1. Simulate an airline seat reservation system using a sparse matrix where only selected seats are booked, and efficiently display booked vs. unbooked seats. (Hint: Use 2D sparse representation and display only non-zero entries.)

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

// Structure to represent a booked seat in sparse format
class Seat {
public:
  int row;
  int col;
  string passengerName;

  Seat(int r, int c, string name) {
    row = r;
    col = c;
    passengerName = name;
  }
};

class AirlineReservation {
  int totalRows, totalCols;
  vector<Seat> bookedSeats; // sparse representation: store only booked seats

public:
  AirlineReservation(int rows, int cols) {
    totalRows = rows;
    totalCols = cols;
  }

  void bookSeat(int row, int col, string passengerName) {
    // Check if already booked
    for (Seat &s : bookedSeats) {
      if (s.row == row && s.col == col) {
        cout << "Seat (" << row << ", " << col << ") is already booked by " <<
s.passengerName << ".\n";
        return;
      }
    }
    bookedSeats.push_back(Seat(row, col, passengerName));
    cout << "Seat (" << row << ", " << col << ") booked for " << passengerName
<< ".\n";
  }

  void displayBookedSeats() {
    if (bookedSeats.empty()) {
      cout << "No seats are booked.\n";
      return;
    }
    cout << "\nBooked Seats (row, col) with passenger name:\n";
    for (const Seat &s : bookedSeats) {
      cout << "(" << s.row << ", " << s.col << ") - " << s.passengerName << "\n";
    }
  }

  void displaySeatMap() {
    cout << "\nSeat Map (B = booked, A = available):\n";
    for (int i = 0; i < totalRows; i++) {
      for (int j = 0; j < totalCols; j++) {
        bool booked = false;
        for (const Seat &s : bookedSeats) {
          if (s.row == i && s.col == j) {
            booked = true;
            break;
          }
        }
        cout << (booked ? "B " : "A ");
      }
      cout << "\n";
    }
  }
};

int main() {
  int rows, cols;
  cout << "Enter total rows and columns of seats: ";
  cin >> rows >> cols;

  AirlineReservation system(rows, cols);

  int choice;
  while (true) {
    cout << "\n1. Book a seat\n2. Display booked seats\n3. Display seat
map\n4. Exit\nEnter choice: ";
    cin >> choice;

    if (choice == 1) {
      int r, c;
      string name;
      cout << "Enter seat row and column to book: ";
      cin >> r >> c;
      cout << "Enter passenger name: ";
      cin >> ws; // consume newline
      getline(cin, name);

      if (r >= 0 && r < rows && c >= 0 && c < cols)
        system.bookSeat(r, c, name);
      else
        cout << "Invalid seat coordinates.\n";

    } else if (choice == 2) {
      system.displayBookedSeats();

    } else if (choice == 3) {
      system.displaySeatMap();

    } else if (choice == 4) {
      cout << "Exiting...\n";
      break;

    } else {
      cout << "Invalid choice!\n";
    }
  }

  return 0;
}
```

2. Build a student attendance tracker where attendance records for multiple days are stored in a sparse matrix. Display only the days and students with absences. (Hint: Treat absences as non-zero values in a sparse structure.)

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

class Record {
public:
   int studentID;
   int day;
   int status;   // 1 = Absent

   Record() {
      studentID = 0;
      day = 0;
      status = 0;
   }
};

class AttendanceTracker {
private:
   int totalStudents;
   int totalDays;
   int countAbsences;
   Record data[100];   // Sparse storage

public:
   AttendanceTracker(int s, int d) {
      totalStudents = s;
      totalDays = d;
      countAbsences = 0;
   }

   void takeAttendance() {
      char ch;

      cout << "\nEnter Attendance (P for Present, A for Absent)\n";

      for (int i = 0; i < totalStudents; i++) {
         cout << "\nStudent " << i + 1 << ":\n";
         for (int j = 0; j < totalDays; j++) {
            cout << "Day " << j + 1 << ": ";
            cin >> ch;

            if (ch == 'A' || ch == 'a') {
               data[countAbsences].studentID = i + 1;
               data[countAbsences].day = j + 1;
               data[countAbsences].status = 1;
               countAbsences++;
            }
         }
      }
   }

   void displayAbsenceReport() {
      cout << "\nAbsence Report\n";

      if (countAbsences == 0) {
         cout << "\nNo absences recorded.\n";
         return;
      }

      cout << "\nStudent  Day  Status\n";

      for (int i = 0; i < countAbsences; i++) {
         cout << data[i].studentID << "       "
              << data[i].day << "     "
              << "Absent\n";
      }
   }
};

int main() {
   int students, days;

   cout << "Enter number of students: ";
   cin >> students;

   cout << "Enter number of days: ";
   cin >> days;

   AttendanceTracker tracker(students, days);

   tracker.takeAttendance();
   tracker.displayAbsenceReport();

   return 0;
}
```

3. Develop a movie collection search tool where movie names can be searched using both linear and binary search. Compare search time on sorted vs. unsorted lists. (Hint: Maintain two lists: unsorted for linear search, sorted for binary.)

