Develop the Inventory System for a Grocery Store
Theme: Arrays and Complexity Analysis
GitHub:

Code:

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

using namespace std;

struct InventoryItem {
    int itemID;
    string itemName;
    int quantity;
    float price;

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

```
struct SparseElement {
    int itemID;
    int quantity;
    SparseElement(int id, int qty) : itemID(id), quantity(qty) {}
};

class InventoryManagementSystem {
private:
    InventoryItem* itemArray;
    int capacity;
    int size;
    float** priceQuantityTable;
    int tableRows;
    int tableCols;
    vector<SparseElement> sparseMatrix;

public:
    InventoryManagementSystem(int cap = 100) {
        capacity = cap;
        size = 0;
        itemArray = new InventoryItem[capacity];
        tableRows = 0;
        tableCols = 2;
        priceQuantityTable = nullptr;
    }

    ~InventoryManagementSystem() {
        delete[] itemArray;
```

```
    if (priceQuantityTable != nullptr) {
        for (int i = 0; i < tableRows; i++) {
            delete[] priceQuantityTable[i];
        }
        delete[] priceQuantityTable;
    }
}

// Time: O(1) average, O(n) worst case when resizing needed
// Space: O(1)
void insertItem(int id, string name, int qty, float price) {
    if (size >= capacity) {
        cout << "Inventory full. Cannot insert item.\n";
        return;
    }
    itemArray[size++] = InventoryItem(id, name, qty, price);
    cout << "Item inserted successfully.\n";
}

// Time: O(n) - linear search + shifting elements
// Space: O(1)
bool deleteItem(int itemID) {
    int index = -1;
    for (int i = 0; i < size; i++) {
        if (itemArray[i].itemID == itemID) {
            index = i;
            break;
        }
    }
```

```
    if (index == -1) {
        cout << "Item not found.\n";
        return false;
    }

    for (int i = index; i < size - 1; i++) {
        itemArray[i] = itemArray[i + 1];
    }
    size--;
    cout << "Item deleted successfully.\n";
    return true;
}

// Time: O(n) - linear search
// Space: O(1)
int searchItemByID(int itemID) {
    for (int i = 0; i < size; i++) {
        if (itemArray[i].itemID == itemID) {
            return i;
        }
    }
    return -1;
}

// Time: O(n) - linear search
// Space: O(1)
int searchItemByName(string itemName) {
    for (int i = 0; i < size; i++) {
```

```
            if (itemArray[i].itemName == itemName) {
                return i;
            }
        }
        return -1;
    }

    void displayItem(int index) {
        if (index >= 0 && index < size) {
            cout << "ID: " << itemArray[index].itemID
                 << " | Name: " << itemArray[index].itemName
                 << " | Quantity: " << itemArray[index].quantity
                 << " | Price: $" << fixed << setprecision(2) <<
itemArray[index].price << "\n";
        }
    }

    // Time: O(n)
    // Space: O(1)
    void displayAllItems() {
        cout << "\n========== INVENTORY ==========\n";
        for (int i = 0; i < size; i++) {
            displayItem(i);
        }
        cout << "==============================\n";
    }

    // Time: O(n) - creating 2D array
    // Space: O(n*2)
```

```
    void createPriceQuantityTable() {
        if (priceQuantityTable != nullptr) {
            for (int i = 0; i < tableRows; i++) {
                delete[] priceQuantityTable[i];
            }
            delete[] priceQuantityTable;
        }

        tableRows = size;
        priceQuantityTable = new float*[tableRows];
        for (int i = 0; i < tableRows; i++) {
            priceQuantityTable[i] = new float[tableCols];
        }

        for (int i = 0; i < size; i++) {
            priceQuantityTable[i][0] = itemArray[i].price;
            priceQuantityTable[i][1] = (float)itemArray[i].quantity;
        }
    }

    // Time: O(n*m) where n=rows, m=cols
    // Space: O(1)
    void displayRowMajor() {
        cout << "\n=== Price-Quantity Table (Row-Major) ===\n";
        cout << setw(15) << "Price" << setw(15) << "Quantity" <<
"\n";
        for (int i = 0; i < tableRows; i++) {
            for (int j = 0; j < tableCols; j++) {
```

```
                cout << setw(15) << fixed << setprecision(2) <<
priceQuantityTable[i][j];
            }
            cout << "\n";
        }
    }

    // Time: O(n*m)
    // Space: O(1)
    void displayColumnMajor() {
        cout << "\n=== Price-Quantity Table (Column-Major) ===\n";
        for (int j = 0; j < tableCols; j++) {
            cout << (j == 0 ? "Prices: " : "Quantities: ");
            for (int i = 0; i < tableRows; i++) {
                cout << priceQuantityTable[i][j] << " ";
            }
            cout << "\n";
        }
    }

    // Time: O(n)
    // Space: O(k) where k is number of sparse elements
    void createSparseRepresentation(int threshold = 10) {
        sparseMatrix.clear();
        cout << "\n=== Creating Sparse Matrix for items with
quantity < " << threshold << " ===\n";

        for (int i = 0; i < size; i++) {
            if (itemArray[i].quantity < threshold &&
itemArray[i].quantity > 0) {
```

```
                sparseMatrix.push_back(SparseElement(itemArray[i].it
emID, itemArray[i].quantity));
            }
        }

        cout << "Sparse elements stored: " << sparseMatrix.size() <<
"\n";
    }

    // Time: O(k) where k is sparse elements
    // Space: O(1)
    void displaySparseMatrix() {
        cout << "\n=== Sparse Matrix (Rarely Restocked Items)
===\n";
        cout << setw(15) << "ItemID" << setw(15) << "Quantity" <<
"\n";
        for (const auto& elem : sparseMatrix) {
