Step 1: Understand Asymptotic Notation

Big O Notation:

Big O notation is a mathematical notation that describes the complexity of an algorithm, which is the amount of time or space it requires as the size of the input increases. It's used to classify algorithms according to how their run time or space requirements grow as the input size grows.

Best, Average, and Worst-Case Scenarios:

Best-Case Scenario: The best-case scenario is the minimum time an algorithm takes to complete. For search operations, the best-case scenario is when the target element is found at the first position.

Average-Case Scenario: The average-case scenario is the average time an algorithm takes to complete. For search operations, the average-case scenario is when the target element is found in the middle of the list.

Worst-Case Scenario: The worst-case scenario is the maximum time an algorithm takes to complete. For search operations, the worst-case scenario is when the target element is not found in the list.

Step 2: Setup

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

public class Product {

private String productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

// Getters and setters

public String getProductId() {

return productId;

}

public void setProductId(String productId) {

this.productId = productId;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

}

Step 3: Implementation

Implement linear search and binary search algorithms:

public class Search {

// Linear search algorithm

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

for (Product product : products) {

if (product.getProductName().equals(target)) {

return product;

}

}

return null;

}

// Binary search algorithm

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (products[mid].getProductName().equals(target)) {

return products[mid];

} else if (products[mid].getProductName().compareTo(target) < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

}

Step 4: Analysis

Time Complexity Comparison:

Linear Search: The time complexity of linear search is O(n), where n is the number of products. This is because in the worst-case scenario, the algorithm has to iterate through all products to find the target.

Binary Search: The time complexity of binary search is O(log n), where n is the number of products. This is because the algorithm divides the search space in half at each step, reducing the number of iterations required to find the target.

Algorithm Suitability:

Binary search is more suitable for the e-commerce platform search function because it has a faster time complexity, especially for large datasets. However, binary search requires the products to be sorted, which may not always be the case. If the products are not sorted, linear search may be a better option.