**WEEK 1**

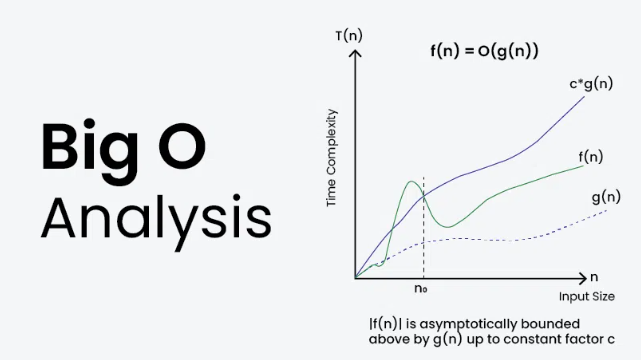
**DATA STRUCTURES AND ALGORITHM**

Exercise 2: E-commerce Platform Search Function

1.Explain Big O notation and how it helps in analyzing algorithms.

Big O notation is a powerful tool used in computer science to describe the time complexity or space complexity of algorithms. Big-O is a way to express the upper bound of an algorithm’s time or space complexity.

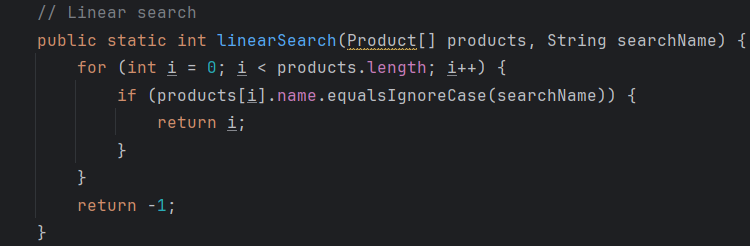
* Describes the asymptotic behavior (order of growth of time or space in terms of input size) of a function, not its exact value.
* Can be used to compare the efficiency of different algorithms or data structures.
* It provides an upper limit on the time taken by an algorithm in terms of the size of the input. We mainly consider the worst case scenario of the algorithm to find its time complexity in terms of Big O
* It’s denoted as O(f(n)), where f(n) is a function that represents the number of operations (steps) that an algorithm performs to solve a problem of size n.

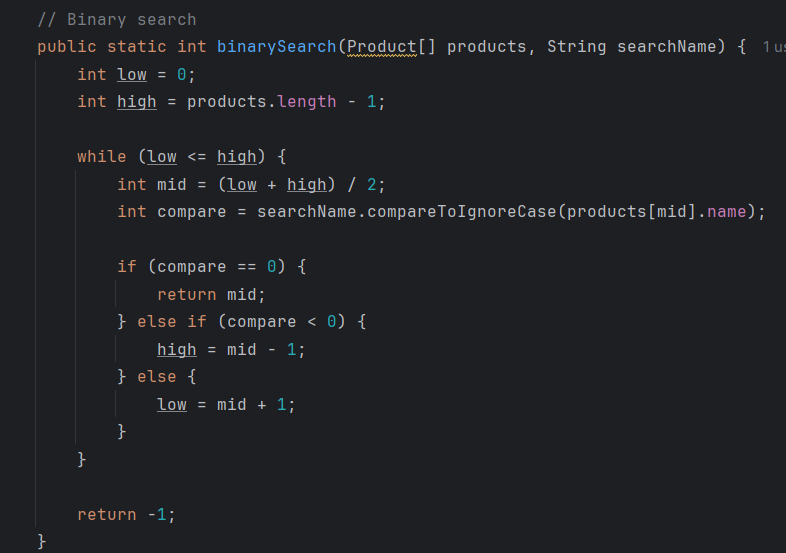


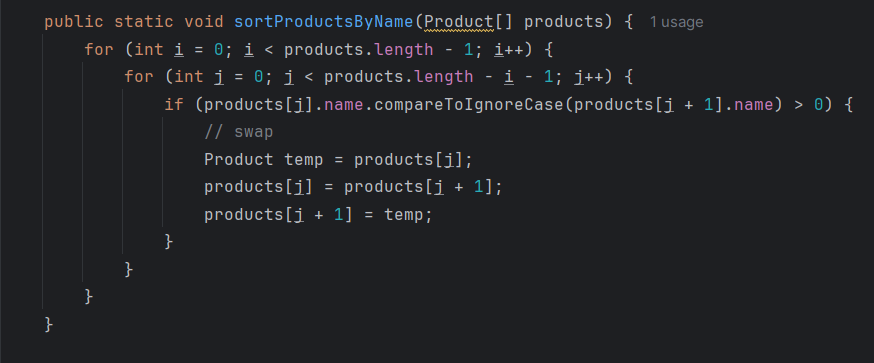
2.Describe the best, average, and worst-case scenarios for search operations.