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

int linearSearch(vector<string> &arr, string key) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == key)
            return i;
    }
    return -1;
}

int binarySearchCustom(vector<string> &arr, string key) {
    int l = 0, r = arr.size() - 1;
    while (l <= r) {
        int mid = (l + r) / 2;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
            l = mid + 1;
        else
            r = mid - 1;
    }
    return -1;
}

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

    vector<string> unsortedList(n), sortedList;
    cout << "Enter movie names:\n";
    for (int i = 0; i < n; i++)
        cin >> unsortedList[i];

    sortedList = unsortedList;
    sort(sortedList.begin(), sortedList.end());

    string key;
    cout << "Enter movie name to search: ";
    cin >> key;

    // Repeat count to make time measurable
    const int REPEAT = 50000;

    // Linear Search Timing
    auto t1 = chrono::high_resolution_clock::now();
    int pos1 = -1;
    for (int i = 0; i < REPEAT; i++)
        pos1 = linearSearch(unsortedList, key);
    auto t2 = chrono::high_resolution_clock::now();
    auto linearTime = chrono::duration_cast<chrono::microseconds>(t2 -
t1).count();

    // Binary Search Timing
    auto t3 = chrono::high_resolution_clock::now();
    int pos2 = -1;
    for (int i = 0; i < REPEAT; i++)
        pos2 = binarySearchCustom(sortedList, key);
    auto t4 = chrono::high_resolution_clock::now();
    auto binaryTime = chrono::duration_cast<chrono::microseconds>(t4 -
t3).count();

    // Output
    if (pos1 != -1)
        cout << "\nLinear Search: Found at position " << pos1 + 1;
    else
        cout << "\nLinear Search: Not Found";

    cout << "\nTime taken: " << linearTime << " microseconds";

    if (pos2 != -1)
        cout << "\nBinary Search: Found at position " << pos2 + 1 << " (in sorted
list)";
    else
        cout << "\nBinary Search: Not Found";

    cout << "\nTime taken: " << binaryTime << " microseconds\n";

    return 0;
}
```

4. Create a hotel room management system where room prices can be sorted using insertion, bubble and selection sort. Compare the efficiency of each for large room lists. (Hint: Use same data for all sorting methods and record time.)

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

// -------- Bubble Sort --------
void bubbleSort(vector<int> &arr)
{
   int n = arr.size();
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (arr[j] > arr[j + 1])
            swap(arr[j], arr[j + 1]);
      }
   }
}

// -------- Selection Sort --------
void selectionSort(vector<int> &arr)
{
   int n = arr.size();
   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;
      }
      swap(arr[i], arr[minIndex]);
   }
}

// -------- Insertion Sort --------
void insertionSort(vector<int> &arr)
{
   int n = arr.size();
   for (int i = 1; i < n; i++)
   {
      int curr = arr[i];
      int prev = i - 1;
      while (prev >= 0 && arr[prev] > curr)
      {
         arr[prev + 1] = arr[prev];
         prev--;
      }
      arr[prev + 1] = curr;
   }
}

// -------- Print Function --------
void printArray(const vector<int> &arr)
{
   for (int x : arr)
      cout << x << " ";
   cout << endl;
}

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

   vector<int> original(n);

   // Generate random data
   srand(time(0)); // Seed for random number generator
   cout << "Generating " << n << " random room prices...\n";
   for (int i = 0; i < n; i++)
   {
      original[i] = rand() % 10000; // Random prices between 0-9999
   }

   // Same data for all sorts
   vector<int> arr1 = original;
   vector<int> arr2 = original;
   vector<int> arr3 = original;

   // ---- Bubble Sort ----
   clock_t t1 = clock();
   bubbleSort(arr1);
   clock_t t2 = clock();
   double bubbleTime = double(t2 - t1) / CLOCKS_PER_SEC * 1000;

   // ---- Selection Sort ----
   clock_t t3 = clock();
   selectionSort(arr2);
   clock_t t4 = clock();
   double selectionTime = double(t4 - t3) / CLOCKS_PER_SEC * 1000;

   // ---- Insertion Sort ----
   clock_t t5 = clock();
   insertionSort(arr3);
   clock_t t6 = clock();
   double insertionTime = double(t6 - t5) / CLOCKS_PER_SEC * 1000;

   // ---- Output ----
   cout << "\n--- Time Comparison ---\n";
   cout << "Bubble Sort Time    : " << bubbleTime << " milliseconds\n";
   cout << "Selection Sort Time : " << selectionTime << " milliseconds\n";
   cout << "Insertion Sort Time : " << insertionTime << " milliseconds\n";

   // Show first 20 sorted prices as proof
   cout << "\nFirst 20 Sorted Prices (from all three - they're identical):\n";
   for (int i = 0; i < 20 && i < n; i++)
   {
      cout << arr1[i] << " ";
   }
   cout << endl;

   // Show last 20 sorted prices
   cout << "\nLast 20 Sorted Prices:\n";
   for (int i = max(0, n - 20); i < n; i++)
   {
      cout << arr1[i] << " ";
   }
   cout << endl;

   return 0;
}
```

5. Create a hotel room management system where room prices can be sorted using quick sort. Compare the efficiency of each for large room lists. (Hint: Use same data for all sorting methods and record time.)

```
#include <iostream>
#include <vector>
#include <algorithm> // For std::swap
#include <chrono>    // For measuring time
#include <cstdlib>   // For random number generation

using namespace std;
using namespace std::chrono;

// Structure to represent a Hotel Room
struct Room {
   int roomNumber;
   double price;
};

class HotelSystem {
public:
   int partition(vector<Room>& arr, int low, int high) {
      double pivot = arr[high].price;
      int i = low - 1;

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

   void quickSort(vector<Room>& arr, int low, int high) {
      if (low < high) {
         int pi = partition(arr, low, high);
         quickSort(arr, low, pi - 1);
         quickSort(arr, pi + 1, high);
      }
   }

   void bubbleSort(vector<Room>& arr) {
      int n = arr.size();
      for (int i = 0; i < n - 1; i++) {
         for (int j = 0; j < n - i - 1; j++) {
            if (arr[j].price > arr[j + 1].price) {
               swap(arr[j], arr[j + 1]);
            }
         }
      }
   }

   void generateData(vector<Room>& arr, int n) {
      for (int i = 0; i < n; i++) {
         double rPrice = 1000 + (rand() % 9000);
         arr.push_back({i + 101, rPrice});
      }
   }

   void displayRooms(const vector<Room>& arr, int limit = 10) {
      cout << "Room Number | Price\n";
      for (int i = 0; i < min((int)arr.size(), limit); i++) {
         cout << arr[i].roomNumber << " | " << arr[i].price << "\n";
      }
      if (arr.size() > limit)
         cout << "...and " << arr.size() - limit << " more rooms\n";
      cout << "\n";
   }
};

int main() {
   HotelSystem hs;
   int n;

   cout << "Hotel Room Sorting Efficiency Test\n";
   cout << "Enter number of rooms to generate (e.g., 1000): ";
   cin >> n;

   vector<Room> dataSet1;
   hs.generateData(dataSet1, n);
   vector<Room> dataSet2 = dataSet1;

   cout << "\nData generated. Starting sorting...\n";

   auto start = high_resolution_clock::now();
   hs.bubbleSort(dataSet1);
   auto stop = high_resolution_clock::now();
   auto durationBubble = duration_cast<microseconds>(stop - start);

   cout << "Bubble Sort time: " << durationBubble.count() << " microseconds\n";

   start = high_resolution_clock::now();
   hs.quickSort(dataSet2, 0, n - 1);
   stop = high_resolution_clock::now();
   auto durationQuick = duration_cast<microseconds>(stop - start);

   cout << "Quick Sort time: " << durationQuick.count() << " microseconds\n";

   if (durationBubble.count() > durationQuick.count()) {
      cout << "Quick Sort was "
           << (float)durationBubble.count() / durationQuick.count()
           << " times faster than Bubble Sort.\n";
   }

   char show;
   cout << "\nDo you want to see top 10 sorted rooms? (y/n): ";
   cin >> show;
   if (show == 'y' || show == 'Y') {
      hs.displayRooms(dataSet2);
   }

   return 0;
}
```

