Week \_01

**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.

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

**Big O Notation** is a mathematical concept used in computer science to describe the **efficiency of algorithms** in terms of time and space complexity. Big O notation describes the **upper bound** of an algorithm’s growth rate. It focuses on the **worst-case scenario**, which helps developers understand the **maximum time or space** the algorithm might require

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

**Best Case:** where the desired element is found immediately.

In a linear search, the target element is the first element in the list

O(1) -> is the time complexity

**Average Case:** The performance expected over all possible inputs, assuming each case is equally likely.

In linear search, the element is somewhere in the middle of the list.

O(n) -> linear search, O(log n) -> binary search

**Worst Case:** The scenario where the search operation takes the **longest time** to complete.

The element is not present in the list (or)The element is last in the list.

O(n) -> linear search.

O(log n) -> binary search (still efficient).

**Ecommerce.java:**

import java.util.\*;

class Product{

    int pId;

    String PName;

    String PCategory;

    Product(int pId,String PName,String PCategory){

        this.pId=pId;

        this.PName=PName;

        this.PCategory=PCategory;

}

    public String toString(){

       return("PId : "+pId+", " + "PName: " + PName+", "+"PCategory: "+PCategory);

    }

}

class Search{

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

        for(int i=0;i<p.length;i++){

            if(p[i].PName.equalsIgnoreCase(name)){

                return p[i];

            }

        }

       return null;

    }

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

        int left =0;

        int right = p.length-1;

        while(left<=right){

            int mid = (left+right)/2;

            int count = p[mid].PName.compareToIgnoreCase(name);

            if(count==0){

                return p[mid];

            }

            else if(count<0){

                left = mid+1;

            }

            else{

                right = mid-1;

            }

        }

        return null;

    }

}

public class Ecommerce{

public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);

    Product[] p = {

            new Product(1, "Jeans", "Clothing"),

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

            new Product(3, "Journals", "Books"),

            new Product(4, "FaceWash", "SkinCare"),

            new Product(5, "Slippers", "Footwear"),

            new Product(6, "Shampoo", "HairCare"),

            new Product(7, "HandBags", "Bags"),

            new Product(8, "Pens", "Stationary"),

       };

       System.out.println("Enter the Product to Search: ");

       String ProName = sc.nextLine();

       Product resLinear = Search.linearSearch(p, ProName);

       if(resLinear != null){

         System.out.println("Product Found!"+resLinear +"With the help of LinearSearch");

       }

       else{

         System.out.println("Product Not Found! With the help of LinearSearch");

       }

       Arrays.sort(p, Comparator.comparing(product -> product.PName.toLowerCase()));

      Product resBinary = Search.binarySearch(p, ProName);

       if(resBinary != null){

         System.out.println("Product Found!"+resLinear +"With the help of BinarySearch");

       }

       else{

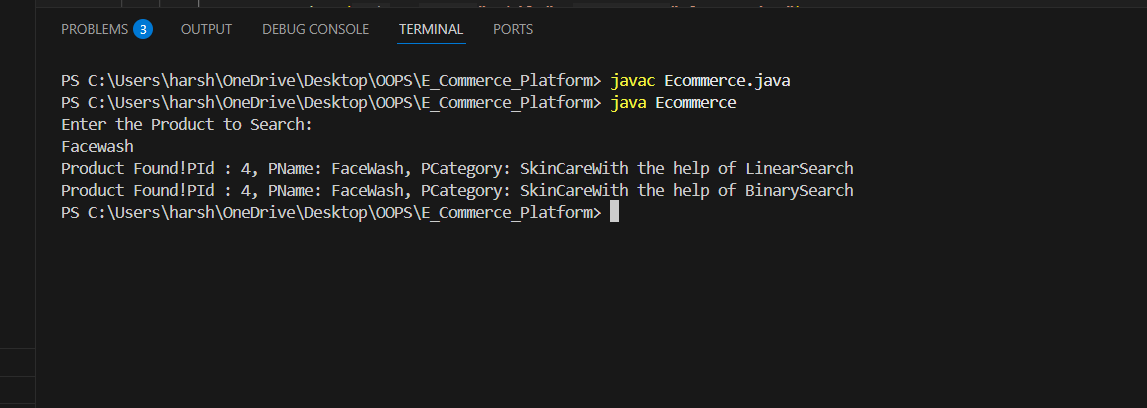
         System.out.println("Product Not Found! With the help of LinearSearch");

       }

}

}

**Output:**



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

Linear Search:

Sequentially checks each element in the list from start to end.

**Best Case**: O(1) – when the element is at the first position.

**Average Case**: O(n) – when the element is in the middle.

**Worst Case**: O(n) – when the element is at the end or not present.

Binary Search:

Repeatedly divides the **sorted list** into halves to find the target.

**Best Case**: O(1) – if the middle element is the target.

**Average Case**: O(log n) – halves the search space every step.

**Worst Case**: O(log n) – logarithmic time in the size of input.

• Discuss which algorithm is more suitable for your platform and why?

**Linear Search is More Suitable When:**

The data is **small or unsorted**.

Real-time sorting is not feasible.

**Binary Search is More Suitable When:**

The dataset is large and pre-sorted.

Memory and time efficiency are important.

**Exercise 7: Financial Forecasting**

**Scenario:**

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

ForeCasting.java:

import java.util.Scanner;

public class Forecasting {

    public static double future(double amount, double rate, int years) {

        if (years == 0) {

            return amount;

        }

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

    }

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter amount:");

        double amount = sc.nextDouble();

        System.out.println("Enter rate:");

        double rate = sc.nextDouble();

        double growrate = rate / 100;

        System.out.println("Enter years:");

        int years = sc.nextInt();

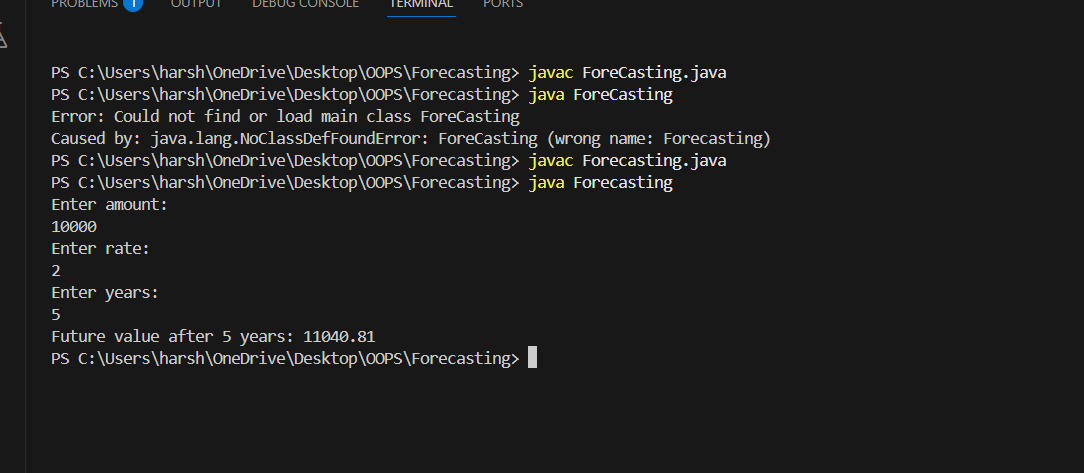
double futureres = future(amount, growrate, years);

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

    }

}

**Output:**

****