# Exercise 2: E-commerce Platform Search Function

***Understanding Asymptotic Notation:***

Big O notation describes how an algorithm's runtime or space requirements grow as the input size grows. It provides an upper bound on complexity:

* O(1) : Constant time (best case)
* O(long n) : Logarithmic time (very efficient)
* O(n) : Linear time
* O(n2) : Quadratic time (less efficient)

***Search Operation Scenarios:***

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

***Implementation:***

Product.java:

package com.example;

public class Product {

private String productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

}

SearchAlgorithms.java:

package com.example;

import java.util.Arrays;

public class SearchAlgorithms {

//Linear search

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

for(Product product : products) {

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

return product;

}

}

return null;

}

//Binary search

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

int left = 0;

int right = sortedProducts.length-1;

while(left <= right) {

int mid = (left + right)/2;

int comparison = sortedProducts[mid].getProductId().compareTo(targetId);

if(comparison == 0) {

return sortedProducts[mid];

} else if(comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

// sort products by ID

public static void sortProducts(Product[] products) {

Arrays.*sort*(products, (p1, p2) -> p1.getProductId().compareTo(p2.getProductId()));

}

}

Main.java:

package com.example;

import java.util.\*;

public class Main {

public static void main(String[] args) {

// Sample products

Product[] products = {

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

new Product("P203", "Smartphone", "Electronics"),

new Product("P045", "Desk Chair", "Furniture"),

new Product("P302", "Coffee Maker", "Appliances")

};

// Make a copy for sorting using binary search

Product[] sortedProducts = Arrays.*copyOf*(products, products.length);

SearchAlgorithms.*sortProducts*(sortedProducts);

String targetId = "P045";

// Test linear search

Product resultLinear = SearchAlgorithms.*linearSearch*(products, targetId);

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

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

// Test binary search

Product resultBinary = SearchAlgorithms.*binarySearch*(sortedProducts, targetId);

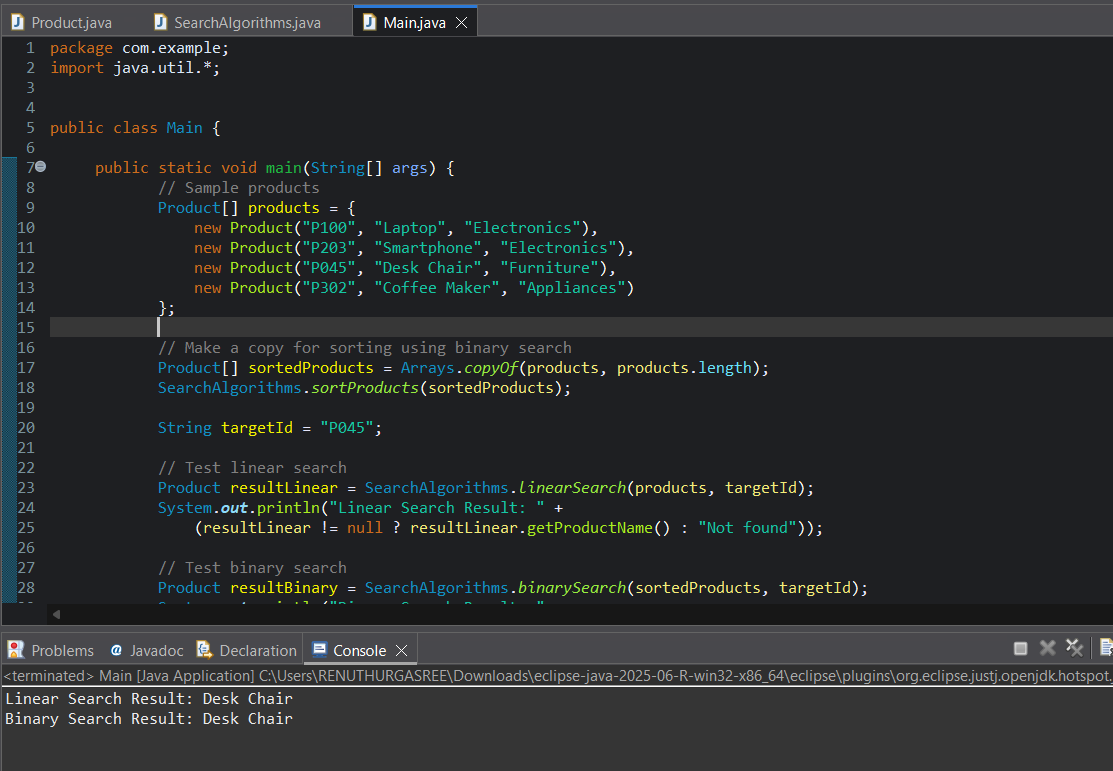
System.***out***.println("Binary Search Result: " +

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

}

}

Output:



***Analysis:***

Time Complexity comparison:

| **Scenario** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Best case** | O(1) - First element | O(1) - Middle element |
| **Average case** | O(n) | O(log n) |
| **Worst case** | O(n) | O(log n) |

Suitable Algorithm:

Binary search will be more suitable for an E-commerce Search Function. As it works well for large , static product catalogs that can be sorted by productId.