6. Implement an expression validator to check for balanced parentheses using a stack. Display valid or invalid based on proper symbol matching. (Hint: Push opening brackets; pop when matching closing bracket appears.)

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

int main() {
   string exp;
   cout << "Enter expression: ";
   getline(cin, exp);

   stack<char> st;

   for (char ch : exp) {

      // Push opening brackets
      if (ch == '(' || ch == '{' || ch == '[') {
         st.push(ch);
      }

      // Check closing brackets
      else if (ch == ')' || ch == '}' || ch == ']') {

         if (st.empty()) {
            cout << "Expression is INVALID\n";
            return 0;
         }

         char top = st.top();
         st.pop();

         if ((ch == ')' && top != '(') ||
            (ch == '}' && top != '{') ||
            (ch == ']' && top != '[')) {
            cout << "Expression is INVALID\n";
            return 0;
         }
      }
   }

   if (st.empty())
      cout << "Expression is VALID\n";
   else
      cout << "Expression is INVALID\n";

   return 0;
}
```

7. Design a bus ticket counter simulation using a circular queue where passengers board and exit efficiently. (Hint: Use front and rear logic to manage flow.)

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

#define SIZE 5   // Fixed size circular queue

class CircularQueue {
private:
   int front, rear;
   string arr[SIZE];

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

   bool isFull() {
      return (front == 0 && rear == SIZE - 1) || (rear + 1 == front);
   }

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

   void enqueue(string name) {
      if (isFull()) {
         cout << "Queue is FULL! Passenger cannot enter.\n";
         return;
      }
      if (isEmpty()) {
         front = rear = 0;
      } else {
         rear = (rear + 1) % SIZE;
      }
      arr[rear] = name;
      cout << name << " boarded the bus.\n";
   }

   void dequeue() {
      if (isEmpty()) {
         cout << "Queue is EMPTY! No passenger to exit.\n";
         return;
      }
      cout << arr[front] << " exited from the bus.\n";

      if (front == rear) {
         front = rear = -1; // queue becomes empty
      } else {
         front = (front + 1) % SIZE;
      }
   }

   void display() {
      if (isEmpty()) {
         cout << "No passengers in queue.\n";
         return;
      }

      cout << "Passengers in queue: ";
      int i = front;
      while (true) {
         cout << arr[i] << " ";
         if (i == rear) break;
         i = (i + 1) % SIZE;
      }
      cout << endl;
   }
};

int main() {
   CircularQueue q;
   int choice;
   string name;

   cout << "=== BUS TICKET COUNTER SIMULATION (Circular Queue) ===\n";

   while (true) {
      cout << "\n1. Passenger Board (Enqueue)";
      cout << "\n2. Passenger Exit (Dequeue)";
      cout << "\n3. Display Queue";
      cout << "\n4. Exit";
      cout << "\nEnter choice: ";
      cin >> choice;

      switch (choice) {
         case 1:
            cout << "Enter passenger name: ";
            cin >> name;
            q.enqueue(name);
            break;
         case 2:
            q.dequeue();
            break;
         case 3:
            q.display();
            break;
         case 4:
            cout << "Simulation Ended.\n";
            return 0;
         default:
            cout << "Invalid choice!\n";  }  }  }
```

8. Simulate a music playlist using a doubly linked list where users can play next, previous, add, or delete songs. (Hint: Use bidirectional navigation + insert/delete operations.)

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

class Node {
public:
    string song;
    Node* next;
    Node* prev;

    Node(string s) {
        song = s;
        next = prev = NULL;
    }
};

class Playlist {
    Node* head;
    Node* current;

public:
    Playlist() {
        head = current = NULL;
    }

    void addSong(string s) {
        Node* n = new Node(s);
        if (head == NULL) {
            head = current = n;
            cout << "Song added.\n";
            return;
        }

        Node* temp = head;
        while (temp->next != NULL) temp = temp->next;

        temp->next = n;
        n->prev = temp;

        cout << "Song added.\n";
    }

    void deleteCurrent() {
        if (current == NULL) {
            cout << "No song to delete.\n";
            return;
        }

        cout << "Deleted: " << current->song << "\n";

        if (current->prev) current->prev->next = current->next;
        else head = current->next;

        if (current->next) current->next->prev = current->prev;

