**Algorithms & Data Structure**

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

**Q1. What is Big O notation and how it helps in analyzing algorithms?**

**Ans –** It describes the upper bound of an algorithms running time as input size grows. It also helps in comparing algorithm efficiency(not depending on hardwares).

**Q2. What is best, average, and worst case scenarios for search operations ?**

**Ans –** Best Case: It means the element is found in the first place.

Average Case: It means the element is found somewhere in between the array.

Worst Case: It means the element is found at last place or after most number of the steps.

**Solution:**

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;

    }

    public String getProductName() {

        return productName;

    }

}

class algo1 {

    // Linear Search

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

        int steps = 0;

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

            steps++;

            if (products[i].getProductName().equalsIgnoreCase(target)) {

                System.out.println("Linear Search (Steps taken): " + steps);

                return i;

            }

        }

        System.out.println("Linear Search (Steps taken): " + steps);

        return -1;

    }

    // Binary Search

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

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

        int steps = 0;

        while (left <= right) {

            steps++;

            int mid = (left + right) / 2;

            int comparison = products[mid].getProductName().compareToIgnoreCase(target);

            if (comparison == 0) {

                System.out.println("Binary Search (Steps taken): " + steps);

                return mid;

            } else if (comparison < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        System.out.println("Binary Search (Steps taken): " + steps);

        return -1;

    }

    public static void main(String[] args) {

        Product[] productList = {

            new Product(1, "Apple", "Fruit"),

            new Product(2, "Banana", "Fruit"),

            new Product(3, "Carrot", "Vegetable"),

            new Product(4, "Dates", "Dry fruit"),

            new Product(5, "eggplant", "Vegetable")

        };

        //sorting

        java.util.Arrays.sort(productList, (a, b) -> a.getProductName().compareToIgnoreCase(b.getProductName()));

        String target = "dates";

        System.out.println("Searching for: " + target);

        int result1 = linearSearch(productList, target);

        int result2 = binarySearch(productList, target);

        if (result1 != -1) {

            System.out.println("Found at index (Linear): " + result1);

            System.out.println("Found at index (Binary): " + result2);

        } else {

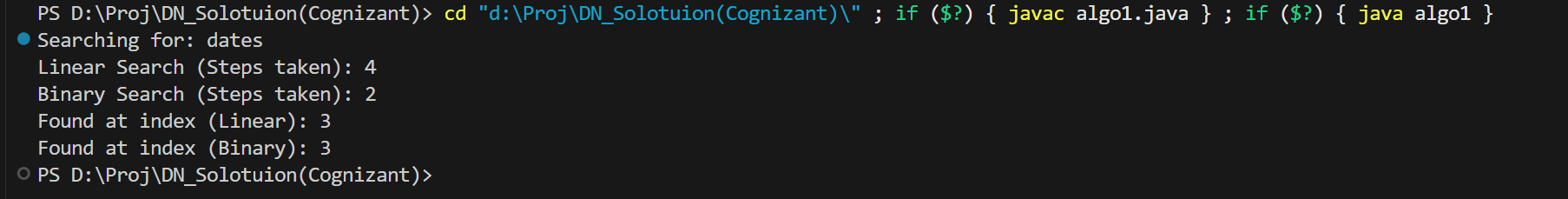
            System.out.println("Product not found.");

        }

    }

}

**Output:**

****

**Q3. Compare the time complexity of linear and binary search algorithms.**

**Ans –** Linear Search(worst case) – O(n)

Binary Search(worst case) – O(log n)

**Q4. What algorithm is more suitable for your platform and why ?**

**Ans –** Binary Search is more suitable for this platform as it have faster speed and have better time complexity.

**Exercise 7: Financial Forecasting**

**Solution:**

class algo2 {

    //Recursive function

    public static double forecastValue(int years, double base, double rate) {

        if (years == 0) {

            return base;

        }

        return (1 + rate) \* forecastValue(years - 1, base, rate);

    }

    public static void main(String[] args) {

        double baseValue = 1000.0;

        double growthRate = 0.10;

        int years = 5;

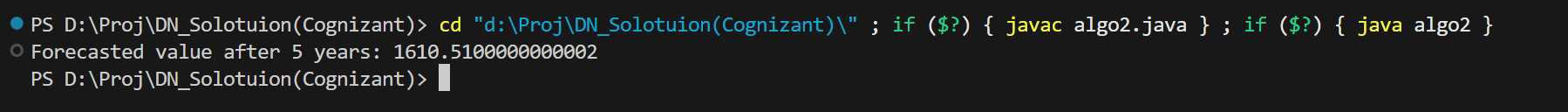
        double futureValue = forecastValue(years, baseValue, growthRate);

        System.out.println("Forecasted value after " + years + " years: " + futureValue);

    }

}

**Output:**

****

**Analysis:**

**Q1. Explain the concept of recursion and how it can simplify certain problems ?**

**Ans -** Recursion is defined as a process which calls itself to solve smaller subproblems.

It simplifies the problems by dividing it into smaller problems and than solving the smaller problems.

**Q2. Discuss the time complexity of your recursive algorithm.**

**Ans -** Time complexity is : O(n)

**Q3. Explain how to optimize the recursive solution to avoid excessive computation.**

**Ans –** We can use memorization to solve almost all the recursive problem , which reduces the time complexity but increases the space complexity(space-time trade off).