Hands-on 1:

Data Structures and Algorithms

Exercise 2: E-commerce Platform Search Function :

Product.java

package EcomSerachFunction;

import java.util.Comparator;

import java.util.List;

import java.util.ArrayList;

import java.util.Collections;

public class Product {

String prductName;

String category;

int productId;

List<Product> listForLinearSearch;

List<Product>listForBinarySearch;

public Product(){

listForLinearSearch=new ArrayList<>();

listForBinarySearch=new ArrayList<>();

}

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

this.prductName=name;

this.category=category;

this.productId=id;

}

public void addProduct(Product product){

listForBinarySearch.add(product);

listForLinearSearch.add(product);

}

//Linear Search

public Product searchProductUsingLinearSearch(int id ){

for(Product product: listForLinearSearch){

if(product.productId==id){

return product;

}

}

return null;

}

//Binary Search

public Product searchProductUsingBinarySearch(int id){

Collections.sort(listForBinarySearch , Comparator.comparingInt(p->p.productId));

int start=0;

int end=listForBinarySearch.size()-1;

while(start<end){

int mid=start+(end-start)/2;

int mi Productid=listForBinarySearch.get(mid).productId;

if(midProductId==id){

return listForBinarySearch.get(mid);

}else if(midProductId<id){

start=mid+1;

}else if(midProductId>id){

end=mid-1;

}

}

return null;

}

//Printing Product

public static void printProduct(Product product){

if(product==null){

System.out.println("Product not found");

}

System.out.println("Product Id : "+product.productId);

System.out.println("Product Name : "+product.prductName);

System.out.println("Product Category : "+product.category);

}

}

Main.java

package EcomSerachFunction;

public class Main {

public static void main(String[]args){

Product store=new Product();

Product product1=new Product("Laptop" , "Electronics" , 1);

Product product2=new Product("Mobile Phone" , "Electronics" , 2);

Product product3=new Product("Charger" , "Electronics" , 3);

Product product4=new Product("Shoe" , "Fashion" , 10);

store.addProduct(product1);

store.addProduct(product2);

store.addProduct(product3);

store.addProduct(product4);

System.out.println("Product search using Linear search O(n) :");

Product product=store.searchProductUsingLinearSearch(1);

Product.printProduct(product);

System.out.println();

System.out.println("Product searched using Binary search O(logn) :");

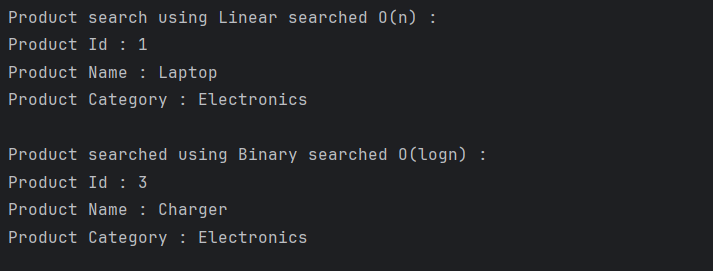
Product secondProduct=store.searchProductUsingBinarySearch(3);

Product.printProduct(secondProduct);

}

}

OUTPUT:



Analysis:

1. **Binary search is more efficient (O(log n)) than linear search (O(n))** for large datasets, making it ideal for fast product lookups on an e-commerce platform.
2. **Binary search requires sorted data**, so it’s best used when products are pre-sorted or rarely modified; otherwise, linear search is simpler for small or dynamic datasets.

Exercise 7: Financial Forecasting :

FinancialForecast.java

public class FinancialForecast {

public static double forecastValue(int year, double initialValue, double annualGrowthRate) {

if (year == 0) {

return initialValue;

}

return forecastValue(year - 1, initialValue, annualGrowthRate) \* (1 + annualGrowthRate);

}

public static void main(String[] args) {

double startingValue = 10000; // ₹10,000

double growthRate = 0.10; // 10% annual growth

int forecastYears = 5;

double result = forecastValue(forecastYears, startingValue, growthRate);

System.out.printf("Forecasted Value after %d years: Rs : %.2f\n", forecastYears, result);

}

}

OUTPUT:



Analysis :

1. **Time Complexity:** The recursive algorithm has a time complexity of **O(n)**, where n is the number of years, since it makes one recursive call per year.