**Data Structures and Algorithms – Exercise 2**

**E-Commerce Platform Search Function**

**Big O Notation (Asymptotic Analysis)**

Big O notation is used to estimate the efficiency of algorithms based on their input size (n). It is used to analyse:  
**Time Complexity:** The growth of the running time of an algorithm with input size.  
**Space Complexity:** The way the memory requirement grows.  
It generally refers to the worst-case, allowing fair comparisons between algorithms regardless of system or hardware performance.

**Search Operations Time Complexity Comparison**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search Method** | **Best Case** | **Average Case** | **Worst Case** | **Note** |
| Linear Search | O(1) | O(n) | O(n) | Does not require sorted data |
| Binary Search | O(1) | O(log n) | O(log n) | Requires sorted data beforehand |

**EcommerceSearch.java**

import java.util.\*;

public class ECommerceSearch {

public static void main(String[] args) {

Product[] productsLinear = {

new Product(303, "Camera", "Electronics"),

new Product(101, "Laptop", "Electronics"),

new Product(220, "Shoes", "Footwear"),

new Product(150, "Phone", "Electronics"),

new Product(400, "Book", "Stationery")

};

Product[] productsBinary = {

new Product(101, "Laptop", "Electronics"),

new Product(150, "Phone", "Electronics"),

new Product(220, "Shoes", "Footwear"),

new Product(303, "Camera", "Electronics"),

new Product(400, "Book", "Stationery")

};

int targetId = 150;

System.out.println("\n🔎 Linear Search Result:\n");

Product linearResult = linearSearch(productsLinear, targetId);

if (linearResult != null)

System.out.println("Product Found: " + linearResult);

else

System.out.println("Product not found using Linear Search.");

System.out.println("\nBinary Search Result:\n");

Product binaryResult = binarySearch(productsBinary, targetId);

if (binaryResult != null)

System.out.println("Product Found: " + binaryResult);

else

System.out.println("Product not found using Binary Search.");

}

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

for (Product p : products) {

if (p.productId == targetId)

return p;

}

return null;

}

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

int low = 0, high = products.length - 1;

while (low <= high) {

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

if (products[mid].productId == targetId)

return products[mid];

else if (products[mid].productId < targetId)

low = mid + 1;

else

high = mid - 1;

}

return null;

}

}

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 "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

**Output**

A computer screen shot of a black screen

AI-generated content may be incorrect.

**Choosing Better Algorithm for the Platform:**

For a site which is an e-commerce site with massive product listings, Binary Search would be the appropriate choice. As soon as the products are indexed based on productId (or any searchable column), binary search substantially reduces the time to search for products, hence the site becomes faster and more efficient.