INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

School of Computer Science and Engineering

-----*-----**

PROJECT REPORT

STORE MANAGER

Advisor: Dr Tran Thanh Tung and Thai Trung Tin

Course: ALGORITHMS AND DATA STRUCTURES

No.	Full Name	Student's ID
1	Đỗ Thanh Bảo Anh	ITDSIU22175
2	Đào Ngọc Lan Hồng	ITDSIU21088
3	Phạm Tuấn Đăng Khoa	ITITIU22087
4	Nguyễn Thị Ngọc Mai	ITDSIU21098
5	Trần Triệu Như	ITDSIU21029

CHAPTER 1: INTRODUCTION

1. Objectives

The goal of the project is to develop a system that analyzes Java source code using advanced data structures and algorithms. The key objectives are as follows:

- **Efficient Data Handling**: The system is designed to utilize suitable data structures, such as trees and linked lists, to effectively manage and organize the data under analysis. By employing a tree structure, it facilitates quick retrieval and storage of information related to products and purchase records.
- **Optimized Searching and Sorting**: A focus on implementing algorithms that enhance the speed of searching and sorting operations is paramount. The use of sorted linked priority queues enables efficient management and retrieval of data based on specific criteria, ensuring rapid access to relevant information.
- **Robust Exception Handling**: It is essential for the system to incorporate effective exception handling, particularly when managing file operations and user inputs. This capability guarantees that the analysis process remains reliable and robust, allowing for meaningful feedback even in the event of errors.
- **Scalability**: The algorithms and data structures are crafted to scale effectively with the increasing size of data. As the volume of the codebase or analyzed data expands, the system will maintain its performance without significant degradation.
- **User-Friendly Interaction**: By structuring data and algorithms to support clear outputs and interfaces, the system aims to enhance user interaction. This includes providing detailed explanations of code functionality and structure, thereby simplifying the interpretation of analysis results for developers.

2. The Tools Used

- IDE for programming and debugging: IntelliJ, VSCode.
- Design: Piskel, Simple2DTileEditor.
- Java Development Kit: 21.
- Mean of code version management: GitHub.
- Means of contacting: Facebook

CHAPTER 2: TIME COMPLEXITY

1. List of Time Complexity

	Algorithm / DataStructure	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Discounted Products	***addQueue***	O(n)	O(n)	O(n)	O(n)/O(1)
	add	O(1)	O(n)	O(n)	O(n)/O(1)
	loadDtatatoSale	O(1)	O(1)	O(1)	O(1)
Best-Selling Products	***sortAmountSold***	O(n · k)	O(n · k)	O(n · k)	O(n)
	loadDtatatoBestSeller	O(n+n.k)	O(n.(k+log m))	O(n.(k+log m))	O(n)
Search Product by attribute	***getInOrderList()***	O(n)	O(n)	O(n)	O(n)
	searchElementName	O(n)	O(n)	O(n)	O(n)

Filter Products by Price	***sortPriceList***	O(n-logn)	O(n-logn)	O(n-logn)	O(n)
Search Product by Product Components	***searchProductsByComponentsInGraph***	O(m)	O(n^2)	O(n^2)	O(n)
Undo	***functionStack.pop()***	O(1)	O(1)	O(1)	O(n)
Inventory statistics	***checkStockAndExpiry***	O(n^2)	O(n^2)	O(n^2)	O(n)

CHAPTER 3: DATA STRUCTURE

Data structures to implement established system's components are selected considering their prominent features.

