**Big O Notation:**

Big O notation is used to describe the **time complexity** or **space complexity** of an algorithm in terms of **input size (n)**. It helps us understand how an algorithm's performance scales as the data grows.

**Why is it useful?**

* Allows comparison of algorithms' efficiency.
* Helps in selecting the right algorithm for large-scale applications (like e-commerce).
* Focuses on worst-case performance which is crucial for performance guarantees.

📈 **Search Operations: Best, Average, Worst Case**

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** | **Sorted Required?** |
| --- | --- | --- | --- | --- |
| **Linear Search** | O(1) | O(n/2) → O(n) | O(n) | ❌ No |
| **Binary Search** | O(1) | O(log n) | O(log n) | ✅ Yes |

* Linear search checks elements one by one.
* Binary search divides the list into halves and is much faster for large sorted data.

Code:

//Product class

**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 + ")";

}

}

Main class:

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** EcommerceSearch {

// Linear Search

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

**for** (Product p : products) {

**if** (p.productName.equalsIgnoreCase(targetName)) {

**return** p;

}

}

**return** **null**;

}

// Binary Search

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

**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**;

}

**public** **static** **void** main(String[] args) {

Product[] products = {

**new** Product(101, "Laptop", "Electronics"),

**new** Product(102, "Shoes", "Footwear"),

**new** Product(103, "Watch", "Accessories"),

**new** Product(104, "Phone", "Electronics"),

**new** Product(105, "Backpack", "Bags")

};

// Linear Search

Product result1 = *linearSearch*(products, "Watch");

System.***out***.println("Linear Search Result: " + (result1 != **null** ? result1 : "Product not found"));

// Sort before Binary Search

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

// Binary Search

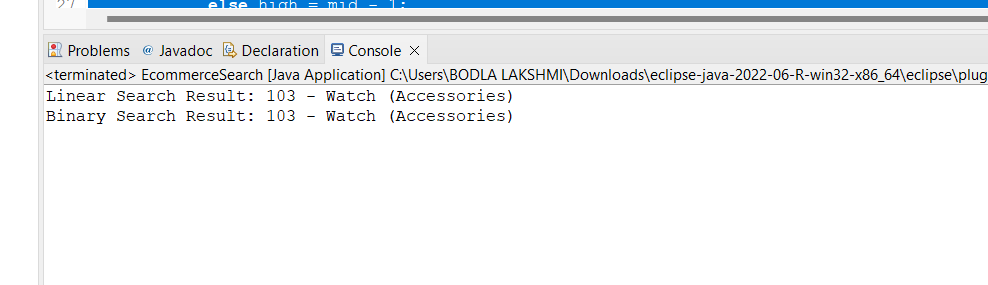
Product result2 = *binarySearch*(products, "Watch");

System.***out***.println("Binary Search Result: " + (result2 != **null** ? result2 : "Product not found"));

}

}

OUTPUT:



**4. Analysis and Comparison**

**✅ Time Complexity**

* **Linear Search** = O(n)
* **Binary Search** = O(log n)

Which to Use for E-commerce?

| **Criterion** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Performance** | Slower for large lists | Fast (logarithmic performance) |
| **Sorting Required** | No | Yes |
| **Best Use Case** | Small, unsorted data | Large, sorted product lists |