**MODULE 2**

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

**Understand Asymptotic Notation**

**Big O Notation**

**Big O notation** is used to describe the **time complexity** (or space complexity) of an algorithm in the worst-case scenario.

It shows how the performance scales as the input size increases

### Best, Average, Worst-case for Search:

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Linear Search** | O(1) (first match) | O(n/2) ≈ O(n) | O(n) (last or not found) |
| **Binary Search** | O(1) (middle match) | O(log n) | O(log n) |

**Program:**

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(103, "Mobile", "Electronics"),

new Product(101, "Shirt", "Clothing"),

new Product(102, "Book", "Education"),

new Product(104, "Laptop", "Electronics")

};

System.out.println("--Linear Search--");

Product result1=Search.linearSearch(products,106);

System.out.println(result1!=null ? result1:"Product not found");

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

Search.Sort(products);

Product result2=Search.BinarySearch(products,102);

System.out.println(result2!=null ? result2:"Product not found");

}

public 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;

}@Override

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

public static class Search{

public static Product linearSearch(Product[] products, int targetId){

for(Product product:products){

if(product.productId==targetId){return product;}

} return null;

}

public static Product BinarySearch(Product[] products,int targetId) {

int start=0;

int end=products.length-1;

while(start<=end){

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

if(targetId==products[mid].productId){return products[mid];}

if(targetId<products[mid].productId){end=mid-1;}

if(targetId>products[mid].productId){start=mid+1;}

} return null;

}

public static Product[] Sort(Product[] products){

if(products.length==1){return products;}

int mid=products.length/2;

Product [] left=Sort(Arrays.copyOfRange(products,0,mid));

Product [] right=Sort(Arrays.copyOfRange(products,mid,products.length));

return Merge(left,right);

}

public static Product[] Merge(Product[] first,Product[] second ) {

Product[] mix=new Product[first.length+second.length];

int i=0;

int j=0;

int k=0;

while(i<first.length && j<second.length){

if(first[i].productId<second[j].productId){

mix[k]=first[i];

i++;

}

else{

mix[k]=second[j];

j++;

} k++;

}

while(i<first.length){

mix[k]=first[i];

i++; k++;

}

while(j<second.length){

mix[k]=second[j];

j++; k++;

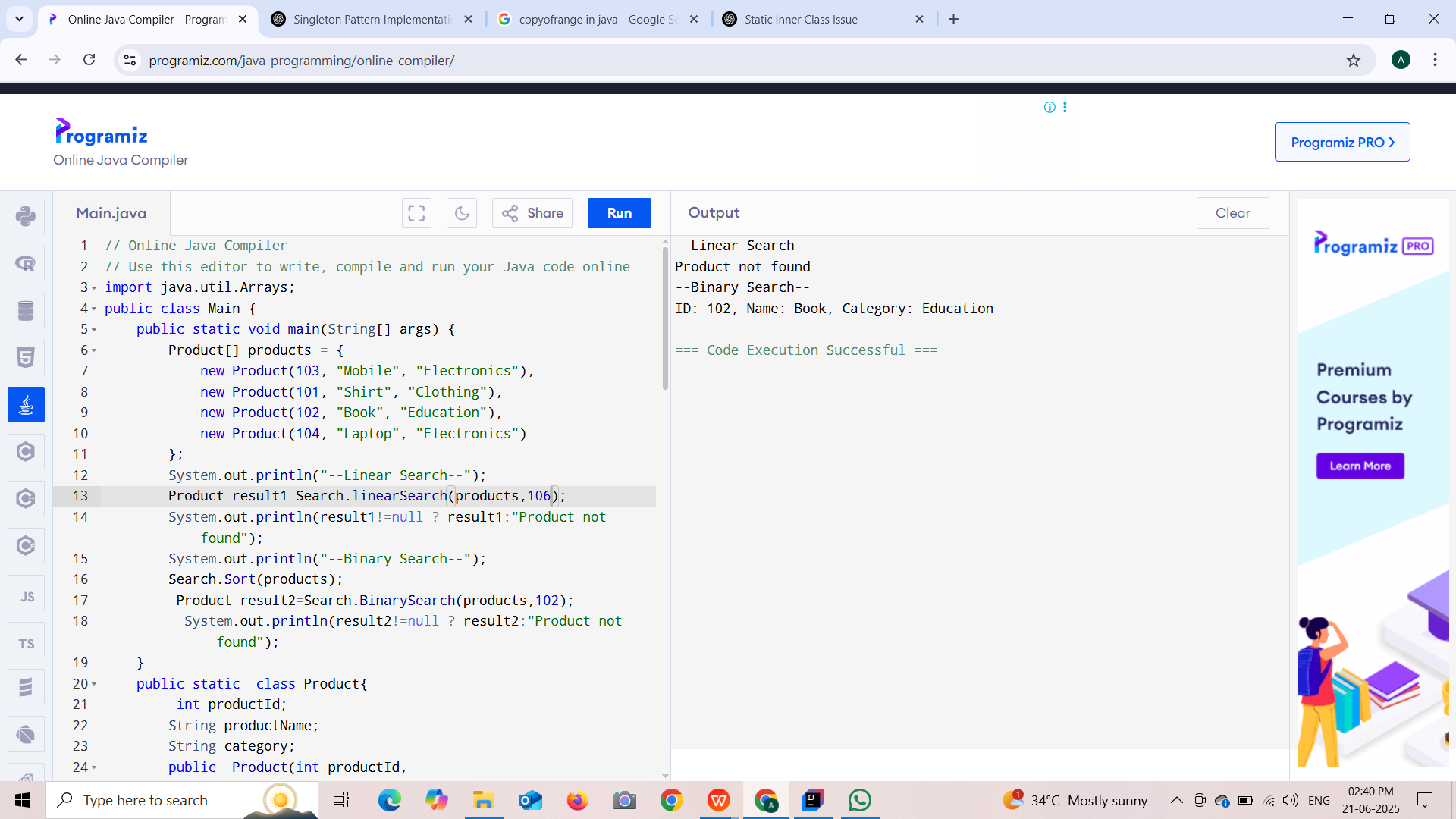
} return mix;

}

}

}

OUTPUT:



**Analysis**

| **Algorithm** | **Time Complexity** | **When to Use** |
| --- | --- | --- |
| Linear Search | O(n) | Small datasets, unsorted arrays |
| Binary Search | O(log n) | Large datasets, **sorted** arrays only |