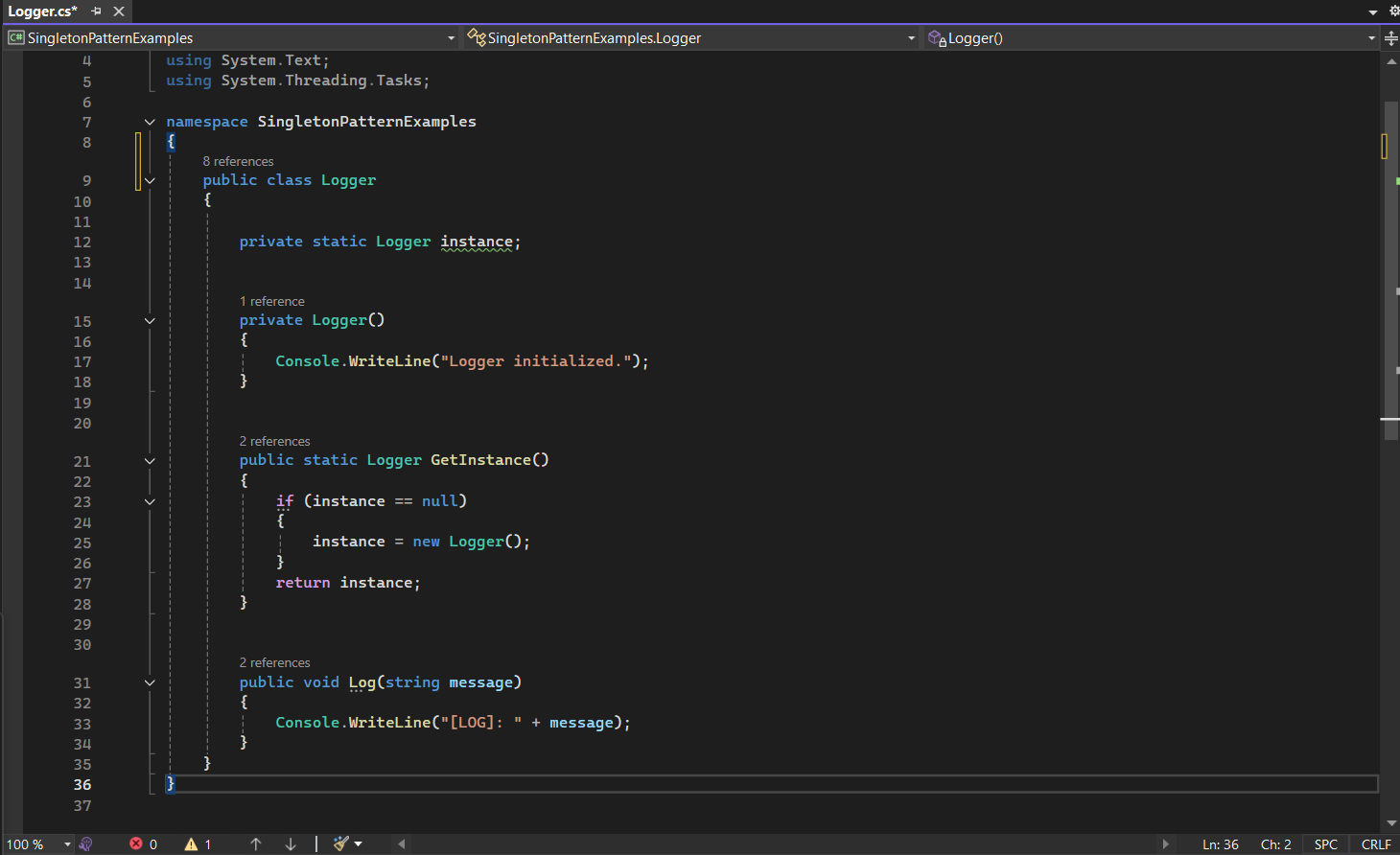
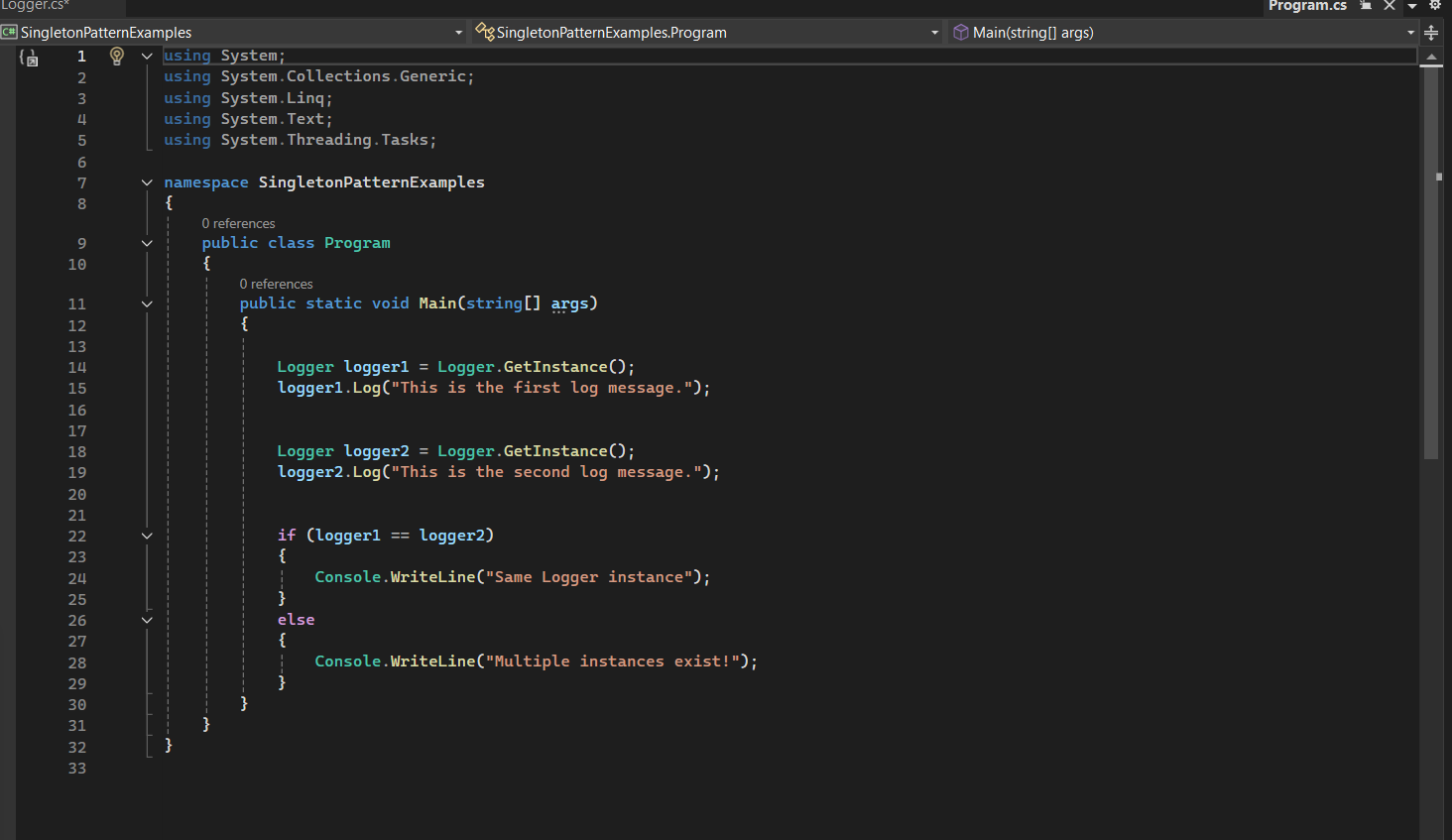
