array: ";
    printArray(A, n);

    MergeSort(A, n);

    cout << "Sorted array: ";
    printArray(A, n);

    return 0;
}
```

Output:

Batch ID	Material Name	Delivery Date	Supplier	Quantity (Units)
1	Steel Plates	05-01-2024	Supplier A	1000
8	Rubber Rolls	15-01-2024	Supplier H	900
14	Steel Wire	25-01-2024	Supplier N	1400
4	Plastic	28-02-2024	Supplier D	1200
9	Wood	15-02-2024	Supplier I	1000
2	Aluminum	12-03-2024	Supplier B	1500
12	Aluminum	08-03-2024	Supplier L	1100
6	Iron Rods	07-04-2024	Supplier F	1100
16	Rubber	12-04-2024	Supplier P	1200
3	Copper Wire	25-05-2024	Supplier C	2000
10	Plastic	05-05-2024	Supplier J	1400
7	Glass Panels	14-06-2024	Supplier G	1300
13	Copper	01-06-2024	Supplier M	900
5	Steel Pipes	19-07-2024	Supplier E	800
17	Glass	30-07-2024	Supplier Q	1000
11	Steel Sheets	22-08-2024	Supplier K	1500
15	Plastic Pipes	10-09-2024	Supplier O	1100
18	Aluminum	15-10-2024	Supplier R	1300
19	Iron Sheets	05-11-2024	Supplier S	1500
20	Steel Bars	01-12-2024	Supplier T	1400

2. Sub-task: Search for specific industry types in a large dataset
SDG Goal: 11
Target: 11.3
Indicator: 11.3.2

Description:
The goal is to efficiently search for and categorize industries such as "Manufacturing", "IT", or "Energy" within a large collection of industry names or descriptions. By using the Rabin-Karp algorithm, we aim to optimize the search process, especially for datasets with multiple patterns.

Input Data:

Index	Industry
0	Manufacturing
1	Technology
2	IT
3	Energy
4	Healthcare
5	Retail
6	Agriculture
7	IT
8	Construction
9	Energy

Patterns to Search:

- IT

- Energy

Algorithm Used: Rabin-Karp Algorithm

Code:

```
void RabinKarp(string text, string pattern) {
    int n = text.length();
    int m = pattern.length();
    int d = 256;
    int q = 101;
    int h = 1;
    int p = 0;
    int t = 0;

    for (int i = 0; i < m - 1; i++) {
        h = (h * d) % q;
    }

    for (int i = 0; i < m; i++) {
        p = (d * p + pattern[i]) % q;
        t = (d * t + text[i]) % q;
    }

    for (int i = 0; i <= n - m; i++) {
        if (p == t) {
            bool found = true;
            for (int j = 0; j < m; j++) {
                if (text[i + j] != pattern[j]) {
                    found = false;
                    break;
                }
            }
            if (found) {
```

```
            cout << "Pattern found at index " << i << endl;
        }
    }

    if (i < n - m) {
        t = (d * (t - text[i] * h) + text[i + m]) % q;
        if (t < 0) t = (t + q);
    }
  }
}

int main() {
    string text = "ABABDABACDABABCABAB";
    string pattern = "ABABCABAB";
    RabinKarp(text, pattern);
    return 0;
}
```

Output:

The **pattern "IT"** is found at index positions **2** and **7**.

The **pattern "Energy"** is found at index positions **3** and **9**.

3.Smart Industrial Parks

Sub-task: Optimize industrial park layouts for efficient energy and resource usage.

SDG Goal: 11 (Sustainable Cities and Communities)

Target: 11.3

Indicator: 11.3.1

Description:

Design industrial zones with optimized layouts using clustering and graph algorithms to minimize energy loss and streamline operations.

Input Data:

Industrial Unit ID	Connected To	Energy Usage (kWh)	Distance (km)	Edge Weight (Energy × Distance)
A	B	300	2.5	750
A	C	500	1.8	900
B	C	400	2	800
B	D	600	3.5	2,100
C	D	450	2.2	990
D	E	700	2.8	1,960

Algorithm Used:

- **Kruskal's Algorithm** for creating a minimum spanning tree (MST).

Code :

```
#include < iostream >

using namespace std;

class d {
public:
int u;
int v;
int w;
};


int find(int arr[50], int u, int v) {
if (arr[u] == arr[v]) {
return 1;
} else {
return 0;
}
}

void union_set(int arr[50], int u, int v, int n) {
int temp = arr[u];
for (int i = 0; i < n; i++) {
if (arr[i] == temp) {
```

