**Data Structures and Algorithms**

**Exercise 2 – Ecommerce Platform Search Functions**

**Code:**

**1. Asymptotic Notations:**

Big O notation helps in showing the worst case time complexity of any algorithm

It can be used to compare different algorithms and choose the best algorithm for our practices.

**Linear Search Binary Search**

Best case O(1) O(1)

Average case O(n) O(log n)

Worst case O(n) O(log n)

**2. Product class with productID, productName, Category**

package EcommerceSearch;

public class Product{

public int ProductID;

public String productName;

public String Category;

Product(int ProductID, String productName, String Category)

{

this.ProductID=ProductID;

this.productName=productName;

this.Category=Category;

}

}

**3. Implementing Binary Search and Linear Search algorithms with Class**

**ProductSearch**

**Code:**

package EcommerceSearch;

import java.util.\*;

public class ProductSearch

{

public Product[] LinearSearch(Product[] products,String ProductName)

{ Product[] result=new Product[products.length];

int i=0;

for(Product p: products)

{

if(p.productName.toLowerCase().contains(ProductName.toLowerCase())||p.Category.toLowerCase().contains(ProductName.toLowerCase()))

{

result[i]=p ;

i++;

}

}

return result;

}

public Product BinarySearch(Product[] products,String ProductName)

{

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

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

while(left<=right)

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

Int cmp=ProductName.compareToIgnoreCase(products[mid].productName);

if(cmp == 0)

{

return products[mid];

}

else if(cmp<0){

right = mid-1;

}

else {

left= mid+1;

}

}

return null;

}

}

**Tested with a Main class**

package EcommerceSearch;

import java.util.Scanner;

public class Main{

public static void main(String [] args)

{

Product[] products = {new Product(1,"HP Pavilion 14","Laptop"),

new Product(2,"IQOO Neo 9 Pro","Mobile"),

new Product(3, "HP Pavilion 13","Laptop"),

new Product(4,"Dell vostro","Laptop"),

new Product(5,"Realme 6","Mobile")

};

Scanner s=new Scanner(System.*in*);

String name= s.nextLine();

ProductSearch search=new ProductSearch();

Product[] n= search.LinearSearch(products,name);

System.*out*.println("Linear Search:");

for(Product p : n)

{

if(p!=null)

{ System.*out*.println(p!=null? p.ProductID + "\n"+ p.productName +"\n" + p.Category + "\n"+" ":"Not Available");

}

}

Product k=search.BinarySearch(products,name);

System.*out*.println("Binary Search:");

System.*out*.println(k!=null?k.ProductID +"\n"+ k.productName +"\n"+ k.Category:"Not Available");

s.close();

