**Module 2 - Data Structures and Algorithms**

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

**1. Understand Asymptotic Notation:**

**Big O Notation and Algorithm Analysis**

Asymptotic notation is used to describe the efficiency of algorithms, particularly their time and space complexity, in relation to the size of the input (denoted as **n**). It helps us understand how an algorithm will perform as the input size becomes very large.

Among the notations, Big O (O) is the most commonly used. It gives an upper bound on the time or space required by the algorithm. This means it tells us the maximum time or memory an algorithm could take in the worst-case scenario.

**Best, Average, and Worst-Case Scenarios**

When analyzing algorithms, especially search algorithms, we usually describe their performance in three cases:

* **Best Case (Omega)**: The scenario where the input is ideal, and the algorithm performs the minimum number of steps.
* **Average Case (Theta)**: The scenario that represents a typical or expected case, assuming random input.
* **Worst Case (Big O)**: The scenario where the algorithm performs the maximum number of steps – this is the one Big O describes.

**Linear Search Analysis**

In linear search, each element is checked one by one until the required element is found or the list ends.

* **Best Case**: O(1) → If the element is found at the very beginning.
* **Average Case**: O(n/2) ≈ O(n) → Element is somewhere in the middle.
* **Worst Case**: O(n) → If the element is the last one or not present at all.

This algorithm is simple and works for unsorted arrays, but it becomes inefficient for large datasets.

**Binary Search Analysis**

Binary search works only on sorted arrays. It divides the array into halves repeatedly to locate the target.

* **Best Case**: O(1) → If the middle element is the target.
* **Average Case**: O(log n) → Requires log n comparisons.
* **Worst Case**: O(log n) → Still only log n steps in total.

Binary search is far more efficient for large datasets, provided the array is sorted.

* Big O notation helps us predict how an algorithm scales as input grows.
* Linear search is suitable for small or unsorted lists.
* Binary search is ideal for large sorted lists due to its logarithmic efficiency.
* For optimizing an e-commerce search function, binary search is recommended when working with sorted product IDs.

**2.Setup**

To create a class named Product with attributes for searching, such as productId, productName, and category.

**3. Implementation**

**AIM:**

To Implement linear search and binary search algorithms for E-Commerce Platform.

**Code:**

import java.util.\*;

 class Product{

    int productId;

    String productName;

    String category;

}

public class EcommercePlatformSearch  {

    static void linearsearch(Product[] p,int x){

        boolean o = false;

        for(int i=0;i<3;i++){

            if(p[i].productId==x){

                System.out.println("Linear search: Product found at index "+" " +i);

                System.out.println();

                o = true;

                break;

            }

        }

        if(!o){

            System.out.println("Linear search: Product not found");

        }

        System.out.println();

}

static void binarysearch(Product[] p,int x){

    int low = 0 ;

        int high =2;

        int mid=0;

         boolean o = false;

Arrays.sort(p, Comparator.comparingInt(prod -> prod.productId));

        while(low<=high){

            mid=(low+high)/2;

            if(p[mid].productId==x){

                System.out.println("Binary search: Product found at index "+" "+mid);

                System.out.println();

                o=true;

                break;

            }

            else if(p[mid].productId>x){

                high=mid-1;

            }

            else{

                low=mid+1;

            }

        }

         if(!o){

            System.out.println("Binary search: Product not found");

        }

        System.out.println();

}

static void linearsearchbypn(Product[] p,String q){

         boolean o = false;

        for(int i=0;i<3;i++){

            if(p[i].productName.equalsIgnoreCase(q)){

                System.out.println("Linear search by product name: Product found at index "+" " +i);

                System.out.println();

                o = true;

                break;

            }

        }

        if(!o){

            System.out.println("Linear search by product name: Product not found");

        }

        System.out.println();

}

    public static void main(String[] args){

        Product[] p = new Product[3];

          Scanner s = new Scanner(System.in);

          System.out.println("Enter the productId to be searched");

        int x=s.nextInt();

        s.nextLine();

        System.out.println();

        System.out.println("Enter the productName to be searched");

        String q=s.nextLine();

        System.out.println();

        p[0] = new Product();

        p[1] = new Product();

        p[2] = new Product();

    System.out.println("Enter productId,productName,category");

      System.out.println();

        for(int i =0;i<3;i++){

            p[i].productId=s.nextInt();

            s.nextLine();

            p[i].productName=s.nextLine();

            p[i].category=s.nextLine();

            System.out.println();

        }

        System.out.println();

          for(int i=0;i<3;i++){

        System.out.println("["+p[i].productId+","+p[i].productName+","+p[i].category+"]");

        }

        System.out.println();

        s.close();

       linearsearch(p,x);