Package	File / Function	Explain
AVL	***AccountManagerTree***: manages account information for users	We utilize an AVL tree, a self-balancing Binary Search Tree (BST), to efficiently manage account information. This structure organizes Account, Product, PhieuMua, and AmountSold objects, enabling quick addition, removal, and search operations. With O(log n) time complexity for search, insert, and delete operations, the AVL tree outperforms unbalanced binary trees, which can degrade to O(n). It also allows faster searches than linked lists and supports dynamic resizing, unlike arrays. Overall, the AVL tree ensures efficient data storage and retrieval, making it ideal for applications requiring quick access and clear organization of data.
	AmountSoldManagerTree: tracks sales data for products.	
	PhieuMuaManagerTree: manages billing information for purchases	
	ProductManagerTree: manages product information	
Graph	***Edge:*** use array list to store edges (product names)	The graph data structure consists of a set of vertices and edges connecting them, allowing for the modeling of complex relationships between components, such as in product management. Using graphs for searching product components offers several advantages, including the ability to represent complex relationships, quick searches through algorithms, and easy expansion to add new products. Compared to other data structures, graphs do not require a fixed size like arrays, allow for more flexible relationships than linked lists, and support multiple connections between vertices unlike trees. In summary, using graphs optimizes the search process and organizes information effectively.
	ListGraph: represents the relationship between two vertices (product components)	
Sorted Linked Priority Queue	***QueueSale***	We designed a Sorted Linked Priority Queue for managing products on sale, prioritizing those with a quantity greater than 30 and favorable expiration dates for quick processing, such as a 20% discount to reduce inventory waste. By maintaining a sorted state based on quantity and expiration, this structure minimizes search time for products that need urgent attention.
HashMap	***totalBoughtQuantities***	With the advantage of O(1) lookup, add and update speed, HashMap is a structure that is very suitable for mapping operations between keys and values. Therefore, we have taken advantage of it to store and quickly look up the total number of purchased

Hash-based		***componentVertexMap***	products to check for mismatches between the purchased and sold quantities in the MismatchInfo function. In addition, HashMap is also used to map component names with vertex indices in the graph to support the conversion of trees into graphs.
	HashSet	***componentSet***	To avoid duplication in collection and storage, we used a HashSet in the Run class to store the product components. Using this data structure is efficient in determining the exact number of components and provides an average retrieval and insertion time of O(1). This ensures that adding components to the set is fast, even when the product list is large. Additionally, storing all unique components in a HashSet before they are used to create vertices in the productGraph ensures that only necessary and unique components are added to the graph, avoiding memory waste.
ArrayList		***chiTietPhieuList***	In addition, we also use ArrayList to conveniently store product lists, PhieuMua, ChiTietPhieu... These lists are retrieved sequentially (getInOrderList) from AVL trees to serve other processing tasks or used to store the results of functions.
		mismatchedProducts	
		productNames	
Stack		***functionStack***	We use the stack to store previously performed account-related operations that need to be undone (add, delete, edit information). Each account-related operation (add, delete, edit) will be saved to functionStack as a FunctionWrapper<Account> object. Because the stack rule is LIFO, the most recent operation will be taken first to perform undo.

CHAPTER 4: ALGORITHM

1. List features

	Quản lý kho	Admin	Customer
Login to system	**Y**	**Y**	**Y**
See the produce	**Y**	**Y**	**Y**
Add/Remove products	**Y**	**Y**	**Y**
Edit product information	**Y**	**Y**	**N**
Search for product information	**Y**	**Y**	**Y**
Product arrangement	**Y**	**Y**	**Y**
Add/Remove account	**N**	**Y**	**N**
Edit account information	**N**	**Y**	**Y**
Search for account information	**N**	**Y**	**N**
Best-seller Statistics	**Y**	**N**	**Y**
Inventory Statistics	**Y**	**N**	**N**
Update tracking goods	**N**	**Y**	**Y**
Cancellation Form	**Y**		**Y**
Discount method	**Y**	**N**	**Y**
Edit orders	**N**	**N**	**Y**

2. Customer

a) Discounted Products:

- The addQueue function will read each row of data, create a Product object, and add it to the queue.

```
    public static void addQueue(SortedLinkedPriorityQueue<Integer, Product> queue, Scanner
scanner) {
        //<-----
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

        while (scanner.hasNext()) {
            try {
                String maSanPham = scanner.nextLine();
                String tenSanPham = scanner.nextLine();
                String soLuong = scanner.nextLine();
                String donGia = scanner.nextLine();
                //LocalDate ngaySanXuat = LocalDate.parse(scanner.nextLine(), formatter);
                //LocalDate hanSuDung = LocalDate.parse(scanner.nextLine(), formatter);
                String ngaySanXuat = scanner.nextLine();
                String hanSuDung = scanner.nextLine();
                String thanhPhan = scanner.nextLine();
                String khoiLuong = scanner.nextLine();
                //LocalDate ngayNhapKHo = LocalDate.parse(scanner.nextLine(), formatter);
                String ngayNhapKHo = scanner.nextLine();
                String soNgayGiaoHang = scanner.nextLine();

                // In ra thông tin sản phẩm đọc được
                System.out.println("Đang thêm sản phẩm: " + maSanPham + ", " + tenSanPham +
", " + soLuong);

                Product a = new Product(maSanPham, tenSanPham, Integer.parseInt(soLuong),
                        Double.parseDouble(donGia), ngaySanXuat,
                        hanSuDung, thanhPhan, khoiLuong,
                        ngayNhapKHo, Integer.parseInt(soNgayGiaoHang));

                queue.add(a, 30, queue); // 30 là số lượng hàng trong kho, nếu lớn hơn 30
thì sẽ sale
                System.out.println("Sản phẩm đã được thêm vào hàng đợi: " + a);
                System.out.println("Số lượng sản phẩm trong hàng đợi hiện tại: " +
queue.size());
            } catch (Exception e) {
                System.out.println("Lỗi khi đọc sản phẩm: " + e.getMessage());
            }
        }

        //<-----
    }
```

- Algorithm logic: Products with a shelf life of less than 30 days will be added to the queue with priority.

```
     public static void add(Product product, int a,
SortedLinkedPriorityQueue<Integer,Product> productSortedLinkedPriorityQueue){
             // Kiểm tra số lượng sản phẩm
             if (product.getSoLuong() > a) {
                 // Kiểm tra ngày hết hạn
                 LocalDate today = LocalDate.now();
                 if (product.getHanSuDung().isBefore(today.plusDays(30))) { // Nếu hạn sử
dụng còn dưới 30 ngày
                     productSortedLinkedPriorityQueue.insert(product.getSoLuong(), product);
                     System.out.println("Sản phẩm " + product.getTenSanPham() + " đã được
thêm vào hàng đợi do sắp hết hạn sử dụng.");
                 } else {
                     System.out.println("Sản phẩm " + product.getTenSanPham() + " không sắp
hết hạn sử dụng.");
                 }
             } else {
                 System.out.println("Sản phẩm " + product.getTenSanPham() + " không đủ số
lượng để thêm vào hàng đợi.");
             }
    }
```

- Get the product name from the first node in the discount queue.

```
    public void loadDtatatoSale() {
        // Kiểm tra xem hàng đợi có rỗng không
        if (!Run.queueSale.isEmpty()) {
            // Lấy tên sản phẩm từ nút đầu tiên trong hàng đợi giảm giá
            String s = Run.queueSale.top().getValue().getTenSanPham();
            System.out.println("loadDatatoSale_line 65: " + s);
            jLabel11.setText(s);
        } else {
            // Xử lý khi hàng đợi rỗng
            jLabel11.setText("Không có sản phẩm giảm giá");
        }
    }
```

b) Best-Selling Products

- sortAmountSold(): Sort list by sales quantity by algorithm RadixSort (*Radix Sort is a non-comparative sorting algorithm, commonly used to sort integers. It works by ordering numbers based on each of their digits, from lowest digit to highest digit (or vice versa, depending on the implementation*).

```
    public static void sortAmountSold(List<AmountSold> list, boolean desc) {

        radixSort(list, desc);
```

```
    }
    //thuật toán sắp xếp radixSort <-----
    public static void radixSort(List<AmountSold> list, boolean desc) {
        int max = maxSold(list);

        for (int exp = 1; max >= exp; exp *= 10) { // k = log10(max)
            countingSort(list, exp, desc);
        }
    }
```