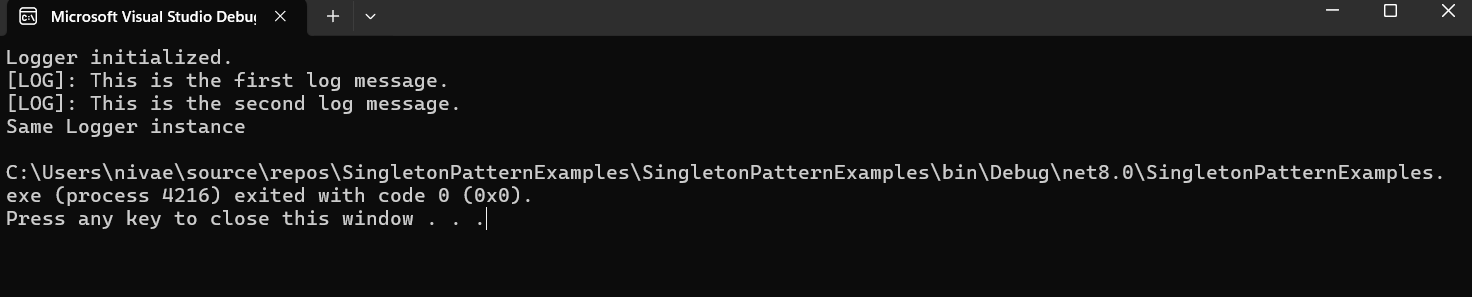
**Design principles & Patterns**

