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

Big(O) Notation is a type of asymptomatic notation which helps us to understand the **Worst-case** time complexity of a program

Linar Search

Time Complexity:

Best Case – When the target is at first index – O(1)

Average Case – When target is somewhere in the middle – O(n/2) ≈ O(n)

Worst Case – When the target is at last index – O(n)

Binary Search

Time Complexity:

Best Case – When the target is at the middle– O(1)

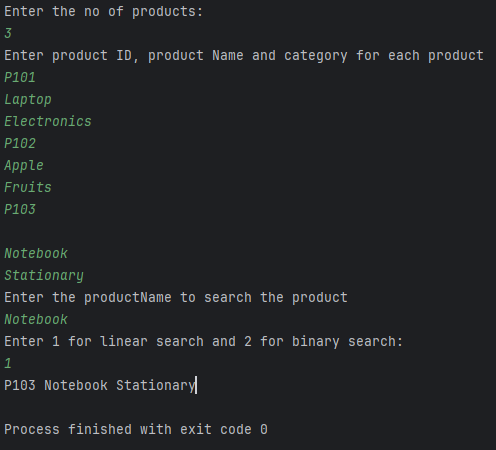
Average Case – When target is found after cutting into halves multiple times – O(log n)

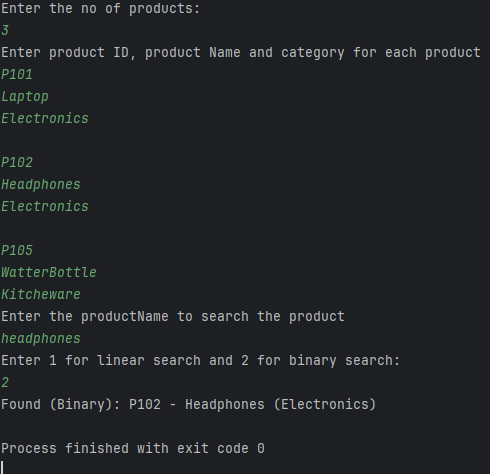
Worst Case – When the target is found when only one item remains– O(log n)

**Code:**

import java.util.\*;  
public class Product {  
 static String[] *category*;  
 static String[] *productName*;  
 static String[] *productId*;  
 static int *n*;  
 public static void main(String[] args)  
 {  
 Scanner sc=new Scanner(System.*in*);  
 System.*out*.println("Enter the no of products:");  
 *n*=sc.nextInt();  
 *productId*=new String[*n*];  
 *productName*=new String[*n*];  
 *category*=new String[*n*];  
 System.*out*.println("Enter product ID, product Name and category for each product");  
 for(int i=0;i<*n*;i++)  
 {  
 *productId*[i]=sc.next();  
 *productName*[i]=sc.next();  
 *category*[i]=sc.next();  
 }  
 System.*out*.println("Enter the productName to search the product");  
 String pdt=sc.next();  
 Product ob=new Product();  
 System.*out*.println("Enter 1 for linear search and 2 for binary search:");  
 int ch=sc.nextInt();  
 switch (ch)  
 {  
 case 1:  
 ob.linear(pdt);  
 break;  
 case 2:  
 ob.binary(pdt);  
 break;  
 default:  
 System.*out*.println("Wrong Choice");  
 }  
 }  
 public void linear(String pdt)  
 {  
 int f=0;  
 for(int i=0;i<*n*;i++)  
 {  
 if(*productName*[i].equalsIgnoreCase(pdt)) {  
 System.*out*.println(*productId*[i] + " " + *productName*[i] + " " + *category*[i]);  
 f=1;  
 }  
 }  
 if(f==0)  
 System.*out*.println("Product Not Found");  
 }  
 public void binary(String pdt)  
 {  
 String[][] combined=new String[*n*][3];  
 for(int i=0;i<3;i++)  
 {  
 combined[i][0]=*productId*[i];  
 combined[i][1]=*productName*[i];  
 combined[i][2]=*category*[i];  
 }  
 Arrays.*sort*(combined, Comparator.*comparing*(a -> a[1].toLowerCase()));  
 int left = 0, right = *n* - 1;  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 int cmp = combined[mid][1].compareToIgnoreCase(pdt);  
 if (cmp == 0) {  
 System.*out*.println("Found (Binary): " + combined[mid][0] + " - " + combined[mid][1] + " (" + combined[mid][2] + ")");  
 return;  
 } else if (cmp < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 System.*out*.println("Product not found (Binary)");  
 }  
}

**Output:**

****

****

Binary Search will be more suitable for my program is I have to search a product among huge amount of data but linear search is more suitable if the amount of data is less

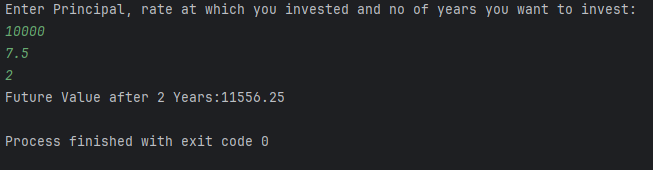
**Exercise 7: Financial Forecasting**

**Recursion** is a programming technique where a method calls **itself** to solve a problem. Instead of solving the entire problem at once, recursion breaks it into **smaller subproblems** of the same type until a **base case** is reached.

**Code:**

import java.util.\*;  
public class Forecast {  
 public static double fValue(double pv, double rate, int n) {  
 if (n == 0)  
 return pv;  
 return (1 + rate) \* *fValue*(pv, rate, n - 1);  
 }  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*);  
 System.*out*.println("Enter Principal, rate at which you invested and no of years you want to invest:");  
 int Principal = sc.nextInt();  
 double Rate = sc.nextDouble();  
 int Time = sc.nextInt();  
 Rate/=100;  
 double future = *fValue*(Principal, Rate, Time);  
 System.*out*.println("Future Value after "+Time+" Years:"+ future);  
 }  
}

**Output:**



Time Complexity Analysis:

O(n) where n is the number of years as for each year the function recursively calls itself once.

Optimization:

The recursive solution can be done in time complexity O(1) by using Math.pow() and calculating the future value.