

Data Structures and Algorithms

Exercise 2:

E-Commerce Platform Search Function

This project implements a fast and optimized search functionality for an E-Commerce Platform using Java. It compares linear and binary search algorithms to efficiently retrieve products based on user queries. Here's a detailed explanation:

Step 1: Understanding Asymptotic Notations

- **Big O Notation:** Describes algorithm efficiency based on input size
 - Linear Search:
 - **Best Case:** $O(1)$ (if target is at beginning)
 - **Average/Worst Case:** $O(n)$
 - Binary Search:
 - **Best Case:** $O(1)$ (middle element is the target)
 - **Average/Worst Case:** $O(\log n)$
 - Requires Sorted array.
-

Step 2: Setup

- A *Product* class is created with *productId*, *productName*, and *category*.
-

Step 3: Implementation

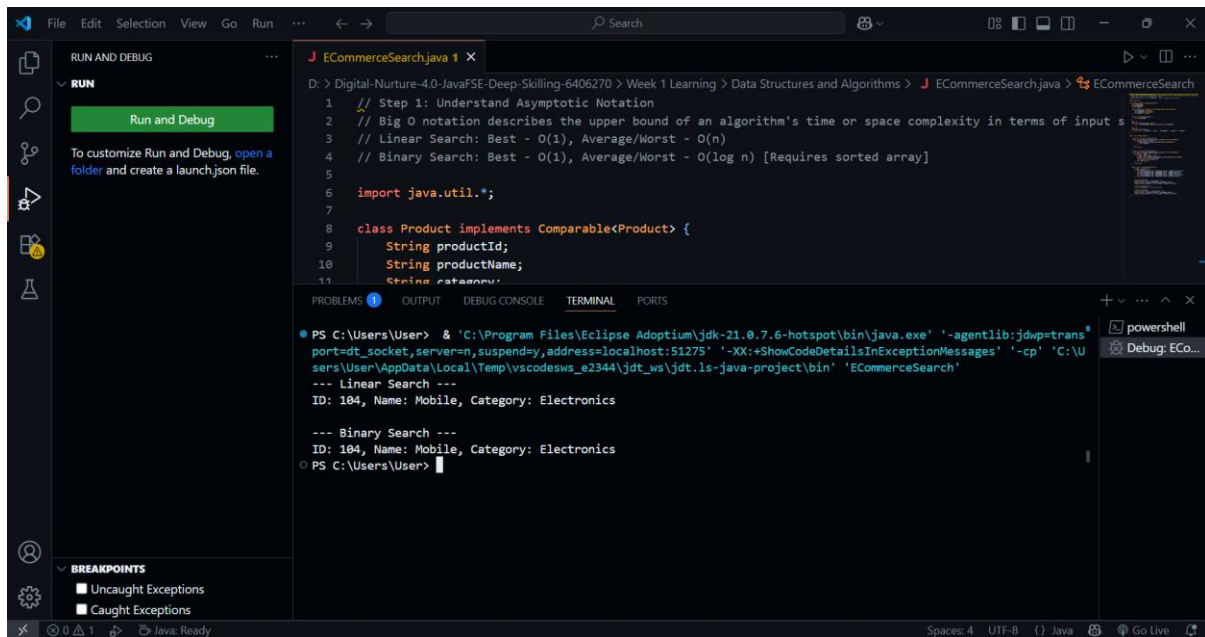
- Two search methods:
 - *linearSearch()* – works on unsorted arrays.
 - *binarySearch()* – works on sorted arrays.
 - Products are stored in an array and sorted using *Arrays.sort()* for binary search.
-

Step 4: Time Complexity Analysis

| Search Method | Time Complexity | Requirement |
|---------------|-----------------|-------------------|
| Linear Search | $O(n)$ | Unsorted allowed |
| Binary Search | $O(\log n)$ | Sorted array only |

- **Binary Search** is preferred for large, sorted datasets due to its logarithmic efficiency.

Output



The screenshot shows an IDE with a Java file named `ECommerceSearch.java`. The code implements a `Product` class that implements `Comparable<Product>`. It includes comments for Step 1: Understand Asymptotic Notation, explaining that Big O notation describes the upper bound of an algorithm's time or space complexity. The code also includes comments for Linear Search (Best - $O(1)$, Average/Worst - $O(n)$) and Binary Search (Best - $O(1)$, Average/Worst - $O(\log n)$ [Requires sorted array]).

```
1 // Step 1: Understand Asymptotic Notation
2 // Big O notation describes the upper bound of an algorithm's time or space complexity in terms of input s
3 // Linear Search: Best - O(1), Average/Worst - O(n)
4 // Binary Search: Best - O(1), Average/Worst - O(log n) [Requires sorted array]
5
6 import java.util.*;
7
8 class Product implements Comparable<Product> {
9     String productId;
10    String productName;
11    String category;
```

The terminal output shows the execution of the program. It displays the results of a Linear Search and a Binary Search for a product with ID 104, Name: Mobile, and Category: Electronics.

```
PS C:\Users\User> & 'C:\Program Files\Eclipse Adoptium\jdk-21.0.7.6-hotspot\bin\java.exe' '-agentlib:jdwp=trans
port=dt_socket,server=n,suspend=y,address=localhost:51275' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\U
sers\User\AppData\Local\Temp\vscodesws_e2344\jdt_ws\jdt.ls-java-project\bin' 'ECommerceSearch'
--- Linear Search ---
ID: 104, Name: Mobile, Category: Electronics
--- Binary Search ---
ID: 104, Name: Mobile, Category: Electronics
PS C:\Users\User>
```