**Algorithms\_Data Structures  
  
Exercise 2: E-commerce Platform Search Function**

CODE:

import java.util.\*;

   class Product {

     int proId;

     String proName;

     String proCat;

     Product( int proId,String proName, String proCat)

    {

      this.proName=proName;

      this.proId=proId;

      this.proCat=proCat;

    }

    void display()

    {

        System.out.println("ID:"+proId+" Name:"+proName+" Category:"+proCat);

    }

}

class LinearSearch

{

    static Product line(Product[] pro,String tar)

    {

        for(Product p:pro)

        {

            if(p.proName.equalsIgnoreCase(tar))

            {

                return p;

            }

        }

        return null;

    }

}

class BinarySearch

{

    static Product bisear(Product[] p,String tar)

    {

        int l=0;

        int r=p.length-1;

        while(l<=r)

        {

            int mid=(l+r)/2;

            int comp=p[mid].proName.compareToIgnoreCase(tar);

            if(comp==0)

            {

                return p[mid];

            }

            else if(comp<0)

            {

                l=mid+1;

            }

            else

            {

                r=mid-1;

            }

        }

       return null;

    }

   static void sortByProName(Product[] p)

   {

    Arrays.sort(p,Comparator.comparing(P->P.proName.toLowerCase()));

   }

}

class Main{

    public static void main(String[] args)

    {

       Product[] p={

        new Product(1,"Lap","elect"),

        new Product(2,"sha","health"),

        new Product(3,"shoe","foot"),

        new Product(4,"book","stat"),

        new Product(5,"tab","elect"),

       };

       Product f1=LinearSearch.line(p,"shoe");

       System.out.println("Linear search");

       if(f1!=null)

       {

        f1.display();

       }

        else

        {

           System.out.println("Product");

        }

        Product f2=BinarySearch.bisear(p,"lap");

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

        if(f2!=null)

        {

            f2.display();

        }

        else{

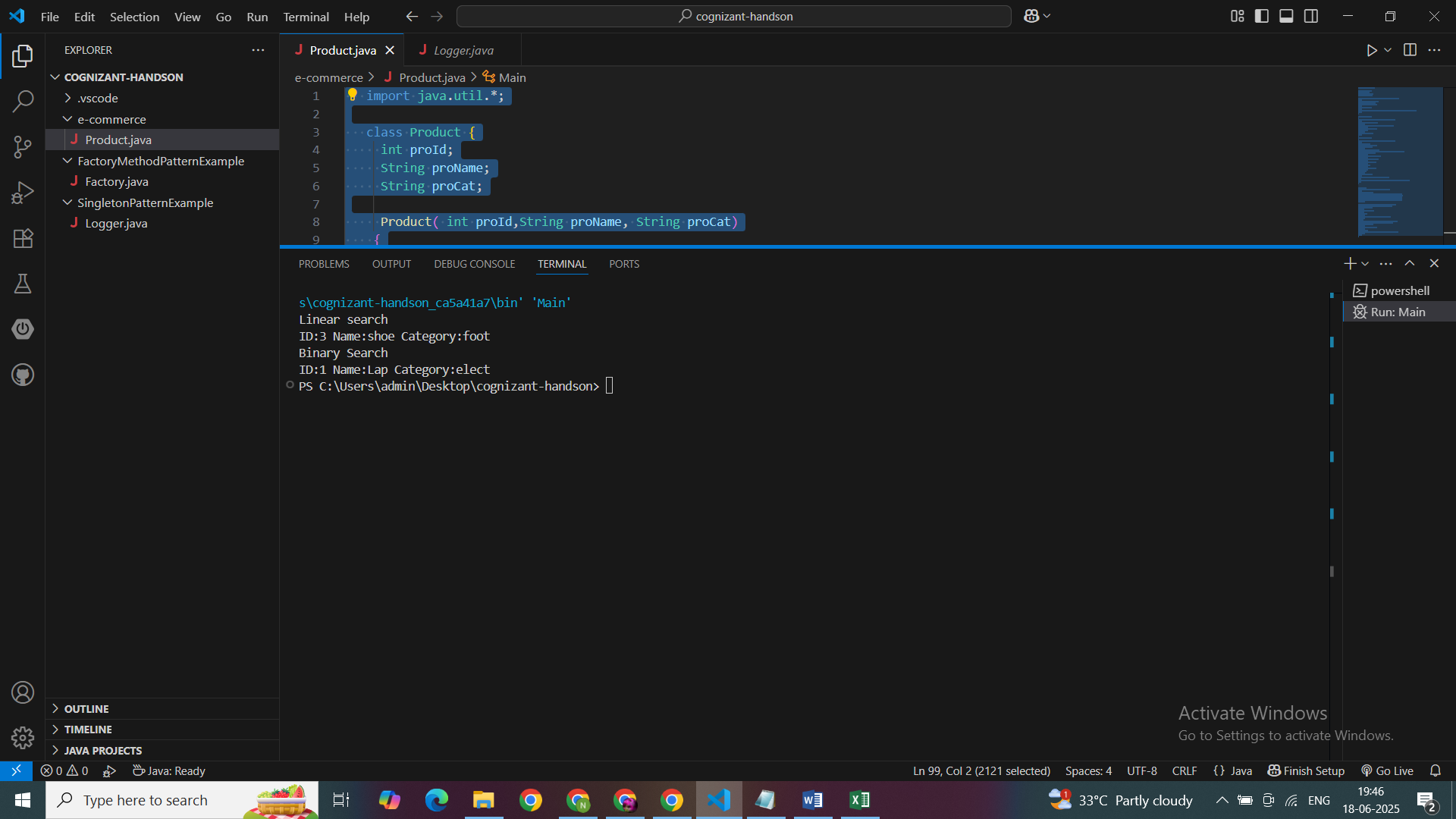
            System.out.println("Poduct not found");

