E-commerce Platform Search Function in Java:

1. Understanding Asymptotic Notation (Same concepts apply in Java)

### Time Complexity Comparison:

**Linear Search**:

Best case: O(1) (first element)

Average case: O(n)

Worst case: O(n) (last element or not present)

**Binary Search**:

Best case: O(1) (middle element)

Average case: O(log n)

Worst case: O(log n)

2. Setup: Product Class

public 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;

}

public String getProductName(){

return productName;

}

public String getCategory(){

return category;

}

public double getPrice(){

return price;

}

@Override

public String toString(){

return String.format("Product(ID: %d, Name: %s, Category: %s,

Price: $%.2f)" productId, productName, category, price);

}

}

3. Implementation

Linear Search Implementation:

public class SearchAlgorithms {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

}

Binary Search Implementation:

public static Product binarySearch(Product[] sortedProducts, int targetId) {

int low = 0;

int high = sortedProducts.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

int currentId = sortedProducts[mid].getProductId();

if (currentId == targetId) {

return sortedProducts[mid];

} else if (currentId < targetId) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

Example:

import java.util.util\*;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Wireless Mouse", "Electronics", 26),

new Product(203, "Coffee Maker", "Home Appliances", 50.99),

new Product(305, "Running Shoes", "Sports", 79.99),

new Product(412, "Bluetooth Speaker", "Electronics", 59.99),

new Product(523, "Desk Lamp", "Home Appliances", 374.99)

};

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

// Search for product with ID 305 (exists)

Product result1 = SearchAlgorithms.linearSearch(products, 305);

System.out.println("Search for ID 305: " + result1);

// Search for product with ID 999 (doesn't exist)

Product result2 = SearchAlgorithms.linearSearch(products, 999);

System.out.println("Search for ID 999: " + result2);

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

// Sort the array by productId for binary search

Arrays.sort(products, Comparator.comparingInt(Product::getProductId));

// Search for product with ID 412 (exists)

Product result3 = SearchAlgorithms.binarySearch(products, 412);

System.out.println("Search for ID 412: " + result3);

}

}

4. Analysis

Which algorithm is more suitable?

For an e-commerce platform:

* **Binary search** is better for large, static product catalogs that can be kept sorted (O(log n) vs O(n)).
* **Linear search** may be simpler for small datasets or when products are frequently added/removed.
* In practice, databases typically use B-tree indexes (O(log n) complexity) for efficient searching.

For optimal performance, you might:

1. Keep products sorted by ID in memory.
2. Use binary search for ID-based lookups.
3. For other searches (name, category), consider a hash map or search index
4. For very large catalogs, implement database indexing.

