**Module 2 Algorithms Data Structures**

**Exercise 1: Inventory Management System**

import java.util.HashMap;

import java.util.Scanner;

class Product {

int productId;

String productName;

int quantity;

double price;

public Product(int productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public void display() {

System.out.println("ID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: ₹" + price);

}

}

// Inventory Management System

public class InventorySystem {

static HashMap<Integer, Product> inventory = new HashMap<>();

static Scanner scanner = new Scanner(System.in);

public static void main(String[] args) {

int choice;

do {

System.out.println("\n---- Inventory Menu ----");

System.out.println("1. Add Product");

System.out.println("2. View Product");

System.out.println("3. Update Quantity");

System.out.println("4. Remove Product");

System.out.println("5. View All");

System.out.println("0. Exit");

System.out.print("Enter choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1 -> addProduct();

case 2 -> viewProduct();

case 3 -> updateQuantity();

case 4 -> removeProduct();

case 5 -> viewAllProducts();

}

} while (choice != 0);

scanner.close();

}

static void addProduct() {

System.out.print("Enter Product ID: ");

int id = scanner.nextInt();

scanner.nextLine(); // consume newline

System.out.print("Enter Name: ");

String name = scanner.nextLine();

System.out.print("Enter Quantity: ");

int qty = scanner.nextInt();

System.out.print("Enter Price: ");

double price = scanner.nextDouble();

Product p = new Product(id, name, qty, price);

inventory.put(id, p);

System.out.println("Product added.");

}

static void viewProduct() {

System.out.print("Enter Product ID to view: ");

int id = scanner.nextInt();

Product p = inventory.get(id);

if (p != null)

p.display();

else

System.out.println("Product not found.");

}

static void updateQuantity() {

System.out.print("Enter Product ID: ");

int id = scanner.nextInt();

Product p = inventory.get(id);

if (p != null) {

System.out.print("Enter new quantity: ");

p.quantity = scanner.nextInt();

System.out.println("Quantity updated.");

} else {

System.out.println("Product not found.");

}

}

static void removeProduct() {

System.out.print("Enter Product ID to remove: ");

int id = scanner.nextInt();

if (inventory.remove(id) != null)

System.out.println("Product removed.");

else

System.out.println("Product not found.");

}

static void viewAllProducts() {

if (inventory.isEmpty()) {

System.out.println("No products in inventory.");

} else {

for (Product p : inventory.values()) {

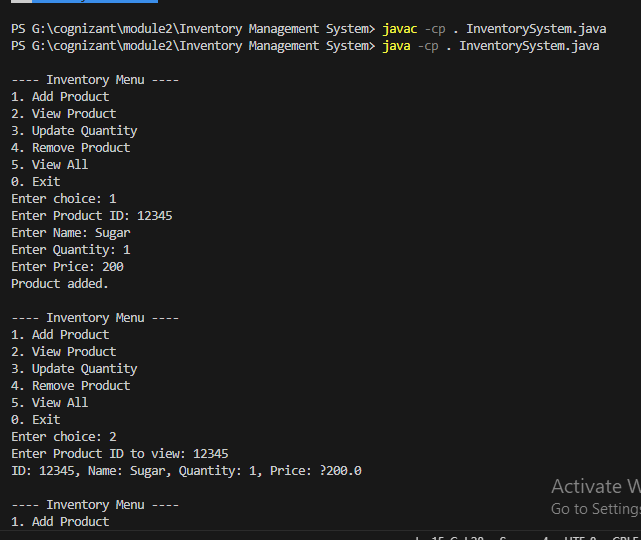
p.display();

}

}

}

}



**Exercise 2: E-commerce Platform Search Function**

import java.util.Arrays;

import java.util.Comparator;

class Product {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

@Override

public String toString() {

return "Product(" + productId + ", '" + productName + "', '" + category + "')";

}

}

public class EcommerceSearch {

public static Product linearSearch(Product[] products, String targetName) {

for (Product product : products) {

if (product.productName.equalsIgnoreCase(targetName)) {

return product;

}

}

return null;

}

public static Product binarySearch(Product[] products, String targetName) {

int left = 0;

int right = products.length - 1;

targetName = targetName.toLowerCase();

while (left <= right) {

int mid = (left + right) / 2;

String midName = products[mid].productName.toLowerCase();

if (midName.equals(targetName)) {

return products[mid];

} else if (midName.compareTo(targetName) < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Shampoo", "Beauty"),

new Product(103, "Notebook", "Stationery"),

new Product(104, "Keyboard", "Electronics"),

new Product(105, "Tablet", "Electronics")

};

String searchTerm = "Notebook";

System.out.println("\nLinear Search:");

