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

// -------------------- INVENTORY ITEM ADT --------------------
struct InventoryItem {
    int itemID;
    string itemName;
    int quantity;
    float price;

    InventoryItem(int id = 0, string name = "", int qty = 0, float p =
0.0) {
        itemID = id;
        itemName = name;
        quantity = qty;
        price = p;
    }
};

// -------------------- INVENTORY MANAGEMENT SYSTEM --------------------
class InventorySystem {
private:
    vector<InventoryItem> items;

public:
    // Insert a new item
    void insertItem(int id, string name, int qty, float price) {
        for (auto &it : items) {
            if (it.itemID == id) {
                cout << "Item with this ID already exists!\n";
                return;
            }
        }
        items.push_back(InventoryItem(id, name, qty, price));
        cout << "Item inserted successfully!\n";
    }

    // Delete an item by ID
    void deleteItem(int id) {
        for (auto it = items.begin(); it != items.end(); ++it) {
            if (it->itemID == id) {
                items.erase(it);
                cout << "Item deleted successfully!\n";
                return;
            }
        }
        cout << "Item not found!\n";
    }

    // Search item by ID or Name
    void searchItem() {
        int choice;
        cout << "Search by: 1. Item ID  2. Item Name: ";
        cin >> choice;
```

```
        if (choice == 1) {
            int id;
            cout << "Enter Item ID: ";
            cin >> id;
            for (auto &it : items) {
                if (it.itemID == id) {
                    cout << "Item Found -> ID: " << it.itemID << ", Name:
" << it.itemName
                         << ", Qty: " << it.quantity << ", Price: " <<
it.price << "\n";
                    return;
                }
            }
            cout << "Item not found!\n";
        } else if (choice == 2) {
            string name;
            cout << "Enter Item Name: ";
            cin >> name;
            for (auto &it : items) {
                if (it.itemName == name) {
                    cout << "Item Found -> ID: " << it.itemID << ", Name:
" << it.itemName
                         << ", Qty: " << it.quantity << ", Price: " <<
it.price << "\n";
                    return;
                }
            }
            cout << "Item not found!\n";
        } else {
            cout << "Invalid choice!\n";
        }
    }

    // Display all items
    void displayItems() {
        if (items.empty()) {
            cout << "Inventory is empty!\n";
            return;
        }
        cout << "\n--- Inventory Items ---\n";
        cout << setw(8) << "ID" << setw(15) << "Name" << setw(10)
             << "Quantity" << setw(10) << "Price\n";
        cout << "-----------------------------------------\n";
        for (auto &it : items) {
            cout << setw(8) << it.itemID << setw(15) << it.itemName
                 << setw(10) << it.quantity << setw(10) << it.price <<
"\n";
        }
    }

    // Row-Major and Column-Major access
    void managePriceQuantity() {
        if (items.empty()) {
            cout << "Inventory is empty!\n";
            return;
        }
        int n = items.size();
        vector<vector<float>> pqTable(2, vector<float>(n, 0));
```

```
        for (int i = 0; i < n; i++) {
            pqTable[0][i] = items[i].price;
            pqTable[1][i] = items[i].quantity;
        }

        cout << "\nRow-Major Order (Price first, then Quantity):\n";
        for (int row = 0; row < 2; row++) {
            for (int col = 0; col < n; col++) {
                cout << setw(8) << pqTable[row][col] << " ";
            }
            cout << "\n";
        }

        cout << "\nColumn-Major Order:\n";
        for (int col = 0; col < n; col++) {
            for (int row = 0; row < 2; row++) {
                cout << setw(8) << pqTable[row][col] << " ";
            }
            cout << "\n";
        }
    }

    // Sparse Representation (Only store items with quantity <= 2)
    void optimizeSparseStorage() {
        cout << "\n--- Sparse Representation (Low Stock Items) ---\n";
        for (auto &it : items) {
            if (it.quantity <= 2) {
                cout << "ID: " << it.itemID << " Name: " << it.itemName
                     << " Qty: " << it.quantity << " Price: " << it.price
<< "\n";
            }
        }
    }

    // Complexity Analysis
    void analyzeComplexity() {
        cout << "\n--- Complexity Analysis ---\n";
        cout << "Insertion: O(1) (amortized for vector push_back)\n";
        cout << "Deletion: O(n) (may need to shift elements)\n";
        cout << "Search: O(n) (linear search by ID or Name)\n";
        cout << "Row/Column Access: O(n)\n";
        cout << "Sparse Representation: O(n)\n";
        cout << "Space: O(n) where n = number of items\n";
    }
};

// -------------------- MAIN FUNCTION (MENU) --------------------
int main() {
    InventorySystem system;
    int choice;

    do {
        cout << "\n------ GROCERY STORE INVENTORY SYSTEM ------\n";
        cout << "1. Insert Item\n";
        cout << "2. Delete Item\n";
        cout << "3. Search Item\n";
        cout << "4. Display All Items\n";
```

```
        cout << "5. Price-Quantity Table\n";
        cout << "6. Sparse Representation (Low Stock)\n";
        cout << "7. Complexity Analysis\n";
        cout << "8. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        int id, qty;
        string name;
        float price;

        switch (choice) {
            case 1:
                cout << "Enter Item ID: ";
                cin >> id;
                cout << "Enter Item Name: ";
                cin >> name;
                cout << "Enter Quantity: ";
                cin >> qty;
                cout << "Enter Price: ";
                cin >> price;
                system.insertItem(id, name, qty, price);
                break;

            case 2:
                cout << "Enter Item ID to delete: ";
                cin >> id;
                system.deleteItem(id);
                break;

            case 3:
                system.searchItem();
                break;

            case 4:
                system.displayItems();
                break;

            case 5:
                system.managePriceQuantity();
                break;

            case 6:
                system.optimizeSparseStorage();
                break;

            case 7:
                system.analyzeComplexity();
                break;

            case 8:
                cout << "Exiting...\n";
                break;

            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 8);
```

```
    return 0;
}
```