Algorithms Data Structures Assignment

* (Superset ID - 6364580) Rishabh Shrivastav

Exercise – 2

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

**Ans - Big O notation** is used to describe the **time or space complexity** of an algorithm in terms of input size (n). It helps us understand how the algorithm performs as the input grows, especially in worst-case scenarios. This allows developers to compare algorithms and choose the most efficient one for their use case.

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

Ans - Best Case : The item is found at the first position (O(1) for linear, O(1) for binary).

Average Case : The item is somewhere in the middle of the data (O(n/2) → O(n) for linear).

Worst Case : The item is not found or at the last index (O(n) for linear, O(log n) for binary).

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

Ans - **Linear Search** checks each item one by one, which is useful for unsorted data. It has the time complexity of O(n).

**Binary Search** repeatedly divides the search range, which is useful for **sorted** data. It has the time complexity of O(logn).

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

Ans - For an e-commerce platform where **fast search is critical**, **Binary Search** is more suitable due to its **logarithmic time** performance — provided the product data is **sorted**.

**Code –**

import java.util.\*;

public class ECommerceSearch {

    public static void main(String[] args) {

        List<Product> products = new ArrayList<>();

        products.add(new Product(1, "Shoes", "Footwear"));

        products.add(new Product(2, "Laptop", "Electronics"));

        products.add(new Product(3, "Phone", "Electronics"));

        products.add(new Product(4, "Watch", "Accessories"));

        // Linear Search

        Product found = linearSearch(products, "Laptop");

        if (found != null)

            System.out.println("Linear Found: " + found.productName);

        else

            System.out.println("Linear: Not found");

        // Sorting the list

        products.sort(Comparator.comparing(p -> p.productName.toLowerCase()));

        // Binary Search

        Product foundBin = binarySearch(products, "Laptop");

        if (foundBin != null)

            System.out.println("Binary Found: " + foundBin.productName);

        else

            System.out.println("Binary: Not found");

    }

    static Product linearSearch(List<Product> products, String targetName) {

        for (Product product : products) {

            if (product.productName.equalsIgnoreCase(targetName)) {

                return product;

            }

        }

        return null;

    }

    static Product binarySearch(List<Product> products, String targetName) {

        int low = 0, high = products.size() - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            Product midProduct = products.get(mid);

            int compare = midProduct.productName.compareToIgnoreCase(targetName);

            if (compare == 0) return midProduct;

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

            else high = mid - 1;

        }

        return null;

    }

}

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;

    }

}

**Output –**

**A screenshot of a computer

AI-generated content may be incorrect.**

Exercise – 7

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

**Ans -** Recursion is a technique where a function calls itself to solve smaller instances of the same problem. e.g., factorial, Fibonacci, tree traversal, financial compounding. It simplifies problems that have a natural repetitive or divide-and-conquer structure.

Q. Discuss the time complexity of your recursive algorithm

Ans - The time complexity of the recursive approach is O(n), where n is the number of years. Each year leads to one recursive call

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

Ans – For optimizing the recursive solution we have 2 ways :

1. Memoization (store results of previous calls to avoid recomputation)
2. Convert to Iterative (use a loop instead of recursion to save stack space)

**Code –**

public class FinancialForecasting {

    public static void main(String[] args) {

        double initialAmount = 1000.0;

        double growthRate = 0.15; // taking 15%

        int years = 5;

        double futureValue = calculateFutureValue(initialAmount, growthRate, years);

        System.out.printf("Predicted value after %d years: %.2f\n", years, futureValue);

    }

    // Recursive method to calculate future value

    static double calculateFutureValue(double amount, double rate, int years) {

        if (years == 0) return amount;

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

    }

}

**Output –**

**A screenshot of a computer

AI-generated content may be incorrect.**

Name – Rishabh Shrivastav

Roll Number – 22053972

Superset ID - 6364580

College – Kalinga Institute of Industrial Technology (KIIT), Bhubaneswar