        Node* temp = current;
        current = (current->next != NULL) ? current->next : current->prev;

        delete temp;
    }

    void playNext() {
        if (current && current->next) {
            current = current->next;
            cout << "Playing: " << current->song << "\n";
        } else
            cout << "No next song.\n";
    }

    void playPrev() {
        if (current && current->prev) {
            current = current->prev;
            cout << "Playing: " << current->song << "\n";
        } else
            cout << "No previous song.\n";
    }

    void showCurrent() {
        if (current)
            cout << "Currently Playing: " << current->song << "\n";
        else
            cout << "Playlist empty.\n";
    }
};

int main() {
    Playlist p;
    int ch;
    string name;

    while (1) {
        cout << "\n1. Add Song\n2. Delete Current Song\n3. Play Next\n4. Play
Previous\n5. Show Current\n6. Exit\nEnter: ";
        cin >> ch;

        switch (ch) {
            case 1:
                cout << "Song name: ";
                cin >> name;
                p.addSong(name);
                break;

            case 2:
                p.deleteCurrent();
                break;

            case 3:
                p.playNext();
                break;

            case 4:
                p.playPrev();
                break;

            case 5:
                p.showCurrent();
                break;

            case 6:
                return 0;
        }
    }
}
```

9. Implement a hospital patient record system where records are stored in a BST. Allow insertion, deletion, and searching based on patient ID. (Hint: Use integer IDs as keys in BST.)

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

class Node {
public:
    int id;
    Node* left;
    Node* right;

    Node(int x) {
        id = x;
        left = right = NULL;
    }
};

class BST {
public:
    Node* root;

    BST() { root = NULL; }

    Node* insert(Node* r, int x) {
        if (r == NULL) return new Node(x);
        if (x < r->id) r->left = insert(r->left, x);
        else if (x > r->id) r->right = insert(r->right, x);
        return r;
    }

    bool search(Node* r, int x) {
        if (r == NULL) return false;
        if (r->id == x) return true;
        if (x < r->id) return search(r->left, x);
        else return search(r->right, x);
    }

    Node* findMin(Node* r) {
        while (r->left != NULL) r = r->left;
        return r;
    }

    // Modified deleteNode to return NULL if node not found and use a flag pointer to detect deletion
    Node* deleteNode(Node* r, int x, bool &deleted) {
        if (r == NULL) return NULL;

        if (x < r->id)
            r->left = deleteNode(r->left, x, deleted);
        else if (x > r->id)
            r->right = deleteNode(r->right, x, deleted);
        else {
            deleted = true;
            // Node with only one child or no child
            if (r->left == NULL) {
                Node* temp = r->right;
                delete r;
                return temp;
            }
            else if (r->right == NULL) {
                Node* temp = r->left;
                delete r;
                return temp;
            }
            // Node with two children
            Node* temp = findMin(r->right);
            r->id = temp->id;
            r->right = deleteNode(r->right, temp->id, deleted);
        }
        return r;
    }

    void inorder(Node* r) {
        if (r == NULL) return;
        inorder(r->left);
        cout << r->id << " ";
        inorder(r->right);
    }
};

int main() {
    BST tree;
    int ch, id;

    while (1) {
        cout << "\n1. Insert Patient\n2. Search Patient\n3. Delete Patient\n4. Display Records (Inorder)\n5. Exit\nEnter choice: ";
        cin >> ch;

        switch (ch) {
            case 1:
                cout << "Enter Patient ID: ";
                cin >> id;
                tree.root = tree.insert(tree.root, id);
                cout << "Patient added.\n";
                break;

            case 2:
                cout << "Enter Patient ID to search: ";
                cin >> id;
                if (tree.search(tree.root, id))
                    cout << "Patient FOUND.\n";
                else
                    cout << "Patient NOT found.\n";
                break;

            case 3: {
                cout << "Enter Patient ID to delete: ";
                cin >> id;
                bool deleted = false;
                tree.root = tree.deleteNode(tree.root, id, deleted);
                if (deleted)
                    cout << "Patient successfully deleted.\n";
                else
                    cout << "Patient NOT found.\n";
                break;
            }

            case 4:
                cout << "Patient Records (Sorted IDs): ";
                tree.inorder(tree.root);
                cout << endl;
                break;

            case 5:
                return 0;
        }
    }
}
```

10. Create a social network friend suggestion system using graphs where each user is a node. Suggest friends using BFS-based shortest paths. (Hint: Suggest nodes at level-2 distance.) Enter number of users: 5 Friend connections (edges): 1 2 1 3 2 4 3 5 Suggest friends for user: 1

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

class SocialNetwork {
   int users;
   vector<vector<int>> adjList;

public:
   SocialNetwork(int n) {
      users = n;
      adjList.resize(n + 1);  // 1-based indexing
   }

   void addConnection(int u, int v) {
      adjList[u].push_back(v);
      adjList[v].push_back(u);  // undirected graph (friendship)
   }

   void suggestFriends(int startUser) {
      vector<int> level(users + 1, -1);
      queue<int> q;

      level[startUser] = 0;
      q.push(startUser);

      while (!q.empty()) {
         int curr = q.front();
         q.pop();

         for (int neighbor : adjList[curr]) {
            if (level[neighbor] == -1) {
               level[neighbor] = level[curr] + 1;
               q.push(neighbor);
            }
         }
      }

      cout << "Friend suggestions for user " << startUser << " (friends at level
2):\n";
      bool found = false;
      for (int i = 1; i <= users; i++) {
         if (level[i] == 2) { // level-2 nodes = friends-of-friends
            cout << i << " ";
            found = true;
         }
      }
      if (!found) cout << "No friend suggestions available.";
      cout << endl;
   }
};

int main() {
   int n, u, v, user;
   cout << "Enter number of users: ";
   cin >> n;

