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

1. **Understanding Asymptotic Notation -**

When we analyse how quickly an algorithm works, we use a special notation called Big O that acts like a speedometer for code. Imagine you're comparing delivery trucks - Big O tells you how much longer a truck will take as it gets more packages to deliver, without worrying about exact minutes or seconds.

**What Big O Notation Tells Us -**

* **Growth Patterns:** It reveals how an algorithm slows down as it handles more data. Like knowing whether adding more stops to a delivery route will slightly increase your time (good) or make it take dramatically longer (bad).
* **Worst-Case Planning:** It prepares us for the toughest scenarios, like searching for an item that might not even be in our inventory.
* **Comparison Tool:** It lets us objectively compare different approaches to solving the same problem.

**Common Efficiency Classes -**

* **Instant Access (O(1)):** Like knowing exactly where your most popular product is stored - you can grab it immediately no matter how big your warehouse gets
* **Divide and Conquer (O(log n)):** Similar to how you'd find a word in a dictionary - with each comparison, you eliminate half of the remaining possibilities.
* **One-by-One Search (O(n)):** Like checking every shelf in order until you find what you need - takes longer in bigger stores.
* **Exponential Slowdown (O(n²)):** Like comparing every product with every other product - becomes impractical quickly as inventory grows.

**Real-World Search Scenarios -**

**Best-Case Performance (Dream Scenario) -**

* **Linear Search:** You get lucky and find the product on your first try, like spotting the exact shirt you want right at the store entrance.
* **Binary Search**: The item happens to be exactly in the middle of your sorted inventory - one quick check and you're done.

**Average Performance (Typical Day) -**

* **Linear Search**: On average, you'll need to look through about half your inventory before finding what you need. More products means proportionally more time.
* **Binary Search:** Each step dramatically narrows down the possibilities, meaning even with thousands of products, you'll only need to check a handful.

**Worst-Case Performance (Tough Day at Work) -**

* **Linear Search:** The item isn't in stock or is the very last one you check - you've wasted time examining everything.
* **Binary Search:** Even in the worst case, the number of checks grows very slowly compared to the inventory size - what takes 10 checks for 1,000 items might only take 15 checks for 10,000 items.

**2,3 – Setup and Implementation -**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.Scanner;

class Product {

private int productId;

private String productName;

private String category;

private double price;

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

this.productId = productId;

this.productName = productName;

this.category = category;

this.price = price;

}

public int getProductId() {

return productId;

}

@Override

public String toString() {

return String.format("Product(ID: %d, Name: '%s', Category: '%s', Price: $%.2f)",

productId, productName, category, price);

}

}

public class ECommerceSearch {

// Linear search implementation

public static Product linearSearch(List<Product> products, int targetId) {

for (Product product : products) {

if (product.getProductId() == targetId) {

return product;

}

}

return null;

}

// Binary search implementation

public static Product binarySearch(List<Product> sortedProducts, int targetId) {

int low = 0;

int high = sortedProducts.size() - 1;

while (low <= high) {

int mid = (low + high) / 2;

int currentId = sortedProducts.get(mid).getProductId();

if (currentId == targetId) {

return sortedProducts.get(mid);

} else if (currentId < targetId) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

// Initialize product list

List<Product> products = new ArrayList<>();

products.add(new Product(101, "Laptop", "Electronics", 999.99));

products.add(new Product(203, "Smartphone", "Electronics", 699.99));

products.add(new Product(305, "Headphones", "Electronics", 149.99));

products.add(new Product(412, "Desk Chair", "Furniture", 199.99));

products.add(new Product(523, "Coffee Maker", "Appliances", 79.99));

// Create sorted copy for binary search

List<Product> sortedProducts = new ArrayList<>(products);

Collections.sort(sortedProducts, Comparator.comparingInt(Product::getProductId));

Scanner scanner = new Scanner(System.in);

System.out.println("E-commerce Product Search System");

System.out.println("--------------------------------");

while (true) {

System.out.println("\nSearch options:");

System.out.println("1. Linear Search");

System.out.println("2. Binary Search");

System.out.println("3. Exit");

System.out.print("Enter your choice (1-3): ");

String choice = scanner.nextLine();

if (choice.equals("3")) {

System.out.println("Exiting...");

break;

}

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

try {

int targetId = Integer.parseInt(scanner.nextLine());

Product result = null;

if (choice.equals("1")) {

System.out.println("\n Performing LINEAR SEARCH...");

result = linearSearch(products, targetId);

} else if (choice.equals("2")) {

System.out.println("\n Performing BINARY SEARCH...");

result = binarySearch(sortedProducts, targetId);