```
arr[i] = arr[v];
}
}
}


void Merge(d B[], int p, d C[], int q, d A[]) {
int i = 0, j = 0, k = 0;
while (i < p && j < q) {
    if (B[i].w <= C[j].w) {
        A[k++] = B[i++];
    } else {
        A[k++] = C[j++];
    }
}
while (i < p) {
    A[k++] = B[i++];
}
while (j < q) {
    A[k++] = C[j++];
}
}


void MergeSort(d A[], int n) {
if (n > 1) {
int mid = n / 2;
```

```
d B[50], C[50];
for (int i = 0; i < mid; i++) {
    B[i] = A[i];
}
for (int i = mid; i < n; i++) {
    C[i - mid] = A[i];
}

MergeSort(B, mid);
MergeSort(C, n - mid);
    Merge(B, mid, C, n - mid, A);
}
}


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


d d1[50];
    cout << "Enter the edges (u v w):" << endl;
    }
```

Output:

Edge (Connection)	Weight (Energy × Distance)	Included in MST
A - B	750	Yes
B - C	800	Yes
C - D	990	Yes
D - E	1,960	Yes

Windmills

1.Site Selection for New Wind Farms

Sub-task: Rank potential locations based on wind speed and other criteria.
SDG Goal: 11 (Sustainable Cities and Communities)

Target: 11.3

Indicator: 11.3.1
Description:
Sort and rank potential locations for wind farms to identify the most suitable sites.

Input Data:

Site ID	Location Name	Wind Speed (m/s)	Feasibility Score (1-10)	Accessibility Score (1-10)	Environmental Impact (1-10)
1	Greenfield Valley	6.2	9	7	6
2	Oak Ridge	5.5	8	6	5
3	Sunrise Hills	7.3	9	8	7
4	Coastal Plains	8	10	9	8
5	Silver Creek	6.8	7	7	6
6	Mountain Range	9	10	10	9
7	Meadowland Plains	6	6	5	7
8	Riverbank Hills	7.5	8	7	8
9	Red Valley	5.8	7	6	5
10	Forest Heights	6.6	8	8	7
11	Windy Ridge	8.2	9	9	9
12	Blue Lake Plateau	6.3	8	7	6
13	North Peak	7	8	7	7
14	East Horizon	5.2	6	5	4
15	Crystal Brook	9.5	10	10	10
16	Sunset Valley	7.8	9	8	8
17	Eagle Mountain	8.5	10	9	9
18	Hilltop Ridge	6.9	8	7	7
19	Prairie View	6.1	7	6	6
20	Boulder Heights	7.2	8	7	7

Algorithm Used:

- **Quick sort** for efficient site ranking.

Code :

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


void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[high];
        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]);
int pi = i + 1;
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
    }
}

int main() {
    int arr[] = {12, 4, 7, 9, 3, 5, 6, 8, 2, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;     }
```

Output:

Site ID	Location Name	Wind Speed (m/s)	Feasibility Score (1-10)	Accessibility Score (1-10)	Environmental Impact (1-10)
15	Crystal Brook	9.5	10	10	10
6	Mountain Range	9	10	10	9
17	Eagle Mountain	8.5	10	9	9
4	Coastal Plains	8	10	9	8
11	Windy Ridge	8.2	9	9	9
3	Sunrise Hills	7.3	9	8	7
16	Sunset Valley	7.8	9	8	8
2	Oak Ridge	5.5	8	6	5
10	Forest Heights	6.6	8	8	7
20	Boulder Heights	7.2	8	7	7
13	North Peak	7	8	7	7
8	Riverbank Hills	7.5	8	7	8
12	Blue Lake Plateau	6.3	8	7	6
18	Hilltop Ridge	6.9	8	7	7
5	Silver Creek	6.8	7	7	6
19	Prairie View	6.1	7	6	6
1	Greenfield Valley	6.2	9	7	6
7	Meadowland Plains	6	6	5	7
14	East Horizon	5.2	6	5	4
9	Red Valley	5.8	7	6	5

2. Maintenance Scheduling for Windmills

Sub-task: Schedule routine maintenance for windmills to ensure optimal performance and reduce downtime. **SDG Goal:** 9 (Industry, Innovation, and Infrastructure)
Target: 9.4
Indicator: 9.4.1

Description:
The goal is to ensure that windmills are maintained regularly to maximize their efficiency. Scheduling maintenance based on performance data helps reduce downtime and ensures continuous energy production.

Input Data:

Windmill (Node)	Connected to (Edges)	Performance (Weight)
Windmill 1	Windmill 2, Windmill 3	45
Windmill 2	Windmill 1, Windmill 4	70
Windmill 3	Windmill 1, Windmill 5	30
Windmill 4	Windmill 2, Windmill 5	55
Windmill 5	Windmill 3, Windmill 4	25

Algorithm Used:
AVL Tree for balancing performance data and maintenance scheduling
Queue for managing maintenance requests

Code :