            cout << setw(15) << elem.itemID << setw(15) <<
elem.quantity << "\n";
        }

        int originalSpace = size * sizeof(int);
        int sparseSpace = sparseMatrix.size() *
sizeof(SparseElement);
        cout << "Space saved: " << originalSpace - sparseSpace << "
bytes\n";
    }

    // Time: O(n)
    // Space: O(1)
    void checkLowStock(int threshold = 10) {
```

```
        cout << "\n=== Low Stock Alert (Quantity < " << threshold <<
") ===\n";
        bool found = false;
        for (int i = 0; i < size; i++) {
            if (itemArray[i].quantity < threshold) {
                displayItem(i);
                found = true;
            }
        }
        if (!found) {
            cout << "No low stock items.\n";
        }
    }

    // Time: O(n)
    // Space: O(1)
    void generateSummaryReport() {
        if (size == 0) {
            cout << "\nNo items in inventory.\n";
            return;
        }

        float totalValue = 0;
        int totalItems = 0;
        float avgPrice = 0;

        for (int i = 0; i < size; i++) {
            totalValue += itemArray[i].price *
itemArray[i].quantity;
```

```
            totalItems += itemArray[i].quantity;
            avgPrice += itemArray[i].price;
        }

        avgPrice /= size;

        cout << "\n========== SUMMARY REPORT ==========\n";
        cout << "Total Items Types: " << size << "\n";
        cout << "Total Items Count: " << totalItems << "\n";
        cout << "Total Inventory Value: $" << fixed <<
setprecision(2) << totalValue << "\n";
        cout << "Average Item Price: $" << avgPrice << "\n";
        cout << "====================================\n";
    }

    int getSize() { return size; }
};

void displayMenu() {
    cout << "\n========== INVENTORY MENU ==========\n";
    cout << "1. Add Item\n";
    cout << "2. Delete Item\n";
    cout << "3. Search Item by ID\n";
    cout << "4. Search Item by Name\n";
    cout << "5. Display All Items\n";
    cout << "6. Create & Display Price-Quantity Table (Row-
Major)\n";
    cout << "7. Display Column-Major Order\n";
    cout << "8. Create Sparse Representation\n";
```

```
    cout << "9. Display Sparse Matrix\n";
    cout << "10. Check Low Stock\n";
    cout << "11. Generate Summary Report\n";
    cout << "0. Exit\n";
    cout << "====================================\n";
    cout << "Enter choice: ";
}

int main() {
    InventoryManagementSystem ims(100);

    ims.insertItem(101, "Rice", 50, 25.50);
    ims.insertItem(102, "Wheat", 30, 22.00);
    ims.insertItem(103, "Sugar", 5, 40.00);
    ims.insertItem(104, "Salt", 8, 15.00);
    ims.insertItem(105, "Oil", 45, 120.00);
    ims.insertItem(106, "Tea", 3, 350.00);
    ims.insertItem(107, "Coffee", 2, 450.00);

    int choice;
    do {
        displayMenu();
        cin >> choice;

        switch(choice) {
            case 1: {
                int id, qty;
                string name;
                float price;
```

```
        cout << "Enter Item ID: ";
        cin >> id;
        cout << "Enter Item Name: ";
        cin.ignore();
        getline(cin, name);
        cout << "Enter Quantity: ";
        cin >> qty;
        cout << "Enter Price: ";
        cin >> price;
        ims.insertItem(id, name, qty, price);
        break;
    }
    case 2: {
        int id;
        cout << "Enter Item ID to delete: ";
        cin >> id;
        ims.deleteItem(id);
        break;
    }
    case 3: {
        int id;
        cout << "Enter Item ID to search: ";
        cin >> id;
        int index = ims.searchItemByID(id);
        if (index != -1) {
            cout << "Item found:\n";
            ims.displayItem(index);
        } else {
            cout << "Item not found.\n";
```

```
    }
    break;
}
case 4: {
    string name;
    cout << "Enter Item Name to search: ";
    cin.ignore();
    getline(cin, name);
    int index = ims.searchItemByName(name);
    if (index != -1) {
        cout << "Item found:\n";
        ims.displayItem(index);
    } else {
        cout << "Item not found.\n";
    }
    break;
}
case 5:
    ims.displayAllItems();
    break;
case 6:
    ims.createPriceQuantityTable();
    ims.displayRowMajor();
    break;
case 7:
    if (ims.getSize() > 0) {
        ims.createPriceQuantityTable();
        ims.displayColumnMajor();
    } else {
```

```
                cout << "No items in inventory.\n";
            }
            break;
        case 8: {
            int threshold;
            cout << "Enter threshold for sparse representation:
";
            cin >> threshold;
            ims.createSparseRepresentation(threshold);
            break;
        }
        case 9:
            ims.displaySparseMatrix();
            break;
        case 10: {
            int threshold;
            cout << "Enter low stock threshold: ";
            cin >> threshold;
            ims.checkLowStock(threshold);
            break;
        }
        case 11:
            ims.generateSummaryReport();
            break;
        case 0:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice.\n";
```

```
        }
    } while(choice != 0);

    return 0;
}
```

OUTPUT:

Output

```
========== INVENTORY MENU ==========
1. Add Item
2. Delete Item
3. Search Item by ID
4. Search Item by Name
5. Display All Items
6. Create & Display Price-Quantity Table (Row-Major)
7. Display Column-Major Order
8. Create Sparse Representation
9. Display Sparse Matrix
10. Check Low Stock
11. Generate Summary Report
0. Exit
====================================
Enter choice: 1
Enter Item ID: 001
Enter Item Name: Red Bull
Enter Quantity: 1000
Enter Price: 60
Item inserted successfully.

========== INVENTORY MENU ==========
```