Product resultLinear = linearSearch(products, searchTerm);

System.out.println("Result: " + (resultLinear != null ? resultLinear : "Product not found."));

Product[] sortedProducts = Arrays.copyOf(products, products.length);

Arrays.sort(sortedProducts, Comparator.comparing(p -> p.productName.toLowerCase()));

System.out.println("\nBinary Search (on sorted product names):");

Product resultBinary = binarySearch(sortedProducts, searchTerm);

System.out.println("Result: " + (resultBinary != null ? resultBinary : "Product not found."));

System.out.println("\n--- Time Complexity Analysis ---");

System.out.println("Linear Search: O(n) - Scans every product until match is found or end is reached.");

System.out.println("Binary Search: O(log n) - Requires sorted array, fast lookup by dividing search space.");

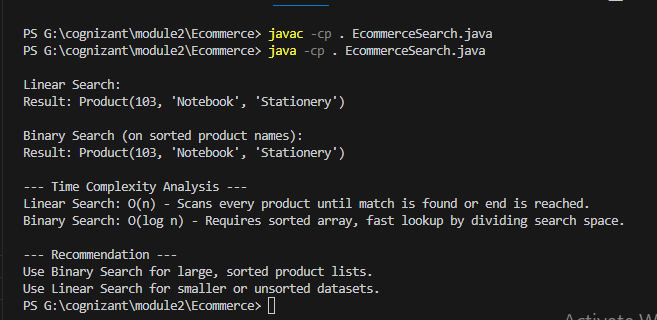
System.out.println("\n--- Recommendation ---");

System.out.println("Use Binary Search for large, sorted product lists.");

System.out.println("Use Linear Search for smaller or unsorted datasets.");

}

}



**Exercise 6: Library Management System**

import java.util.\*;

class Book implements Comparable<Book> {

int bookId;

String title;

String author;

Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title.toLowerCase();

this.author = author.toLowerCase();

}

@Override

public int compareTo(Book b) {

return this.title.compareTo(b.title);

}

@Override

public String toString() {

return "ID: " + bookId + ", Title: " + title + ", Author: " + author;

}

}

public class LibraryManagementSystem {

static Book linearSearch(List<Book> books, String key) {

key = key.toLowerCase();

for (Book book : books) {

if (book.title.equals(key)) {

return book;

}

}

return null;

}

static Book binarySearch(List<Book> books, String key) {

key = key.toLowerCase();

int left = 0, right = books.size() - 1;

while (left <= right) {

int mid = (left + right) / 2;

int cmp = books.get(mid).title.compareTo(key);

if (cmp == 0)

return books.get(mid);

else if (cmp < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

List<Book> books = new ArrayList<>();

books.add(new Book(1, "The Alchemist", "Paulo Coelho"));

books.add(new Book(2, "To Kill a Mockingbird", "Harper Lee"));

books.add(new Book(3, "1984", "George Orwell"));

books.add(new Book(4, "Pride and Prejudice", "Jane Austen"));

books.add(new Book(5, "The Great Gatsby", "F. Scott Fitzgerald"));

System.out.print("Enter title to search: ");

String searchTitle = sc.nextLine();

System.out.println("\n--- Linear Search ---");

Book result1 = linearSearch(books, searchTitle);

if (result1 != null)

System.out.println("Book Found: " + result1);

else

System.out.println("Book not found");

System.out.println("\n--- Binary Search ---");

Collections.sort(books); // Sort list by title

Book result2 = binarySearch(books, searchTitle);

if (result2 != null)

System.out.println("Book Found: " + result2);

else

System.out.println("Book not found");

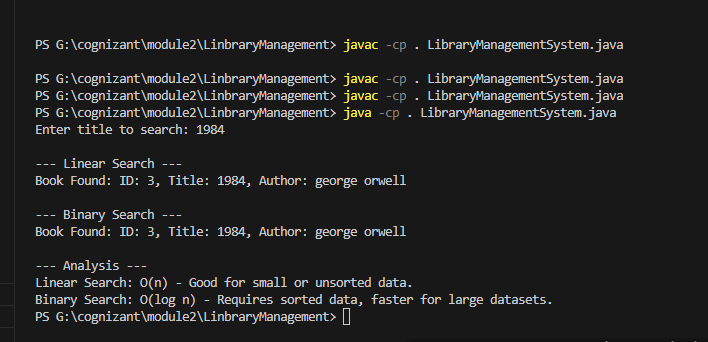
System.out.println("\n--- Analysis ---");

System.out.println("Linear Search: O(n) - Good for small or unsorted data.");

System.out.println("Binary Search: O(log n) - Requires sorted data, faster for large datasets.");

