1. A warehouse system stores package IDs in the order they arrive. To prepare for dispatch, the IDs must be sorted in ascending order. Write a program using Bubble Sort to arrange the following IDs: [5, 4, 3, 2, 1]

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

int main() {
  // Array of package IDs
  int package_ids[] = {5, 4, 3, 2, 1};
  int n = sizeof(package_ids) / sizeof(package_ids[0]);

  // Bubble Sort algorithm
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (package_ids[j] > package_ids[j + 1]) {
        // Swap elements
        int temp = package_ids[j];
        package_ids[j] = package_ids[j + 1];
        package_ids[j + 1] = temp;
      }
    }
  }

  // Display sorted package IDs
  cout << "Sorted package IDs in ascending order: ";
  for (int i = 0; i < n; i++) {
    cout << package_ids[i] << " ";
  }
  cout << endl;

  return 0;
}
```

2. A warehouse system stores package IDs in the order they arrive. To prepare for dispatch, the IDs must be sorted in ascending order. Write a program using Insertion Sort to arrange the following IDs: [5,4,3,2,1]

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

int main() {
  // Array of package IDs
  int package_ids[] = {5, 4, 3, 2, 1};
  int n = sizeof(package_ids) / sizeof(package_ids[0]);

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

```
           // Move elements of package_ids[0..i-1] that are greater than key
        // to one position ahead of their current position
        while (j >= 0 && package_ids[j] > key) {
            package_ids[j + 1] = package_ids[j];
            j--;
        }
        package_ids[j + 1] = key;
    }

    // Display sorted package IDs
    cout << "Sorted package IDs in ascending order: ";
    for (int i = 0; i < n; i++) {
        cout << package_ids[i] << " ";
    }
    cout << endl;

    return 0;
}
```

3 . A warehouse system stores package IDs in the order they arrive. To prepare for dispatch, the IDs must be sorted in ascending order. Write a program using Selection Sort to arrange the following IDs: [5,4,3,2,1]

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

int main() {
    // Array of package IDs
    int package_ids[] = {5, 4, 3, 2, 1};
    int n = sizeof(package_ids) / sizeof(package_ids[0]);

    // Selection Sort algorithm
    for (int i = 0; i < n - 1; i++) {
        int min_index = i;

        // Find the minimum element in the unsorted part
        for (int j = i + 1; j < n; j++) {
            if (package_ids[j] < package_ids[min_index]) {
                min_index = j;
            }
        }
        // Swap the found minimum element with the first element
        int temp = package_ids[i];
        package_ids[i] = package_ids[min_index];
        package_ids[min_index] = temp;
    }

    // Display sorted package IDs
    cout << "Sorted package IDs in ascending order: ";
    for (int i = 0; i < n; i++) {
        cout << package_ids[i] << " ";
    }
    cout << endl;

    return 0;
}
```

4. A hospital management system stores patient IDs in a linked list to maintain their admission order. You are given the following sequence of patient IDs: 111 → 123 → 124 → NULL Write a program to create and display this linked list.

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

// Define the structure for a node
struct Node {
    int patientID;
    Node* next;
};

int main() {
    // Create nodes
    Node* head = new Node();
    Node* second = new Node();
    Node* third = new Node();
    // Assign patient IDs
    head->patientID = 111;
    head->next = second;

    second->patientID = 123;
    second->next = third;

    third->patientID = 124;
    third->next = NULL;

    // Display linked list
    Node* current = head;
    cout << "Patient IDs in admission order: ";
    while (current != NULL) {
        cout << current->patientID;
        if (current->next != NULL) {
            cout << " -> ";
        }
        current = current->next;
    }
    cout << " -> NULL" << endl;

    // Free allocated memory
    delete head;
    delete second;
    delete third;

    return 0;
}
```

5. A social networking app wants to represent user connections as a graph, where each user is a node and friendships are edges between them. Given a graph showing user connections, create the adjacency list representation for it. Create the adjacency matrix for the given graph.

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

```
using namespace std;


int main() {
  int n, e;
  cout << "Enter number of users (nodes): ";
  cin >> n;
  cout << "Enter number of friendships (edges): ";
  cin >> e;


  // Initialize adjacency list and matrix
  vector<int> adjList[n];
  int adjMatrix[n][n];


  // Initialize matrix to 0
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      adjMatrix[i][j] = 0;


  // Input edges
  cout << "Enter friendships (two users per line, 0-based index):\n";
  for (int i = 0; i < e; i++) {
    int u, v;
    cin >> u >> v;
    // Add to adjacency list
    adjList[u].push_back(v);
    adjList[v].push_back(u); // since undirected graph


    // Add to adjacency matrix
    adjMatrix[u][v] = 1;
    adjMatrix[v][u] = 1;
  }


  // Display adjacency list
```

```
    cout << "\nAdjacency List Representation:\n";
    for (int i = 0; i < n; i++) {
        cout << i << ": ";
        for (int j : adjList[i])
            cout << j << " ";
        cout << endl;
    }

