**DATA STRUCTURES AND ALGORITHMS (Mandatory hands-on)**

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

**CODE:**

import java.util.Arrays;

import java.util.Comparator;

public class ecommerce {

static 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 "id: " + productid + ", name: " + productname + ", category: " + category;

}

}

public static product linearsearch(product[] products, String name) {

for (product p : products) {

if (p.productname.equalsIgnoreCase(name)) {

return p;

}

}

return null;

}

public static product binarysearch(product[] products, String name) {

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

while (left <= right) {

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

int cmp = products[mid].productname.compareToIgnoreCase(name);

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

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

else right = mid - 1;

}

return null;

}

public static void main(String[] args) {

product[] products = {

new product(201, "laptop", "electronics"),

new product(202, "t-shirt", "clothing"),

new product(203, "book", "education"),

new product(204, "phone", "electronics"),

new product(205, "shoes", "footwear")

};

System.out.println("linear search for 'phone':");

product r1 = linearsearch(products, "phone");

System.out.println(r1 != null ? r1 : "product not found");

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

System.out.println("\nbinary search for 'phone':");

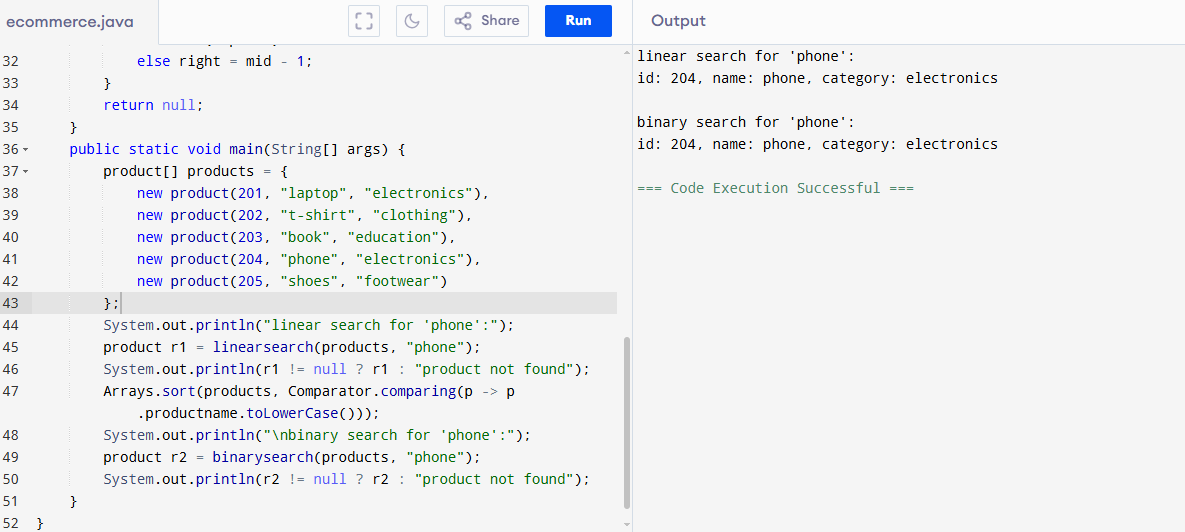
product r2 = binarysearch(products, "phone");

System.out.println(r2 != null ? r2 : "product not found");

}

}

**OUTPUT:**

****

**Exercise 7: Financial Forecasting**

**CODE:**

import java.util.HashMap;

public class financialforecast {

static class forecaster {

HashMap<Integer, Double> memo = new HashMap<>();

public double forecastrecursive(double current, double growthrate, int years) {

if (years == 0) return current;

return forecastrecursive(current \* (1 + growthrate), growthrate, years - 1);

}

public double forecastoptimized(double current, double growthrate, int years) {

if (years == 0) return current;

if (memo.containsKey(years)) return memo.get(years);

double result = forecastoptimized(current \* (1 + growthrate), growthrate, years - 1);

memo.put(years, result);

return result;

}

}

public static void main(String[] args) {

forecaster f = new forecaster();

double currentvalue = 1000.0;

double growthrate = 0.10;

int years = 5;

System.out.println("recursive forecast:");

double r1 = f.forecastrecursive(currentvalue, growthrate, years);

System.out.println("future value after " + years + " years: " + r1);

System.out.println("\noptimized forecast with memoization:");

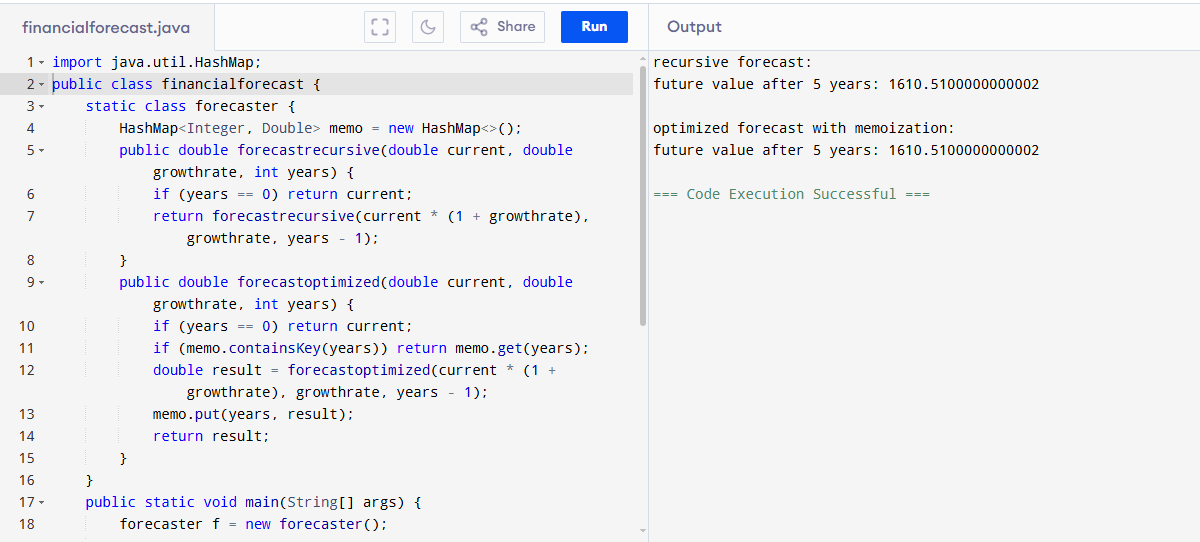
double r2 = f.forecastoptimized(currentvalue, growthrate, years);

System.out.println("future value after " + years + " years: " + r2);

}

}

**OUTPUT:**

****