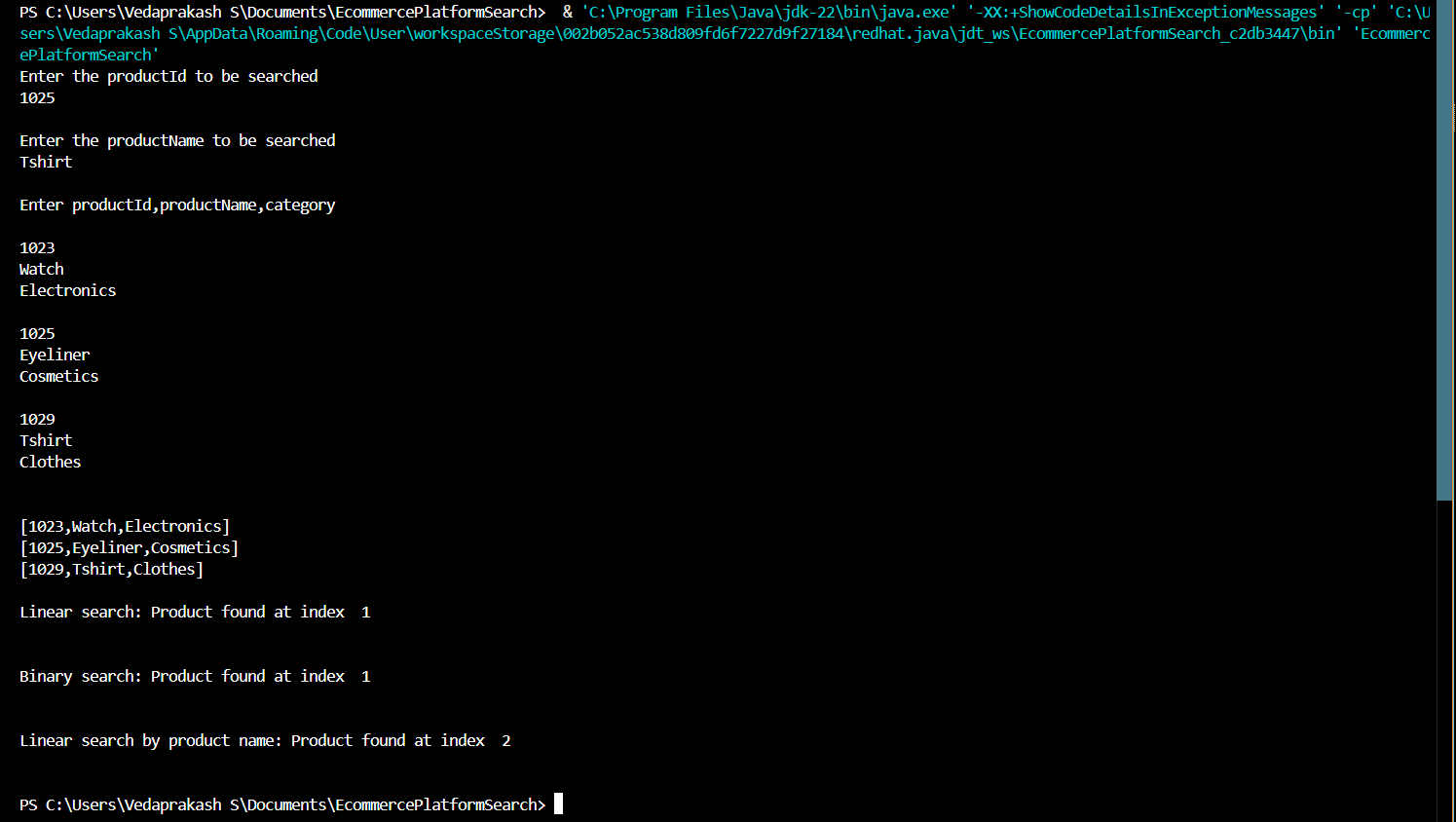
       binarysearch(p ,x);

       linearsearchbypn(p, q);

    }

  }

**OUTPUT:**



**4.Analysis**

**Comparison of Time Complexity**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** | **Sorted Required** |
| --- | --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) | No |
| Binary Search | O(1) | O(log n) | O(log n) | Yes |

* **Linear Search**:
  + The best case occurs when the element is at the beginning.
  + The worst case is when the element is not present or is at the end.
  + The average case depends on the position of the element (usually O(n/2), but simplified to O(n)).
  + It does not require the data to be sorted.
* **Binary Search**:
  + Performs exceptionally well with large datasets due to its divide-and-conquer strategy.
  + It reduces the search space by half with each step.
  + Requires the input array to be sorted beforehand.
  + Time complexity remains logarithmic in best, average, and worst cases (except best case can be O(1) when found at mid).

**Which is More Suitable for the E-commerce Platform?**

For an e-commerce platform, binary search is more suitable due to the following reasons:

* The product data (like product IDs) can be maintained in sorted order.
* Search speed is very important for user experience; binary search offers better performance for large datasets.
* Binary search reduces the number of comparisons drastically compared to linear search.
* In a real-world scenario, databases or in-memory data structures can maintain sorted indexes for fast lookup (similar to binary search).

However, in cases where the product list is not sorted or dynamicallychanging (like newly added items frequently), linear search might be used temporarily — though this is inefficient in the long run.

* **Binary search** is the preferred choice for fast, efficient searches in sorted product data.
* **Linear search** is only suitable for small or unsorted datasets and should be avoided in production-level e-commerce platforms.