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

**1. Understand Asymptotic Notation**

Big O notation is a powerful tool used in computer science to describe the time complexity or space complexity of algorithms. Big-O is a way to express the upper bound of an algorithm’s time or space complexity.

* Describes the asymptotic behavior (order of growth of time or space in terms of input size) of a function, not its exact value.
* Can be used to compare the efficiency of different algorithms or data structures.
* It provides an upper limit on the time taken by an algorithm in terms of the size of the input. We mainly consider the worst-case scenario of the algorithm to find its time complexity in terms of Big O
* It’s denoted as O(f(n)), where f(n) is a function that represents the number of operations (steps) that an algorithm performs to solve a problem of size n.

|  |  |  |
| --- | --- | --- |
| Case | Linear Search | Binary Search |
| Best | O(1) - first element is the key | O(1) - middle element is the key |
| Average | O(n/2) ≈ O(n) | O(log n) |
| Worst | O(n) - last element is the key/key absent | O(log n) |

**2. Setup**

public class Product{

    int productId;

    String productName;

    String category;

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

        this.productId=id;

        this.productName=name;

        this.category=category;

    }

}

**3. Implementation**

public class Search{

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

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

            if (products[i].productName.equalsIgnoreCase(name)) {

                return products[i];

            }

        }

        return null;

    }

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

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

        while (low <= high) {

            int mid = low + (high-low)/2;

            int cmp = products[mid].productName.compareToIgnoreCase(name);

            if (cmp == 0) {

                return products[mid];

            } else if (cmp < 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

}

public class ProductTest {

    public static void main(String[] args) {

        Product[] products = {

            new Product(1, "Laptop", "Electronics"),

            new Product(2, "Shoes", "Footwear"),

            new Product(3, "Keyboard", "Electronics"),

            new Product(4, "Book", "Stationery")

        };

        Product linearResult=Search.linearSearch(products,"keyboard");

        System.out.println("id: "+linearResult.productId+"\nCategory: "+linearResult.category);

        Product[] sorted\_products = {

            new Product(1, "Book", "Stationery"),

            new Product(2, "Keyboard", "Electronics"),

            new Product(3, "Laptop", "Electronics"),

            new Product(4, "Shoes", "Footwear")

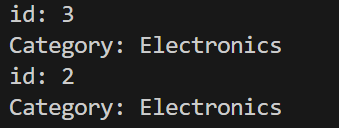
        };

        Product binaryResult=Search.binarySearch(sorted\_products,"keyboard");

        System.out.println("id: "+binaryResult.productId+"\nCategory: "+binaryResult.category);

    }

}



**4. Analysis**

Linear Search – O(n)

Binary Search – O(log n)

Binary search is better for performance, but only if the product list is sorted. For dynamic/unsorted data or very small datasets, linear search is simpler.

**Exercise 7: Financial Forecasting**

**1. Understand Recursive Algorithms**

Recursion is a process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

**2. Setup**

**3. Implementation**

public class Forecast {

    public static double calculateFutureValue(double principal, double rate, int years){

        if(years==0)

            return principal;

        return calculateFutureValue(principal\*(1+rate), rate, years-1);

    }

    public static void main(String[] args) {

        double principal=10000.00;

        double rate=0.1;

        int years=5;

        double futureValue=calculateFutureValue(principal, rate, years);

        System.out.println("Future value = "+futureValue);

    }

}



**4. Analysis**

Time Complexity: O(n) - because for each year, we perform one recursive call.

Recursion can lead to stack overflow if n is large. Use tail recursion or iterative approach or memoization for optimized results.