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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

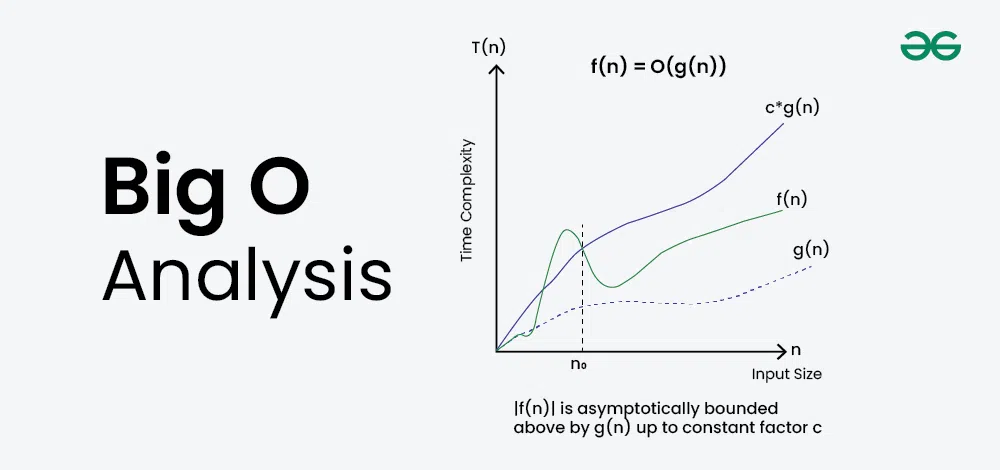
**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class Product with attributes for searching, such as productId, productName, and category.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**Big O Notation**

Big O notation is a powerful tool used in computer science to describe the time complexity or space complexity of algorithms. Big-O is a way to express the upper bound of an algorithm’s time or space complexity.

* Describes the asymptotic behavior (order of growth of time or space in terms of input size) of a function, not its exact value.
* Can be used to compare the efficiency of different algorithms or data structures.
* It provides an**upper limit** on the time taken by an algorithm in terms of the size of the input. We mainly consider the worst case scenario of the algorithm to find its time complexity in terms of Big O
* It’s denoted as**O(f(n))**, where**f(n)** is a function that represents the number of operations (steps) that an algorithm performs to solve a problem of size **n**.



**Big O in Searching:**

* **Linear Search**

**Best case**: O(1) → If the item is found at the beginning

**Average case**: O(n) → You search half the array on average

**Worst case**: O(n) → You search the entire array

* **Binary Search** (Only works on **sorted** arrays)

**Best case**: O(1) → If the item is found at the middle

**Average case**: O(log n)

**Worst case**: O(log n)

**Setup:**

class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

}

**Implementation:**

**For Linear Search**

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

for (Product p : products) {

if (p.productName.equalsIgnoreCase(name)) {

return p;

}

}

return null;

}

**For Binary Search**

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int compare = name.compareToIgnoreCase(products[mid].productName);

if (compare == 0) {

return products[mid];

} else if (compare < 0) {

right = mid - 1;

} else {

left = mid + 1;

}

}

return null;

}

**Analysis:**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |

For an e-commerce platform, binary search is a better choice than linear search because it provides faster performance, especially when dealing with a large number of products. Since customers expect quick search results, binary search can help by reducing the number of comparisons using a divide-and-conquer approach. However, it requires the product list to be sorted by product name or ID, which is usually manageable in such platforms as the data is often structured and can be kept sorted. On the other hand, linear search is slower because it checks each product one by one and is only suitable for small datasets. Therefore, to ensure a smooth and responsive user experience, binary search is the more efficient and practical option for a real-world e-commerce application.