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

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| Linear | O(1) | O(n/2) ≈ O(n) | O(n) |
| Binary | O(1) | O(log n) | O(log n) |

**CODE:**

package ECommerceSearchFinal;

class Product {

int productId;

String productName;

String productCategory;

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

this.productId = id;

this.productName = name;

this.productCategory = cat;

}

public String toString() {

return "Product Id: " + this.productId + " | Product Name: " + this.productName + " | Product Category: " + this.productCategory;

}

}

// Sort class

class Sort {

public static Product[] sortById(Product[] p) {

int n = p.length;

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

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

if (p[j].productId > p[j + 1].productId) {

Product temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

return p;

}

}

// Binary Search

class Binary {

public static int bsById(Product[] a, int key) {

Product[] p = Sort.sortById(a); // Sort before binary search

int l = 0, h = p.length - 1;

while (l <= h) {

int mid = (l + h) / 2;

if (p[mid].productId == key) {

return mid;

} else if (p[mid].productId < key) {

l = mid + 1;

} else {

h = mid - 1;

}

}

return -1;

}

}

// Linear Search

class Linear {

public static int lsByName(Product[] p, String key) {

for (int i = 0; i < p.length; i++) {

if (p[i].productName.equalsIgnoreCase(key)) {

return i;

}

}

return -1;

}

}

// Main class

public class Main {

public static void main(String[] args) {

Product[] p = {

new Product(101, "Fan", "Home Appliances"),

new Product(102, "TV", "Home Appliances"),

new Product(103, "Chocolate", "Food"),

new Product(104, "Coco cola", "Beverages"),

new Product(105, "Bat", "Sports"),

new Product(106, "Chips", "Food")

};

// Linear Search

String keyName = "Bat";

int lsPos = Linear.lsByName(p, keyName);

if (lsPos != -1) {

System.out.println("Linear Search:\n" + p[lsPos] + "\nFound at index: " + lsPos);

} else {

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

}

// Binary Search

int keyId = 102;

int bsPos = Binary.bsById(p, keyId);

if (bsPos != -1) {

System.out.println("Binary Search:\n" + p[bsPos] + "\nFound at index: " + bsPos + "\n");

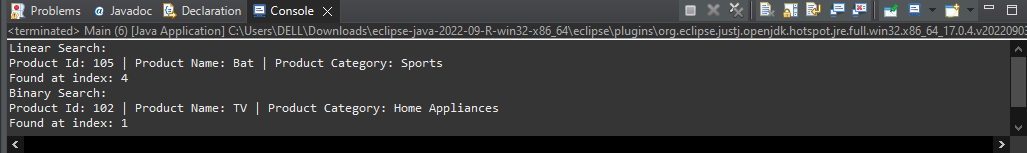
} else {

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

}

}

}



**Exercise 7: Financial Forecasting**

**CODE:**

public class Forecast {

public static double forecast(double[] r, int y, double currentValue) {

if (y == 0) return currentValue;

// Get previous year's value and apply this year's growth rate

double pre = forecast(r, y - 1, currentValue);

double growthRate = r[y - 1];

return pre + (pre \* growthRate);

}

public static void main(String[] args) {

// Initial amount

double n = 10000;

// Growth rate changes every year for 5 years

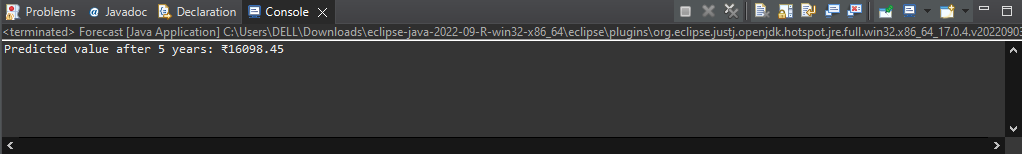
double[] yearly = {0.10, 0.08, 0.12, 0.09, 0.11}; // 5 years

double res = forecast(yearly, yearly.length,n);

System.out.printf("Predicted value after %d years: ₹%.2f\n", yearly.length, res);

}

}



## **Time Complexity Analysis**

* Each recursive call goes down one year.
* So total calls = n (number of years)
* Work per call = O(1)

**Time complexity = O(n)**