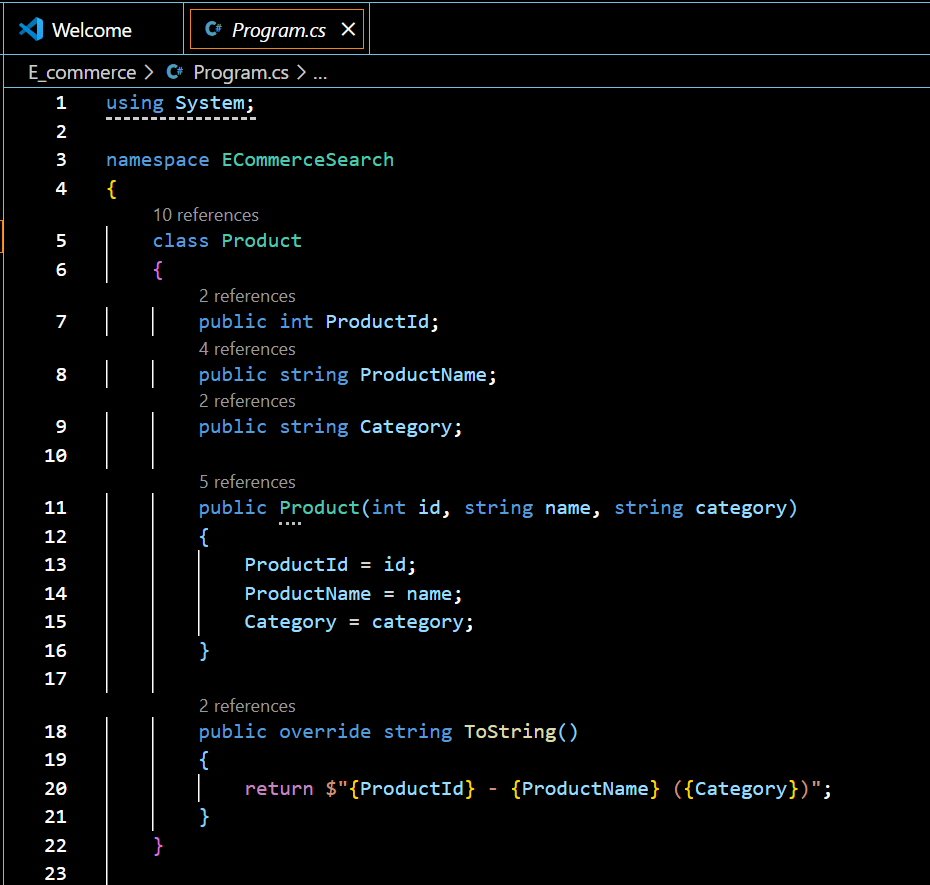
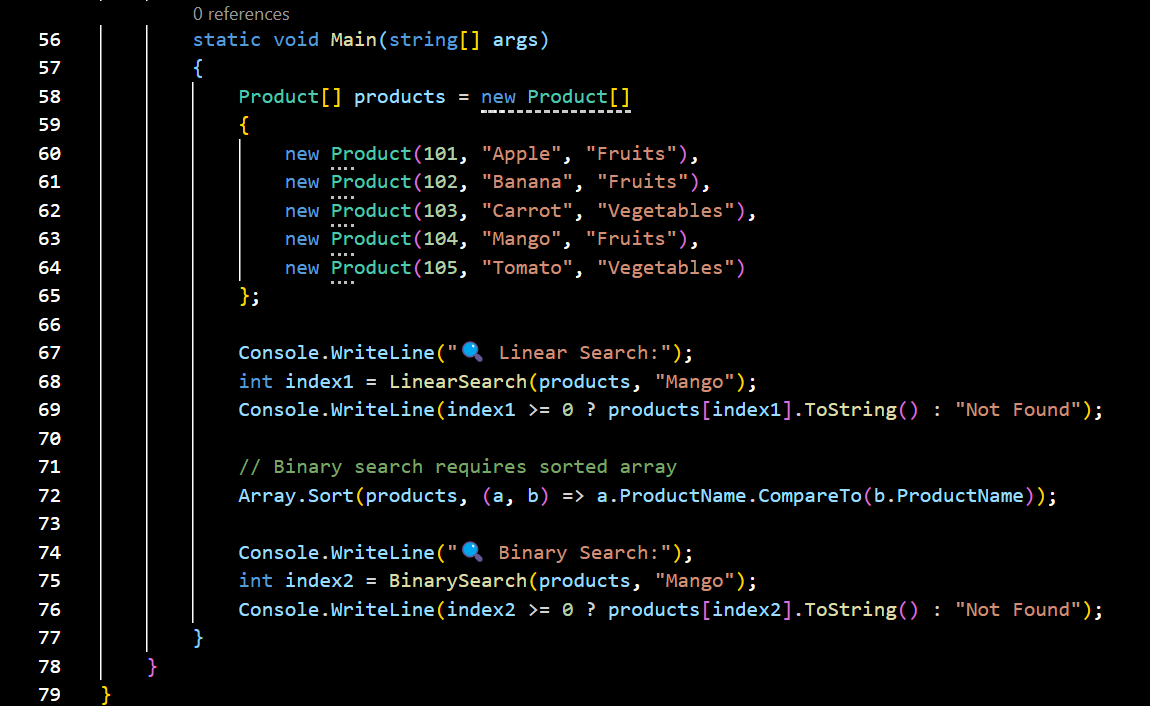
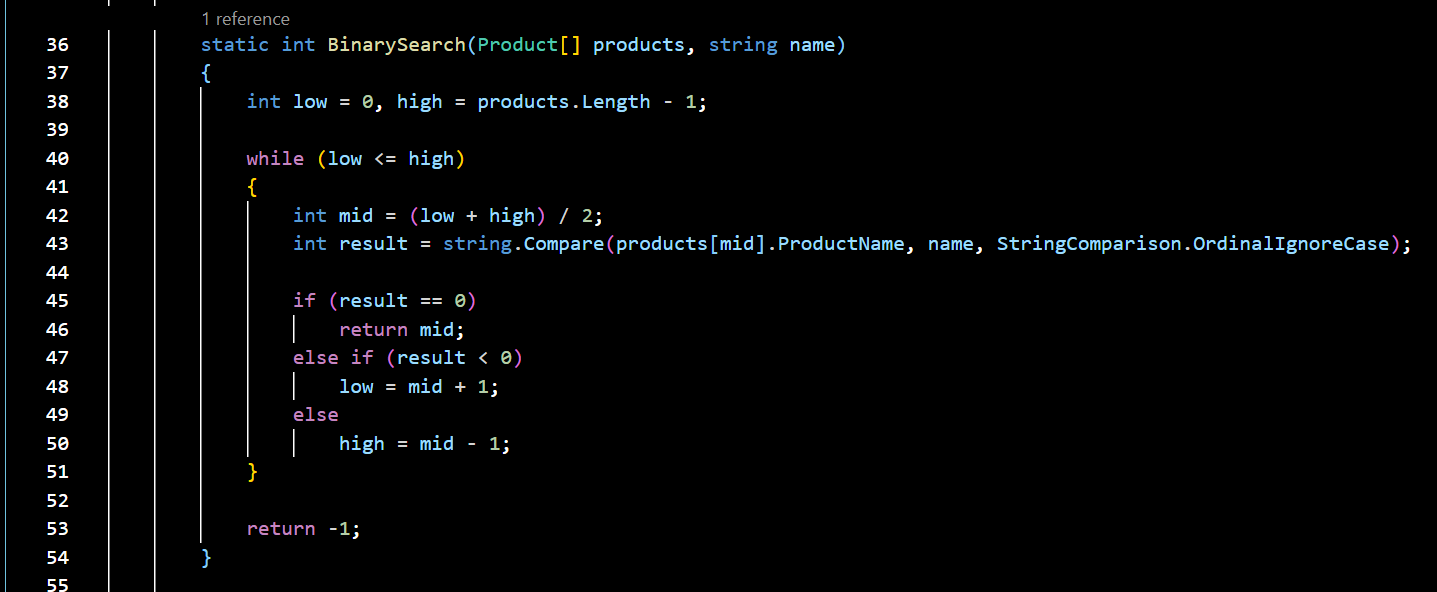
