**E-commerce Platform Search Function**

**Step 1: Understand Asymptotic Notation**

**Big O Notation:**

* Big O notation is used to describe the performance or complexity of an algorithm.
* It tells how fast or slow an algorithm works as the input size increases.
* It helps in comparing different algorithms.
* It guides us to choose the best approach for scalability and efficiency.

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

**1. Best-Case Scenario**

* **Definition:**The best-case scenario happens when the item being searched is found immediately, with the least number of steps.
* **Linear Search Example:**The product is the first item in the array.  
  Only one comparison is needed.  
  Time Complexity: O(1)
* **Binary Search Example:**The product is the middle element of the sorted array in the first attempt.  
  Time Complexity: O(1)

**2. Average-Case Scenario**

* **Definition:**The average case represents the expected number of steps taken when the item’s position is random or unknown.
* **Linear Search Example:**The product is somewhere in the middle of the list.  
  Time Complexity: O(n)
* **Binary Search Example:**The product is not found immediately, but found after a few divisions of the array.  
  Time Complexity: O(log n)

**3. Worst-Case Scenario**

* **Definition:**The worst-case scenario occurs when the item is not present at all, or is the last item to be found.
* **Linear Search Example:**The product is the last item, or does not exist in the list.  
  Requires checking all items.  
  Time Complexity: O(n)
* **Binary Search Example:**The product is not in the sorted list, and we have to divide until the search range becomes empty.  
  Time Complexity: O(log n)

**Step 2: Setup**

class Product {

    int id;

    String name;

    String type;

    public Product(int id, String name, String type) {

        this.id = id;

        this.name = name;

        this.type = type;

    }

}

**Step 3: Implementation**

**Linear Search Code:**

class Product {

    int id;

    String name;

    String type;

    public Product(int id, String name, String type) {

        this.id = id;

        this.name = name;

        this.type = type;

    }

    public void Details() {

        System.out.println("Product ID: " + id + ", Name: " + name + ", Category: " + type);

    }

}

public class LinearSearchExample {

    public static Product findProductByName(Product[] items, String searchName) {

        for (int i = 0; i < items.length; i++) {

            Product product = items[i];

            if (product.name.equalsIgnoreCase(searchName)) {

                return product;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] inventory = {

            new Product(1001, "Television", "Electronics"),

            new Product(1002, "Laptop", "Electronics"),

            new Product(1003, "Smartphone", "Electronics"),

            new Product(1004, "Smartwatch", "Electronics")

        };

        String query = "Laptop";

        System.out.println("Searching for product: " + query);

        Product match = findProductByName(inventory, query);

        if (match != null)

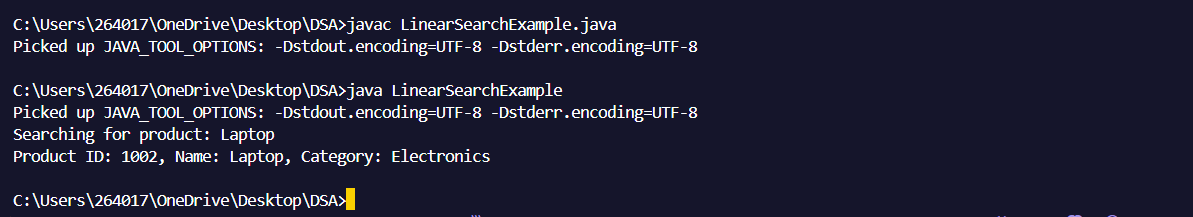
            match.Details();

        else

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

    } }

**Output:**

****

**Binary Search Code:**

import java.util.Arrays;

class Product implements Comparable<Product> {

    int id;

    String name;

    String category;

    public Product(int id, String name, String category) {

        this.id = id;

        this.name = name;

        this.category = category;

    }

    @Override

    public int compareTo(Product other) {

        return Integer.compare(this.id, other.id);

    }

    public void show() {

        System.out.println("Product ID: " + id + ", Name: " + name + ", Category: " + category);

    }

}

public class BinarySearchExample {

    public static Product searchById(Product[] items, int targetId) {

        int start = 0;

        int end = items.length - 1;

        while (start <= end) {

            int mid = (start + end) / 2;

            Product midProduct = items[mid];

            if (midProduct.id == targetId) {

                return midProduct;

            } else if (midProduct.id < targetId) {

                start = mid + 1;

            } else {

                end = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

         Product[] products = {

            new Product(1001, "Television", "Electronics"),

            new Product(1002, "Laptop", "Electronics"),

            new Product(1003, "Smartphone", "Electronics"),

            new Product(1004, "Smartwatch", "Electronics")

        };

        Arrays.sort(products);

        int search = 1003;

        System.out.println("Searching for Product ID: " + search);

        Product found = searchById(products, search);

        if (found != null)

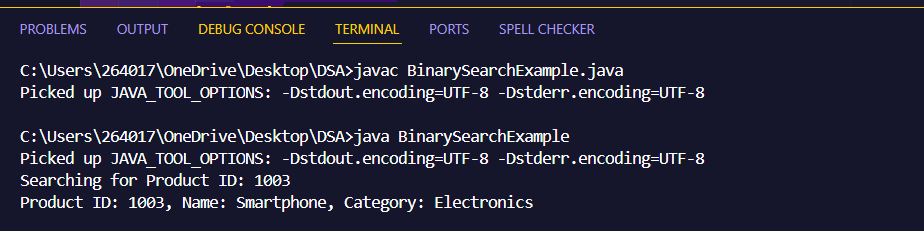
            found.show();

        else

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

    }}

**Output:**

****

**Step 4: Analysis**

**1. Linear Search – Time Complexity**

| Case | Time Complexity |
| --- | --- |
| Best Case | O(1) |
| Average Case | O(n) |
| Worst Case | O(n) |

* Best Case: Product is found at the first position.
* Average Case: Product is found somewhere in the middle.
* Worst Case: Product is found at the end or not present at all.
* Linear search checks every item one by one, so performance drops as the product list grows.

**2. Binary Search – Time Complexity**

| Case | Time Complexity |
| --- | --- |
| Best Case | O(1) |
| Average Case | O(log n) |
| Worst Case | O(log n) |

* Best Case: Product is found in the first middle comparison.
* Average/Worst Case: Each step cuts the search area in half.
* Binary search is much faster and works well with large sorted data.

Suitable Algorithm:

**Binary Search is more suitable for an e-commerce platform.**

Faster Performance:

* Binary search is much faster for large product lists since it reduces the number of comparisons dramatically.
* For example, in a list of 1,000 items, binary search only needs around 10 steps to find a product.

Large Product Database:

* E-commerce platforms usually deal with thousands or even millions of products.
* Linear search becomes slow and inefficient in such scenarios.

Sorted Data Availability:

* Product lists are usually sorted (Mostly alphabetically, by price or by category, etc.), making binary search easy to implement.

User Experience:

* Fast search means quicker results, which improves customer experience on the platform.