# Exercise 2: E-commerce Platform Search Function

### **Understand Asymptotic Notation**

**Big O Notation:**

* **Definition:** Big O notation is a mathematical notation that describes the upper bound of the time complexity of an algorithm. It helps us understand how the runtime of an algorithm grows as the size of the input increases.
* **Purpose:** It provides a high-level understanding of the algorithm's efficiency and performance in terms of time and space.

**Scenarios for Search Operations:**

* **Best Case:** The minimum time an algorithm takes to complete. For example, in a linear search, the best case is when the target element is the first element in the array O(1)O(1)O(1).
* **Average Case:** The expected time an algorithm takes to complete, assuming the input is random. For linear search, it’s O(n/2)O(n/2)O(n/2), which simplifies to O(n)O(n)O(n).
* **Worst Case:** The maximum time an algorithm takes to complete. For linear search, the worst case is when the target element is the last element or not present at all O(n)O(n)O(n).

### **Step 2: Setup**

**Product Class:**

class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

// Getters

public int getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

}

### **Step 3: Implementation**

**Linear Search:**

public class LinearSearch {

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

for (Product product : products) {

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

return product;

}

}

return null; // Return null if the product is not found

}

}

**Binary Search:**

import java.util.Arrays;

public class BinarySearch {

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

Arrays.sort(products, (a, b) -> a.getProductName().compareTo(b.getProductName())); // Sort the array

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int comparison = products[mid].getProductName().compareTo(productName);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null; // Return null if the product is not found

}

}

### **Step 4: Analysis**

**Time Complexity:**

* **Linear Search:**
  + **Best Case:** O(1)O(1)O(1)
  + **Average Case:** O(n)O(n)O(n)
  + **Worst Case:** O(n)O(n)O(n)
* **Binary Search:**
  + **Best Case:** O(1)O(1)O(1)
  + **Average Case:** O(log⁡n)O(\log n)O(logn)
  + **Worst Case:** O(log⁡n)O(\log n)O(logn)

**Comparison and Suitability:**

* **Linear Search:**
  + Simple and does not require the array to be sorted.
  + Suitable for small datasets or unsorted data.
* **Binary Search:**
  + Requires the array to be sorted.
  + Much faster for large datasets due to its logarithmic time complexity.
  + More suitable for an e-commerce platform where the product list can be pre-sorted and fast search performance is critical.

**Conclusion:** For an e-commerce platform, **binary search** is more suitable due to its efficient performance with larger datasets. Pre-sorting the array ensures that search operations are fast and scalable, which is essential for a platform with potentially thousands of products.