

Exercise 3: 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.

Steps:


- 1. **Understand Asymptotic Notation:**
 - Explain Big O notation and how it helps in analyzing algorithms.
 - Describe the best, average, and worst-case scenarios for search operations.
- 2. **Setup:**
 - Create a class **Product** with attributes for searching, such as **productId**, **productName**, and **category**.
- 3. **Implementation:**
 - Implement linear search and binary search algorithms.
 - Store products in an array for linear search and a sorted array for binary search.
- 4. **Analysis:**
 - Compare the time complexity of linear and binary search algorithms.
 - Discuss which algorithm is more suitable for your platform and why.

What is Big O Notation?

Big O notation describes the **performance or complexity** of an algorithm in terms of **input size n**.

Case	Meaning
Best	Fastest scenario (e.g., item is first in list)
Average	Typical scenario (e.g., item is somewhere in the middle)
Worst	Slowest scenario (e.g., item is last or not found at all)

Linear Search vs Binary Search:

Algorithm	Best Case	Average Case	Worst Case	Requirements
Linear Search	$O(1)$	$O(n/2) \rightarrow O(n)$	$O(n)$	Unsorted data
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$	 Requires sorted data

Product.java:

```
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=" + productId +  
            ", productName=\"" + productName + "\" +  
            ", category=\"" + category + "\" +  
            '}';  
    }  
}
```

SearchDemo.java

```
public class SearchDemo {  
  
    public static Product linearSearch(Product[] products, int targetId) {  
        for (Product product : products) {  
            if (product.productId == targetId) {  
                return product;  
            }  
        }  
        return null;  
    }  
  
    public static Product binarySearch(Product[] products, int targetId) {  
        int left = 0;  
        int right = products.length - 1;  
  
        while (left <= right) {  
            int mid = left + (right - left) / 2;  
            if (products[mid].productId == targetId) {
```

```
        return products[mid];
    } else if (products[mid].productId < targetId) {
        left = mid + 1;
    } else {
        right = mid - 1;
    }
}
return null;
}
```

```
public static void main(String[] args) {
    Product[] products = {
        new Product(101, "Mobile Phone", "Electronics"),
        new Product(202, "T-shirt", "Apparel"),
        new Product(303, "Laptop", "Electronics"),
        new Product(404, "Headphones", "Accessories"),
        new Product(505, "Shoes", "Apparel")
    };

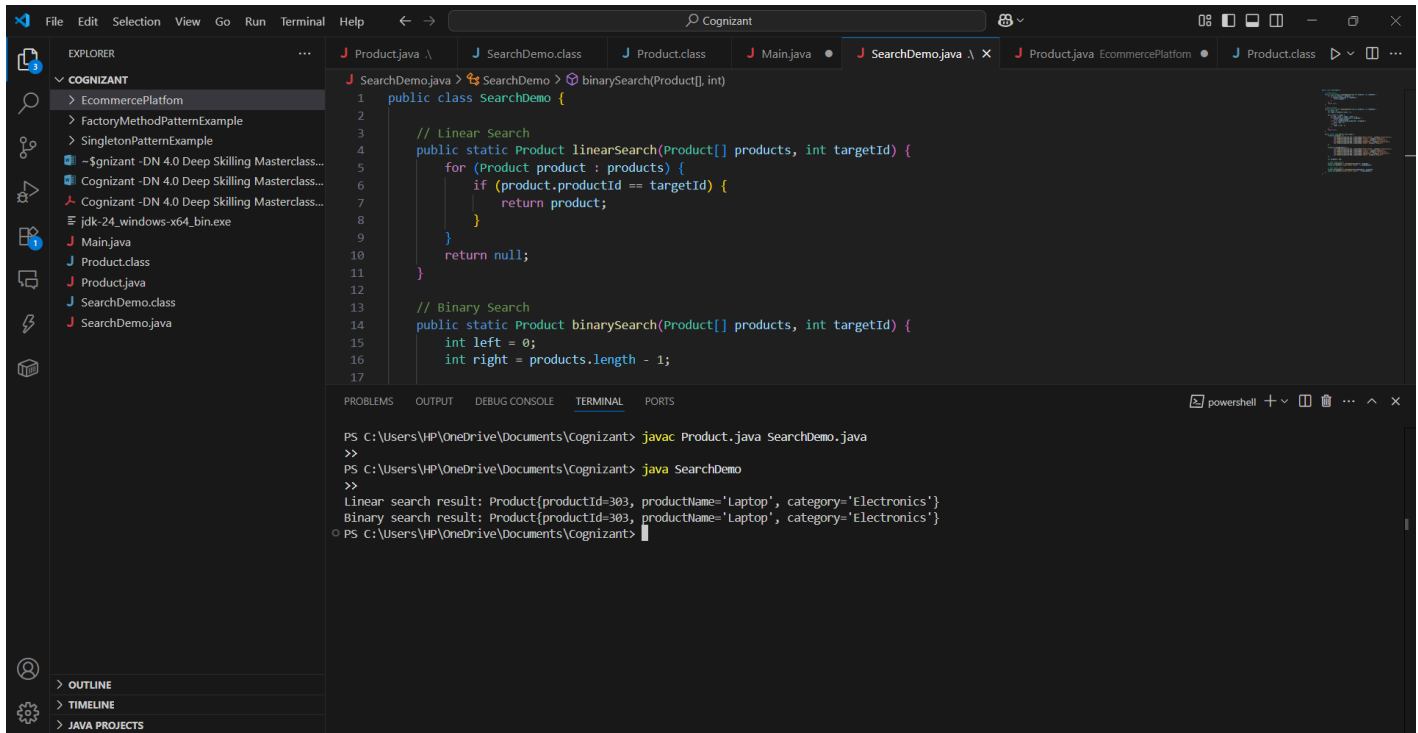
    Product[] sortedProducts = {
        new Product(101, "Mobile Phone", "Electronics"),
        new Product(202, "T-shirt", "Apparel"),
        new Product(303, "Laptop", "Electronics"),
        new Product(404, "Headphones", "Accessories"),
        new Product(505, "Shoes", "Apparel")
    };

    int targetId = 303;

    Product linearResult = linearSearch(products, targetId);
    System.out.println("Linear search result: " + linearResult);

    Product binaryResult = binarySearch(sortedProducts, targetId);
    System.out.println("Binary search result: " + binaryResult);
}
}
```

OUTPUT:



Analysis & Recommendation

Metric	Linear Search	Binary Search
Time Complexity	$O(n)$	$O(\log n)$
Space Complexity	$O(1)$	$O(1)$
Data Requirement	Can work on unsorted	Requires sorted data
Implementation	Simple	Slightly complex

Which Algorithm is better and Why?

For **E-commerce platforms** (large product catalog):

- Use **Binary Search** on **sorted lists**
- For huge data: use **HashMap** or **search indexing** (e.g., ElasticSearch)

Binary Search is preferred for sorted lists of limited size. For production, you'll use more advanced indexing.