}

}



**Exercise 3: Sorting Customer Orders**

import java.util.\*;

class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

@Override

public String toString() {

return orderId + " - " + customerName + " - $" + totalPrice;

}

}

public class SortOrders {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (orders[j].totalPrice > orders[j + 1].totalPrice) {

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

public static void quickSort(Order[] orders, int low, int high) {

if (low < high) {

int pi = partition(orders, low, high);

quickSort(orders, low, pi - 1);

quickSort(orders, pi + 1, high);

}

}

public static int partition(Order[] orders, int low, int high) {

double pivot = orders[high].totalPrice;

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders[j].totalPrice < pivot) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp;

return i + 1;

}

public static void display(String message, Order[] orders) {

System.out.println("\n" + message);

for (Order order : orders) {

System.out.println(order);

}

}

public static void main(String[] args) {

Order[] originalOrders = {

new Order(101, "Alice", 250.0),

new Order(102, "Bob", 150.5),

new Order(103, "Charlie", 300.25),

new Order(104, "Diana", 120.0),

new Order(105, "Eve", 400.0)

};

Order[] bubbleSortedOrders = Arrays.copyOf(originalOrders, originalOrders.length);

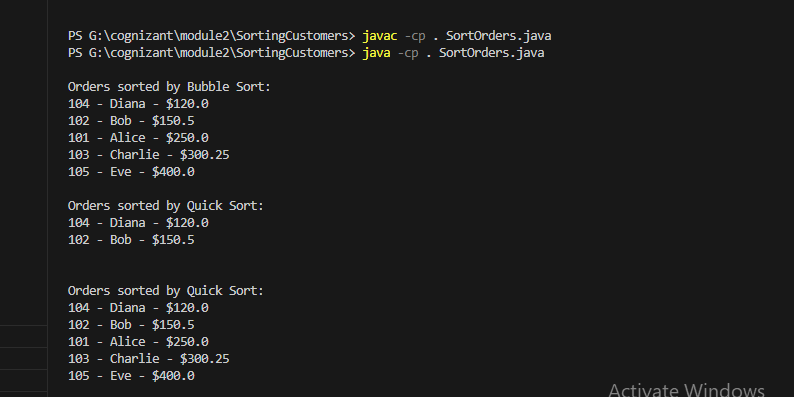
bubbleSort(bubbleSortedOrders);

display("Orders sorted by Bubble Sort:", bubbleSortedOrders);

Order[] quickSortedOrders = Arrays.copyOf(originalOrders, originalOrders.length);

quickSort(quickSortedOrders, 0, quickSortedOrders.length - 1);

display("Orders sorted by Quick Sort:", quickSortedOrders);



**Exercise 5: Task Management System**

import java.util.Scanner;

class Task {

int taskId;

String taskName;

String status;

Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public String toString() {

return "Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status;

}

}

class Node {

Task task;

Node next;

Node(Task task) {

this.task = task;

this.next = null;

}

}

class TaskLinkedList {

private Node head;

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null)

temp = temp.next;

temp.next = newNode;

}

}

public void traverseTasks() {

Node temp = head;

while (temp != null) {

System.out.println(temp.task);

temp = temp.next;

}

}

public Task searchTask(int taskId) {

Node temp = head;

while (temp != null) {

if (temp.task.taskId == taskId)

return temp.task;

temp = temp.next;

}

return null;

}

public boolean deleteTask(int taskId) {

if (head == null)

return false;

if (head.task.taskId == taskId) {

head = head.next;

return true;

}

Node prev = head;

Node curr = head.next;

while (curr != null) {

if (curr.task.taskId == taskId) {

prev.next = curr.next;

return true;

}

prev = curr;

curr = curr.next;

}

return false;

}

}

public class TaskManagementSystem {

public static void main(String[] args) {

TaskLinkedList taskList = new TaskLinkedList();

Scanner scanner = new Scanner(System.in);

taskList.addTask(new Task(1, "Design UI", "Pending"));

taskList.addTask(new Task(2, "Implement Backend", "In Progress"));

taskList.addTask(new Task(3, "Write Tests", "Pending"));

System.out.println("All Tasks:");

taskList.traverseTasks();

System.out.print("\nEnter task ID to search: ");

int searchId = scanner.nextInt();

Task found = taskList.searchTask(searchId);

System.out.println(found != null ? "Found: " + found : "Task not found");

System.out.print("\nEnter task ID to delete: ");

int deleteId = scanner.nextInt();

boolean deleted = taskList.deleteTask(deleteId);

System.out.println(deleted ? "Task deleted." : "Task not found.");

System.out.println("\nTasks after deletion:");

taskList.traverseTasks();

scanner.close();

}

}