        }

    }

}

**OUTPUT:**

****

**Time Complexity**

| **Algorithm** | **Time Complexity (Worst Case)** |
| --- | --- |
| Linear Search | - O(n) |
| Binary Search | - O(log n) |

**Linear Search**:

✅ Works on **unsorted data**  
❌ Slower for large datasets

**Binary Search**:  
✅ Much **faster** (log n), but  
❌ Requires the data to be **sorted**

**Efficiency:** Binary Search is ideal for performance — if data is sorted

 We use Linear Search for simplicity when data is small or unsorted

**Exercise 7: Financial Forecasting**

package FinancialForecasting;

import java.util.\*;

public class Finance {

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        double present=sc.nextDouble();

        double rate=sc.nextDouble();

        int years=sc.nextInt();

        double futureValue=future(present,rate,years);

        System.out.printf("%.2f",futureValue);

    }

    public static double future(double p,double r,double y)

    {

        if(y==0)

        {

            return p;

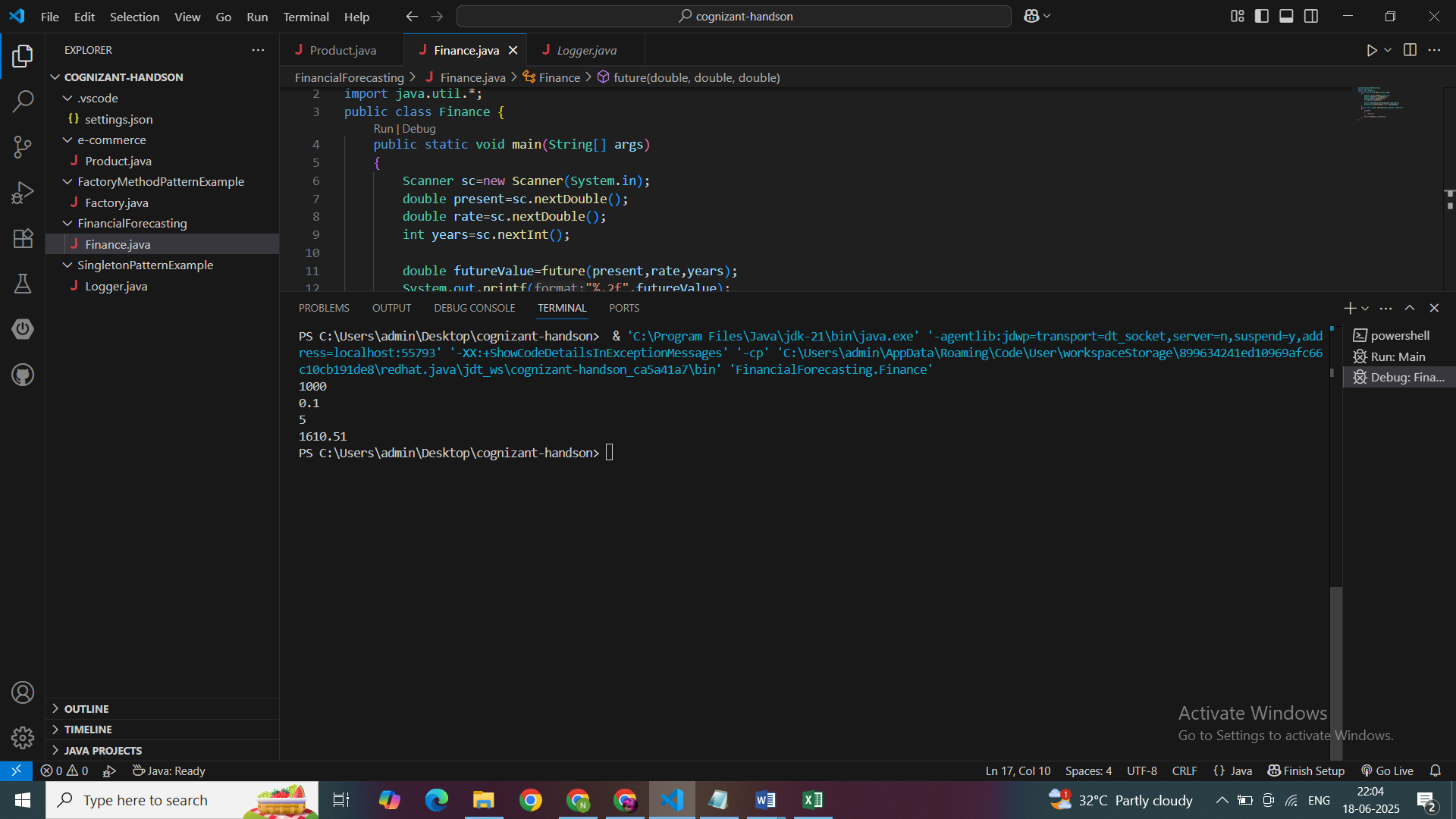
        }

        return future(p,r,y-1)\*(1+r);

    }

}

**OUTPUT:**

****

#### 🔸 Time Complexity:

Each recursive call reduces years by 1.

So:

* **Time Complexity = O(n)**
* **Space Complexity = O(n)** (due to call stack)

#### 🔸 Problem: Recursive overhead

For large years, too many recursive calls can lead to:

* Stack overflow
* Slow performance

**Optimization Techniques**

**1.Iteration  
2.Math formula  
  
1.Iteration**

static double futureValueIterative(double presentValue, double rate, int years) {

double result = presentValue;

for (int i = 1; i <= years; i++) {

result \*= (1 + rate);

}

return result;

}  
  
 **Time Complexity**: O(n)

 **Space Complexity**: O(1)

**2.Math Formula(Exponentation)**static double futureValueFormula(double presentValue, double rate, int years) {

return presentValue \* Math.pow(1 + rate, years);

}  
  
**Time Complexity**: O(1)

**Space Complexity:** O(1)