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]

- **Aim:**
  To write a C++ program to sort package IDs using Bubble Sort.
- **Objective:**
  To arrange the given IDs [5, 4, 3, 2, 1] in ascending order using Bubble Sort technique.
- **Procedure:**
  1. Start the program.
  2. Input the array elements.
  3. Compare each pair of adjacent elements.
  4. Swap if the first is greater than the second.
  5. Repeat the process for all passes.
  6. Display the sorted array.
  7. Stop.
- **Code:**

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

int main() {
  int arr[] = {5, 4, 3, 2, 1};
  int n = 5;

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

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

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

- **Aim:**
  To write a C++ program to sort package IDs using Insertion Sort.
- **Objective:**
  To arrange [5, 4, 3, 2, 1] in ascending order using Insertion Sort technique.
- **Procedure:**
  1. Start the program.
  2. Take array input.
  3. Compare each element with previous elements.
  4. Insert it in the correct position.
  5. Repeat for all elements.
  6. Display sorted array.
  7. Stop.
- **Code:**

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

int main() {
  int arr[] = {5, 4, 3, 2, 1};
  int n = 5;

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

  for(int i=1; i<n; i++) {
    int key = arr[i];
    int j = i-1;
    while(j >= 0 && arr[j] > key) {
      arr[j+1] = arr[j];
      j--;
    }
    arr[j+1] = key;
  }

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

- **Aim:**
  To write a C++ program to sort package IDs using Selection Sort.
- **Objective:**
  To arrange [5, 4, 3, 2, 1] in ascending order using Selection Sort.
- **Procedure:**
  1. Start the program.
  2. Find the smallest element in the array.
  3. Swap it with the first element.
  4. Repeat for remaining elements.
  5. Display sorted array.
  6. Stop.
- **Code:**

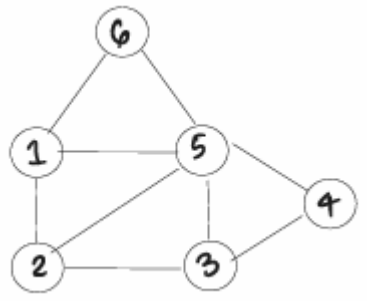
```
#include <iostream>
using namespace std;

int main() {
  int arr[] = {5, 4, 3, 2, 1};
  int n = 5;

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

  for(int i=0; i<n-1; i++) {
    int minIndex = i;
    for(int j=i+1; j<n; j++) {
      if(arr[j] < arr[minIndex]) {
        minIndex = j;
      }
    }
    int temp = arr[i];
    arr[i] = arr[minIndex];
    arr[minIndex] = temp;
  }

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

- **Aim:**
  To write a C++ program to create and display a linked list of patient IDs.
- **Objective:**
  To represent and display the patient admission order using a linked list.
- **Procedure:**
  1. Start the program.
  2. Create a structure Node having data and a pointer to the next node.
  3. Dynamically create nodes for given IDs (111 → 123 → 124).
  4. Link each node to the next one.
  5. Traverse the linked list to display all patient IDs.
  6. Stop the program.
- **Code:**

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

struct Node {
  int data;
  Node* next;
};

int main() {
  Node *head = new Node{111, NULL};
  Node *second = new Node{123, NULL};
  Node *third = new Node{124, NULL};

  head->next = second;
  second->next = third;

  cout << "Patient IDs in linked list: ";
  Node *temp = head;
  while(temp != NULL) {
    cout << temp->data << " -> ";
    temp = temp->next;
  }
  cout << "NULL";
  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.**



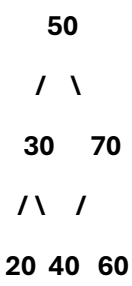
- **Aim:**
  To write a C++ program to represent user connections as a graph using adjacency list and matrix.
- **Objective:**
  To store and display the connections (edges) between users (nodes) using both representations.
- **Procedure:**
  1. Start the program.
  2. Represent users as nodes and friendships as edges.
  3. Create adjacency list and matrix.
  4. Display both representations.
  5. Stop.
- **Code:**

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

int main() {
  int vertices = 4;
  vector<int> adjList[4];

  adjList[0] = {1, 2};
  adjList[1] = {2};
  adjList[2] = {0, 3};
  adjList[3] = {3};

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

