**Big O Notation**

Big O notation describes the upper bound (worst-case) time or space complexity of an algorithm in terms of the input size *n*.

It helps you:

* Predict performance as input grows.
* Compare different algorithms fairly.
* Optimize code in time-critical systems (like an e-commerce search engine 👀).

**Search Scenarios:**

* **Best Case:** Item is found at the first try.
* **Average Case:** Item is somewhere in the middle.
* **Worst Case:** Item is last or not found at all.

| Algorithm | Best Case | Average Case | Worst Case |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |
|  |  |  |  |

#### **Which One’s Better for E-commerce?**

* **Small datasets**: Linear search is simple and good enough.
* **Large datasets (1000+ items)**: Binary search shines due to faster performance, especially when products are sorted alphabetically or by ID.
* **Real-world scalability**: Binary search or advanced indexing/search trees (or even search engines like Elasticsearch) are preferred.

**Implementation**

**Product.java**

public 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;

}

@Override

public String toString() {

return productId + " - " + productName + " [" + category + "]";

}

}

**LinearSearch.java**

public class LinearSearch {

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

for (Product product : products) {

if (product.productName.equalsIgnoreCase(targetName)) {

return product;

}

}

return null;

}

}

**BinarySearch.java**

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

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

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) {

return products[mid];

} else if (cmp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

}

**SearchTest.java (Main Class)**

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

new Product(1, "iPhone", "Electronics"),

new Product(2, "Shoes", "Fashion"),

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

new Product(4, "Watch", "Accessories")

};

String searchTerm = "Laptop";

Product resultLinear = LinearSearch.linearSearch(products, searchTerm);

System.out.println("Linear Search Result: " + (resultLinear != null ? resultLinear : "Not Found"));

Product resultBinary = BinarySearch.binarySearch(products, searchTerm);

System.out.println("Binary Search Result: " + (resultBinary != null ? resultBinary : "Not Found"));

}

}

**Output**

