Data Structures and Algorithm

Exercise 2:

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: 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: Compare the time complexity of linear and binary search algorithms. o Discuss which algorithm is more suitable for your platform and why.

| **Time Complexity** | **Description** |
| --- | --- |
| O(1) | Constant time |
| O(log n) | Logarithmic time |
| O(n) | Linear time |
| O(n log n) | Linearithmic time |
| O(n²) | Quadratic time |

**Search Scenarios:**

* **Best Case:** Fastest possible scenario (e.g., first element matched).
* **Average Case:** Expected time over all possible inputs.
* **Worst Case:** Slowest possible scenario (e.g., last element or not found).

Define Product Class

public 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 toString() {

return productId + " - " + productName + " (" + category + ")";

}

}

Implementation - Linear & Binary Search

import java.util.Arrays;

import java.util.Comparator;

public class SearchService {

// Linear Search: O(n)

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

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

// Binary Search: O(log n)

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

int left = 0;

int 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;

}

// Utility method to sort by productId

public static void sortProducts(Product[] products) {

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

}

}

Analysis and Testing

public class TestSearch {

public static void main(String[] args) {

Product[] products = {

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

new Product(205, "Shirt", "Apparel"),

new Product(150, "Laptop", "Electronics"),

new Product(110, "Shoes", "Footwear")

};

// Linear Search Test

Product result1 = SearchService.linearSearch(products, 150);

System.out.println("Linear Search Result: " + result1);

// Sort products before binary search

SearchService.sortProducts(products);

// Binary Search Test

Product result2 = SearchService.binarySearch(products, 150);

System.out.println("Binary Search Result: " + result2);

}

}

Exercise 7:

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

Recursive Forecast Implementation

public class FinancialForecaster {

public static double forecastValue(int n, double initialValue, double growthRate) {

if (n == 0) {

return initialValue;

}

return forecastValue(n - 1, initialValue, growthRate) \* (1 + growthRate);

}

public static void main(String[] args) {

double initialValue = 10000.0;

double growthRate = 0.05;

int years = 5;

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

System.out.printf("Forecasted Value after %d years: ₹%.2f\n", years, futureValue);

}

}

**Output:**

**Forecasted Value after 5 years: ₹12762.82**

**Time Complexity**

* **Recursive Time Complexity:** O(n)  
  Each call computes one year, making it linear with respect to n.