**Exercise 1: Implementing the Singleton Pattern**

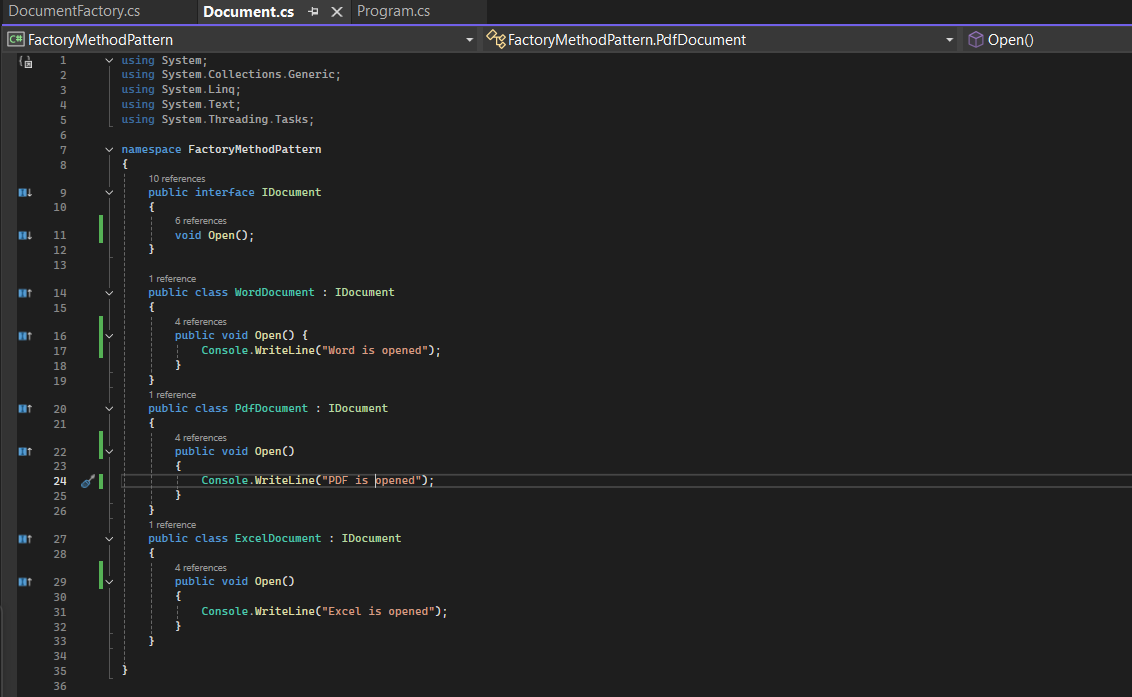