- Get a list of sold products. Sort the list by quantity sold in descending order using the sort function. Then, remove any products with a quantity of 0. Finally, retrieve the products in order from the top of the list down.

```
    // Lấy danh sách sản phẩm bán chạy
    public void loadDtatatoBestSeller() {
        // Lấy danh sách sản phẩm đã bán
        List<AmountSold> list = Run.AmountSoldTree.getInOrderList();

        // Sắp xếp danh sách theo số lượng bán ra
        Run.AmountSoldTree.sortAmountSold(list, true); // hàm sortAmountSold trong file
AmountSoldManagerTree, line 370

        // Loại bỏ sản phẩm nào có số lượng bằng 0
        list.removeIf(amountSold -> {
            String maSanPham = amountSold.getMaSanPham();
            return Run.ProductTree.search(maSanPham).isEmpty() ||
                    Run.ProductTree.search(maSanPham).get(0).getSoLuong() == 0;
        });

        // Kiểm tra nếu danh sách không rỗng trước khi đặt giá trị vào các trường văn bản
        if (!list.isEmpty()) {
            if (list.size() > 0) {
                txtTop1.setText(list.get(0).getMaSanPham());
            }
            if (list.size() > 1) {
                txtTop2.setText(list.get(1).getMaSanPham());
            }
            if (list.size() > 2) {
                txtTop3.setText(list.get(2).getMaSanPham());
            }
        } else {
            // Xử lý khi danh sách rỗng (có thể đặt giá trị mặc định hoặc thông báo)
            txtTop1.setText("Không có sản phẩm bán chạy");
            txtTop2.setText("");
            txtTop3.setText("");
        }
```

```
}
```

c) Search Product by Name

- getInOrderList(): This method performs a mid-order binary tree traversal (in order), meaning the tree is traversed in the order: left - root - right, list sorted ascending by key (name).

```
public List<Product> getInOrderList() {
    return getInOrderList(getRoot(), new ArrayList<>());
}

private List<Product> getInOrderList(ProductNode node, List<Product> productList) {
    if (node == null) {
        return productList;
    }

    if (node.getLeft() != null) {
        getInOrderList(node.getLeft(), productList);
    }

    productList.add(node.product);

    if (node.getRight() != null) {
        getInOrderList(node.getRight(), productList);
    }

    return productList;
}
```

- Create an empty list productList to store the products that match the search query. Convert the name to lowercase once (lowerName = name.toLowerCase()) to save time and resources, minimizing the need to call toLowerCase() multiple times in the loop. Use a for loop to iterate through each product in the list:Get the product name using getNameOfProduct(). Check if the product name is null before calling toLowerCase(). If the product name is not null and contains the string lowerName, add the product to the productList.After completing the loop, return the productList, containing the products that match the search query.

```
public List<Product> search(String name) {
    return new ArrayList<>(searchElementName(name));
}
public List<Product> searchElementName(String name) {
```

```
        // Nếu name là null, không thể tìm kiếm, trả về danh sách trống
        if (name == null) {
            return new ArrayList<>();
        }

        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();


        // Chuyển name sang lowercase một lần duy nhất để tránh gọi nhiều lần
        String lowerName = name.toLowerCase();


        for (Product product : list) {
            String tenSP = product.getTenSanPham(); // Lấy tên sản phẩm
            // Kiểm tra null trước khi gọi toLowerCase()
            if (tenSP != null && tenSP.toLowerCase().contains(lowerName)) {
                productList.add(product);
            }
        }


        return productList;
    }
```

d) Filter Products by Price

- sortPriceList(List<Product> list, boolean desc): return sorted list.

```
    public static List<Product> sortPriceList(List<Product> list, boolean desc) {

        sort(list, 0, list.size() - 1, desc);
        return list;
    }
```

- sort(List<Product> list, int low, int high, boolean desc): Performs the merge sort algorithm.

  How it works: Check Stop Condition: If low is less than high, it means there is more than one element in the list to sort. Calculate Mid Index: Calculate the mid index to split the list into two halves. Recursion: Call sort() method again for the left half (low to mid) and right half (mid + 1 to high). Call Merge Method: After the two halves are sorted, call merge() method to merge them into a sorted list.

```
    private static void sort(List<Product> list, int low, int high, boolean desc) {

        if (low < high) {
            int mid = (low + high) / 2;
```

```
            sort(list, low, mid, desc);
            sort(list, mid + 1, high, desc);

            merge(list, low, mid, high, desc);
        }
    }
```