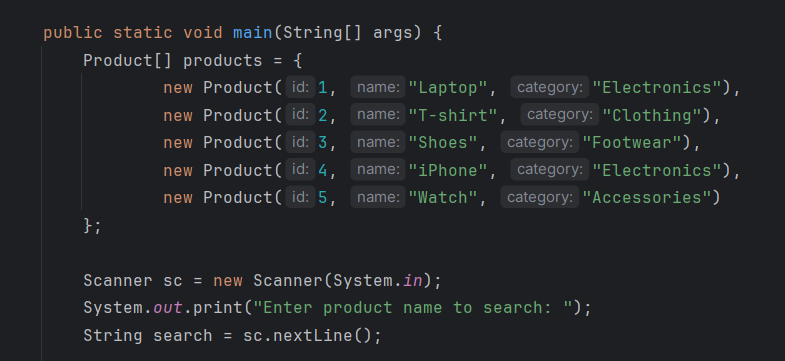
|  |  |  |
| --- | --- | --- |
| Case | Linear search | Binary search |
| Best | O(1) – First item | O(1) – Middle item |
| Average | O(n/2) → O(n) | O(log n) |
| Worst | O(n) – Last/not found | O(log n) – Not found |

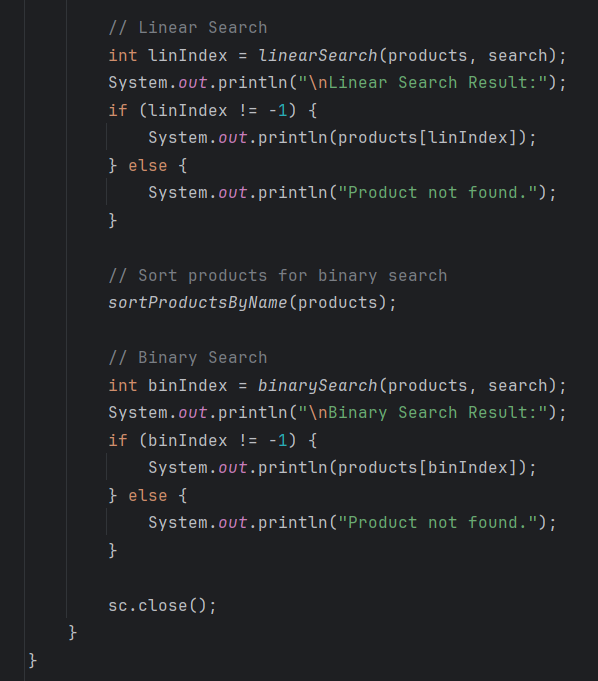




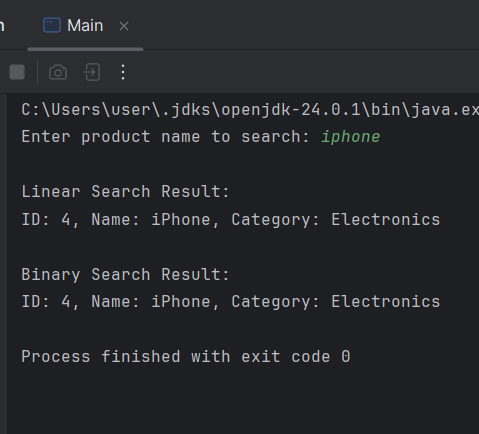








OUTPUT



Time complexity between linear search and binary search

|  |  |  |  |
| --- | --- | --- | --- |
| 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) |

3. Discuss which algorithm is more suitable for your platform and why  
Binary search is more suitable for an e-commerce platform because   
 \*It has faster search speed with O(log n) time  
\*Improved performance

**Exercise 7: Financial Forecasting**

Introduction to Recursion  
The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

* A recursive algorithm takes one step toward solution and then recursively call itself to further move. The algorithm stops once we reach the solution.
* Since called function may further call itself, this process might continue forever. So it is essential to provide a base case to terminate this recursion process.