   SocialNetwork sn(n);

   cout << "Enter friend connections (edges):\n";
   for (int i = 0; i < n - 1; i++) {  // usually n-1 edges for connected graph, but here
no restriction given
      cin >> u >> v;
      sn.addConnection(u, v);
   }

   cout << "Suggest friends for user: ";
   cin >> user;

   sn.suggestFriends(user);

   return 0;
}
```

11. Develop a product lookup system using hashing where items are stored by SKU codes. Handle collisions using linear probing. (Hint: Use open addressing to resolve collisions.) Enter number of products: 5 SKUs: 101, 205, 330, 412, 101 (duplicate to test collision) Search SKU: 330

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

class ProductHash {
  vector<int> table;
  int size;
  int empty = -1;  // denotes empty slot

  int hashFunction(int sku) {
    return sku % size;
  }

public:
  ProductHash(int s) {
    size = s;
    table.resize(size, empty);
  }

  bool insert(int sku) {
    int idx = hashFunction(sku);
    int startIdx = idx;

    do {
      if (table[idx] == empty) {
        table[idx] = sku;
        cout << "Inserted SKU " << sku << " at index " << idx << endl;
        return true;
      }
      else if (table[idx] == sku) {
        cout << "Duplicate SKU " << sku << " found at index " << idx << ". Not
inserting.\n";
        return false;
      }
      idx = (idx + 1) % size;
    } while (idx != startIdx);

    cout << "Hash table full! Cannot insert SKU " << sku << endl;
    return false;
  }

  bool search(int sku) {
    int idx = hashFunction(sku);
    int startIdx = idx;

    do {
      if (table[idx] == sku) {
        cout << "SKU " << sku << " found at index " << idx << endl;
        return true;
      }
      else if (table[idx] == empty) {
        // Empty slot means not found
        return false;
      }
      idx = (idx + 1) % size;
    } while (idx != startIdx);

    return false;
  }

  void display() {
    cout << "\nHash Table Contents:\n";
    for (int i = 0; i < size; i++) {
      if (table[i] == empty)
        cout << "Index " << i << ": Empty\n";
      else
        cout << "Index " << i << ": " << table[i] << endl;
    }
  }
};

int main() {
  int n, sku;
  cout << "Enter number of products: ";
  cin >> n;

  ProductHash ph(n);

  cout << "Enter SKUs:\n";
  for (int i = 0; i < n; i++) {
    cin >> sku;
    ph.insert(sku);
  }

  cout << "\nEnter SKU to search: ";
  cin >> sku;

  if (!ph.search(sku)) {
    cout << "SKU " << sku << " not found.\n";
  }

  ph.display();

  return 0;
}
```

12. Simulate a book-arranging robot that sorts books by their IDs using merge sort.

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

void merge(vector<int> &arr, int st, int mid, int end)
{
   vector<int> temp;
   int i = st, j = mid + 1;

   while (i <= mid && j <= end)
   {
      if (arr[i] <= arr[j])
      { // use <= for stability
         temp.push_back(arr[i]);
         i++;
      }
      else
      {
         temp.push_back(arr[j]);
         j++;
      }
   }

   while (i <= mid)
   {
      temp.push_back(arr[i]);
      i++;
   }

   while (j <= end)
   {
      temp.push_back(arr[j]);
      j++;
   }

   for (int idx = 0; idx < temp.size(); idx++)
   {
      arr[idx + st] = temp[idx];
   }
}

void mergeSort(vector<int> &arr, int st, int end)
{
   if (st < end)
   {
      int mid = st + (end - st) / 2; // fixed mid calculation
      mergeSort(arr, st, mid);       // left half
      mergeSort(arr, mid + 1, end);  // right half

      merge(arr, st, mid, end);
   }
}

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

   vector<int> books(n);
   cout << "Enter book IDs:\n";
   for (int i = 0; i < n; i++)
   {
      cin >> books[i];
   }

   mergeSort(books, 0, n - 1);
   cout << "Books sorted by ID:\n";
   for (int id : books)
   {
      cout << id << " ";
   }
   cout << endl;
   return 0;
}
```

13. Build a browser history manager using two stacks for back and forward navigation. (Hint: Push/Pop operations simulate real browser behavior.)

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

int main() {
   stack<string> backStack;
   stack<string> forwardStack;
   string currentPage = "Home";   // starting page
   int choice;
   string url;

   cout << "=== BROWSER HISTORY MANAGER USING TWO STACKS ===\n";
   cout << "Current Page: " << currentPage << "\n";

   while (true) {
      cout << "\n1. Visit New Page";
      cout << "\n2. Back";
      cout << "\n3. Forward";
      cout << "\n4. Exit";
      cout << "\nEnter your choice: ";
      cin >> choice;

      switch (choice) {

         case 1: // Visit new page
            cout << "Enter URL: ";
            cin >> url;

            backStack.push(currentPage);   // current page stored in back stack
            currentPage = url;

            // visiting a new page clears forward stack
            while (!forwardStack.empty())
               forwardStack.pop();

            cout << "You are now on: " << currentPage << "\n";
            break;

         case 2: // Back navigation
            if (backStack.empty()) {
               cout << "No pages in BACK history.\n";
            } else {
               forwardStack.push(currentPage);  // save current page into forward
stack
               currentPage = backStack.top();   // move to previous page
               backStack.pop();

               cout << "Went BACK to: " << currentPage << "\n";
            }
            break;

         case 3: // Forward navigation
            if (forwardStack.empty()) {
               cout << "No pages in FORWARD history.\n";
            } else {
               backStack.push(currentPage);     // save current into back stack
               currentPage = forwardStack.top(); // go forward
               forwardStack.pop();

               cout << "Went FORWARD to: " << currentPage << "\n";
            }
            break;

         case 4:
            cout << "Exiting Browser History Manager.\n";
            return 0;

         default:
            cout << "Invalid choice! Try again.\n";
      }
   }
}
```

14. Develop an ambulance dispatch system using a max priority queue to serve highest priority cases first. (Hint: Priority determines removal order.)