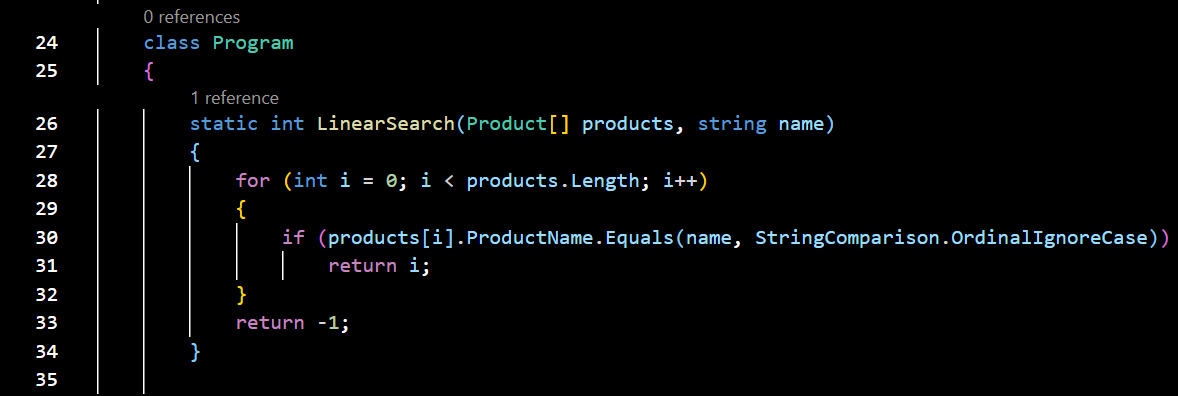
## **Exercise 2: E-commerce Platform Search Function**