}

}

**4. Analysis:**

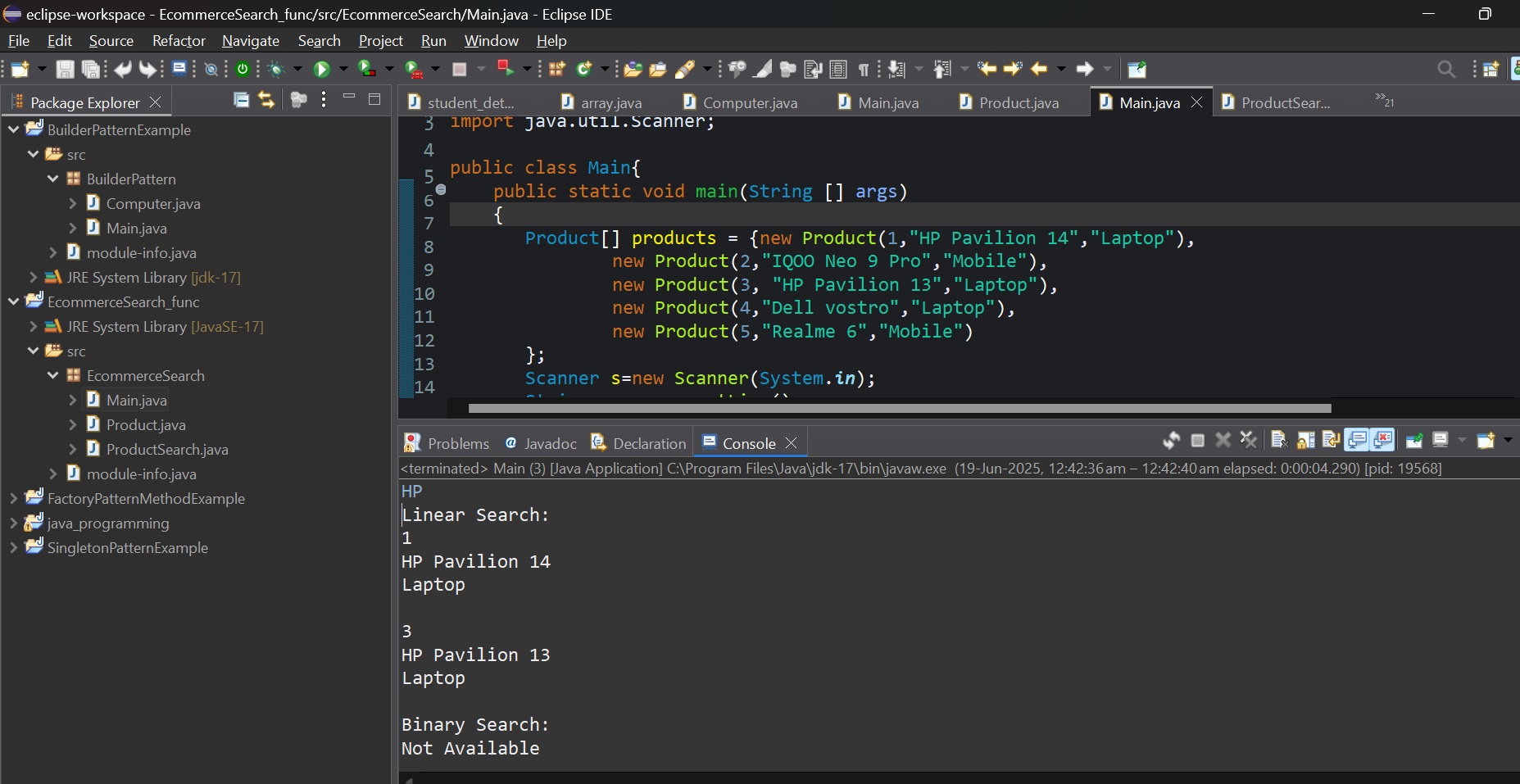
Linear Search time complexity is O(n) as the worst and average case which is greater than the time complexity of Binary Search which is O(log n) for both Worst and average cases.

In terms of time complexity, Binary search is best for searching any element than Linear Search

But, in E-commerce Platform Linear Search is the best algorithm to use cause, it can show all the matches related to the input which are present in the platform that to by searching in all atributes, where as binary search can only search for single element in either of the attributes.

The best algorithm for E-commerce Search Function is Linear Search.

**Output:** User Input is in Blue colour on the console.

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 7 – Financial Forecasting**

**1. Recurssion**

It is defined as a function calling itself.

It can simplify some iterative problems using iterative and base cases.

It is used to solve sub problems

**2. Created a method named as Forecast to calculate the future financial values using forecasting**

public double Forecast(double principle, double rate, int years)

{

if(years==0)

{

return principle;

}

return Forecast(principle,rate, years-1)\*(1+rate);

}

**3. Implemented the recursive algorithm that predicts future values depending on growing past values and also tested it in Main class**

package FinancialForecast;

public class FinancialForecast{

public double Forecast(double principle, double rate, int years)

{

if(years==0)

{

return principle;

}

return Forecast(principle,rate, years-1)\*(1+rate);

}

}

package FinancialForecast;

import java.util.Scanner;

public class Main{

public static void main(String [] args)

{

Scanner s=new Scanner(System.***in***);

double principle= s.nextDouble();

double rate = s.nextDouble();

int years = s.nextInt();

FinancialForecast predict= new FinancialForecast();

double k= predict.Forecast(principle, rate, years);

System.***out***.println(k);

s.close();