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

struct Case {
    string patientName;
    int priority;

    // Overload < operator for max-priority queue
    bool operator<(const Case &c) const {
        return priority < c.priority;  // Higher priority served first
    }
};

int main() {
    priority_queue<Case> pq;
    int n;

    cout << "AMBULANCE DISPATCH SYSTEM (Max Priority Queue)\n";
    cout << "-----------------------------------------------\n";

    cout << "Enter number of emergency cases: ";
    cin >> n;
    cin.ignore();

    // Input cases
    for (int i = 0; i < n; i++) {
        Case c;
        cout << "\nEnter patient name: ";
        getline(cin, c.patientName);

        cout << "Enter priority (higher = more critical): ";
        cin >> c.priority;
        cin.ignore();

        pq.push(c);
    }

    cout << "\nDispatching Ambulances (Highest Priority First):\n";

    // Process cases by priority
    while (!pq.empty()) {
        Case current = pq.top();
        pq.pop();

        cout << "Dispatching ambulance for: " << current.patientName
             << " (Priority: " << current.priority << ")\n";
    }

    cout << "\nAll emergency cases handled.\n";

    return 0;
}
```

15. Create a book-return bin simulator where returned books are stacked and then processed in LIFO order. (Hint: Simple stack push/pop.)

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

#define SIZE 10

class Stack {
private:
  int top;
  string arr[SIZE];

public:
  Stack() {
    top = -1;
  }

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

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

  void push(string book) {
    if (isFull()) {
      cout << "Return bin is FULL! Cannot add more books.\n";
      return;
    }
    arr[++top] = book;
    cout << book << " returned and added to bin.\n";
  }

  void pop() {
    if (isEmpty()) {
      cout << "Return bin is EMPTY! No books to process.\n";
      return;
    }
    cout << arr[top] << " processed from the bin.\n";
    top--;
  }

  void display() {
    if (isEmpty()) {
      cout << "No books in return bin.\n";
      return;
    }
    cout << "Books in return bin (top to bottom): ";
    for (int i = top; i >= 0; i--) {
      cout << arr[i] << " ";
    }
    cout << endl;
  }
};

int main() {
  Stack s;
  int choice;
  string book;

  cout << "=== BOOK RETURN BIN SIMULATOR (STACK - LIFO) ===\n";

  while (true) {
    cout << "\n1. Return Book (Push)";
    cout << "\n2. Process Book (Pop)";
    cout << "\n3. Display Books in Bin";
    cout << "\n4. Exit";
    cout << "\nEnter choice: ";
    cin >> choice;

    switch (choice) {
      case 1:
        cout << "Enter book name: ";
        cin >> book;
        s.push(book);
        break;

      case 2:
        s.pop();
        break;

      case 3:
        s.display();
        break;

      case 4:
        cout << "Simulation Ended.\n";
        return 0;

      default:
        cout << "Invalid choice!\n";
    }
  }
}
```

16. Implement a railway parcel management system where parcels are stored in hash tables using their tracking numbers. (Hint: Use modulo hashing + chaining.)

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

class ParcelHashTable {
   int tableSize;
   vector<list<int>> table;  // each list stores tracking numbers (chaining)

   int hashFunction(int trackingNumber) {
      return trackingNumber % tableSize;
   }

public:
   ParcelHashTable(int size) {
      tableSize = size;
      table.resize(tableSize);
   }

   void insert(int trackingNumber) {
      int index = hashFunction(trackingNumber);
      table[index].push_back(trackingNumber);
      cout << "Parcel " << trackingNumber << " inserted at index " << index <<
endl;
   }

   bool search(int trackingNumber) {
      int index = hashFunction(trackingNumber);
      for (int num : table[index]) {
         if (num == trackingNumber) {
            return true;
         }
      }
      return false;
   }

   void display() {
      cout << "\nParcel Hash Table:\n";
      for (int i = 0; i < tableSize; i++) {
         cout << "Index " << i << ": ";
         for (int num : table[i]) {
            cout << num << " -> ";
         }
         cout << "NULL\n";
      }
   }
};

int main() {
   int size, choice, trackingNum;

   cout << "Enter hash table size: ";
   cin >> size;

   ParcelHashTable parcels(size);

   while (true) {
      cout << "\n1. Insert Parcel\n2. Search Parcel\n3. Display Table\n4.
Exit\nEnter choice: ";
      cin >> choice;

      switch(choice) {
         case 1:
            cout << "Enter tracking number to insert: ";
            cin >> trackingNum;
            parcels.insert(trackingNum);
            break;

         case 2:
            cout << "Enter tracking number to search: ";
            cin >> trackingNum;
            if (parcels.search(trackingNum))
               cout << "Parcel " << trackingNum << " found.\n";
            else
               cout << "Parcel " << trackingNum << " not found.\n";
            break;

         case 3:
            parcels.display();
            break;

         case 4:
            cout << "Exiting...\n";
            return 0;

         default:
            cout << "Invalid choice! Try again.\n";
      }
   }
}
```

17. Simulate a food delivery route planner using graphs to show all delivery paths using DFS. (Hint: Treat restaurants and deliveries as nodes.)

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

class Graph {
   int V;
   vector<vector<int>> adj;

public:
   Graph(int v) {
      V = v;
      adj.resize(V);
   }

   void addEdge(int u, int v) {
      adj[u].push_back(v);
      adj[v].push_back(u);   // Undirected graph
   }

   void dfsPaths(int src, int dest, vector<bool> &visited, vector<int> &path) {
      visited[src] = true;
      path.push_back(src);

      if (src == dest) {
         for (int node : path)
            cout << node << " ";
         cout << endl;
      }
      else {
         for (int nbr : adj[src]) {
            if (!visited[nbr]) {
               dfsPaths(nbr, dest, visited, path);
            }
         }
      }

      path.pop_back();
      visited[src] = false;
   }

   void findAllPaths(int start, int end) {
      vector<bool> visited(V, false);
      vector<int> path;
      cout << "\nAll possible delivery paths (DFS):\n";
      dfsPaths(start, end, visited, path);
   }
};

int main() {
   int V, E, u, v;
   cout << "Enter number of nodes: ";
   cin >> V;

   Graph g(V);

   cout << "Enter number of paths: ";
   cin >> E;

   cout << "Enter paths (u v):\n";
   for (int i = 0; i < E; i++) {
      cin >> u >> v;
      g.addEdge(u, v);
   }

   int start, dest;
   cout << "Enter Restaurant node (start): ";
   cin >> start;
   cout << "Enter Delivery node (destination): ";
   cin >> dest;

   g.findAllPaths(start, dest);

   return 0;
}
```

18. Simulate a food delivery route planner using graphs to show the shortest path estimations using BFS. (Hint: Treat restaurants and deliveries as nodes.)

