# Exercise 2: E-commerce Platform Search Function Scenario:

1. **Understand Asymptotic Notation:**

Big O notation is used to describe the performance of an algorithm in terms of time or space complexity as the input size increases. It gives an upper bound on the growth rate, helping to understand how the algorithm scales.

Common examples include:

* O(1): Constant time – the operation takes the same time regardless of input size.
* O(n): Linear time – performance grows linearly with input size.
* O(log n): Logarithmic time – performance grows slowly even as input increases.
* O(n²): Quadratic time – performance decreases rapidly as input grows.
* Linear Search:
  + Best case: O(1) – when the target is the first element.
  + Average case: O(n) – when the target is somewhere in the middle.
  + Worst case: O(n) – when the target is at the end or not found.
* Binary Search:
  + Best case: O(1) – when the target is at the middle position.
  + Average and worst case: O(log n) – the search space is halved at each step.

# Implementation

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;

}

}

class SearchFunctions {

// Linear Search

public static int linearSearch(Product[] products, String targetName) { for (int i = 0; i < products.length; i++) {

if (products[i].productName.equalsIgnoreCase(targetName)) { return i; // product found

}

}

return -1; // product not found

}

// Binary Search (array must be sorted by productName)

public static int binarySearch(Product[] products, String targetName) { int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2; int cmp =

products[mid].productName.compareToIgnoreCase(targetName); if (cmp == 0)

return mid; // product found else if (cmp < 0)

left = mid + 1; else

right = mid - 1;

}

return -1; // product not found

}

// Example main method to test both search functions

public static void main(String[] args) { Product[] products = {

new Product(101, "Laptop", "Electronics"), new Product(102, "Phone", "Electronics"), new Product(103, "Book", "Stationery"), new Product(104, "Pen", "Stationery")

};

// For binary search, make sure array is sorted by productName java.util.Arrays.sort(products, (a, b) ->

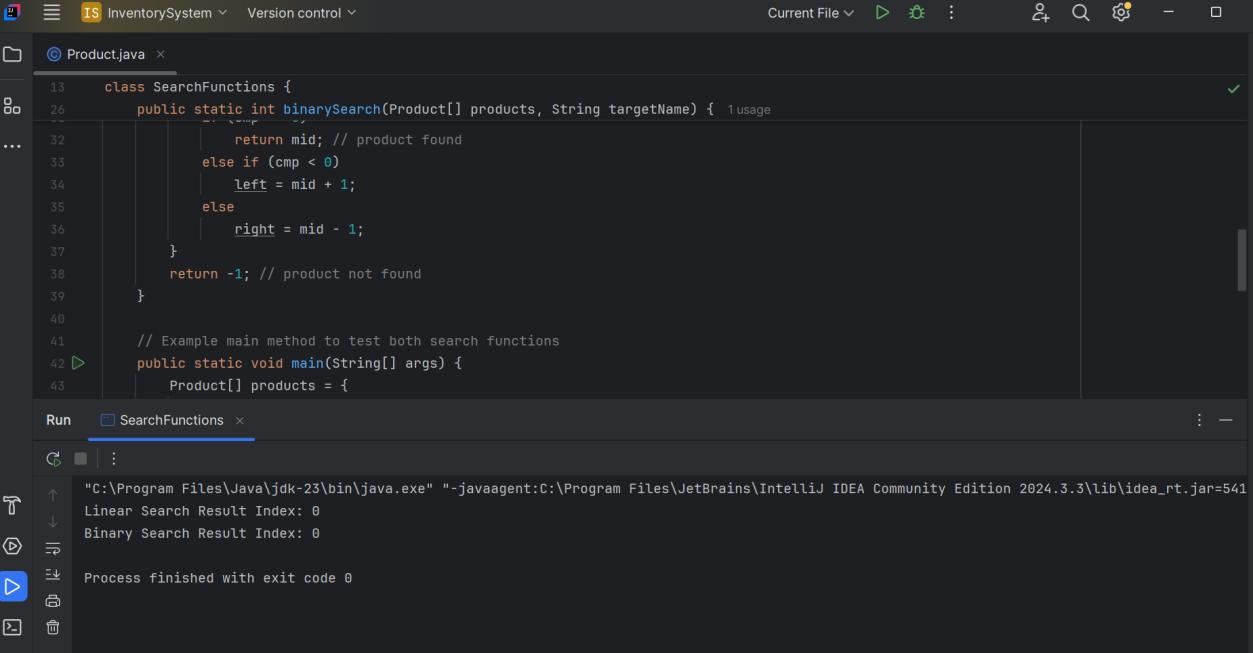
a.productName.compareToIgnoreCase(b.productName));

String searchTarget = "Book";

int linearResult = linearSearch(products, searchTarget); System.out.println("Linear Search Result Index: " + linearResult);

int binaryResult = binarySearch(products, searchTarget); System.out.println("Binary Search Result Index: " + binaryResult);

}

}

1. **Analysis:**
2. Linear Search
   * Time Complexity:
     + Best Case: O(1) → Target is at the beginning.
     + Worst Case: O(n) → Target is at the end or not present.
   * Space Complexity: O(1) → No extra space used.
   * Use Case: Suitable for unsorted arrays or when the dataset is small and simplicity is preferred.
3. Binary Search
   * Time Complexity:
     + Best Case: O(1) → Target is at the middle.
     + Worst Case: O(log n) → Repeated halving of search space.
   * Space Complexity: O(1) → Iterative version uses no extra space.
   * Use Case: Efficient for sorted arrays with large datasets.

