# **Assignment: Exercise 2 – E-commerce Platform Search Function**

## **1. Understanding Asymptotic Notation**

In this assignment, I understood how **Big O notation** is used to check the performance of algorithms. It mainly shows how fast or slow the code runs as the data increases.

* **Best Case:** The element we’re searching for is right at the beginning. Very quick.
* **Average Case:** The element is somewhere in the middle. This happens in most real-world cases.
* **Worst Case:** The element is either not there or is at the end, so we have to go through everything.

Big O helps us choose which algorithm to use based on how much data we have and how fast we want it.

## **2. Setup: Creating a Product Class**

I created a Product class in Java to store some basic details like ID, name, and category.

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;

}

}

## **3. Implementation: Linear and Binary Search**

I implemented both **Linear Search** and **Binary Search** in Java. Here's the code:

### **Linear Search:**

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

for (Product product : products) {

if (product.productName.equals(targetName)) {

return product;

}

}

return null;

}

### **Binary Search (sorted by product name):**

public static Product binarySearch(Product[] products, String targetName) {

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int compare = products[mid].productName.compareTo(targetName);

if (compare == 0) {

return products[mid];

} else if (compare < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

### **Sample Product List & Sorting for Binary Search:**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Laptop", "Electronics"),

new Product(103, "Shirt", "Clothing"),

new Product(104, "Shoes", "Footwear"),

new Product(105, "Book", "Stationery")

};

Arrays.sort(products, Comparator.comparing(p -> p.productName));

Product result1 = linearSearch(products, "Shirt");

if (result1 != null) {

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

} else {

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

}

Product result2 = binarySearch(products, "Laptop");

if (result2 != null) {

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

} else {

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

}

}

}

**4. Analysis of the Two Searches**

The time complexity of **Linear Search** is **O(n)**, which means it goes through each item one by one until it finds the match. It’s simple and works well when we have only a small number of products or when the list is not sorted.

On the other hand, **Binary Search** has a time complexity of **O(log n)**. It’s much faster but only works if the list is already sorted. It keeps dividing the list in half and checks the middle item in each step.

So, for an e-commerce platform where we may have thousands of products, **Binary Search** is more efficient and suitable because of its faster performance. But if the list isn’t sorted or the data is very small, **Linear Search** is easier to use and totally fine.

## **5. Output**

Linear Search Found: Shirt in Clothing

Binary Search Found: Laptop in Electronics