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

**Asymptotic Notation Explanation**

**Big O Notation:**

* **Big O (O-notation)** describes the upper bound of the time complexity.
* It expresses how an algorithm's runtime or space requirement grows relative to input size n.
* Examples:
  + O(1) – Constant time
  + O(n) – Linear time
  + O(log n) – Logarithmic time
  + O(n log n), O(n^2), etc.
* **Search Case Scenarios:**

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Linear Search** | O(1) (first item match) | O(n/2) → O(n) | O(n) (last item or not found) |
| **Binary Search** | O(1) (middle match) | O(log n) | O(log n) |

**Product.java**

package javapro;

public class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

}

**SearchService.java**

package javapro;

import java.util.\*;

public class SearchService {

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

for (Product p : products) {

if (p.productName.equalsIgnoreCase(searchName)) {

return p;

}

}

return null;

}

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int result = products[mid].productName.compareToIgnoreCase(searchName);

if (result == 0) return products[mid];

else if (result < 0) low = mid + 1;

else high = mid - 1;

}

return null;

}

public static void sortProducts(Product[] products) {

Arrays.*sort*(products, new Comparator<Product>() {

public int compare(Product a, Product b) {

return a.productName.compareToIgnoreCase(b.productName);

}

});

}

}

**Main.java**

package javapro;

import java.util.Scanner;

public class Main {

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[] products = new Product[n];

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

System.***out***.println("\nEnter details for product " + (i + 1) + ":");

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

int id = sc.nextInt();

sc.nextLine();

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

String name = sc.nextLine();

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

String category = sc.nextLine();

products[i] = new Product(id, name, category);

}

System.***out***.println("\n--- Product Search ---");

System.***out***.println("1. Linear Search");

System.***out***.println("2. Binary Search");

System.***out***.print("Enter your choice (1 or 2): ");

int choice = sc.nextInt();

sc.nextLine();

System.***out***.print("Enter product name to search: ");

String searchName = sc.nextLine();

Product result = null;

switch (choice) {

case 1:

result = SearchService.*linearSearch*(products, searchName);

break;

case 2:

SearchService.*sortProducts*(products);

result = SearchService.*binarySearch*(products, searchName);

break;

default:

System.***out***.println("Invalid choice.");

return;

}

if (result != null) {

System.***out***.println("\nProduct found: " + result.productName);

System.***out***.println("ID: " + result.productId);

System.***out***.println("Category: " + result.category);

} else {

System.***out***.println("\nProduct not found.");

}

}

}

**OUTPUT:**

Enter number of products: 3

Enter details for product 1:

Product ID: 1

Product Name: Laptop

Category: Electronics

Enter details for product 2:

Product ID: 2

Product Name: Shoes

Category: Footwear

Enter details for product 3:

Product ID: 3

Product Name: Book

Category: Stationery

--- Product Search ---

1. Linear Search

2. Binary Search

Enter your choice (1 or 2): 2

Enter product name to search: laptop

Product found: Laptop

ID: 1

Category: Electronics

**Comparison and Suitability:**

| **Criteria** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Time Complexity** | O(n) | O(log n) |
| **Data Requirement** | Unsorted data | Sorted data required |
| **Use Case** | Small or unsorted datasets | Large, sorted datasets |

**Which is Better for E-Commerce?**

* **Binary Search** is preferred for performance if:
  + Products are stored in sorted order (or can be indexed).
  + Dataset is large (e.g., thousands+ of products).
* **Linear Search** is simple and works without sorting — best for small or unsorted lists.