# Exercise 7: Financial Forecasting

1. **Understand Recursive Algorithms**

**Recursion** is a programming technique where a method calls itself to solve a smaller instance of the same problem.

It simplifies problems that can be divided into similar subproblems — like:

* + Calculating factorial
  + Fibonacci numbers
  + Predicting future values in time-series

# Implementation

public class FinancialForecast {

public static double futureValue(double principal, double rate, int years) { if (years == 0) {

return principal;

}

return (1 + rate) \* futureValue(principal, rate, years - 1);

}

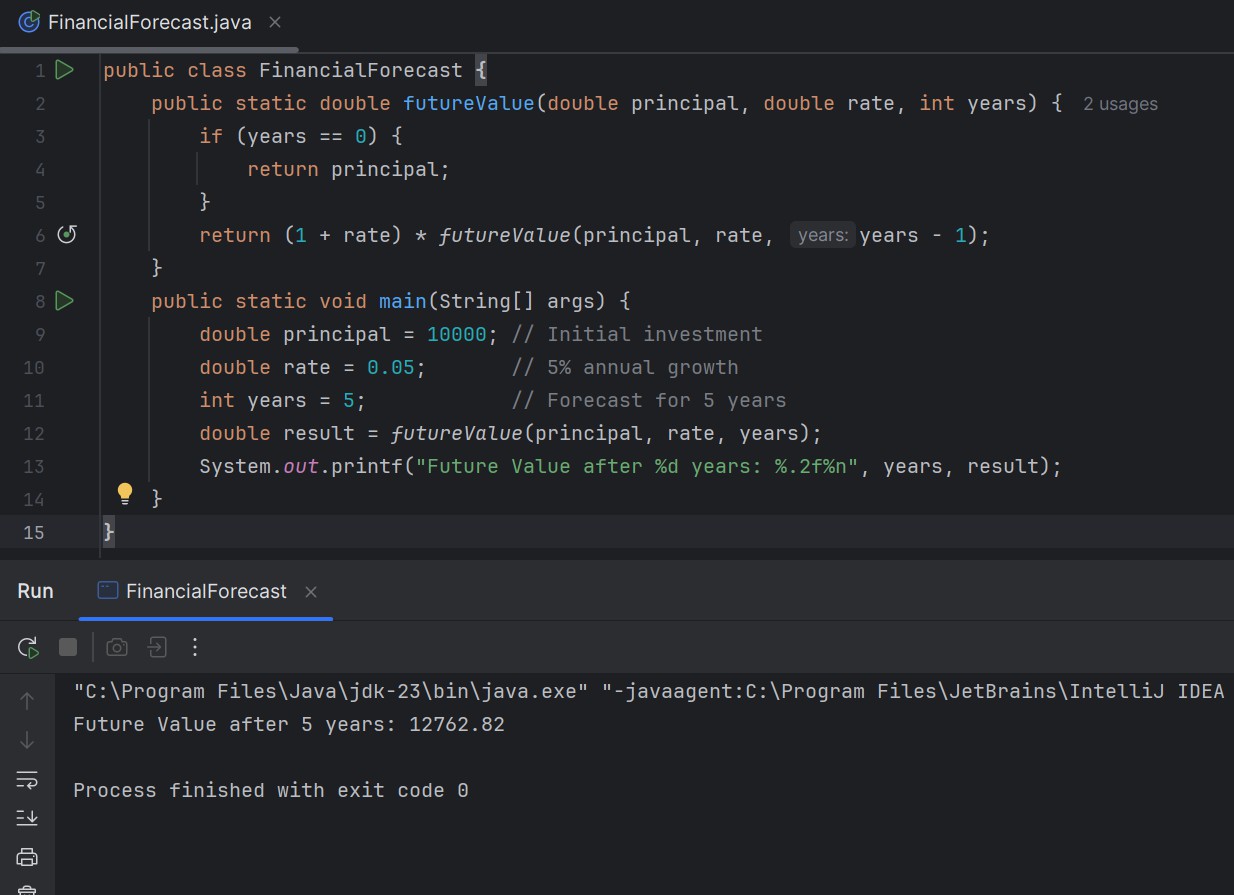
public static void main(String[] args) {

double principal = 10000; // Initial investment double rate = 0.05; // 5% annual growth int years = 5; // Forecast for 5 years

double result = futureValue(principal, rate, years); System.out.printf("Future Value after %d years: %.2f%n", years, result);

}

}



# Analysis

* + - Time Complexity: Time Complexity: O(n)
    - Space Complexity: O(n)