**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.

**1. Understand Asymptotic Notation**

**Big O Notation**:

Big O notation is a way to describe the performance or complexity of an algorithm in terms of the size of the input data. It provides an upper bound on the time or space complexity of an algorithm, which helps in understanding how the algorithm scales with larger inputs.

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

* **Best-Case Scenario**: This is the scenario where the algorithm performs the minimum number of operations. For instance, in a search operation, the best case is when the item is found at the very beginning of the data structure.
* **Average-Case Scenario**: This scenario considers the average performance of the algorithm across all possible inputs. For a linear search, this would be the case where the item is located somewhere in the middle of the data structure.
* **Worst-Case Scenario**: This is the scenario where the algorithm performs the maximum number of operations. For a linear search, the worst case is when the item is not in the data structure or is located at the very end.

**LINEAR AND BINARY SEARCH:**

package searchingdsa;

import java.util.Arrays;

public class SearchingDSA {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Smartphone", "Electronics"),

new Product(3, "Tablet", "Electronics"),

new Product(4, "Headphones", "Accessories"),

new Product(5, "Keyboard", "Accessories")

};

// Linear search example

int targetId = 3;

Product result = SearchAlgorithms.linearSearch(products, targetId);

System.out.println("Linear Search Result: " + result);

// Binary search example

Arrays.sort(products, (a, b) -> a.getProductId() - b.getProductId()); // Ensure array is sorted

result = SearchAlgorithms.binarySearch(products, targetId);

System.out.println("Binary Search Result: " + result);

}

}

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 + '\'' +

'}';

}

}

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) (item is found at the first position)
  + **Average Case**: O(n) (item is found somewhere in the middle)
  + **Worst Case: O(n) (item is not found or is at the end)**
* **Binary Search**:
  + **Best Case: O(1) (item is found at the middle position)**
  + **Average Case: O(log n) (performance improves as the list size increases)**
  + **Worst Case: O(log n) (item is found or not found after repeatedly halving the search space)**

**Which Algorithm is More Suitable?**:

For an e-commerce platform:

* **Binary Search** is generally more suitable when you have a sorted list and need fast search times, as it provides a logarithmic time complexity, making it much more efficient for large datasets.
* **Linear Search** might be more appropriate for unsorted data or when dealing with small datasets. However, its time complexity grows linearly with the size of the dataset, which is less efficient for large datasets compared to binary search.