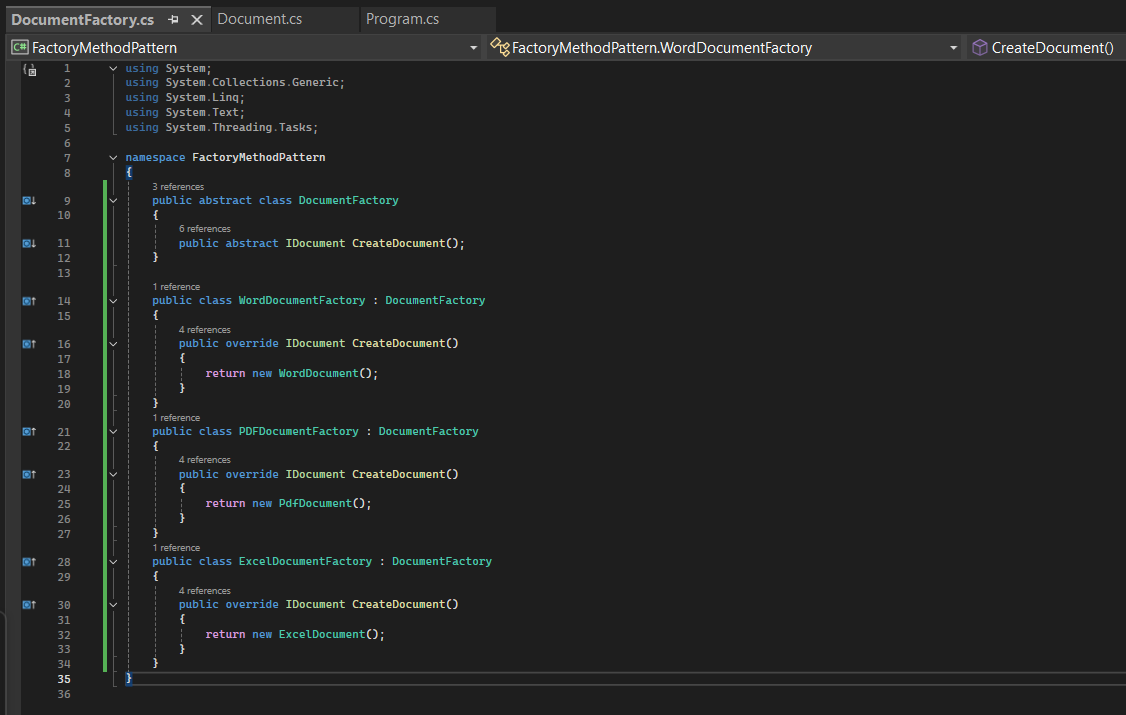


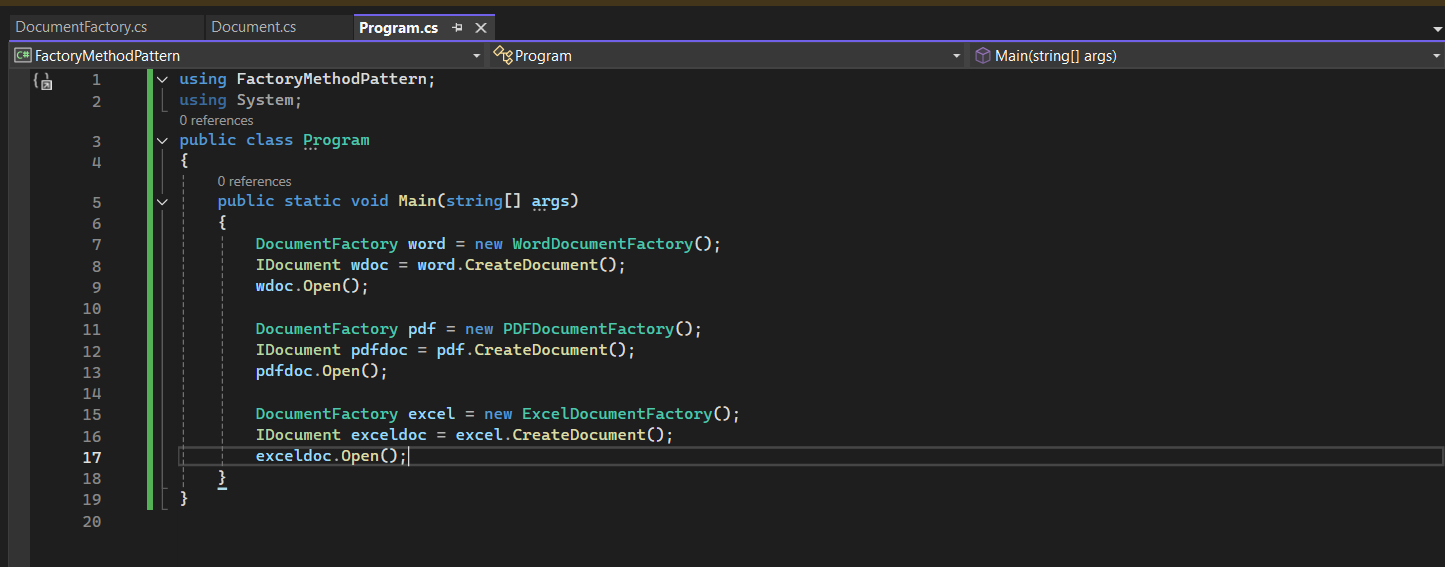
OUTPUT:



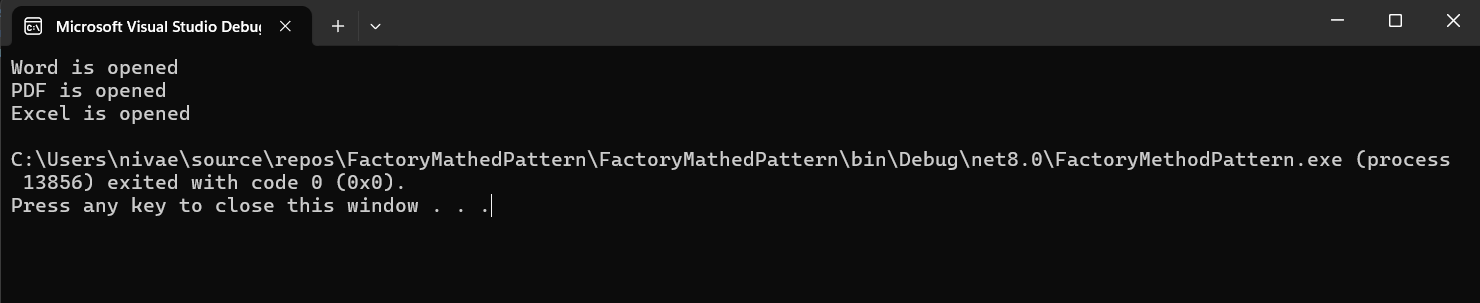
**Exercise 2: Implementing the Factory Method Pattern**







OUTPUT:



**Data structures and Algorithms**

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

Common Big O Notations:

O(1) – constant

O(n) – Linear

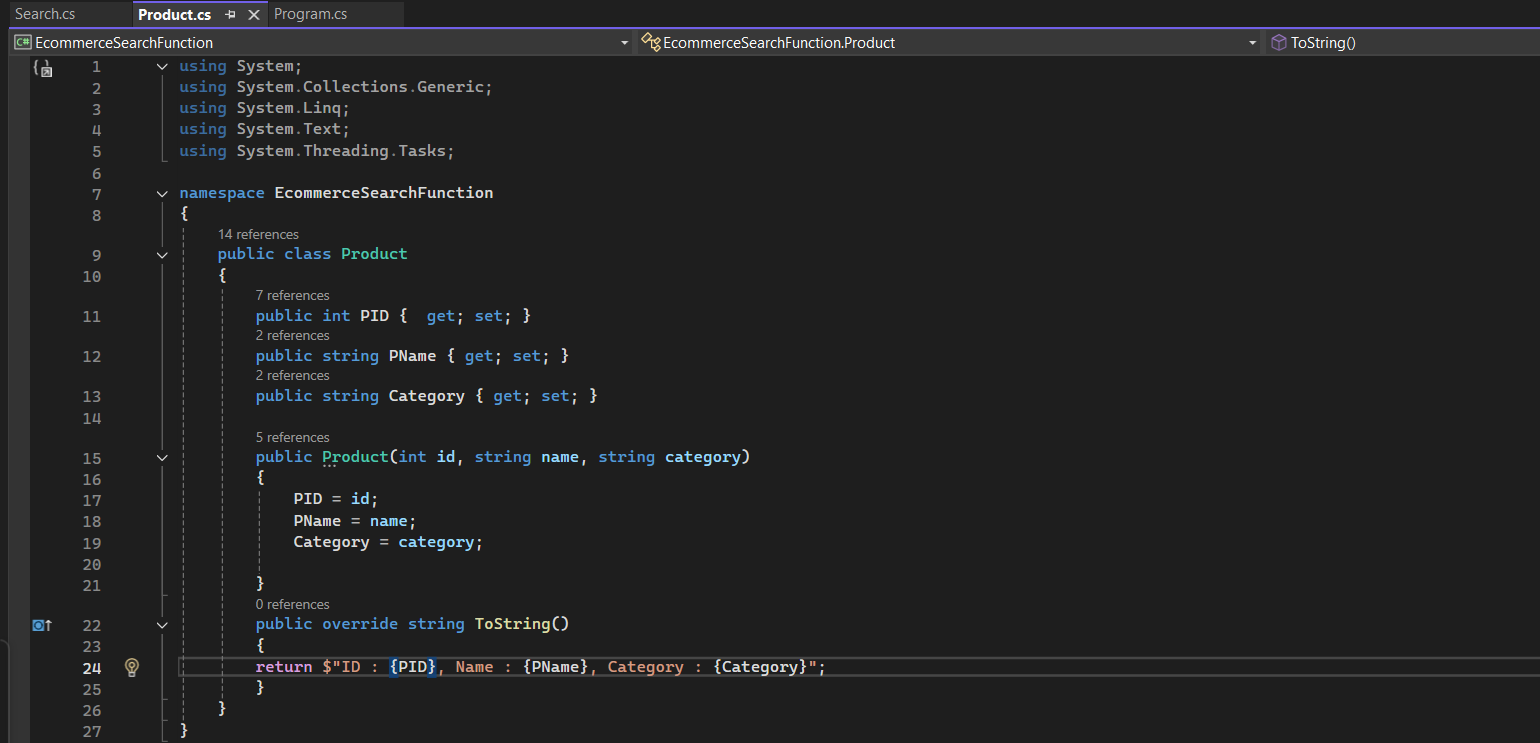
O(log n) – Logarithmic

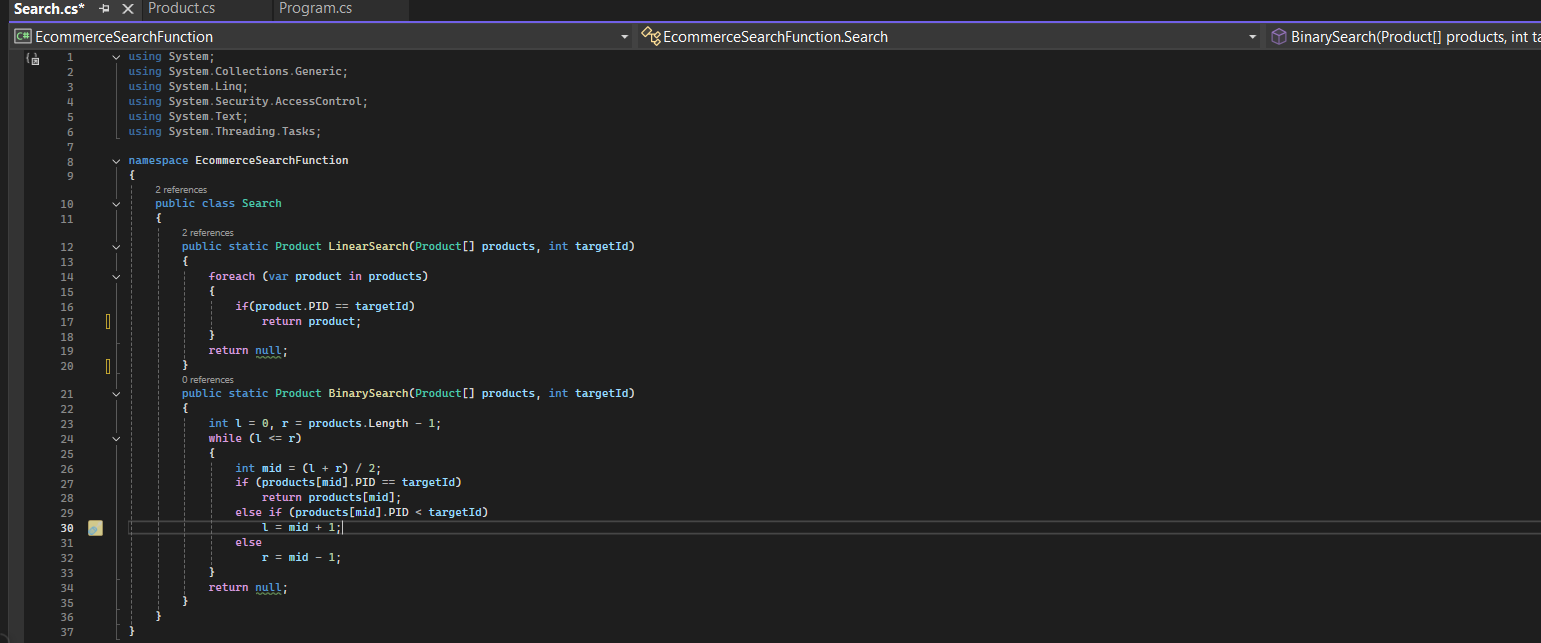
O(n log n)- Linearithmic

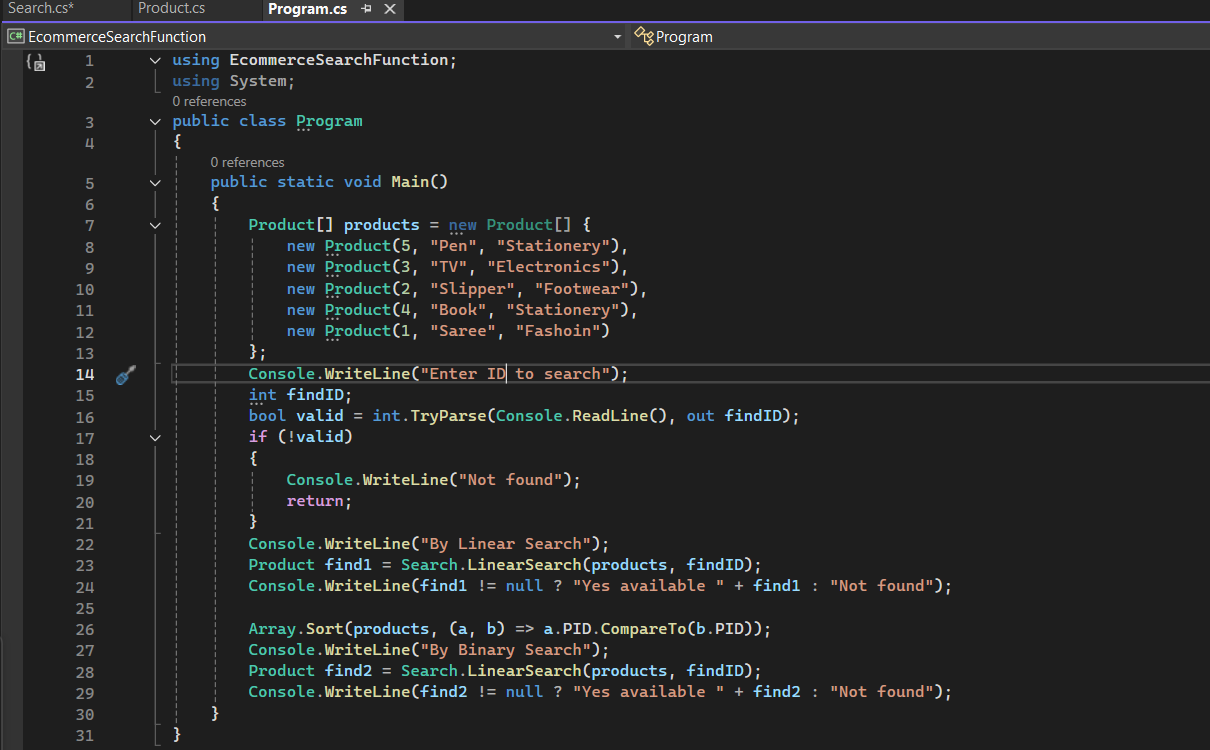
O(n^2) – Quadratic

|  |  |  |
| --- | --- | --- |
| **LINEAR SEARCH** | | |
| CASE | TIME | EXAMPLE |
| Best | O(1) | Found at first |
