**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.

**1. Understanding Asymptotic Notation**

**Big O Notation**

Big O notation is a mathematical notation used to describe the upper bound of an algorithm's running time. It provides an approximation of the time complexity, representing how the runtime of an algorithm grows with the input size. It helps in comparing the efficiency of different algorithms, especially for large inputs.

**Best, Average, and Worst-Case Scenarios**

* **Best Case:** The scenario where the algorithm performs the minimum number of operations. For a search operation, this occurs when the target element is found at the beginning of the data structure.
* **Average Case:** The scenario representing the expected number of operations across all possible inputs. It provides a more realistic measure of an algorithm's performance.
* **Worst Case:** The scenario where the algorithm performs the maximum number of operations. For a search operation, this occurs when the target element is not in the data structure, or is at the end.

**2. Setup**

We'll create a Product class with attributes productId, productName, and category.

java

class Product {

private int productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

@Override

public String toString() {

return "Product{" +

"productId=" + productId +

", productName='" + productName + '\'' +

", category='" + category + '\'' +

'}';

}

}

**3. Implementation**

We'll implement both linear search and binary search algorithms. Linear search works on an unsorted array, while binary search requires the array to be sorted.

class SearchAlgorithms {

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

for (Product product : products) {

if (product.getProductId() == targetId) {

return product;

}

}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

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

return products[mid];

}

if (products[mid].getProductId() < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

}

**4. Analysis**

**Time Complexity Comparison**

* **Linear Search:**
  + Best Case: O(1) - The target element is the first element in the array.
  + Average Case: O(n) - The target element is somewhere in the middle.
  + Worst Case: O(n) - The target element is the last element or not present.
* **Binary Search:**
  + Best Case: O(1) - The target element is the middle element.
  + Average Case: O(log n) - The target element is found after a few divisions.
  + Worst Case: O(log n) - The target element is not present, and the array is divided until the search space is empty.

**Suitability for the Platform**

* **Linear Search:** Suitable for small datasets where the overhead of sorting is not justified. It's simple and does not require the data to be sorted.
* **Binary Search:** More suitable for larger datasets where the efficiency of search operations is critical. The dataset must be sorted, but the search operations are significantly faster due to the logarithmic time complexity.