**Exercise 2: E-commerce Platform Search Function**

**Understanding Asymptotic Notation:**

* **Explain Big O notation and how it helps in analysing algorithms.**
* **Big O notation** is a mathematical way to describe the **performance or complexity of an algorithm** as the input size grows.
* It helps developers understand how efficiently an algorithm scales with larger data.
* It describes the **upper bound (worst-case scenario)** of an algorithm’s running time or space requirement.
* **Best, Average, and Worst-case scenarios for search operations.**
* **Best Case**

The scenario where the algorithm performs the minimum number of steps.

Found at the first position.

* **Average Case**

The expected scenario considering all possible positions.

Found somewhere at the middle.

* **Worst Case**

The scenario where the algorithm performs the maximum number of steps.

Found at the end or not present.

**Setup: Product Class**

**Product.java**

public 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;

}

}

**Implementation: Linear and Binary Search**

**SearchFunctions.java**

public class SearchFunctions {

// Linear Search by productid

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

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

// Binary Search by productId (array must be sorted by productId)

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

return products[mid];

}

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

left = mid + 1;

}

else {

right = mid - 1;

}

}

return null;

}

}

**TestSearch.java**

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

public class TestSearch {

public static void main(String[] args) {

Product[] products = {

new Product(3, "Headphones", "Electronics"),

new Product(1, "Laptop", "Electronics"),

new Product(5, "Shoes", "Fashion"),

new Product(2, "Phone", "Electronics"),

new Product(4, "T-shirt", "Fashion")

};

Scanner sc=new Scanner(System.in);

int targetId = sc.nextInt();

// Linear Search

Product result1 = SearchFunctions.linearSearch(products, targetId);

if (result1 != null)

System.out.println("Linear Search: Found " + result1.productName);

else

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

// Sort products by productId for Binary Search

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

// Binary Search

Product result2 = SearchFunctions.binarySearch(products, targetId);

if (result2 != null)

System.out.println("Binary Search: Found " + result2.productName);

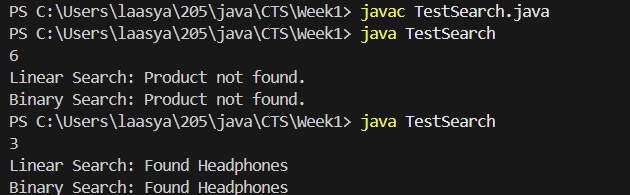
else

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

}

}

**OUTPUT:**

****

**ANALYSIS:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Search Algorithm** | |  | | --- | |  |  |  | | --- | | **Time Complexity** | | **Best Case** | **Average Case** | **Worst Case** | **Suitable for** |
| |  | | --- | | **Linear Search** |  |  | | --- | |  | | |  | | --- | | **O(n)** |  |  | | --- | |  | | |  | | --- | | **O(1)** |  |  | | --- | |  | | |  | | --- | | **O(n/2)** |  |  | | --- | |  | | |  | | --- | | **O(n)** |  |  | | --- | |  | | |  | | --- | | **Small or unsorted product lists** |  |  | | --- | |  | |
| |  | | --- | | **Binary Search** |  |  | | --- | |  | | |  | | --- | | **O(log n)** |  |  | | --- | |  | | |  | | --- | | **O(1)** |  |  | | --- | |  | | |  | | --- | | **O(log n)** |  |  | | --- | |  | | |  | | --- | | **O(log n)** |  |  | | --- | |  | | **Large, sorted product lists** |

**Which Algorithm is More Suitable for an E-Commerce Platform and Why?**

* Binary Search is generally more suitable for large product databases because it is much faster (O(log n)) compared to Linear Search (O(n)), provided the data is sorted by the search key (like productId).
* Linear Search can be acceptable for small, unsorted product lists but will become inefficient as the product list grows.

**Conclusion**:

Use **Binary Search** after sorting products by productId for efficient, scalable search functionality

an e-commerce platform.