| Average | O(n) | In middle |
| Worst | O(n) | At last or not at all present |

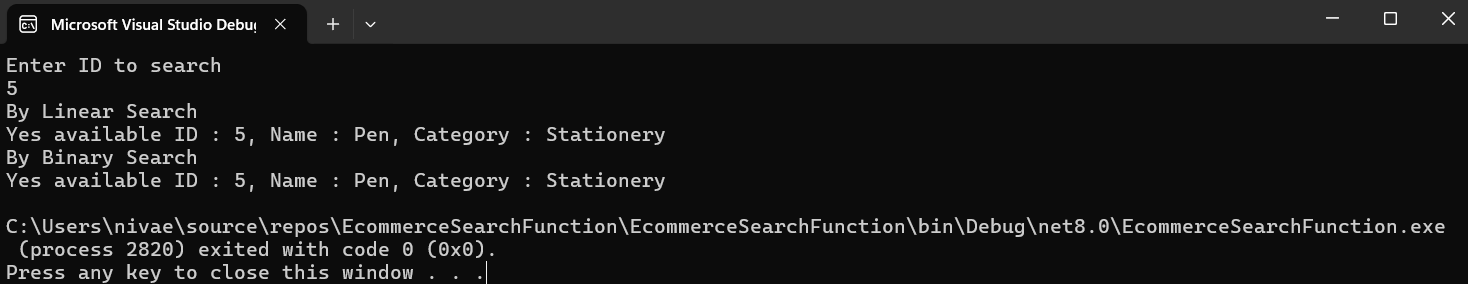
|  |  |  |
| --- | --- | --- |
| **BINARY SEARCH** | | |
| CASE | TIME | EXAMPLE |
| Best | O(1) | Found at first |
| Average | O( log n) | In middle |
| Worst | O(log n) | At last or not at all present |

