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

**Understand Asymptotic Notation**

**Big O Notation**

Big O notation is a mathematical notation used to describe the upper bound of an algorithm's runtime or space requirements in terms of the size of the input. It helps in analyzing the efficiency of algorithms by focusing on their behavior as the input size grows.

* O(1): Constant time - The algorithm's runtime does not depend on the input size.
* O(n): Linear time - The algorithm's runtime grows linearly with the input size.
* O(log n): Logarithmic time - The algorithm's runtime grows logarithmically with the input size.
* O(n^2): Quadratic time - The algorithm's runtime grows quadratically with the input size.
* O(2^n): Exponential time - The algorithm's runtime grows exponentially with the input size.

Best, Average, and Worst-case Scenarios for Search Operations

* Best Case: The scenario where the search operation performs the minimum number of steps. For example, in a linear search, this happens when the target element is the first element of the array.
* Average Case: The scenario that represents the expected number of steps for a search operation considering all possible cases.
* Worst Case: The scenario where the search operation performs the maximum number of steps. For example, in a linear search, this happens when the target element is the last element or not present in the array.

**Implementation process**

1. Requirement Analysis:
   * Understand the need for a search function in an e-commerce platform.
   * Identify the key features: ability to search products and compare different search algorithms.
2. Design:
   * Create a class structure: Product, LinearSearch, BinarySearch, and Main.
   * Define the relationships between these classes.
3. Implementation: a. Product Class:
   * Define attributes: productId, productName, category.
   * Implement constructor, getters, and toString method.

b. LinearSearch Class:

* + Implement static search method that iterates through an array of products.

c. BinarySearch Class:

* + Implement static search method that first sorts the array, then performs binary search.

d. Main Class:

* + Implement main method to orchestrate the program flow.
  + Create sample products.
  + Get user input for search method and search term.
  + Perform the search and measure execution time.
  + Display results.

1. Testing:
   * Test each class individually.
   * Perform integration testing to ensure all components work together.
2. Optimization:
   * Analyze performance and optimize if necessary.

Now, let's dive deeper into the implementation of each class:

1. Product Class:
   * This class is straightforward. We define the attributes, constructor, getters, and toString method.
   * The toString method is overridden to provide a custom string representation of the Product object.
2. LinearSearch Class:
   * The search method takes an array of Products and a search term.
   * It uses a for-each loop to iterate through all products.
   * It compares each product's name with the search term (case-insensitive).
   * If a match is found, it returns the Product; otherwise, it returns null.
3. BinarySearch Class:
   * The search method also takes an array of Products and a search term.
   * First, it sorts the array based on product names using Arrays.sort with a custom Comparator.
   * Then it performs the binary search algorithm:
     + It repeatedly divides the search interval in half.
     + If the search term is less than the middle element, it continues searching the lower half.
     + If greater, it searches the upper half.
     + If equal, it returns the found Product.
   * If the product is not found, it returns null.
4. Main Class:
   * We start by creating sample products using the createSampleProducts method.
   * We use a Scanner to get user input for the search method and search term.
   * Based on the user's choice, we call either LinearSearch or BinarySearch.
   * We use System.nanoTime() to measure the execution time of the search.
   * Finally, we display the search result and the time taken.

Implementation Process:

1. We start by creating the Product class as it's the foundation of our system.
2. Next, we implement the search algorithms in separate classes for modularity.
3. We then create the Main class to tie everything together and provide user interaction.
4. Throughout the process, we test each component to ensure it works correctly.
5. Finally, we optimize and refine the code as needed.

This implementation demonstrates several key programming concepts:

* Object-Oriented Programming (encapsulation, method overriding)
* Static methods
* Array manipulation
* Basic algorithms (linear search, binary search)
* User input handling
* Performance measurement

**Class Diagram Description**

* Product Class: Represents the product with attributes such as product Id, product Name, and category.
* Search Interface: Defines the contract for search methods.
* Linear Search Class: Implements the linear search algorithm.
* Binary Search Class: Implements the binary search algorithm.

