# 

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

## Scenario:

## You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

## 

**Big O notation** helps us understand how algorithms perform as data size increases. It focuses on worst-case scenarios for search functions:

* **Best case**: When we find the item right away
* **Average case**: Typical performance with random data
* **Worst case**: When item is missing or found last

## **Product Class Implementation**

public class Product {

private int productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

}

**Search Algorithm Implementation**

public class SearchManager {

// Main search method with algorithm selection

// 0 = Linear Search, 1 = Binary Search

public static Product searchProduct(Product[] products, int targetId, int searchType) {

if (searchType == 0) {

return linearSearchMethod(products, targetId);

} else {

return binarySearchMethod(products, targetId);

}

}

// Linear Search Implementation

private static Product linearSearchMethod(Product[] products, int targetId) {

for (int i = 0; i < products.length; i++) {

if (products[i].getProductId() == targetId) {

return products[i];

}

}

return null;

}

// Binary Search Implementation

private static Product binarySearchMethod(Product[] products, int targetId) {

int start = 0;

int end = products.length - 1;

while (start <= end) {

int middle = (start + end) / 2;

if (products[middle].getProductId() == targetId) {

return products[middle];

} else if (products[middle].getProductId() > targetId) {

end = middle - 1;

} else {

start = middle + 1;

}

}

return null;

}

}

## **Usage Example**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

// Sample product data

Product[] productList = {

new Product(101, "Wireless Mouse", "Electronics"),

new Product(102, "Gaming Keyboard", "Electronics"),

new Product(103, "LED Monitor", "Electronics"),

new Product(104, "Office Chair", "Furniture")

};

// Using Linear Search (searchType = 0)

Product result1 = SearchManager.searchProduct(productList, 102, 0);

System.out.println("Linear Search Result: " +

(result1 != null ? result1.getProductName() : "Not found"));

// Sort array for binary search

Arrays.sort(productList, Comparator.comparing(Product::getProductId));

// Using Binary Search (searchType = 1)

Product result2 = SearchManager.searchProduct(productList, 103, 1);

System.out.println("Binary Search Result: " +

(result2 != null ? result2.getProductName() : "Not found"));

}

## **Performance Comparison**

| **Algorithm Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |

## **When to Use Each Algorithm**

**Binary search works better for e-commerce** because:

1. **Better Performance**: With 1 million products:
   * Linear search: up to 1 million checks needed
   * Binary search: only about 20 checks needed
2. **Faster Response**: Reduces search time from seconds to milliseconds
3. **Practical Implementation**: Product IDs can be easily sorted

Output :

