## **1. Understanding Asymptotic Notation (Big O)**

* **What is Big O?**  
   Big O notation is a mathematical tool used to describe how an algorithm's runtime (or space use) grows as input size increases, ignoring constants and lower-order terms.
* **Why it matters:**  
   It allows you to compare algorithms’ scalability—for instance, knowing that an algorithm runs in O(n log n) rather than O(n²) helps predict performance as data grows.

### **2. Best, Average, and Worst Cases for Search Operations**

In search algorithms, the **best-case** scenario occurs when the target item is found on the first attempt. For **linear search**, this means checking the very first element—yielding a time complexity of *O(1)*. **Binary search**, applied on a sorted array, also achieves *O(1)* in the best case, which happens when the middle element matches the target immediately.

The **average-case** scenario considers a typical search where the target is equally likely to be anywhere in the list. For **linear search**, this averages to about *n/2* comparisons, simplifying to *O(n)* . **Binary search**, in contrast, halves the search space repeatedly, resulting in a time complexity of *O(log n)* .

In the **worst-case**, linear search either inspects the very last element or fails to find the target at all—leading to *O(n).* Binary search, however, continues halving until the search space is exhausted, assuring a bound of *O(log n)* in the worst case too .

## **3. Code: Product Search in Java**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

static class Product {

int productId;

String productName;

String category;

public Product(int id, String name, String cat) {

this.productId = id;

this.productName = name;

this.category = cat;

}

}

// Linear search: returns index or -1

public static int linearSearch(Product[] arr, int targetId) {

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

if (arr[i].productId == targetId) return i;

}

return -1;

}

// Binary search: requires sorted by productId

public static int binarySearch(Product[] arr, int targetId) {

int low = 0, high = arr.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (arr[mid].productId == targetId) return mid;

else if (arr[mid].productId < targetId) low = mid + 1;

else high = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Widget A", "Gadgets"),

new Product(102, "Widget B", "Gadgets"),

new Product(103, "Thing A", "Tools"),

new Product(104, "Thing B", "Tools")

};

System.out.println("Linear search for 103: " +

linearSearch(products, 103));

// Sorting by productId is required before binary search

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

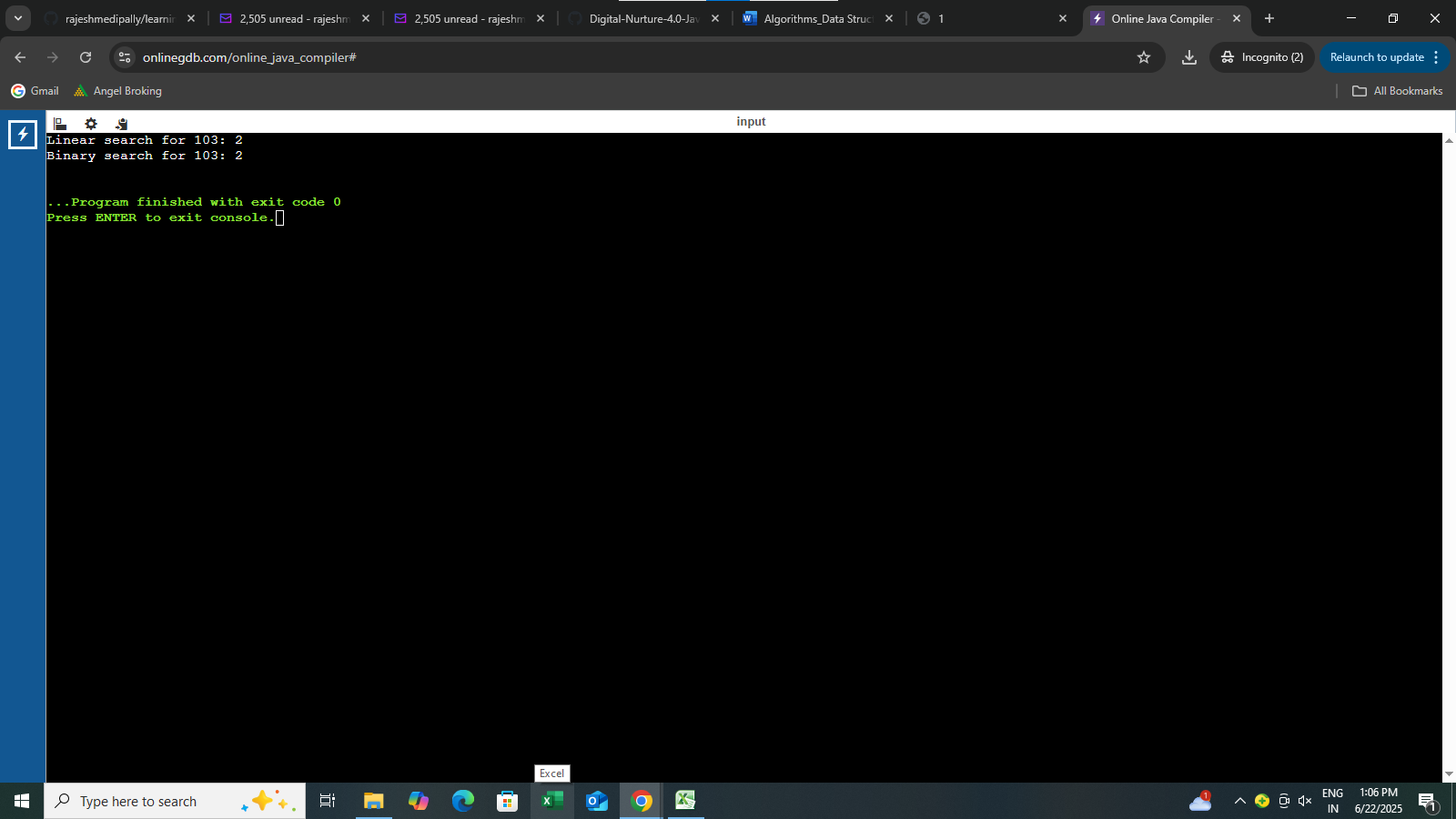
System.out.println("Binary search for 103: " +

binarySearch(products, 103));

}

}

* For **linearSearch**, the array need not be sorted.
* For **binarySearch**, the array **must** be sorted by productId.



### **4. Time Complexity Comparison**

When comparing the two algorithms based on their time complexity: **linear search** operates in constant time *O(1)* in the best case but degrades to *O(n)* for both average and worst cases—because it may have to scan through the entire dataset . In terms of space usage, it operates in-place with *O(1)* additional memory.

**Binary search**, on the other hand, delivers significant performance gains. Even though its best case is only *O(1)*, its average and worst-case time complexity remains *O(log n)*, thanks to its divide‑and‑conquer approach of halving the search range each step. The iterative version also uses *O(1)* extra space, although a recursive implementation would require *O(log n)* call stack.

## **5. Which to Use on the E-commerce Platform?**

* **Linear Search Advantages**:
  + Works on unsorted product lists.
  + Simple and low overhead; ideal for small or rarely searched datasets.
* **Binary Search Advantages**:
  + Highly efficient with O(log n) performance on large product catalogs.
  + Ideal when product list is sorted and searches are frequent.
* **Recommendation**:
  + **Use binary search** for core search operations—just ensure your product list (or index) remains sorted by ID or another quick key.
  + **Use linear search** only for small, ad-hoc lists or one-off quick checks where sorting isn’t justified.