**Class Diagram**

Product

-productId

-productName

-category

+getProdcutId()

+getProductName()

+getCategory()

+toString()

Linear Search

+search()

Binary Search

+search()

Main

-product

+main()

-createSample

Product()

**Explanation of the class diagram:**

1. Product Class:
   * This is the core model class representing a product in the e-commerce system.
   * It has private attributes: productId, productName, and category.
   * It provides public getter methods for each attribute.
   * It overrides the toString() method for easy printing of product information.
2. LinearSearch Class:
   * This class implements the linear search algorithm.
   * It has a static search() method that takes an array of Products and a search term.
   * It returns a Product if found, or null if not found.
3. BinarySearch Class:
   * This class implements the binary search algorithm.
   * Like LinearSearch, it has a static search() method with the same signature.
   * It sorts the array before performing the binary search.
4. Main Class:
   * This is the entry point of the program.
   * It contains the main() method which orchestrates the program flow.
   * It has a private method createSampleProducts() to initialize the product array.
   * It interacts with the user, calls the appropriate search method, and displays results.

Relationships:

* The Main class has a composition relationship with Product (indicated by the diamond). It creates and manages an array of Product objects.
* The Main class uses (depends on) both LinearSearch and BinarySearch classes, as indicated by the dashed arrows.
* There's no direct relationship between Product and the search classes, but the search classes operate on arrays of Product objects.

This design follows several object-oriented principles:

1. Single Responsibility Principle: Each class has a single, well-defined responsibility.
2. Encapsulation: Product attributes are private with public getter methods.
3. Separation of Concerns: Search algorithms are separated into their own classes.
4. Code Reusability: Search methods can be easily reused for different types of searches.

This structure allows for easy extension. For example, if you wanted to add a new search algorithm, you could create a new class without modifying existing code. It also separates the data model (Product) from the algorithms that operate on it, making the code more maintainable and flexible.

**4. Analysis**

To analyze the time complexity of each main component of the program using Big O notation:

1. Product Class:
   * All methods (getters, toString): O(1) These are simple operations that don't depend on the input size.
2. LinearSearch Class:
   * search() method: O(n) Where n is the number of products in the array. In the worst case, it needs to traverse the entire array.
3. BinarySearch Class:
   * search() method: O(log n) Where n is the number of products in the array. However, there's an important caveat:
   * Sorting step: O(n log n) The Arrays.sort() method uses a variant of quicksort, which has an average time complexity of O(n log n).

Therefore, the total time complexity for binary search including sorting is: O(n log n) + O(log n) = O(n log n)

1. Main Class:
   * createSampleProducts(): O(1) This method creates a fixed number of products, so it's constant time.
   * main() method:
     + User input operations: O(1)
     + Calling search methods:
       - For LinearSearch: O(n)
       - For BinarySearch: O(n log n)

Overall Program Time Complexity:

* When using Linear Search: O(n)
* When using Binary Search: O(n log n)

Important notes:

1. While binary search itself is O(log n), the need to sort the array first makes its total complexity O(n log n) in this implementation.
2. For small datasets, linear search might actually be faster due to its simplicity and the overhead of sorting in binary search.
3. If the array was kept sorted at all times (e.g., inserting new elements in the correct position), then binary search would indeed be O(log n), making it more efficient for large datasets.
4. Space complexity for both search algorithms is O(1) as they don't use any extra space that grows with input size.
5. The time complexity of System.nanoTime() is generally considered O(1), although its actual performance can vary depending on the system.

In practice, the choice between linear and binary search would depend on:

* The size of the dataset
* How frequently the data changes
* How often searches are performed

For very small datasets or infrequent searches, the simpler linear search might be preferable. For large datasets with frequent searches, keeping the data sorted and using binary search would be more efficient.