Queue-

```
#include <iostream>
```

```
using namespace std;

#define SIZE 100

class Queue {
    int front, rear, arr[SIZE];

public:
    Queue() {
        front = -1;
    rear = -1;
}

bool isEmpty() {
    return front == -1;
}

bool isFull() {
    return rear == SIZE - 1;
}

void enqueue(int value) {
    if (isFull())
        return;
    if (isEmpty())
        front = 0;
    arr[++rear] = value;
}

int dequeue() {
    if (isEmpty())
```

```
        return -1;
    int value = arr[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front++;
    }
    return value;
}

    int peek() {
        return isEmpty() ? -1 : arr[front];
    }
};

int main() {
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    cout << q.dequeue() << endl;
    cout << q.peek() << endl;
    cout << q.dequeue() << endl;

    return 0;
}
```

AVL tree-

```
#include <iostream>
#include <queue>
```

```
#include <string>
using namespace std;
struct AVLNode {
    int windmillID;
    int performance;
    AVLNode* left;
    AVLNode* right;
    int height;
};
int getHeight(AVLNode* node) {
    return (node == nullptr) ? 0 : node->height;
}
AVLNode* createNode(int windmillID, int performance) {
    AVLNode* node = new AVLNode();
    node->windmillID = windmillID;
    node->performance = performance;
    node->left = node->right = nullptr;
    node->height = 1;
    return node;
}
AVLNode* rightRotate(AVLNode* y) {
    AVLNode* x = y->left;
    AVLNode* T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;

    return x;
```

```
}
AVLNode* leftRotate(AVLNode* x) {
    AVLNode* y = x->right;
    AVLNode* T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

    return y;
}
int getBalance(AVLNode* node) {
    return (node == nullptr) ? 0 : getHeight(node->left) - getHeight(node->right);
}
AVLNode* insert(AVLNode* node, int windmillID, int performance) {
    if (node == nullptr)
        return createNode(windmillID, performance);

    if (windmillID < node->windmillID)
        node->left = insert(node->left, windmillID, performance);
else if (windmillID > node->windmillID)
    node->right = insert(node->right, windmillID, performance);
else
    return node;

node->height = max(getHeight(node->left), getHeight(node->right)) + 1;

int balance = getBalance(node);
```

```
if (balance > 1 && windmillID < node->left->windmillID)
    return rightRotate(node);

if (balance < -1 && windmillID > node->right->windmillID)
    return leftRotate(node);

if (balance > 1 && windmillID > node->left->windmillID) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
}
    if (balance < -1 && windmillID < node->right->windmillID) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}
void displayMaintenance(AVLNode* root, int threshold, queue<int>&
maintenanceQueue) {
    if (root == nullptr)
        return;

displayMaintenance(root->left, threshold, maintenanceQueue);

if (root->performance < threshold) {
    cout << "Windmill ID: " << root->windmillID
        << ", Performance: " << root->performance
        << " (Maintenance Required)\n";
    maintenanceQueue.push(root->windmillID);
}
```

```
    displayMaintenance(root->right, threshold, maintenanceQueue);
}

int main() {
    AVLNode* root = nullptr;
    root = insert(root, 1, 45);
    root = insert(root, 2, 70);
    root = insert(root, 3, 30);
    root = insert(root, 4, 55);
    root = insert(root, 5, 25);

    int performanceThreshold = 50;
    queue<int> maintenanceQueue;

    cout << "Windmills requiring maintenance (Performance < " <<
performanceThreshold << "):\n";
    displayMaintenance(root, performanceThreshold, maintenanceQueue);

    cout << "\nScheduled Maintenance Queue:\n";
    while (!maintenanceQueue.empty()) {
        cout << "Windmill ID: " << maintenanceQueue.front() << "\n";
        maintenanceQueue.pop();
    }

    return 0;
}
```

Output:

Windmills scheduled for maintenance:

Windmill ID: 1, Performance: 45, Last Maintenance: 2024-01-10

Windmill ID: 3, Performance: 30, Last Maintenance: 2023-11-20

Windmill ID: 5, Performance: 25, Last Maintenance: 2023-12-01

4: Predicting Windmill Power Generation

Sub-task: Predict windmill power generation based on historical weather data and wind speed. **SDG Goal:** 7 (Affordable and Clean Energy)
Target: 7.2
Indicator: 7.2.1

Description:
This business case involves predicting the energy output of windmills based on historical weather data, including wind speed and other environmental factors. This prediction can help plan energy storage and distribution in advance.

Input Data:

Day	Windmill ID	Wind Speed (m/s)	Temperature (°C)	Efficiency Rate (%)
1	W1	12.5	20.1	85
1	W2	14.2	19.8	87
1	W3	11.8	21.3	83
2	W1	13.1	19.6	86
2	W2	15	20.4	89

Algorithm Used:

Fenwick Tree (Binary Indexed Tree) for efficient range queries

Output:

Day	Windmill ID	Predicted Output (kWh)	Energy Status
1	W1	1,062	Available
1	W2	1,234	Available
1	W3	987	Low
2	W1	1,108	Available
2	W2	1,305	Available