### Example Idea:

Predict future value based on this formula:

FV=PV×(1+r)nFV = PV \times (1 + r)^nFV=PV×(1+r)n

Where:

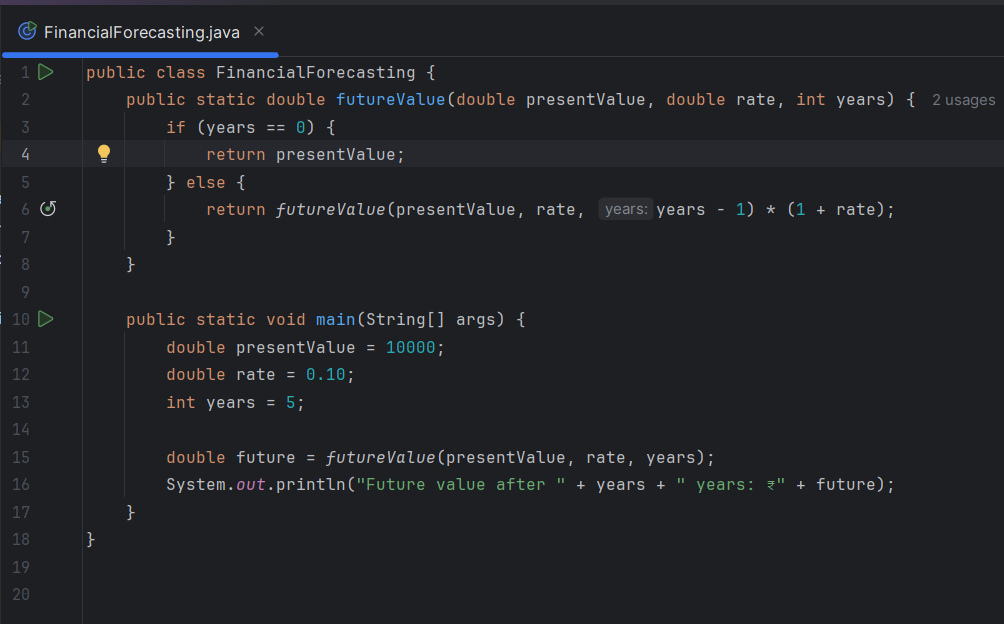
* FV = Future Value
* PV = Present Value
* r = Growth rate per period (as a decimal)
* n = Number of periods

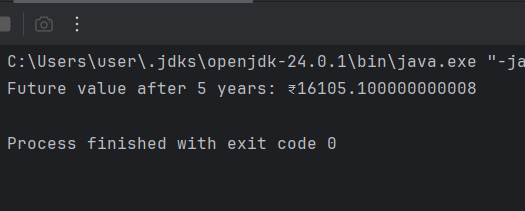
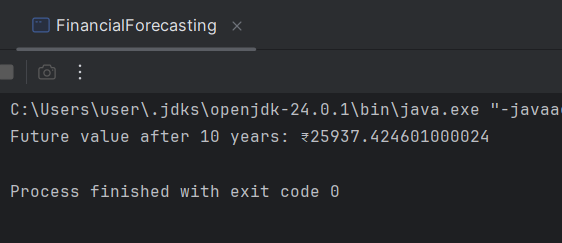
We can calculate this recursively:

FV(n)=FV(n−1)×(1+r)FV(n) = FV(n - 1) \times (1 + r)FV(n)=FV(n−1)×(1+r)

With base case:

* FV(0)=PVFV(0) = PVFV(0)=PV



OUTPUT  
 

### Time Complexity:

* Each recursive call reduces years by 1.
* So total calls = n, where n is the number of years.
* **Time Complexity**: **O(n)**
* **Space Complexity**: **O(n)** (due to call stack)

### Optimization Tips:

1. **Use Iteration Instead of Recursion**
2. Memoization