**E-commerce Platform Search Function Solution**

**Understanding Asymptotic Notation**

Big O Notation:

Big O notation describes how an algorithm's runtime or space requirements grow as input size grows. It helps us:

* Compare algorithm efficiency
* Predict performance at scale
* Identify bottlenecks in our code

Search Operation Scenarios:

* Best case: Item is found immediately (first element checked)
* Average case: Item is found after checking half the elements (for linear search)
* Worst case: Item isn't present or is the last element checked

**CODE:**

import java.util.Arrays;

import java.util.Comparator;

class Product {

String productId;

String productName;

String category;

double price;

public Product(String id, String name, String cat, double price) {

this.productId = id;

this.productName = name;

this.category = cat;

this.price = price;

}

}

public class ECommerceSearch {

// Linear Search Implementation

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

for (Product product : products) {

if (product.productId.equals(targetId)) {

return product;

}

}

return null;

}

// Binary Search Implementation

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int comparison = products[mid].productId.compareTo(targetId);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product("P100", "Laptop", "Electronics", 999.99),

new Product("P101", "Smartphone", "Electronics", 699.99),

new Product("P102", "Headphones", "Accessories", 99.99)

};

// Test multiple searches

testSearch(products, "P100"); // Should find Laptop

testSearch(products, "P101"); // Should find Smartphone

testSearch(products, "P102"); // Should find Headphones

testSearch(products, "P999"); // Should not find anything

}

public static void testSearch(Product[] products, String targetId) {

System.out.println("\nSearching for: " + targetId);

// Linear Search

Product linearResult = linearSearch(products, targetId);

System.out.print("Linear Search: ");

System.out.println(linearResult != null ? linearResult.productName : "Not Found");

// Binary Search (requires sorted array)

Arrays.sort(products, Comparator.comparing(p -> p.productId));

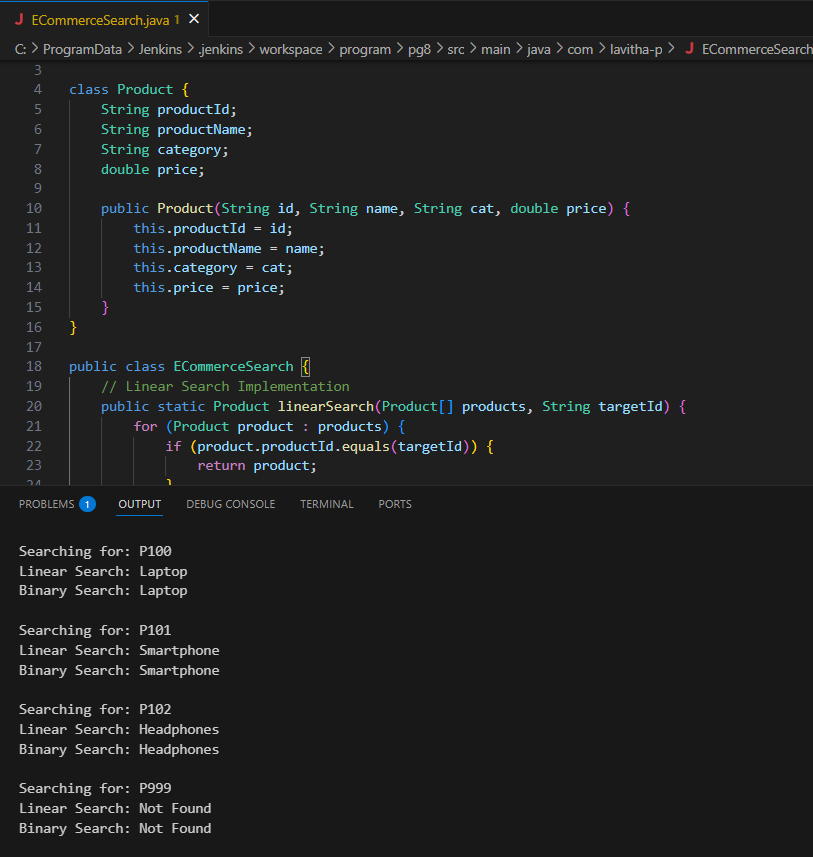
Product binaryResult = binarySearch(products, targetId);

System.out.print("Binary Search: ");

System.out.println(binaryResult != null ? binaryResult.productName : "Not Found");

}

**OUTPUT:**



**Analysis:**

Time Complexity Comparison

| Algorithm | Best Case | Average Case | Worst Case

| Linear Search | O(1) | O(n) | O(n)

| Binary Search | O(1) | O(log n) | O(log n)

Which Algorithm to Choose?

Binary search is better when:

* + Products can be kept sorted (by ID)
  + You have many products (>1000 items)
  + Searches are frequent but updates are rare

Linear search is better when:

* + Products can't be kept sorted
  + You need to search by different fields (name, category)
  + The inventory changes frequently

For most e-commerce platforms, a hybrid approach works best:

1. Use binary search for primary searches by product ID

2. Use linear search (or better, a hash map) for other searches

3. Consider more advanced structures like tries for name searches

Additional Optimization

For real-world e-commerce, consider:

* + Using database indexing for persistent storage
  + Implementing caching for frequent searches
  + Using more advanced data structures like hash tables or search trees