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

// BFS shortest path function
void bfsShortestPath(map<string, vector<string>> &graph,
                     string start, string dest) {

   map<string, bool> visited;
   map<string, string> parent;  // to reconstruct path
   queue<string> q;

   q.push(start);
   visited[start] = true;
   parent[start] = "";

   // BFS
   while (!q.empty()) {
      string node = q.front();
      q.pop();

      if (node == dest)
         break;

      for (string neigh : graph[node]) {
         if (!visited[neigh]) {
            visited[neigh] = true;
            parent[neigh] = node;
            q.push(neigh);
         }
      }
   }

   // If no path found
   if (!visited[dest]) {
      cout << "No delivery path found.\n";
      return;
   }

   // Reconstruct shortest path
   vector<string> path;
   string temp = dest;

   while (temp != "") {
      path.push_back(temp);
      temp = parent[temp];
   }

   // Print in correct order
   cout << "\nShortest Delivery Route: ";
   for (int i = path.size()-1; i >= 0; i--) {
      cout << path[i];
      if (i != 0) cout << " -> ";
   }
   cout << "\n";
}

int main() {
   map<string, vector<string>> graph;
   int edges;

   cout << "Enter number of paths (edges): ";
   cin >> edges;

   cout << "Enter connected locations (Restaurant/Delivery points):\n";
   for (int i = 0; i < edges; i++) {
      string u, v;
      cin >> u >> v;
      graph[u].push_back(v);
      graph[v].push_back(u); // undirected graph
   }

   string start, dest;
   cout << "Enter Restaurant (start): ";
   cin >> start;

   cout << "Enter Delivery Location (destination): ";
   cin >> dest;

   bfsShortestPath(graph, start, dest);

   return 0;
}
```

19. Build a bank transaction recorder using a queue to process transactions in order. (Hint: FIFO processing of deposits/withdrawals.)

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

int main() {
   queue<string> transactions;
   int n;

   cout << "BANK TRANSACTION RECORDER (FIFO Queue)\n";
   cout << "----------------------------------------\n";

   cout << "Enter number of transactions: ";
   cin >> n;
   cin.ignore(); // clear buffer

   // Taking manual input and adding to queue
   for (int i = 0; i < n; i++) {
      string t;
      cout << "Enter transaction " << i + 1 << " (Deposit/Withdraw amount): ";
      getline(cin, t);
      transactions.push(t);
   }

   cout << "\nProcessing Transactions in FIFO Order:\n";

   // Process queue
   while (!transactions.empty()) {
      cout << "Processing: " << transactions.front() << endl;
      transactions.pop();
   }

   cout << "\nAll transactions processed successfully.\n";

   return 0;
}
```

20. Develop a railway timetable sorting tool where arrival times are sorted using insertion sort and selection sort. Compare their performance. (Hint: Time-based sorting comparison.)

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

// Insertion Sort Function
void insertionSortAlgo(vector<int>& arr) {
    int n = arr.size();
    for (int i = 1; i < n; i++) {
        int curr = arr[i];
        int prev = i - 1;
        while (prev >= 0 && arr[prev] > curr) {
            arr[prev + 1] = arr[prev];
            prev--;
        }
        arr[prev + 1] = curr;
    }
}

// Selection Sort Function
void selectionSortAlgo(vector<int>& arr) {
    int n = arr.size();
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIdx])
                minIdx = j;
        }
        swap(arr[i], arr[minIdx]);
    }
}

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

    vector<int> arrival(n);
    cout << "Enter arrival times (in minutes):\n";
    for (int i = 0; i < n; i++) {
        cin >> arrival[i];
    }

    cout << "\n--- Sorting Railway Timetable ---\n";

    // Insertion Sort Timing (Repeated 10,000 times)
    auto start1 = high_resolution_clock::now();
    for (int k = 0; k < 10000; k++) {
        vector<int> temp = arrival;
        insertionSortAlgo(temp);
    }
    auto end1 = high_resolution_clock::now();
    long long t1 = duration_cast<microseconds>(end1 - start1).count();

    // Display final sorted list once only
    vector<int> sorted1 = arrival;
    insertionSortAlgo(sorted1);
    cout << "\nInsertion Sort Result: ";
    for (int x : sorted1) cout << x << " ";
    cout << "\nTime taken (Insertion Sort): " << t1 << " microseconds\n";

    // Selection Sort Timing (Repeated 10,000 times)
    auto start2 = high_resolution_clock::now();
    for (int k = 0; k < 10000; k++) {
        vector<int> temp = arrival;
        selectionSortAlgo(temp);
    }
    auto end2 = high_resolution_clock::now();
    long long t2 = duration_cast<microseconds>(end2 - start2).count();

    vector<int> sorted2 = arrival;
    selectionSortAlgo(sorted2);
    cout << "\nSelection Sort Result: ";
    for (int x : sorted2) cout << x << " ";
    cout << "\nTime taken (Selection Sort): " << t2 << " microseconds\n";

