**Problem Understanding:**

In an e-commerce platform, search functionality must quickly find the relevant product based on the product name. Implementation has two search algorithms: Linear search and Binary search.

**Big O Notation:**

Big O Notation is a way to describe how fast or slow an algorithm runs depending on the size of the input. It helps us measure the efficiency of an algorithm in terms of:

* Time Complexity
* Space Complexity

Big O focuses on the worst-case growth rate ignoring small constants and lower-order terms.

**Linear Search:**

Checks each element one by one until it finds the target or reaches the end.

* Best Case: O(1) → Item found at the beginning.
* Average Case: O(n/2) / O(n) → Item found in the middle.
* Worst Case: O(n) → Item at the end or not found.

**Binary Search:**

Works only on sorted arrays. Divides the array into halves to find the item.

* Best Case: O(1) → Item found in the middle on the first try.
* Average Case: O(long n) → Repeatedly dividing the array.
* Worst Case: O(log n) → Still halves until 1 element is left.

**Implementation:**

**Product class**

class Product {

String productId;

String productName;

String category;

public Product(String productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

public static int linearSearch(Product[] products, String key) {

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

if (products[i].productName.equalsIgnoreCase(key)) {

return i;

}

}

return -1;

}

public static int binarySearch(Product[] products, String key) {

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

while (low <= high) {

int mid = (low + high) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(key);

if (cmp == 0) return mid;

else if (cmp < 0) low = mid + 1;

else high = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Product[] products = { new Product("P001", "Laptop", "Electronics"),

new Product("P002", "Chair", "Furniture"),

new Product("P003", "Pen", "Stationery"),

new Product("P004", "Notebook", "Stationery"),

new Product("P005", "Mobile", "Electronics")

};

String searchKey = "Notebook";

int linearIndex = *linearSearch*(products, searchKey);

if (linearIndex != -1) {

System.***out***.println("Linear Search: Found at index " + linearIndex);

System.***out***.println(products[linearIndex]);

} else {

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

}

Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName.toLowerCase()));

int binaryIndex = *binarySearch*(products, searchKey);

if (binaryIndex != -1) {

System.***out***.println("Binary Search: Found at index " + binaryIndex);

System.***out***.println(products[binaryIndex]);

} else {

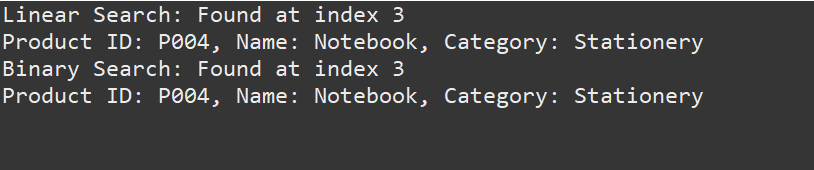
System.***out***.println("Binary Search: Product not found.");

}

}

}

**Output:**



**Most Suitable: Binary Search**

If the platform has sorted product lists binary search is ideal because much faster for large datasets and efficient even when done frequently.

If the data is small or unsorted and don’t want to sort before searching, we can use *Linear Search.*