} else {

System.out.println("Invalid choice. Please try again.");

continue;

}

if (result != null) {

System.out.println(" Product found: " + result);

} else {

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

}

} catch (NumberFormatException e) {

System.out.println("Please enter a valid number for product ID");

}

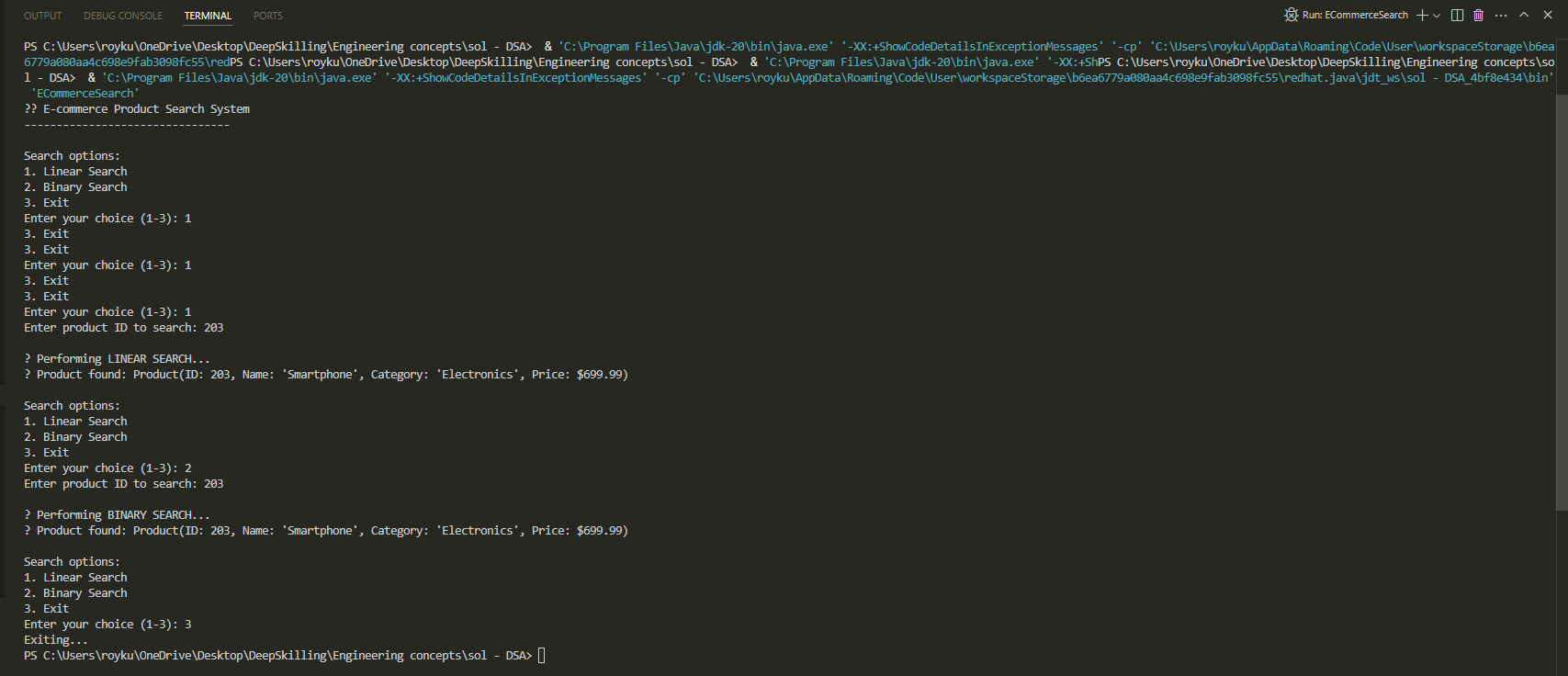
}

scanner.close();

}

}

**Output-**



**Try these test cases:**

* Search for existing product IDs (101, 203, 305, 412, 523)
* Search for non-existent IDs

**4. Analysing Search Algorithms for E-Commerce –**

For e-commerce platforms, search efficiency impacts user experience. Binary search (O(log n)) excels when:

* Products are sorted by ID
* The catalog is large
* Searches are frequent

Linear search (O(n)) works better when:

* Searching by unsortable fields (name, description)
* The inventory changes often
* The product count is small

5. Recommendations:

* Use binary search for ID lookups on sorted data
* Apply linear search for flexible queries
* For massive catalogs, consider hash tables or search engines like Elasticsearch

Binary search is faster but requires sorted data. Linear search is simpler but slower for large datasets. Choose based on your catalog size, search frequency, and performance needs. For most platforms, a hybrid approach works best—optimizing common searches while keeping others flexible.