    return 0;
}
```

21. Implement a polynomial calculator using linked lists where polynomials can be created, added, and simplified. (Hint: Each node contains coefficient and power.)

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

class Node {
public:
  int coef;
  int pow;
  Node* next;

  Node(int c, int p) {
    coef = c;
    pow = p;
    next = NULL;
  }
};

class Polynomial {
public:
  Node* head;

  Polynomial() {
    head = NULL;
  }

  // Insert term in descending order of powers
  void insertTerm(int coef, int pow) {
    Node* newNode = new Node(coef, pow);

    if (!head || pow > head->pow) {
      newNode->next = head;
      head = newNode;
      return;
    }

    Node* temp = head;
    while (temp->next && temp->next->pow >= pow)
      temp = temp->next;

    newNode->next = temp->next;
    temp->next = newNode;
  }

  // Add two polynomials
  Polynomial add(Polynomial &p2) {
    Polynomial result;
    Node* p1Node = head;
    Node* p2Node = p2.head;

    while (p1Node && p2Node) {
      if (p1Node->pow == p2Node->pow) {
        result.insertTerm(p1Node->coef + p2Node->coef, p1Node->pow);
        p1Node = p1Node->next;
        p2Node = p2Node->next;
      }
      else if (p1Node->pow > p2Node->pow) {
        result.insertTerm(p1Node->coef, p1Node->pow);
        p1Node = p1Node->next;
      }
      else {
        result.insertTerm(p2Node->coef, p2Node->pow);
        p2Node = p2Node->next;
      }
    }

    // Remaining terms
    while (p1Node) {
      result.insertTerm(p1Node->coef, p1Node->pow);
      p1Node = p1Node->next;
    }
    while (p2Node) {
      result.insertTerm(p2Node->coef, p2Node->pow);
      p2Node = p2Node->next;
    }
    return result;
  }

  void display() {
    Node* temp = head;
    if (!temp) {
      cout << "0\n";
      return;
    }

    while (temp) {
      cout << temp->coef << "x^" << temp->pow;
      temp = temp->next;
      if (temp)
        cout << " + ";
    }
    cout << endl;
  }
};

int main() {
  Polynomial p1, p2, sum;
  int n, coef, pow;

  cout << "=== POLYNOMIAL CALCULATOR USING LINKED LIST ===\n";

  // First Polynomial
  cout << "\nEnter number of terms in Polynomial 1: ";
  cin >> n;
  for (int i = 0; i < n; i++) {
    cout << "Enter coefficient and power: ";
    cin >> coef >> pow;
    p1.insertTerm(coef, pow);
  }

  // Second Polynomial
  cout << "\nEnter number of terms in Polynomial 2: ";
  cin >> n;
  for (int i = 0; i < n; i++) {
    cout << "Enter coefficient and power: ";
    cin >> coef >> pow;
    p2.insertTerm(coef, pow);
  }

  cout << "\nPolynomial 1: ";
  p1.display();

  cout << "Polynomial 2: ";
  p2.display();

  // Add
  sum = p1.add(p2);

  cout << "\nSUM (Simplified): ";
  sum.display();

  return 0;
}
```

22. Simulate a campus navigation system using weighted graphs where buildings are nodes; display shortest path using BFS (for unweighted). (Use adjacency list for efficiency.)

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

class CampusMap {
   map<string, vector<string>> adj;   // adjacency list

public:
   void addConnection(string a, string b) {
      adj[a].push_back(b);
      adj[b].push_back(a);          // undirected graph
   }

   void shortestPath(string start, string end) {
      map<string, bool> visited;
      map<string, string> parent;
      queue<string> q;

      q.push(start);
      visited[start] = true;
      parent[start] = "";

      while (!q.empty()) {
         string curr = q.front();
         q.pop();

         if (curr == end) break;

         for (string nxt : adj[curr]) {
            if (!visited[nxt]) {
               visited[nxt] = true;
               parent[nxt] = curr;
               q.push(nxt);
            }
         }
      }

      if (!visited[end]) {
         cout << "No path found!\n";
         return;
      }

      // Reconstruct path (backtracking using parent map)
      vector<string> path;
      string temp = end;
      while (temp != "") {
         path.push_back(temp);
         temp = parent[temp];
      }

      cout << "Shortest path from " << start << " to " << end << ":\n";
      for (int i = path.size() - 1; i >= 0; i--) {
         cout << path[i];
         if (i != 0) cout << " -> ";
      }
      cout << endl;
   }
};

int main() {
   CampusMap cm;

   // Adding buildings and connections
   cm.addConnection("Library", "Lab");
   cm.addConnection("Lab", "Hostel");
   cm.addConnection("Canteen", "Library");
   cm.addConnection("Admin", "Hostel");

   string start, end;
   cout << "Find path: ";
   cin >> start >> end;
   cm.shortestPath(start, end);

   return 0;
}
```

23. Build a shipment tracking queue where parcels enter the system and are processed in the order they arrive using a linear queue. (Hint: Simple FIFO process.)

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

#define MAX 100

class Queue {
   string arr[MAX];
   int front, rear;

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

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

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

   void enqueue(string parcel) {
      if (isFull()) {
         cout << "Queue is FULL. Cannot add parcel.\n";
         return;
      }
      if (front == -1)
         front = 0;

      rear++;
      arr[rear] = parcel;
      cout << parcel << " added to shipment queue.\n";
   }

   void dequeue() {
      if (isEmpty()) {
         cout << "Queue is EMPTY. No parcel to process.\n";
         return;
      }
      cout << "Processing parcel: " << arr[front] << endl;
      front++;
   }

   void display() {
      if (isEmpty()) {
         cout << "Queue is EMPTY.\n";
         return;
      }
      cout << "Current shipment queue:\n";
      for (int i = front; i <= rear; i++)
         cout << arr[i] << " ";
      cout << endl;
   }
};

int main() {
   Queue q;
   int n;
   cout << "Enter number of parcels: ";
   cin >> n;

   cout << "Enter parcel IDs (like P100, P102, P105):\n";
   for (int i = 0; i < n; i++) {
      string p;
      cin >> p;
      q.enqueue(p);
   }

   cout << "\n--- Shipment Queue ---\n";
   q.display();

   cout << "\nProcessing parcels in FIFO order:\n";
   while (!q.isEmpty()) {
      q.dequeue();
   }

   return 0;
}
```