}

}

**4. Analysis**

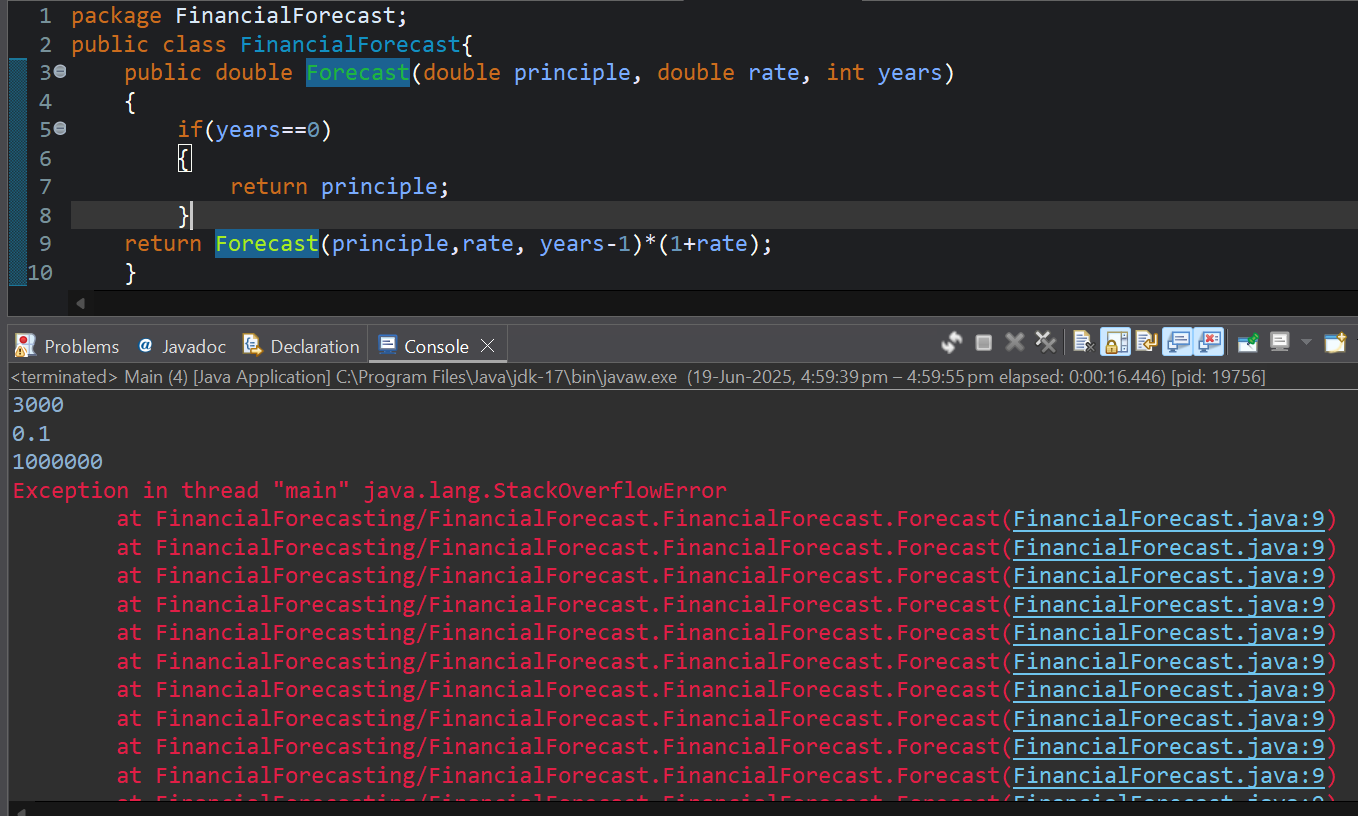
* Time complexity for my recursive algorithm is O(n) cause it recurses N times where N is the no of years.

**Output for Recursive algorithm:**

A screenshot of a computer program

AI-generated content may be incorrect.

* There is a disadvantage in using recursive algorithm cause when a large number of years is given as input, stack may overflow. So instead of recursion, use iterative algorithm which prevents stack overflow in large year number.

**Output for recursive algorithm when year number is large, thrown a stack overflow error:**

**Iterative Algorithm:**

public double ForecastIterative(double principle, double rate, int years) {

double result = principle;

