**E-commerce Platform Search Function.**

**1. Understand Asymptotic Notation**

**What is Big O Notation?**

* Big O notation is a mathematical way to describe how the performance of an algorithm changes as the size of the input grows.
* Big O focuses on the dominant term and ignores constant factors to give us a high-level view of performance.

**Common Big O Complexities:**

| Big O Notation | Meaning | Example Algorithm |
| --- | --- | --- |
| O(1) | Constant time | HashMap lookup |
| O(log n) | Logarithmic time | Binary Search |
| O(n) | Linear time | Linear Search |
| O(n log n) | Log-linear time | Merge Sort |
| O(n²) | Quadratic time | Bubble Sort |

**Best, Average, and Worst-Case Scenarios:**

**Linear Search (unsorted array)**

* Best case: O(1) → item is at the start
* Average case: O(n/2) ≈ O(n)
* Worst case: O(n) → item is at the end or not found

**Binary Search (sorted array)**

* Best case: O(1) → item is at the middle
* Average case: O(log n)
* Worst case: O(log n) → reduces the search space by half each time

**4. Analysis**

**Time Complexity Comparison**

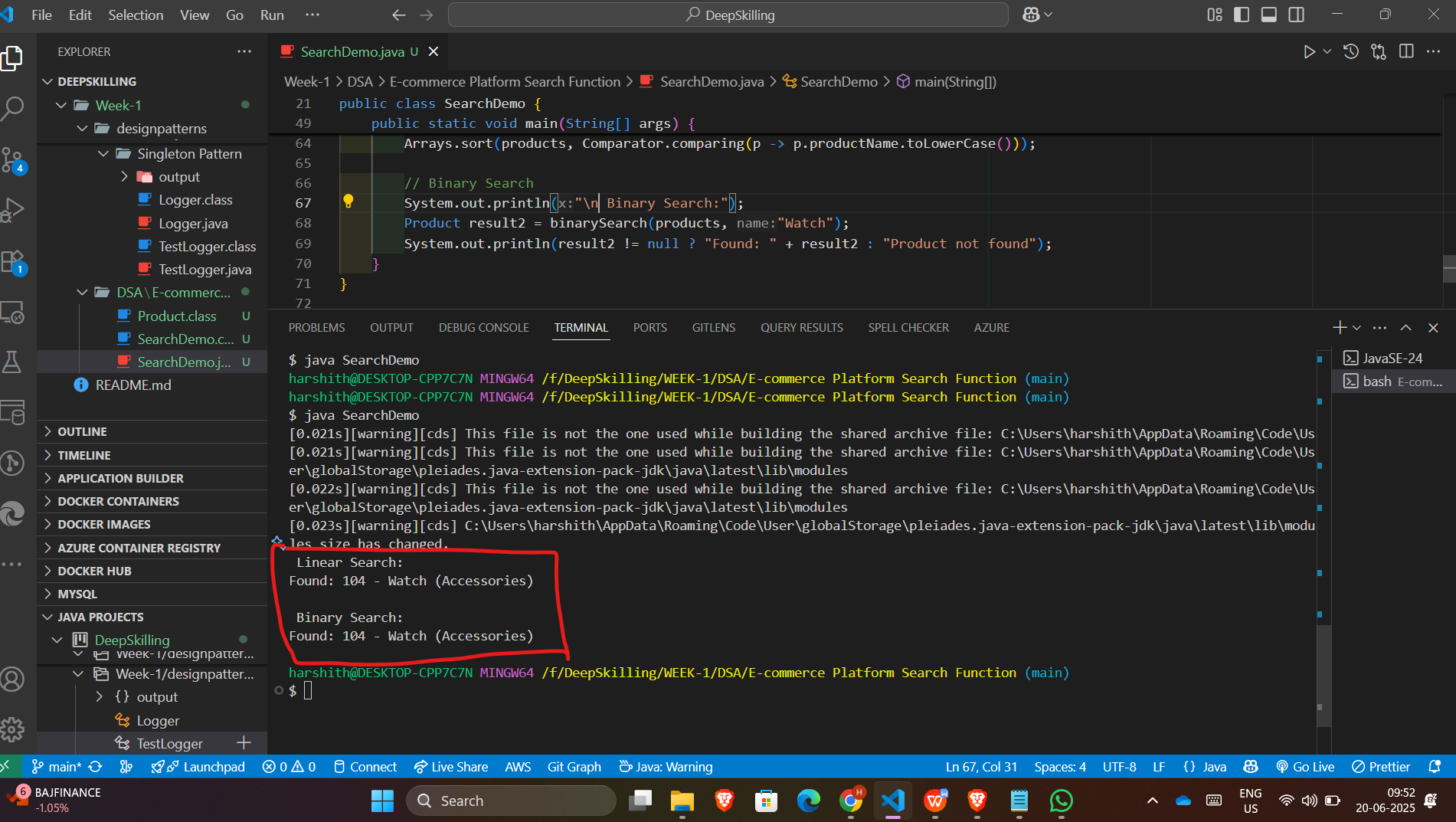
| Operation | Linear Search | Binary Search |
| --- | --- | --- |
| Best Case | O(1) | O(1) |
| Average Case | O(n) | O(log n) |
| Worst Case | O(n) | O(log n) |
| Pre-sorting | Not required | Required |

**Which is More Suitable?**

* Linear Search is simple and works on unsorted data, but gets slow as the product list grows.
* Binary Search is significantly faster, especially for large datasets — but only works on sorted arrays.

**For an e-commerce platform, where performance matters:**

* Products can be sorted once (by ID) during upload or batch updates.
* Binary search is more efficient for fast lookups.



**Financial Forecasting**

**1. Understand Recursive Algorithms**

**What is Recursion?**

* Recursion is a technique in programming where a function calls itself to solve a smaller version of the same problem.
* It is especially useful when:
* The problem can be broken down into repetitive smaller tasks
* Each step depends on the result of the previous one
* It’s easier to express the solution in terms of "do the same thing again with smaller input"

**Example of a Recursive Pattern:**

* Financial growth often follows a pattern where:
* This year’s value = Last year’s value × (1 + growth rate)
* So to calculate the future value after N years:
* FV(N) = FV(N - 1) × (1 + rate)
* This pattern is perfect for solving with recursion.

**2. Analysis**

**Time Complexity:**

| Version | Time Complexity |
| --- | --- |
| Plain Recursion | O(n) |
| With Memoization | O(n) |

* In plain recursion, every call leads to another until base case years == 0 is reached.
* With memoization, we store already computed values, which reduces repeated computation.

**Optimization Insights**

* Recursive functions are clean but can become inefficient if they re-calculate the same values repeatedly.
* Memoization (storing results) improves performance and avoids unnecessary function calls.
* You can also convert recursion to iteration if you need even more control over memory.

**Conclusion**

* Recursion simplifies the implementation of repeated calculations like compounded growth.
* With memoization, we optimize performance and make the algorithm suitable for real-time forecasting.

This is a classic use-case for recursive thinking in financial applications, such as:

* Investment forecasting
* Loan amortization
* Compound interest calculators

A screen shot of a computer

Description automatically generated