Exercise 2: E-commerce Platform Search Function

Big O Notation:

Big O notation describes the upper bound of an algorithm's running time - how performance scales with input size n.

Algorithm	Time Complexity	Meaning
Linear Search	O(n)	Time grows linearly
Binary Search	O(log n)	Time grows logarithmically

Best, Average, Worst Case:

Algorithm	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n/2) ~ O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)

Best Case: Target found in first attempt.

Average Case: Random position.

Worst Case: Target not found (scan all).

Product.java:

```
public 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 productId+" - "+productName+" ["+category+"]";
    }
}
```

SearchAlgorithms.java:

```
public class SearchAlgorithms {
    public static Product linearSearch(Product[] products, String targetName) {
        for (Product p:products) {
            if (p.productName.equalsIgnoreCase(targetName)) {
                return p;
            }
        }
        return null;
    }
    public static Product binarySearch(Product[] products, String targetName) {
        int left=0;
        int right=products.length-1;
        while (left<=right){
            int mid=left+(right-left)/2;
            Product midProduct=products[mid];
            if(midProduct.productName.equalsIgnoreCase(targetName)){
                return midProduct;
            } else if(midProduct.productName.compareToIgnoreCase(targetName)<0){
                left=mid+1;
            } else{
                right=mid-1;
            }
        }
        return null;
    }
}
```

Main.java:

```
import java.util.Arrays;
```

```
public class Main{
   public static void main(String[] args){
      Product[] products={
         new Product(1, "Laptop", "Electronics"),
         new Product(2, "Chair", "Furniture"),
         new Product(3, "Pen", "Stationery"),
         new Product(4, "Mobile", "Electronics"),
         new Product(5, "Desk", "Furniture")
      };
      Product foundLinear=SearchAlgorithms.linearSearch(products, "Pen");
      System.out.println("Linear Search Result: "+(foundLinear!=null?foundLinear:"Not found"));
      Arrays.sort(products,(a,b)->a.productName.compareToIgnoreCase(b.productName));
      Product foundBinary=SearchAlgorithms.binarySearch(products,"Pen");
      System.out.println("Binary Search Result: "+(foundBinary!=null?foundBinary:"Not found"));
   }
}
```

Output:

```
yagnanarayanan@yagnanarayanans-MacBook-Air week1-cog %  /usr/bin/env /Users/yagnanarayanan/Library/Application\ Support/Code/User/globalStorage/pleiades.java-extension-pack-jdk/java/17/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/yagnanarayanan/Library/Application\ Support/Code/User/workspaceStorage/e4795353b6be3a1cfa35ea32f327f068/redhat.java/jdt_ws/week1-cog_1876fd24/bin Main
Linear Search Result: 3 - Pen [Stationery]
Binary Search Result: 3 - Pen [Stationery]
yagnanarayanan@yagnanarayanans-MacBook-Air week1-cog %
```

Analysis:

Use Linear Search if data is small or frequently unsorted.

Use Binary Search when:

- Data is large
- Rarely modified
- Sorted or can be sorted once during load