**Data Structure And Alogorithms**

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

**Explain Big O notation and how it helps in analyzing algorithms:**

Big O notation is a mathematical notation used to describe the time or space complexity of an algorithm in terms of the input size (n).It tells us how the performance of an algorithm scales as the input grows, helping us compare and choose the most efficient algorithm**.**

**Describe the best, average, and worst-case scenarios for search operations.**

**1. Best Case:**  
The scenario in which the algorithm takes the least amount of time to complete. This occurs when the desired element is found immediately, such as at the beginning of the array.

Example – Linear Search:  
When the target element is located at the first index.  
Time Complexity: O(1)

Example – Binary Search:  
When the target element is exactly in the middle of the array.  
Time Complexity: O(1)

**2. Average Case:**  
The scenario that represents the expected performance of the algorithm over a range of typical inputs. It assumes the input is in a random order and calculates the average time taken.

Example – Linear Search:  
When the target element is located somewhere in the middle of the array.  
Time Complexity: O(n)

Example – Binary Search:  
The target is located after several comparisons by dividing the array repeatedly.  
Time Complexity: O(log n)

**3. Worst Case:**  
The scenario in which the algorithm takes the maximum amount of time to complete. This usually happens when the element is not present or is located at the end of the array.

Example – Linear Search:  
When the target element is at the last index or not present at all.  
Time Complexity: O(n)

Example – Binary Search:  
When the algorithm must divide the array until only one element is left or the element is not found.  
Time Complexity: O(log n)

**Program:**

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;

}

public static int linearSearch1(Product[] products, int targetId){

for (int i=0; i<products.length; i++) {

if (products[i].productId==targetId){

return i;

}

}

return -1;

}

public static int binarySearch1(Product[] products, int targetId) {

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

return mid;

} else if (products[mid].productId < targetId){

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

Product[] products = {

new Product(1, "Pencil", "Stationery"),

new Product(2, "Pouch", "Stationery"),

new Product(3, "Laptop", "Electronics"),

new Product(4, "Bottle", "Stationery")

};

int linearSearchResult = linearSearch1(products, 3);

int binarySearchResult = binarySearch1(products, 3);

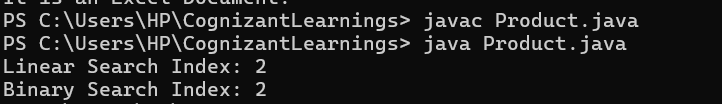
System.out.println("Linear Search: " + linearSearchResult);

System.out.println("Binary Search: " + binarySearchResult);

}

}

Output:



**Analysis:**

**Comparison of Time Complexity**

Linear Search has a time complexity of:

Best Case**:** O(1) – when the target is at the first position

Average Case**:** O(n) – when the target is in the middle

Worst Case**:** O(n) – when the target is at the end or not present

Binary Search has a time complexity of:

Best Case**:** O(1) – when the target is at the middle

Average and Worst Case: O(log n) – as it repeatedly divides the search range by half

**Which Algorithm is More Suitable and Why**

For an e-commerce platform where the product database can be large and searches happen frequently, binary search is more suitable. It is significantly faster for sorted data, especially as the number of products increases. Although it requires sorting the data, the performance benefits make it ideal for optimizing search speed**.** Linear search, on the other hand, is only practical for small or unsorted lists where setup time is minimal.