<u>Data Structure and Algorithm(Week 1)</u>

Exercise 2: E-commerce Platform Search Function

1. **Understand Asymptotic Notation:**
    - Explain Big O notation and how it helps in analyzing algorithms.
    - Describe the best, average, and worst-case scenarios for search operations.

Big O notation describes the upper bound (worst-case scenario) of an algorithm's time or space complexity, helping us understand how it scales with input size.

- Example: O(n) means time increases linearly with the input size n.

Search Type	**Best Case**	**Average Case**	**Worst Case**
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)

Product.java

```java
package ECommerce;

public class Product {
    int id;
    String name;
    String category;

    public Product(int id,String name, String category) {
        this.id=id;
        this.name=name;
        this.category=category;
    }

}
```

SearchOperation.java

```java
package ECommerce;
import java.util.Arrays;
import java.util.Comparator;

public class SearchOperations {

    public static Product linearSearch(Product[] products, String nameToFind) {
        for(Product product : products) {
            if(product.name.equalsIgnoreCase(nameToFind)) {
                return product;
            }
        }
        return null;
    }

    public static Product binarySearch(Product[] products, String nameToFind) {
        Arrays.sort(products,Comparator.comparing(p -> p.name.toLowerCase()));

        int low=0;
        int high=products.length-1;
        while(low<=high) {
        int mid=(low+high)/2;
        int compare=products[mid].name.compareToIgnoreCase(nameToFind);

        if(compare==0) {
            return products[mid];

        }else if(compare<0) {
            low=mid+1;
        }else {
            high=mid-1;
        }
    }
    return null;
}
}
```

Main.java

```java
public class Main {
    public static void main(String[] args) {
        Product[] products= {
                new Product(1,"Laptop","Electronics"),
                new Product(2,"Shoes","Fashion"),
                new Product(3,"Book","Stationary"),
                new Product(4,"Phone","Electronics")

        };

        Product foundLinear=SearchOperations.linearSearch(products, "Laptop");
        if(foundLinear != null) {
            System.out.println("Linear Search Found:" +foundLinear.name+"in"+ foundLinear.category);
        }else {
            System.out.println("Linear Search:Product not found");
        }
        Product foundBinary = SearchOperations.binarySearch(products, "phone");
        if(foundBinary != null) {
            System.out.println("Binary Search Found:"+ foundBinary.name+" in "+ foundBinary.category);
        }else {
            System.out.println("Binary search: Product not found");
        }
    }

}
```

Output:

```
Problems  @ Javadoc  Declaration  Console ×  Co
<terminated> Main (1) [Java Application] C:\Users\stuti\.p2\poo
Linear Search Found:Laptop in Electronics
Binary Search Found:Shoes in Fashion
```

Exercise 7: Financial Forecasting

Recursion is when a method calls itself to solve a smaller part of a larger problem.

Forecast.java

```java
public class Forecast {
    double currentValue;
    double growthRate;
    int years;

    public Forecast(double currentValue, double growthRate,int years) {
        this.currentValue=currentValue;
        this.growthRate=growthRate;
        this.years=years;
    }

}
```

ForecastUtil.java

```java
public class ForecastUtil {
    public static double calculateFutureValue(double currentValue,double growthRate,int years)
        if(years ==0) {
            return currentValue;
        }

        double newValue=currentValue * (1 + growthRate);
        return calculateFutureValue(newValue , growthRate,years-1);
    }

}
```

Main.java

```java
package Financialforecaste;

public class Main {

    public static void main(String[] args) {
        Forecast forecast=new Forecast(1000.0,0.10,5);

        double futureValue=ForecastUtil.calculateFutureValue(
                forecast.currentValue,
                forecast.growthRate,
                forecast.years
                );
        System.out.printf("Future value after %d years : %.2f\n",forecast.years,futureValue);

    }

}
```

Output: