**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.

**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.

**My Answers :**

1. **Understand Asymptotic Notation:**

Big – O => upper bound ( worst case ) 🡪 Linear search O(n)

Omega => Lower bound ( best case ) 🡪 Linear search finds at index omega(1)

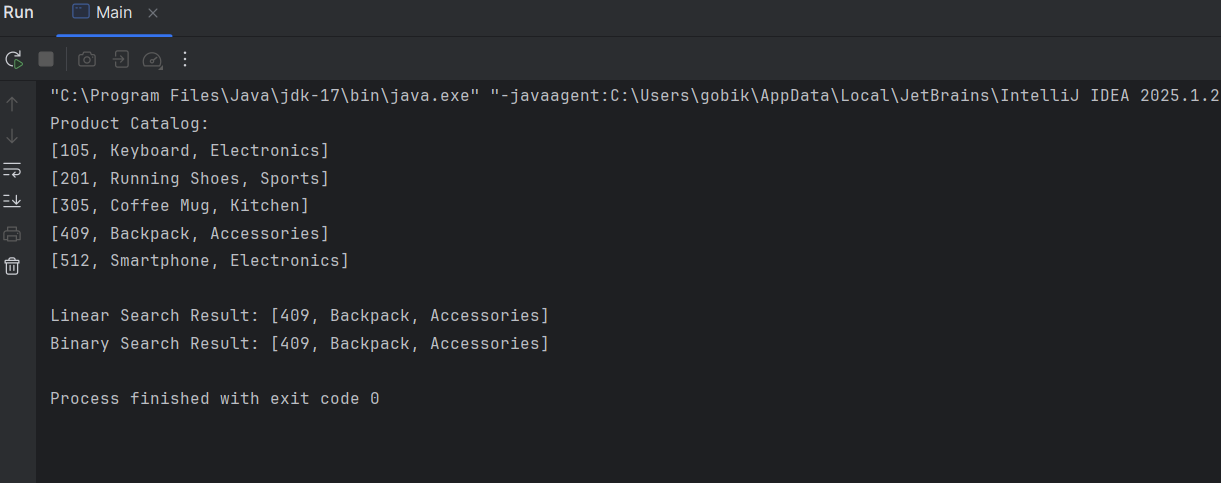
Theta => Tight bound (matches both ) 🡪 Binary search O(log n ) on average & worst case

|  |  |  |  |
| --- | --- | --- | --- |
| **ALGORITHM** | **BEST** | **AVERAGE** | **WORST** |
| Linear search | O(1) | O(n/2) = O(n) | O(n) |
| Binary Search | O(1) | O(log n ) | O(log n ) |

1. **Setup / Implemented Code:**

import java.util.\*;  
  
public class Main {  
  
 static 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 "[" + productId + ", " + productName + ", " + category + "]";  
 }  
 }  
  
 static int linearSearch(Product[] products, int targetId) {  
 for (int i = 0; i < products.length; i++) {  
 if (products[i].productId == targetId) return i;  
 }  
 return -1;  
 }  
 static int binarySearch(Product[] products, int targetId) {  
 int low = 0, high = products.length - 1;  
  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 if (products[mid].productId == targetId) return mid;  
 else if (products[mid].productId < targetId) low = mid + 1;  
 else high = mid - 1;  
 }  
 return -1;  
 }  
 static void sortById(Product[] products) {  
 Arrays.sort(products, Comparator.comparingInt(p -> p.productId));  
 }  
  
 public static void main(String[] args) {  
 Product[] catalog = {  
 new Product(105, "Keyboard", "Electronics"),  
 new Product(201, "Running Shoes", "Sports"),  
 new Product(305, "Coffee Mug", "Kitchen"),  
 new Product(409, "Backpack", "Accessories"),  
 new Product(512, "Smartphone", "Electronics")  
 };  
  
 System.out.println("Product Catalog:");  
 for (Product p : catalog) System.out.println(p);  
  
 // Linear Search  
 int linearResult = linearSearch(catalog, 409);  
 System.out.println("\nLinear Search Result: " + (linearResult != -1 ? catalog[linearResult] : "Not found"));  
  
 sortById(catalog);  
  
 //Binary Search  
 int binaryResult = binarySearch(catalog, 409);  
 System.out.println("Binary Search Result: " + (binaryResult != -1 ? catalog[binaryResult] : "Not found"));  
 }  
}

**OUTPUT :**



1. **Analysis :**

Usage of Binary search :

* + - Searching by productid
    - Data is already sorted manner
    - Fast and repeated searching

Usage of linear search :

**🡪** Dataset is small

**🡪** when list is not sorted // sorting is unnecessary.

🡪 Searching By address ref: name , category

|  |  |  |  |
| --- | --- | --- | --- |
| **ALGORITHM** | **BEST** | **AVERAGE** | **WORST** |
| Linear search | O(1) | O(n/2) = O(n) | O(n) |
| Binary Search | O(1) | O(log n ) | O(log n ) |

**Which is Better for an E-commerce Platform?**

Speed on large products list 🡪 Binary

To use in Unsorted data 🡪 Linear

Performance on the spot 🡪 Binary

Flexibility ( search by hash ( name , address ) ) 🡪 Linear of HashMap