```
    int adjMatrix[4][4] = {0};
    adjMatrix[0][1] = adjMatrix[0][2] = 1;
    adjMatrix[1][2] = 1;
    adjMatrix[2][0] = adjMatrix[2][3] = 1;
    adjMatrix[3][3] = 1;

    cout << "\nAdjacency Matrix Representation:\n";
    for(int i=0; i<vertices; i++) {
      for(int j=0; j<vertices; 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.**

- **Aim:**
  To write a C++ program to create and display a binary tree.
- **Objective:**
  To represent student roll numbers in a binary tree structure and display them using inorder traversal.
- **Given Tree:**



- **Procedure:**

1. Start the program.
2. Define a structure for tree nodes.
3. Manually link nodes according to the given structure.
4. Display the tree using inorder traversal.
5. Stop.

- **Code:**

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

struct Node {
  int data;
  Node* left;
  Node* right;
};

Node* newNode(int value) {
```

```
    Node* node = new Node();
    node->data = value;
    node->left = node->right = NULL;
    return node;
}

void inorder(Node* root) {
    if(root != NULL) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    Node* root = newNode(50);
    root->left = newNode(30);
    root->right = newNode(70);
    root->left->left = newNode(20);
    root->left->right = newNode(40);
    root->right->left = newNode(60);

    cout << "Inorder Traversal of Binary Tree: ";
    inorder(root);
    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.

- **Aim:**
  To write a C++ program to store book IDs using hashing.
- **Objective:**
  To use hash function h(key) = key % table_size for storing book IDs efficiently.
- **Given:**
  Book IDs = [1, 2, 3, 4], Table Size = 3
- **Procedure:**

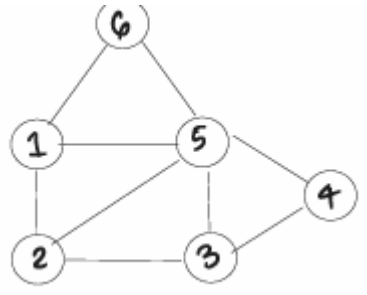
  1. Start the program.
  2. Take given book IDs and table size.
  3. Apply hash function key % size.
  4. Insert elements in hash table.
  5. Display final table.
  6. Stop.

- **Code:**

```
#include <iostream>

using namespace std;


int main() {

   int tableSize = 3;

   int hashTable[3] = {-1, -1, -1};

   int keys[] = {1, 2, 3, 4};


   for(int i=0; i<4; i++) {

      int index = keys[i] % tableSize;
```

```
        if(hashTable[index] == -1)
            hashTable[index] = keys[i];
        else
            cout << "Collision occurred for key " << keys[i] << " at index " << index << endl;
    }

    cout << "\nFinal Hash Table:\n";
    for(int i=0; i<tableSize; i++) {
        cout << "Index " << i << ": ";
        if(hashTable[i] != -1)
            cout << hashTable[i];
        cout << 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.



- **Aim:**
  To write a C++ program to represent road connections between intersections using adjacency matrix.
- **Objective:**
  To show the connection between intersections (nodes) as a matrix representation.
- **Procedure:**

  1. Start the program.
  2. Let intersections be nodes and roads be edges.
  3. Represent connection as 1 and no connection as 0.
  4. Display the adjacency matrix.
  5. Stop.

- **Code:**

```
#include <iostream>

using namespace std;



int main() {

   int vertices = 4;

   int adjMatrix[4][4] = {0};



   adjMatrix[0][1] = adjMatrix[0][2] = 1;

   adjMatrix[1][2] = 1;

   adjMatrix[2][0] = adjMatrix[2][3] = 1;
```

```
    adjMatrix[3][3] = 1;


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


    return 0;
}
```