    // Display adjacency matrix
    cout << "\nAdjacency Matrix Representation:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << adjMatrix[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

6. A university's examination system stores student roll numbers in a binary tree for efficient searching. Given the structure of the tree, implement and display the binary tree. 50 / \ 30 70 / \ / 20 40 60

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

// Define structure for a tree node
struct Node {
    int data;
    Node* left;
    Node* right;
};

// Function to create a new node
Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}

// Inorder traversal (Left, Root, Right)
void inorderTraversal(Node* root) {
    if (root != nullptr) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}

// Function to display tree visually (preorder style)
void displayTree(Node* root, string indent = "", bool last = true) {
    if (root != nullptr) {
        cout << indent;
        if (last) {
            cout << "R----";
            indent += "     ";
        } else {
            cout << "L----";
            indent += "|    ";
        }
        cout << root->data << endl;
        displayTree(root->left, indent, false);
        displayTree(root->right, indent, true);
    }
}

int main() {
    // Manually create the tree
    Node* root = createNode(50);
    root->left = createNode(30);
    root->right = createNode(70);
    root->left->left = createNode(20);
    root->left->right = createNode(40);
    root->right->left = createNode(60);

    // Display tree visually
    cout << "Binary Tree Structure:\n";
    displayTree(root);

    // Display inorder traversal
    cout << "\nInorder Traversal (sorted roll numbers): ";
    inorderTraversal(root);
    cout << endl;

    return 0;
}
```

7. An online library wants to store book IDs efficiently using hashing. The hash function used is: h(key) = key % table_size If the book IDs are [1, 2, 3, 4] and the hash table size is 3, insert the keys into the hash table and show the final table representation.

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

int main() {
    int table_size = 3;
    int book_ids[] = {1, 2, 3, 4};
    int n = sizeof(book_ids) / sizeof(book_ids[0]);

    // Create hash table with chaining
    list<int> hashTable[table_size];
    // Insert keys
    for (int i = 0; i < n; i++) {
        int index = book_ids[i] % table_size;
        hashTable[index].push_back(book_ids[i]);
    }

    // Display hash table
    cout << "Hash Table Representation:\n";
    for (int i = 0; i < table_size; i++) {
        cout << i << ": ";
        for (int key : hashTable[i]) {
            cout << key << " -> ";
        }
        cout << "NULL" << endl;
    }

    return 0;
}
```

8. A city traffic control system represents road connections between intersections as a graph, where each intersection is a node and roads are edges. Given the graph, create the adjacency matrix representation for it.

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

int main() {
    int n = 5; // Number of intersections
    int adjMatrix[5][5] = {0}; // Initialize 5x5 matrix with 0

    // Add roads (edges)
    adjMatrix[0][1] = adjMatrix[1][0] = 1;
    adjMatrix[0][2] = adjMatrix[2][0] = 1;
    adjMatrix[1][2] = adjMatrix[2][1] = 1;
    adjMatrix[1][3] = adjMatrix[3][1] = 1;
    adjMatrix[2][4] = adjMatrix[4][2] = 1;
    adjMatrix[3][4] = adjMatrix[4][3] = 1;

    // Display adjacency matrix
    cout << "Adjacency Matrix Representation:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << adjMatrix[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

1️⃣ Adjacency List Representation (C++):

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

int main() {
    int n = 5; // Number of users
    vector<int> adjList[n]; // Array of vectors

    // Add edges (friendships)
    adjList[0].push_back(1);
    adjList[0].push_back(2);

    adjList[1].push_back(0);
    adjList[1].push_back(2);
    adjList[1].push_back(3);

    adjList[2].push_back(0);
    adjList[2].push_back(1);
    adjList[2].push_back(4);

    adjList[3].push_back(1);
    adjList[3].push_back(4);

    adjList[4].push_back(2);
    adjList[4].push_back(3);

    // Display adjacency list
    cout << "Adjacency List Representation:\n";
    for (int i = 0; i < n; i++) {
        cout << i << ": ";
        for (int j : adjList[i]) {
            cout << j << " ";
        }
        cout << endl;
    }

    return 0;
}
```

2️⃣ Adjacency Matrix Representation (C++)

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

int main() {
    int n = 5; // Number of users
    int adjMatrix[5][5] = {0};

    // Add edges (friendships)
    adjMatrix[0][1] = adjMatrix[1][0] = 1;
    adjMatrix[0][2] = adjMatrix[2][0] = 1;
    adjMatrix[1][2] = adjMatrix[2][1] = 1;
    adjMatrix[1][3] = adjMatrix[3][1] = 1;
    adjMatrix[2][4] = adjMatrix[4][2] = 1;
    adjMatrix[3][4] = adjMatrix[4][3] = 1;

    // Display adjacency matrix
    cout << "Adjacency Matrix Representation:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << adjMatrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```