Exercise 2: E-commerce Platform Search Function

Scenario:

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

Steps:

1. Understand Asymptotic Notation:

o Explain Big O notation and how it helps in analyzing algorithms.

o Describe the best, average, and worst-case scenarios for search operations.

2. Setup:

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

3. Implementation:

o Implement linear search and binary search algorithms.

o Store products in an array for linear search and a sorted array for binary search.

4. Analysis:

o Compare the time complexity of linear and binary search algorithms.

o Discuss which algorithm is more suitable for your platform and why.

**SOLUTIONS:**

**1.** Big O 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.

DEF: Given two functions**f(n)** and **g(n)**, we say that**f(n)** is**O(g(n))** if there exist constants**c > 0** and **n0 >=**0 such that**f(n) <= c\*g(n)** for all **n >= n0**.

* Big O Notation is important because it helps analyze the efficiency of algorithms.
* It provides a way to describe how the**runtime**or **space requirements** of an algorithm grow as the input size increases.
* Allows programmers to compare different algorithms and choose the most efficient one for a specific problem.
* Helps in understanding the scalability of algorithms and predicting how they will perform as the input size grows.
* Best , avg and worst case scenarios:

1. WORST CASE

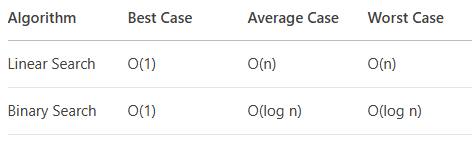
* In the worst-case analysis, we calculate the upper bound on the running time of an algorithm. We must know the case that causes a maximum number of operations to be executed.
* This is the most commonly used analysis of algorithms (We will be discussing below why). Most of the time we consider the case that causes maximum operations.

2. BEST CASE

* In the best-case analysis, we calculate the lower bound on the running time of an algorithm. We must know the case that causes a minimum number of operations to be executed.

3. AVERAGE CASE

* In average case analysis, we take all possible inputs and calculate the computing time for all of the inputs.
* Sum all the calculated values and divide the sum by the total number of inputs.



**2. CODE: setup**

public class Product {

int productId;

String productName;

String category;

public Product(int id, String name, String cat)

{

productId = id;

productName = name;

category = cat;

}

}

**3. Algorithms implementation: Linear search & Binary search**

public class LinearSearch {

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

for (Product p : products) {

if (p.productId == targetId) {

return p;

}

}

return null;

}

}

// Binary search for sorted data

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

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

Arrays.sort(products, Comparator.comparingInt(p -> p.productId));

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

while (left <= right) {

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

if (products[mid].productId == targetId) {

return products[mid];

}

else if (products[mid].productId < targetId) {

left = mid + 1;

} else

{

right = mid - 1;

}

}return null;

}

}

// Testing and output

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(202, "Shirt", "Fashion"),

new Product(303, "Book", "Stationery"),

new Product(404, "Phone", "Electronics")

};

Product linearResult = LinearSearch.search(products, 303);

System.out.println("Linear Search Result: " + (linearResult != null ? linearResult.productName : "Not Found"));

Product binaryResult = BinarySearch.search(products, 202);

System.out.println("Binary Search Result: " + (binaryResult != null ? binaryResult.productName : "Not Found"));

}

}

**OUTPUT:**



**4. ANALYSIS:**

Time complexities of linear and binary search are O(n) and O(log n) respectively.

Since this is an E- commerce website, it is better to go with Binary search as it is faster for large catalogs and linear search becomes inefficient with large data.

Exercise 7: Financial Forecasting

Scenario:

You are developing a financial forecasting tool that predicts future values based on past data.

Steps:

1. Understand Recursive Algorithms:

o Explain the concept of recursion and how it can simplify certain problems.

2. Setup:

o Create a method to calculate the future value using a recursive approach.

3. Implementation:

o Implement a recursive algorithm to predict future values based on past growth rates.

4. Analysis:

o Discuss the time complexity of your recursive algorithm.

o Explain how to optimize the recursive solution to avoid excessive computation.

**SOLUTION:**

**1.** Recursionis technique used in computer science to solve big problems by breaking them into smaller, similar problems. The process in which a function calls itself directly or indirectly is called **recursion**and the corresponding function is called a recursive function.

This is can be used for **problems that can be divided into identical smaller tasks**.

Eg: Factorial and Fibonacci

**2. CODE:** setup and implementation ---

**FORMULA USED : (FutureValue = CurrentValue × (1 + GrowthRate)^Years)**

public class FinancialForecaster {

public static double calculateFutureValue(

double currentValue,

double growthRate,

int years

) {

if (years <= 0) {

return currentValue;

}

double nextValue = currentValue \* (1 + growthRate);

return calculateFutureValue(nextValue, growthRate, years - 1);

}

public static void main(String[] args) {

double initialInvestment = 1000.0;

double annualGrowthRate = 0.05;

int investmentYears = 10;

double futureValue = calculateFutureValue(

initialInvestment,

annualGrowthRate,

investmentYears

);

System.out.printf("Future value after %d years: $%.2f",

investmentYears, futureValue);

}

}

**OUTPUT:**



**4. ANALYSIS:**

Time complexity : O(n) for naïve recursion indicating O(1) as constant

For best optimized solutions:

Convert the recursive solution to Tail- recursive giving us a more optimized solution and thus avoiding excessive computations.

It also reduces the stack overflow risk.