- merge(List<Product> list, int low, int mid, int high, boolean desc): Merge two sorted sublists into one larger list.

```
    private static void merge(List<Product> list, int low, int mid, int high, boolean desc)
{
        int n = high - low + 1;
        Product[] b = new Product[n];
        int left = low, right = mid + 1, bIdx = 0;

        while (left <= mid && right <= high) {
            if (!desc) {
                if (list.get(left).getGiaTien() <= list.get(right).getGiaTien()) {
                    b[bIdx++] = list.get(left++);
                } else {
                    b[bIdx++] = list.get(right++);
                }
            } else {
                if (list.get(left).getGiaTien() >= list.get(right).getGiaTien()) {
                    b[bIdx++] = list.get(left++);
                } else {
                    b[bIdx++] = list.get(right++);
                }
            }
        }

        while (left <= mid) {
            b[bIdx++] = list.get(left++);
        }
        while (right <= high) {
            b[bIdx++] = list.get(right++);
        }

        for (int i = 0; i < n; i++) {
            list.set(low + i, b[i]);
        }
    }
```

e) Search Product by Product Components

- Component Vertex Search

- Split Components: Take a componentsString string, convert it to lowercase, remove spaces, and split it into an array searchComponents.
- Initialize List: Create a list componentVertices to store the vertex indices of the components in the graph and a list result to store the matching products.
- Find Vertices:
  - Iterate through each component in searchComponents.
  - Find the vertex index in the graph corresponding to each component.
  - If found, add it to componentVertices; otherwise, print a message that it was not found.
  - Test Components: If no vertices are found, stop searching and return an empty list.

```
    public static List<Product> searchProductsByComponentsInGraph(String componentsString,
ListGraph graph, List<Product> products) {
        // Tách các thành phần cần tìm
        String[] searchComponents = componentsString.toLowerCase().trim().split(",");
        List<Integer> componentVertices = new ArrayList<>();
        List<Product> result = new ArrayList<>();

        System.out.println("Các thành phần cần tìm: " + Arrays.toString(searchComponents));

        // Tìm đỉnh tương ứng với từng thành phần
        for (String component : searchComponents) {
            component = component.trim();
            boolean found = false;
            for (int i = products.size(); i < graph.getNumV(); i++) { // Các đỉnh thành phần
nằm sau đỉnh sản phẩm
                if (component.equalsIgnoreCase(graph.getVertexName(i))) {
                    componentVertices.add(i);
                    System.out.println("Đã tìm thấy thành phần \"" + component + "\" tại
đỉnh: " + i);
                    found = true;
                    break;
                }
            }
            if (!found) {
                System.out.println("Không tìm thấy thành phần: " + component);
            }
        }

        if (componentVertices.isEmpty()) {
```

```
            System.out.println("Không tìm thấy bất kỳ thành phần nào. Dừng tìm kiếm.");
            return result; // Không có thành phần nào khớp
        }
```

- Iterate through Products:Iterate through each product in the product list. Check if the product contains all the ingredients by checking the edges in the graph.

```
        // Duyệt qua tất cả các sản phẩm để kiểm tra nếu sản phẩm chứa tất cả các thành phần
        System.out.println("Bắt đầu duyệt qua các sản phẩm...");
        for (int i = 0; i < products.size(); i++) {
            boolean matchesAll = true;

            for (int componentVertex : componentVertices) {
                if (!graph.isEdge(i, componentVertex)) { // Nếu không có cạnh giữa sản phẩm
và thành phần
                    System.out.println("Sản phẩm \"" + products.get(i).getTenSanPham() + "\"
không chứa thành phần tại đỉnh: " + componentVertex);
                    matchesAll = false;
                    break;
                }
            }

            if (matchesAll) { // Nếu sản phẩm chứa tất cả thành phần
                System.out.println("Sản phẩm khớp: " + products.get(i).getTenSanPham());
                result.add(products.get(i));
            }
        }

        System.out.println("Số sản phẩm khớp tìm thấy: " + result.size());
        return result;
    }
```