### **1. Big O Notation Summary**

* **Big O notation** expresses the time complexity of algorithms as the input size grows.
* **Best, Average, Worst Case**:  
  + **Linear Search**:  
    - Best: O(1) (First item is the search target)
    - Average: O(n)
    - Worst: O(n)
  + **Binary Search** (on sorted data):  
    - Best: O(1)
    - Average: O(log n)
    - Worst: O(log n)

### **2. C# Code**

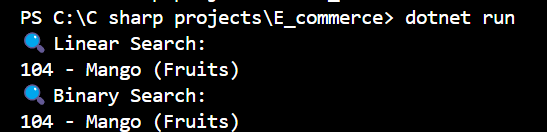




### **3.Analysis**

* **Linear search** has a time complexity of **O(n)**, scanning each element sequentially, making it inefficient for large datasets. **Binary search**, with **O(log n)** complexity, repeatedly divides the search space, requiring a **sorted array**. For an e-commerce platform with large, searchable product catalogs, **binary search is significantly faster and more suitable**.
* **Recommendation**: Use **binary search** on a sorted product list for performance.

### **4.Output**

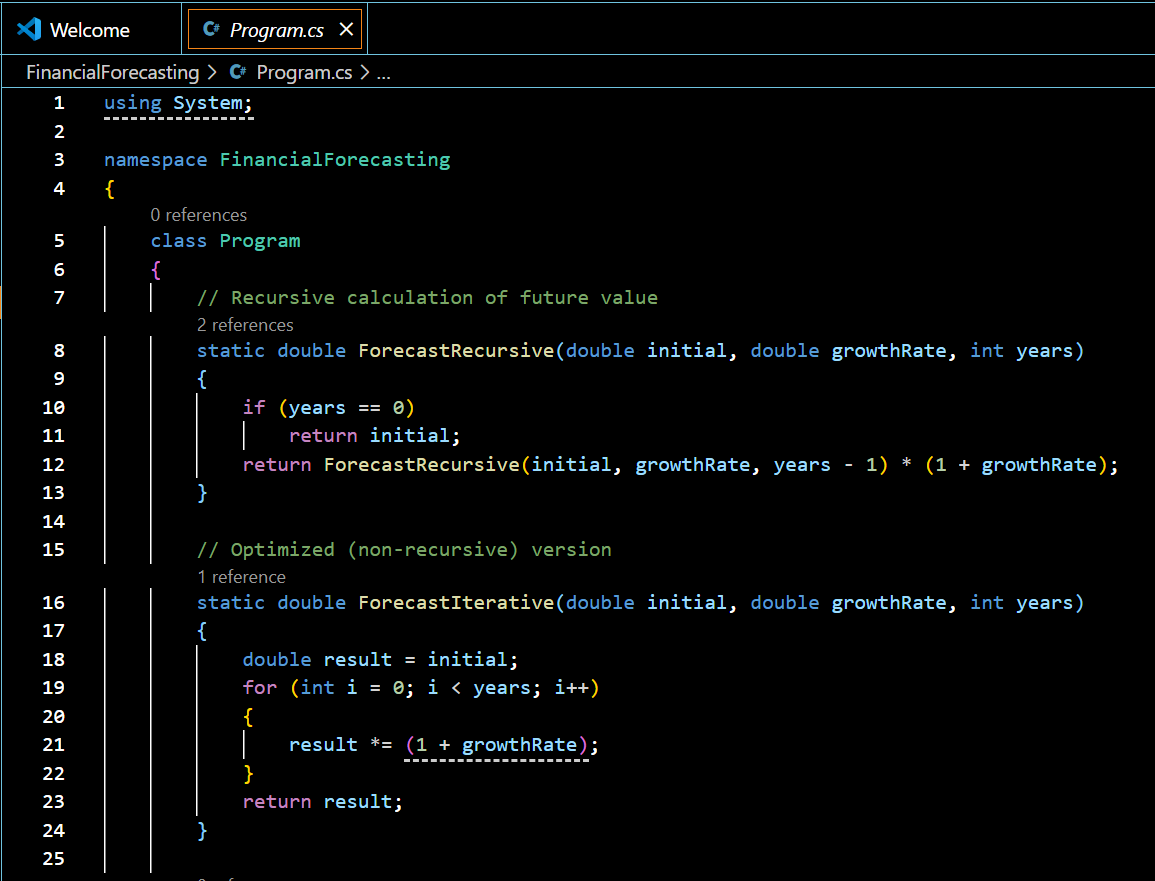


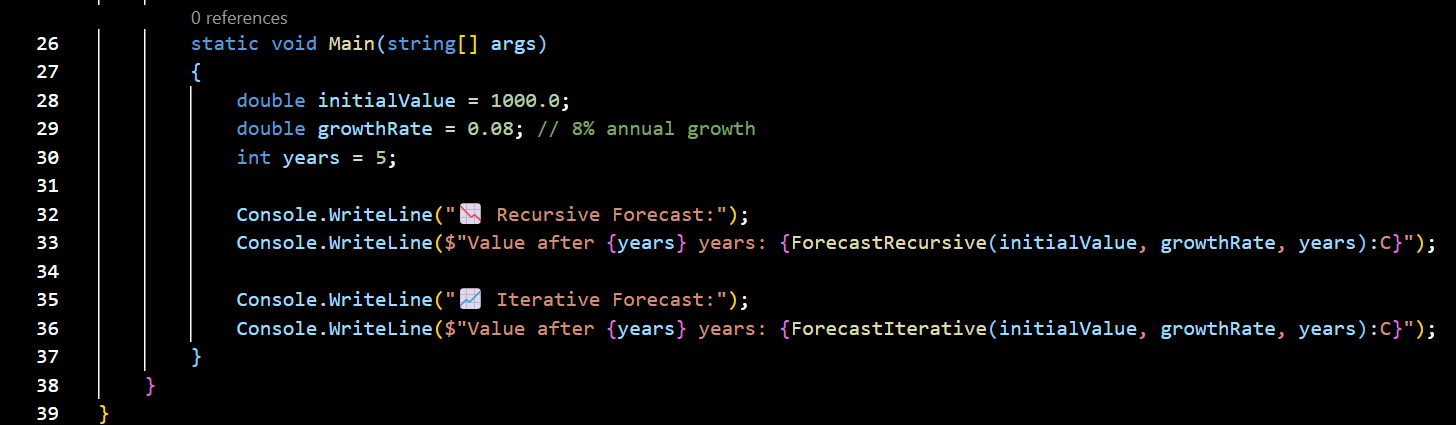
## **Exercise 7: Financial Forecasting**

### **1. Recursion Summary**

* **Recursion** is when a function calls itself to solve smaller instances of a problem.
* It's elegant for problems like **Fibonacci**, **growth models**, etc.
* But can be inefficient if repeated calls overlap.

### **2. C# Code**





### **3.Analysis**

* **Time Complexity (Recursive):** O(n) — calls itself n times.
* **Optimization Tip**:  
  + For simple multiplications like this, **iteration is faster and safer** (avoids stack overflow).
  + Use **memoization** or iterative logic for heavy computations.

### **4.Output**

