**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:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**Solution:**

**Steps:**

1. **Understand Asymptotic Notation:**

* **Big O Notation -** Big O Notation is a powerful tool to measure the time and space complexity of an algorithm. It describes the upper bound of the complexity in the worst-case scenario. 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.
* **Importance of Big O -** Big O Notation is important because it helps to analyze the efficiency of the algorithms.

It provides a way to describe how the runtime or space requirements of an algorithm grow as the input size increases.

It also allows the programmer to compare different algorithms and decide the most efficient and appropriate one.

* **Best, Average, and Worst-case scenarios for search operations-**

|  |  |  |  |
| --- | --- | --- | --- |
| **Search Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| Linear Search | O(1) (if found early) | O(n) | O(n) |
| Binary Search | O(1) (if mid match) | O(log n) | O(log n) |

1. **Setup:**

***// Product.java***

package com.ecommerce;

public class Product {

    private int productId;

    private String productName;

    private String category;

    Product() {

        productId=0;

        productName=null;

        category=null;

    }

    public Product(int productId, String productName, String category) {

        super();

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    public int getProductId() {

        return productId;

    }

    public void setProductId(int productId) {

        this.productId = productId;

    }

    public String getProductName() {

        return productName;

    }

    public void setProductName(String productName) {

        this.productName = productName;

    }

    public String getCategory() {

        return category;

    }

    public void setCategory(String category) {

        this.category = category;

    }

    @Override

    public String toString() {

        return "Product [productId=" + productId + ", productName=" + productName + ", category=" + category + "]";

    }

}

1. **Implementation:**

***// SearchFunction.java***

package com.ecommerce;

public interface SearchFunction {

    public Product Search(Product products[], int sproductId);

}

***// LinearSearchFunction.java***

public class LinearSearchFunction implements SearchFunction {

    @Override

    public Product Search(Product[] products, int sproductId) {

        for(Product prd : products) {

            if(prd.getProductId() == sproductId) {

                return prd;

}

        }

        return null;

    }

}

***// BinarySearchFunction.java***

package com.ecommerce;

public class BinarySearchFunction implements SearchFunction {

    @Override

    public Product Search(Product[] products, int sproductId) {

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

        while (low <= high) {

            int mid = (low + high) / 2;

            int midId = products[mid].getProductId();

            if (midId == sproductId) {

                return products[mid];

            } else if (midId < sproductId) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

    public void SortByProductId(Product[] products) {

        int n = products.length;

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - i - 1; j++) {

                if (products[j].getProductId() > products[j + 1].getProductId()) {

*// Swap products[j] and products[j + 1]*

                    Product temp = products[j];

                    products[j] = products[j + 1];

                    products[j + 1] = temp;

                }

            }

        }

    }

}

***// EcomMain.java***

package com.ecommerce;

import java.util.\*;

public class EcomMain {

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(2, "Shirt", "Clothing"),

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

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

            new Product(5, "Eye Liner", "Cosmetics")

        };

        Scanner sc = new Scanner(System.in);

*// Linear Search (No need to sort)*

        System.out.println("Enter the productid to be searched: ");

        int id = sc.nextInt();

        System.out.println("Applying Linear Search Technique");

        LinearSearchFunction ls = new LinearSearchFunction();

        Product p1 = ls.Search(products, id);

        if(p1!=null)

        {

            System.out.println("The Product is found.");

            System.out.println("The Product Details are: "+p1);

        }

        else

        {

            System.out.println("The Product is Not Found");

        }

        System.out.println("Enter the productid to be searched: ");

        int id1 = sc.nextInt();

        System.out.println("Applying Binary Search Technique");

*// Sort for Binary Search*

        BinarySearchFunction bs = new BinarySearchFunction();

        bs.SortByProductId(products);

*// Binary Search*

        Product p2 = bs.Search(products, id1);

        if(p2!=null)

        {

            System.out.println("The Product is found.");

            System.out.println("The Product Details are: "+p2);