f) Undo

- The algorithm utilizes a stack (functionStack) to store and manage cart operations. Each action performed on the cart (e.g., adding or removing products) is encapsulated in a FunctionWrapper object. The stack enables Last-In-First-Out (LIFO) access to ensure that the most recent action is undone first.

```
    private void btnDetail1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_btnDetail1ActionPerformed
         try {
            FunctionWrapper<ChiTietPhieu> funcWrapper = functionStack.pop();
            funcWrapper.executeFunction(); //gọi lại cái giá trị functionStack.pop() trước
đó
```

```
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Đã hoàn tác hết các bước gần đây");
        }
    }//GEN-LAST:event_btnDetail1ActionPerformed
```

```
public class FunctionWrapper<T> {
    private final Function<T> function;
    private final T parameter;

    public FunctionWrapper(Function<T> function, T parameter) {
        this.function = function;
        this.parameter = parameter;
    }

    public void executeFunction() {
        function.execute(parameter);
    }
    }
```

3. QuanLyKho

a) Search Product by Implement

- The product attribute search methods, including searchMaSanPham, searchSoLuong, searchDonGia, searchThanhPhan, searchKhoiLuong, searchHSD, searchNSX, and searchNNK, all implement the Linear Search algorithm. These methods first retrieve the list of products (list = getInOrderList()) and create an empty list (productList = new ArrayList<>()) to store products that match the search criteria. The methods then iterate through each product in the list, checking its attributes to see if they contain the search string. For date attributes (such as HanSuDung, NgaySanXuat, NgayNhapKho), the date is formatted as a string for comparison with the search term. If a product's attribute meets the search condition, the product is added to the result list. Ultimately, the method returns the result list containing the matching products.

```
    public List<Product> searchMaSanPham(String name) {
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        for (Product product : list) {
            if (product.getMaSanPham().toLowerCase().contains(name.toLowerCase())) {
                productList.add(product);
            }
        }

        return productList;
```

```
    }

    public List<Product> searchSoLuong(String name) {
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        for (Product product : list) {
            if
(Integer.toString(product.getSoLuong()).toLowerCase().contains(name.toLowerCase())) {
                productList.add(product);
            }
        }

        return productList;
    }

    public List<Product> searchDonGia(String price) {
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        for (Product product : list) {
            if (Integer.toString((int) product.getGiaTien()).contains(price)) {
                productList.add(product);
            }
        }

        return productList;
    }
    public List<Product> searchHanSuDung(String name) {
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        // Giả định rằng 'name' là một ngày theo định dạng "dd/MM/yyyy"
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

        for (Product product : list) {
            // Chuyển đổi ngày hết hạn sang định dạng chuỗi để so sánh
            if
(product.getHanSuDung().format(formatter).toLowerCase().contains(name.toLowerCase())) {
                productList.add(product);
            }
        }

        return productList;
    }
```

```
    public List<Product> searchNgaySanXuat(String name) {
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        // Giả định rằng 'name' là một ngày theo định dạng "dd/MM/yyyy"
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

        for (Product product : list) {
            // Chuyển đổi ngày sản xuất sang định dạng chuỗi để so sánh
            if
(product.getNgaySanXuat().format(formatter).toLowerCase().contains(name.toLowerCase())) {
                productList.add(product);
            }
        }

        return productList;
    }


    public List<Product> searchThanhPhan(String name) {
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        for (Product product : list) {
            if (product.getThanhPhan().toLowerCase().contains(name.toLowerCase())) {
                productList.add(product);
            }
        }

        return productList;
    }

    public List<Product> searchKhoiLuong(String name) {
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        for (Product product : list) {
            if (product.getKhoiLuong().toLowerCase().contains(name.toLowerCase())) {
                productList.add(product);
            }
        }

        return productList;
    }

    public List<Product> searchNNK(String name) {
```

```
        List<Product> list = getInOrderList();
        List<Product> productList = new ArrayList<>();

        // Giả định rằng 'name' là một ngày theo định dạng "dd/MM/yyyy"
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

        for (Product product : list) {
            // Chuyển đổi ngày nhập kho sang định dạng chuỗi để so sánh
            if
(product.getNgayNhapKho().format(formatter).toLowerCase().contains(name.toLowerCase())) {
                productList.add(product);
            }
        }

        return productList;
    }
```

