2. E-COMMERCE PLATFORM SEARCH FUNCTION

Big O Notation:

* Big O notation is a representation used to **describe the performance (time or space complexity)** of an algorithm for the input growing size of input.
* This gives details about the **time and space complexity** of the algorithm used for the solution.

CODE:

import java.util.Arrays;

import java.util.Comparator;

//class definition

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 class Main {

// Linear Search

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

for (Product p : products) {

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

return p;

}

}

return null;

}

// Binary Search after sorting by productNames

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

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

while (left <= right) {

int mid = (left + right) / 2;

int compare = products[mid].productName.compareToIgnoreCase(target);

if (compare == 0) return products[mid];

else if (compare < 0) left = mid + 1;

else right = mid - 1;

}

return null;

}

public static void main(String[] args) {

//input array of products

Product[] products = {

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

new Product(102, "Shoes", "Fashion"),

new Product(103, "Mobilephone", "Electronics"),

new Product(104, "Books", "Stationery"),

new Product(105, "Tablet", "Electronics")

};

//target for Linear Search

String prdctoflinear = "Mobilephone";

//target for Binary Search

String prdctofbinary = "Books";

//result of Linear Search

Product result1 = *linearSearch*(products, prdctoflinear);

if (result1 != null)

System.***out***.println("Linear Search: Found " + result1.productName + " in " + result1.category);

else

System.***out***.println("Linear Search: Product not found");

//result of Binary Search

// Sort by name for Binary Search

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

//Search operation

Product result2 = *binarySearch*(products,prdctofbinary);

if (result2 != null)

System.***out***.println("Binary Search: Found " + result2.productName + " in " + result2.category);

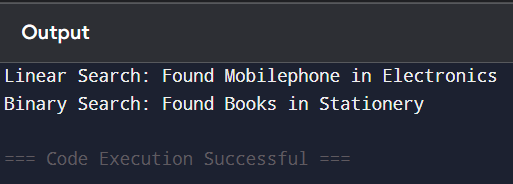
else

System.***out***.println("Binary Search: Product not found");

}

}

OUTPUT:



* Binary Search has a time complexity of **O(log n)**, as it repeatedly divides the sorted array in half to find the target.
* **Binary Search** is more suitable for an e-commerce platform because it offers faster search performance (**O(log n)**) on large datasets. However, it requires the product list to be **sorted**. If the data is unsorted or very dynamic, **Linear Search** may be used temporarily, but **Binary Search** is ideal for optimal efficiency in production.

7. FINANCIAL FORECASTING

Recursion:

Recursion is a technique where a function **calls itself** to solve a problem, until it reaches a **base case**. It breaks the problem into smaller sub-problems until it’s simple enough to solve.

CODE:

public class Main {

//forecast using Recursive method

public static double Recursiveforecast(double currentValue, double growthRate, int years) {

if (years == 0) return currentValue; // base case

//recursion execution

return *Recursiveforecast*(currentValue, growthRate, years - 1) \* (1 + growthRate);

}

//main method

public static void main(String[] args) {

double currentValue = 15000; // rupees 15,000 as an initial amnt

double growthRate = 0.15; // assuming 15% annual growth

int years = 10;

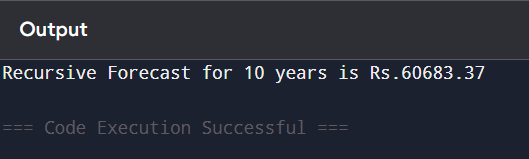
double futureVal = *Recursiveforecast*(currentValue, growthRate, years);

System.***out***.printf("Recursive Forecast for %d years is Rs.%.2f\n", years, futureVal);

}

}

OUTPUT:



* **To optimize the recursive solution and avoid excessive computation, we can use memoization.**  
  This involves storing the results of previous computations in an array or map so that repeated calls with the same inputs do not recompute values.