        }

        else

        {

            System.out.println("The Product is Not Found");

        }

        sc.close();

    }

}

1. **Understand Asymptotic Notation:**

* **Time Complexity Comparison-**

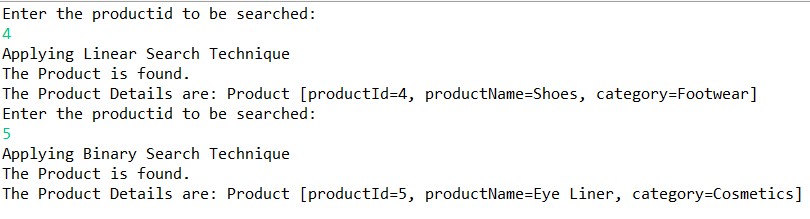
|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | **Sorted Required?** | **Suitable for Large Datasets?** |
| Linear Search | O(n) | No | No, slow for large arrays |
| Binary Search | O(log n) | Yes | Ideal for large, sorted arrays |

* Binary Search is much faster than linear search for large datasets if the data is sorted.

For static or rarely changing product lists using binary search is better.

For dynamic/unsorted data or small datasets using linear search is better.

* **Output:**



**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:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**Solution:**

**Steps:**

1. **Understand Recursive Algorithms:**

* Recursion is a programming technique where a function calls itself repetitively based on a base condition.
* It can significantly simplify certain problems by making the code more readable and concise, especially when dealing with self-similar structures or repetitive tasks.

1. **Setup: &** 3. **Implementation:**

***// FinancialForecast.java***

package com.FinForecast;

public class FinancialForecast {

    private double initialamt;

    private double growthrate;

    private int years;

    FinancialForecast() {

        initialamt=0.0;

        growthrate=0.0;

        years=0;

    }

    public FinancialForecast(double initialamt, double growthrate, int years) {

        super();

        this.initialamt = initialamt;

        this.growthrate = growthrate;

        this.years = years;

    }

    public double getInitialamt() {

        return initialamt;

    }

    public void setInitialamt(double initialamt) {

        this.initialamt = initialamt;

    }

    public double getGrowthrate() {

        return growthrate;

    }

    public void setGrowthrate(double growthrate) {

        this.growthrate = growthrate;

    }

    public int getYears() {

        return years;

    }

    public void setYears(int years) {

        this.years = years;

    }

    @Override

    public String toString() {

  return "FinancialForecast [initialamt=" + initialamt + ", growthrate=" + growthrate + ", years=" + years + "]";

    }

}

***// Forecasting.java***

public class Forecasting {

public static double predictFutureValue(double initialAmount, double growthRate, int years) {

        if (years == 0) {

            return initialAmount;

        }

else {

return predictFutureValue(initialAmount, growthRate, years - 1) \* (1 + growthRate);

        }

    }

}

***// FinancialForecastingMain.java***

package com.FinForecast;

import java.util.\*;

public class FinancialForecastingMain {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the Initial Amount: ");

        double initialAmt = sc.nextDouble();

        System.out.println("Enter the Growth Rate: ");

        double growthRate = sc.nextDouble();

        System.out.println("Enter the Years: ");

        int years = sc.nextInt();

*//Displaying the Values*

        FinancialForecast fp = new FinancialForecast(initialAmt, growthRate, years);

        fp.toString();

*//Predicting the Future Value*

  double finalAmt = Forecasting.predictFutureValue(initialAmt, growthRate, years);

        System.out.println("The Future Predicted Amount is " + finalAmt);

        sc.close();

    }

}

4. **Analysis:**

|  |  |  |
| --- | --- | --- |
| **Method** | **Time Complexity** | **Explanation** |
| Basic Recursion | O(n) | One call per year |

* **Optimization of Recursion can be done by the following ways -**
* Memoization-Reduces redundant computations
* Tail Recursion-Potential compiler optimization
* Iterative Conversion-Best performance
* **Output:**

