**Exercise 2: E-commerce Platform Search Function**  
  
**Problem Statement** : You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.  
  
**CODE** :  
 **1. Using Linear Search**

import java.util.Scanner;

class 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 String toString() {

return "Product [ID=" + id + ", Name=" + name + ", Category=" + category + "]";

}

}

public class LinearSearchDemo {

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

for (Product product : products) {

if (product.id == targetId) {

return product;

}

}

return null;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

sc.nextLine();

Product[] list = new Product[n];

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

System.out.println("\nProduct " + (i + 1));

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

int id = sc.nextInt(); sc.nextLine();

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

String name = sc.nextLine();

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

String cat = sc.nextLine();

list[i] = new Product(id, name, cat);

}

System.out.print("\nEnter ID to search: ");

int searchId = sc.nextInt();

long start = System.nanoTime();

Product result = linearSearch(list, searchId);

long end = System.nanoTime();

if (result != null)

System.out.println("Found: " + result);

else

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

System.out.println("Time taken: " + (end - start) + " ns");

sc.close();

}

}  
  
  
**2. Using Binary Search**  
  
  
import java.util.Scanner;

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);

}

@Override

public String toString() {

return "Product [ID=" + id + ", Name=" + name + ", Category=" + category + "]";

}

}

public class BinarySearchDemo {

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

int left = 0, right = products.length - 1;

while (left <= right) {

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

if (products[mid].id == targetId)

return products[mid];

else if (products[mid].id < targetId)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

sc.nextLine();

Product[] list = new Product[n];

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

System.out.println("\nProduct " + (i + 1));

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

int id = sc.nextInt(); sc.nextLine();

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

String name = sc.nextLine();

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

String cat = sc.nextLine();

list[i] = new Product(id, name, cat);

}

// Must sort for binary search

Arrays.sort(list);

System.out.print("\nEnter ID to search: ");

int searchId = sc.nextInt();

long start = System.nanoTime();

Product result = binarySearch(list, searchId);

long end = System.nanoTime();

if (result != null)

System.out.println("Found: " + result);

else

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

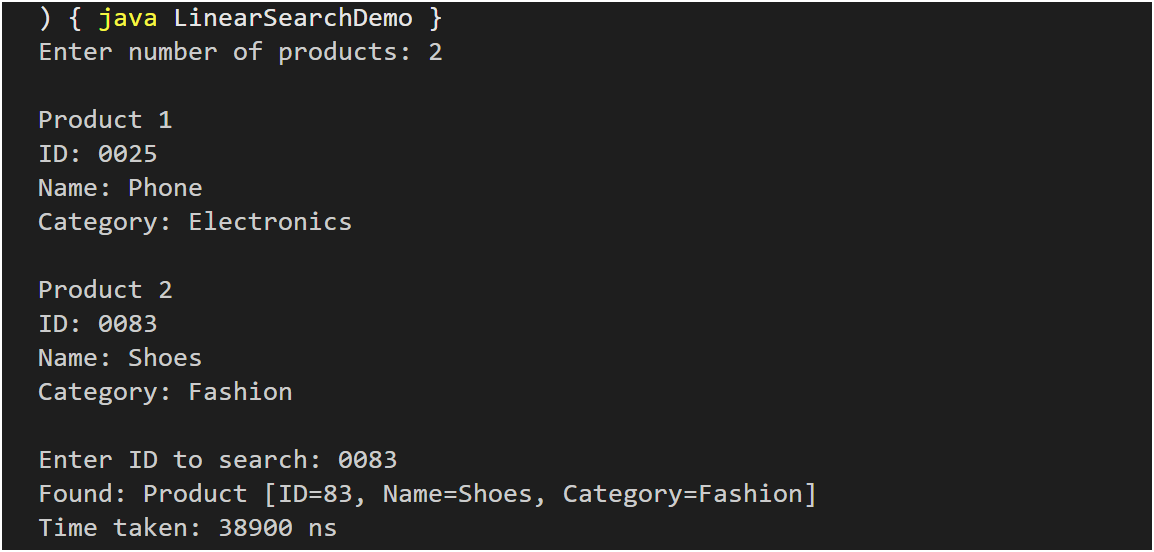
System.out.println("Time taken: " + (end - start) + " ns");