b) Inventory statistics <Thống kê hàng tồn kho: Kiểm tra số lượng &HSD>

- The checkStockAndExpiry method uses in-order traversal on a binary search tree (BST) to retrieve a list of products and coupons, then checks the expiration date and inventory. The method calculates the minimum inventory level based on daily consumption and lead time days (minStockLevel = dailyConsumption * leadTimeDays), and issues a warning if inventory falls below the minimum or the expiration date is below 30%. This helps businesses manage inventory effectively, replenish products on time, and avoid waste.

```
    // Kiểm tra số lượng và hạn sử dụng
    public void checkStockAndExpiry() {
        List<Product> products = Run.ProductTree.getInOrderList();
        List<Phieu> phieulist = Run.PhieuMuaTree.getInOrderList();

        tblCombinedWarning.setRowCount(0);
        tblPredictedStock.setRowCount(0);
        LocalDate currentDate = LocalDate.now();

        for (Product product : products) {
            int stockQuantity = product.getSoLuong();
            LocalDate expiryDate = product.getHanSuDung();

            if (expiryDate == null) {
                tblCombinedWarning.addRow(new Object[]{
                        product.getTenSanPham(),
                        "Hạn sử dụng không hợp lệ."
```

```
                });
                continue;
            }

            double dailyConsumption = product.getDailyConsumption(phieulist);
            int leadTimeDays = product.getSoNgayGiaoHang();

            long daysUntilExpiry = ChronoUnit.DAYS.between(currentDate, expiryDate);
            long totalShelfLife = ChronoUnit.DAYS.between(product.getNgaySanXuat(),
expiryDate);

            if (totalShelfLife <= 0) {
                tblCombinedWarning.addRow(new Object[]{
                        product.getTenSanPham(),
                        "Thông tin ngày sản xuất hoặc hạn sử dụng không hợp lệ."
                });
                continue;
            }

            double minStockLevel = dailyConsumption * leadTimeDays;
            if (stockQuantity < minStockLevel) {
                tblCombinedWarning.addRow(new Object[]{
                        product.getTenSanPham(),
                        "Số lượng hàng tồn kho còn lại: " + stockQuantity + " (Mức tối
thiểu: " + minStockLevel + ")"
                });
            }

            // Dự đoán số lượng cần nhập kho
            int predictedStock = predictStockNeed(product, dailyConsumption, leadTimeDays);
            if (predictedStock > 0) {
                tblPredictedStock.addRow(new Object[]{
                        product.getTenSanPham(),
                        predictedStock
                });
            }

            if (daysUntilExpiry <= (0.3 * totalShelfLife)) {
                tblCombinedWarning.addRow(new Object[]{
                        product.getTenSanPham(),
                        "Hạn sử dụng sắp hết: " + daysUntilExpiry + " ngày (30% hạn sử
dụng)"
                });
            }

            writeToFile(product);
        }
```

```
    }

    // Dự đoán số lượng cần nhập kho bằng cách sử dụng phương pháp đơn giản
    private int predictStockNeed(Product product, double dailyConsumption, int leadTimeDays)
{
        // Tính toán số lượng cần nhập dựa trên tiêu thụ hàng ngày và thời gian giao hàng
        int requiredStock = (int) Math.ceil(dailyConsumption * leadTimeDays);
        return requiredStock; // Trả về giá trị dự đoán
    }
```

CHAPTER 5: CONCLUSION AND FUTURE WORKS

1. Conclusion

Through the Warehouse Management project, we not only learned how to apply algorithms in data structures but also effectively addressed real-world problems related to customer sales management and warehouse management. The system we built has optimized the processes of tracking and controlling goods, from importing and exporting inventory to managing stock levels and tracking customer transaction histories. This has helped businesses save time, enhance operational efficiency, and improve user experience. The project also marks a significant step forward in leveraging technology to optimize management processes, paving the way for potential future management solutions

2. Acknowledgment

We would like to convey our deepest appreciation to our instructor and individuals who assisted us in reaching the goals of this project:

- Dr. Tran Thanh Tung and MSc. Thai Trung Tin
- Original code from https://github.com/kouhoang/store_manager_dsa.git