for (int i = 0; i < years; i++) {

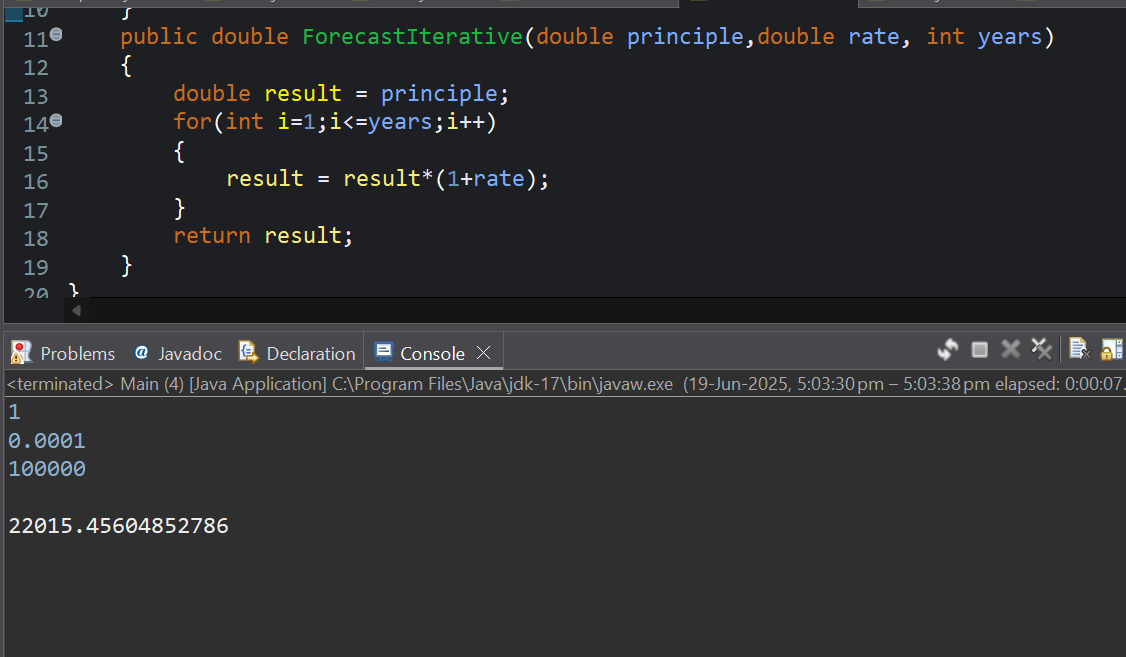
result \*= (1 + rate);

}

return result;

}

**Output:**



**Exercise 1 – Inventory Management System**

1. **Why are data structures and algorithms essential in handling large inventories?**They help store and access data efficiently. With large inventories, quick search, update, and delete operations are important to keep the system fast and responsive.
2. **The types of data structures are suitable for this problem**

* HashMap: Best for fast access using product ID.
* ArrayList: Good for simple lists but slower for searching.
* TreeMap: Keeps data sorted by key but slightly slower than HashMap.

1. **Created a Project for inventory management System.**
2. **Created a Product class with attributes.**

package Inventory\_management;

public class Product

{

int ProductId;

String ProductName;

int Quantity;

double price;

public Product(int ProductID, String ProductName, int Quantity, double price)

{

this.ProductId=ProductID;

this.ProductName=ProductName;

this.Quantity=Quantity;

this.price=price;

}

}

1. **Choosed HashMap as Data Structure.**

Best for fast access using product ID.

1. **Implemented the methods to add, update and delete product from inventory.**

**Code:**

package Inventory\_management;

import java.util.HashMap;

public class InventorySystem{

private HashMap<Integer, Product> Inventory= new HashMap<>();

public void addProduct(Product product)

{

if(Inventory.containsKey(product.ProductId))

{

System.*out*.println("Product already exists");

}

else {

Inventory.put(product.ProductId,product);

}

}

public void updateProduct(int Id,String ProductName, int Quantity,double price)

{

Product oldprod= Inventory.get(Id);

if(oldprod!=null)

{

if(ProductName!=null)

{

oldprod.ProductName=ProductName;

}

if(Quantity!=0)

{

oldprod.Quantity=Quantity;

}

if(price!=0.0)

{oldprod.price=price;

}

System.*out*.println("Updated the Product with Id"+Id+"\n");

}

}

public void deleteProduct(int id)

{

if(Inventory.get(id)!=null)

{

Inventory.remove(id);

}

else {

System.*out*.println("Product not Found");

}

}

public void displayInventory()

{

for(Product p: Inventory.values())

{

System.*out*.println("Id: "+p.ProductId);

System.*out*.println("Name: "+p.ProductName);

System.*out*.println("Quantity: "+p.Quantity);

System.*out*.println("Price: "+p.price+"\n");

}

}

}

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Analysis time complexity**

**Time complexity of each operation (in HashMap):**

Add: O(1)

Update : O(1)

Delete: O(1)

Use HashMap for quick access, avoid duplicate keys, and use indexing if searching by name or other fields.