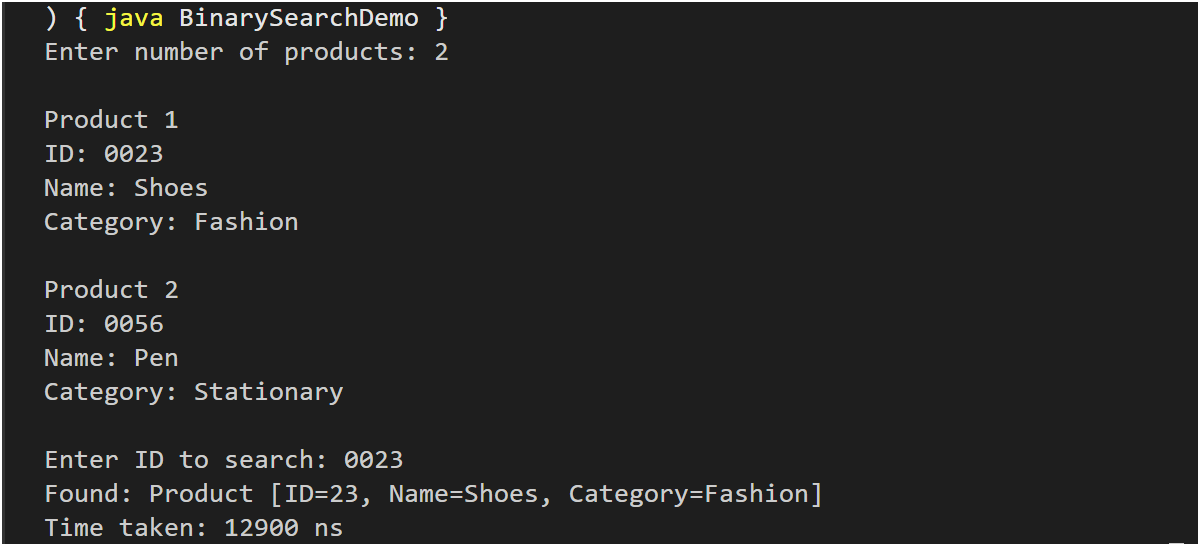
sc.close();

}

}

**OUTPUTS** :

**1 Linear Search Output**:  
  


**2 Binary Search Output:**  
  


**Analysis:  
  
Comparison of Time Complexity:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search Algorithm** | Best Case | Average Case | Worst Case | Time Complexity |
| Linear Search | O(1) | O(n/2) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) | O(log n) |

**Which Algorithm is More Suitable for an E-Commerce Platform :**In the context of an e-commerce platform, where the volume of products can range from hundreds to several millions, the efficiency and speed of search operations are paramount. The choice of the appropriate search algorithm plays a significant role in ensuring a seamless and responsive user experience.

**Linear search** is a straightforward approach that sequentially scans each product in the list until a match is found. While this method is acceptable for small datasets or temporary lists—such as limited search results on a client-side interface—it becomes increasingly inefficient as the dataset size grows. The time complexity of O(n) in the worst-case scenario makes it unsuitable for large-scale applications where frequent searches are expected.

In contrast, **binary search** offers a more scalable and performance-oriented solution. Operating with a time complexity of O(log n), it significantly reduces the number of comparisons required to locate an item. Although it requires the data to be sorted, most e-commerce systems inherently maintain sorted indexes for key attributes such as product IDs, prices, or categories—either through database indexing or in-memory structures. As a result, binary search can be implemented effectively without substantial overhead.

**Conclusion:**  
Given the scale and performance demands of modern e-commerce platforms, **binary search is the more appropriate choice**. Its logarithmic time complexity and compatibility with sorted data structures make it highly efficient and reliable for handling large volumes of search operations.