**Data Structures and Algorithms**

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

import java.util.\*;

class Product {

int productId;

String productName;

String category;

Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

public String toString() {

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

}

}

public class ProductSearch {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Product[] products = {

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

new Product(2, "Shoes", "Fashion"),

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

new Product(4, "Book", "Education"),

new Product(5, "Watch", "Fashion")

};

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

System.out.print("Enter product name to search: ");

String searchName = sc.next();

System.out.println("Using Linear Search:");

boolean found = false;

for (Product p : products) {

if (p.productName.equalsIgnoreCase(searchName)) {

System.out.println(p);

found = true;

}

}

if (!found) System.out.println("Not Found");

System.out.println("Using Binary Search:");

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

found = false;

while (low <= high) {

int mid = (low + high) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(searchName);

if (cmp == 0) {

System.out.println(products[mid]);

found = true;

break;

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

else high = mid - 1;

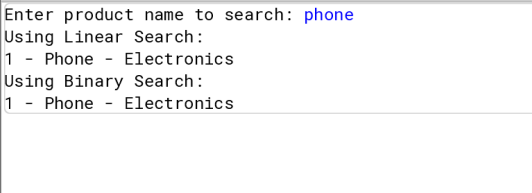
}

if (!found) System.out.println("Not Found");

}

}

**OUTPUT :**



**Analysis**

**Time Complexity Comparison**

* **Linear Search**:
  + Time Complexity: **O(n)**
  + It checks each product one by one.
  + Slower when there are many products.
* **Binary Search**:
  + Time Complexity: **O(log n)**
  + It searches by dividing the sorted product list in half repeatedly.
  + Much faster than linear search.

**Best Algorithm:**

* Use **Binary Search** for large, sorted datasets.
* Use **Linear Search** for small/unsorted data or infrequent searches

**Exercise 7: Financial Forecasting**

import java.util.Scanner;

public class FinancialForecasting {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter initial amount: ");

double amount = sc.nextDouble();

System.out.print("Enter growth rate (%): ");

double rate = sc.nextDouble();

System.out.print("Enter number of years: ");

int years = sc.nextInt();

double futureValue = forecast(amount, rate, years);

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

}

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

if (years == 0)

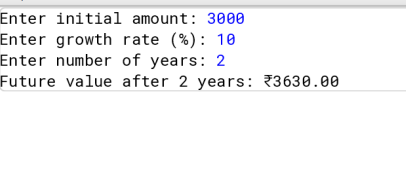
return amount;

return forecast(amount \* (1 + rate / 100), rate, years - 1);

}

}

**OUTPUT:**

****

**Analysis**

**Time Complexity:**

* Recursive solution: Often exponential, e.g., O(2^n) for naive Fibonacci.
* Optimized via memoization or converting to iterative: O(n)

**Optimization:**

* Use Dynamic Programming to cache results.
* Avoids redundant computations in recursive calls.