# E-commerce Platform Search Function

## 1. Understand Asymptotic Notation

Big O notation is a mathematical tool used to describe the efficiency of an algorithm in terms of time or space as the input size grows. It helps developers analyze and compare different algorithms based on how well they scale.

It helps compare algorithms based on:

* Speed (Time Complexity) – how long it takes to run.
* Memory (Space Complexity) – how much memory it uses.

Common time complexities include:

• O(1) – Constant time. (e.g., accessing an element by index in an array)

• O(n) – Linear time. (e.g., linear search in an unsorted array)

• O(log n) – Logarithmic time. (e.g., binary search in a sorted array)

• O(n log n): Linearithmic time complexity. (e.g., efficient sorting algorithms like merge sort, quick sort)

• O(n!): Factorial time complexity. (e.g., generating all permutations of a string or solving the traveling salesman problem with brute force)

• O(n²) – Quadratic time. (e.g., bubble sort, insertion sort, nested loops over the same input)

• O(2^n): Exponential time complexity. (e.g., recursive calculation of Fibonacci numbers without memoization)

**Search Algorithm Scenarios:**

• Best Case:

- Linear Search: O(1) when the element is first

- Binary Search: O(1) when the element is in the middle

• Average Case:

- Linear Search: O(n/2)

- Binary Search: O(log n)

• Worst Case:

- Linear Search: O(n)

- Binary Search: O(log n)

## 2. Setup

We define a Product class with attributes for searching: productId, productName, and category.

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 "[Product ID: " + productId + ", Name: " + productName + ", Category: " + category + "]";

}

}

## **3. Implementation**

We implement both Linear Search and Binary Search algorithms:

**Linear Search (Time Complexity: O(n))**

public static Product linearSearch(Product[] products, int targetId) {  
 for (Product p : products) {  
 if (p.productId == targetId) {  
 return p;  
 }  
 }  
 return null;  
}

**Binary Search (Time Complexity: O(log n))**

public static Product binarySearch(Product[] sortedProducts, int targetId) {  
 int low = 0, high = sortedProducts.length - 1;  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 if (sortedProducts[mid].productId == targetId) {  
 return sortedProducts[mid];  
 } else if (sortedProducts[mid].productId < targetId) {  
 low = mid + 1;  
 } else {  
 high = mid - 1;  
 }  
 }  
 return null;  
}

## 4. Analysis

Comparison of Time Complexities:

• Linear Search: O(n), works on unsorted data.

• Binary Search: O(log n), requires sorted data.

Binary Search is more efficient for large datasets where searching is frequent. If the product data is sorted or can be kept sorted, Binary Search significantly reduces search time compared to Linear Search. Therefore, Binary Search is more suitable for an e-commerce platform.