**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. Understand Asymptotic Notation**

* Explain Big O notation and how it helps in analyzing algorithms.

**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.
* Describe the best, average, and worst-case scenarios for search operations.
* Describe the 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.
* **Binary Search**: The target element is found in the middle of the array on the first comparison. Time complexity is O(1).
* **Linear Search**: The target element is found at the very beginning of the list. Time complexity is O(1).
* Average Case: The scenario that represents the expected number of steps for a search operation considering all possible cases.
* **Binary Search**: The target element is found after examining half the elements in a balanced, sorted array. Time complexity is O(log n).
* **Linear Search**: The target element is found in the middle of the list on average. Time complexity is O(n/2)=O(n).
* 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.
* **Binary Search**: The target element is not present in the array, and the algorithm has to search through all possible positions (logarithmically). Time complexity is O(log n).
* **Linear Search**: The target element is not present, and the algorithm has to check every element in the list. Time complexity is O(n).

2. Setup:

* **Class Creation**: Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.

class Product implements Comparable<Product> {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

* Defines the Product class with attributes productId, productName, and category.
* Implements the Comparable interface to allow sorting by productId.

**Attributes of the Product class:**

* **int productId**: An integer representing the unique identifier for each product.
* **String productName**: A string representing the name of the product.
* **String category**: A string representing the category of the product.

**Constructor:**

* **public Product(int productId, String productName, String category):** The constructor has three parameters: productId (an integer), productName (a string), and category(a string). These parameters are used to initialize the instance variables.
* **this.productId = productid;**: The this keyword is used to refer to the current instance of the class. This line assigns the value of the parameter productId to the instance variable productId.
* **this.productName = productName;**: Similarly, this line assigns the value of the parameter productName to the instance variable productName.
* **this.category = category;**: This line assigns the value of the parameter category to the instance variable category.

3. Implementation:

* Implement linear search and binary search algorithms:

Linear Search

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

for (Product product : products) {

if (product.productId == productId) {

return product;

}

}

return null;

}

Binary Search

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

if (products[mid].productId == productId) {

return products[mid];

}

if (products[mid].productId < productId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

}

* **linearSearch Method**:
* Iterates through the products array and returns the product if the productId matches the search ID.
* **binarySearch Method**:
* Performs a binary search on the sorted products array to find the product with the given productId.
* Store products in an array for linear search and a sorted array for binary search.

public class ECommerceSearch {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of products:");

int n = scanner.nextInt();

scanner.nextLine();

Product[] products = new Product[n];

for (int i = 0; i < n; i++) {

System.out.println("Enter details for product " + (i + 1) + ":");

System.out.print("Product ID: ");

int productId = scanner.nextInt();

scanner.nextLine();

System.out.print("Product Name: ");

String productName = scanner.nextLine();

System.out.print("Category: ");

String category = scanner.nextLine();

products[i] = new Product(productId, productName, category);

}

Product[] sortedProducts = Arrays.copyOf(products, products.length);

Arrays.sort(sortedProducts);

System.out.println("Enter product ID to search:");

int searchId = scanner.nextInt();

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

Product resultLinear = linearSearch(products, searchId);

if (resultLinear != null) {

System.out.println(resultLinear);

} else {

System.out.println("Product not found.");

}

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

Product resultBinary = binarySearch(sortedProducts, searchId);

if (resultBinary != null) {

System.out.println(resultBinary);

} else {

System.out.println("Product not found.");

}

scanner.close();

}

* **ECommerceSearch Class**:
* Contains the main method, which prompts the user to input the number of products and their details.
* Copies and sorts the products array for binary search.
* Prompts the user to enter a productId to search.
* Calls the linearSearch and binarySearch methods and prints the results.

Here is the github repo link –

**4. Analysis**

* Compare the time complexity of linear and binary search algorithms.

**Time Complexity** of Linear Search :

* + linearsearch() method: O(n) Where n is the number of products in the array. In the worst case, it needs to traverse the entire array.

**Time Complexity** of Binary Search :

* + binarysearch() 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)

* Discuss which algorithm is more suitable for your platform and why.
* 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.
* For small datasets, linear search might actually be faster due to its simplicity and the overhead of sorting in binary search.
* 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.
* Space complexity for both search algorithms is O(1) as they don't use any extra space that grows with input size.

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.

Output:

****