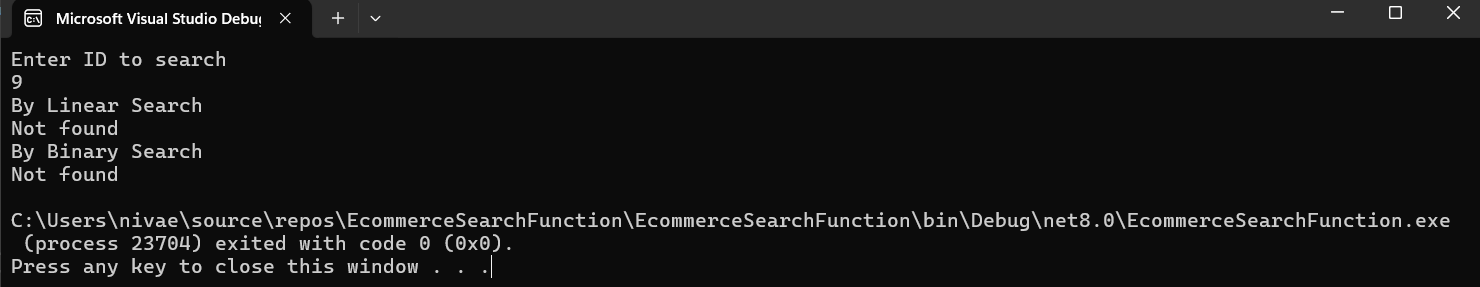
****

****

****

**OUTPUT:**

****

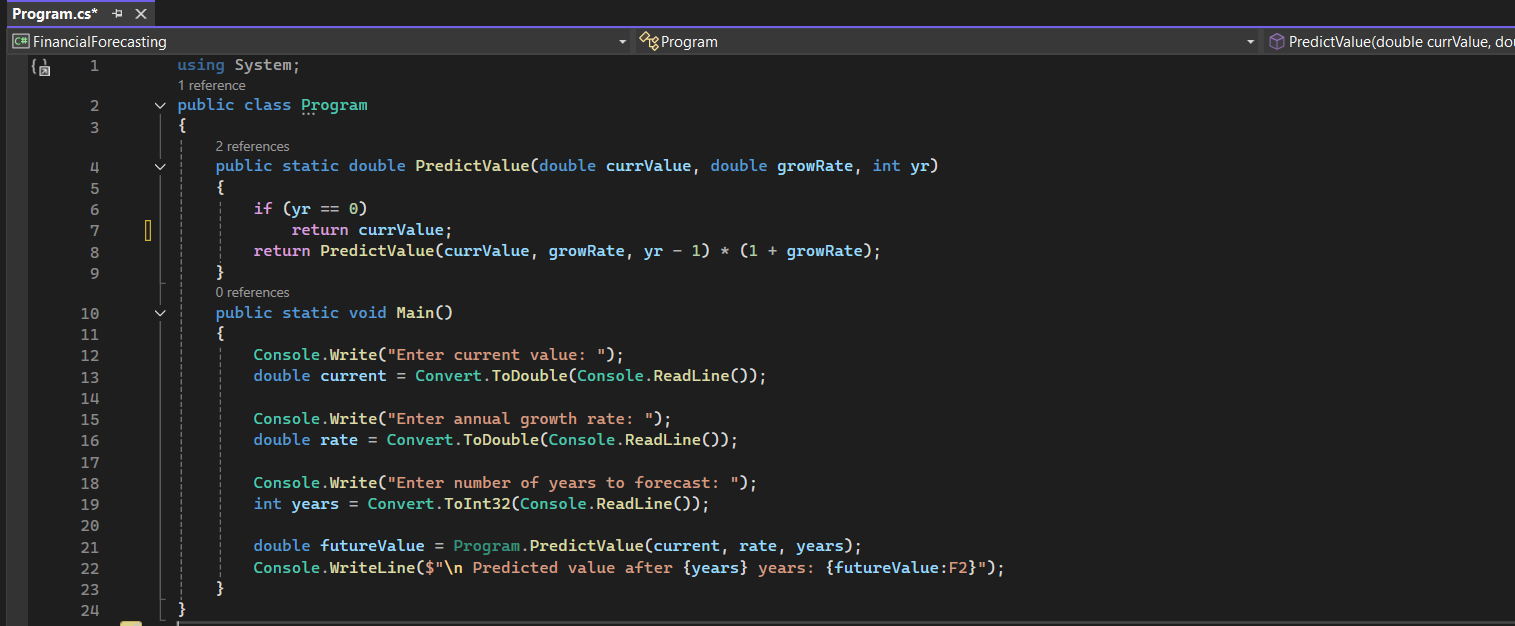
****

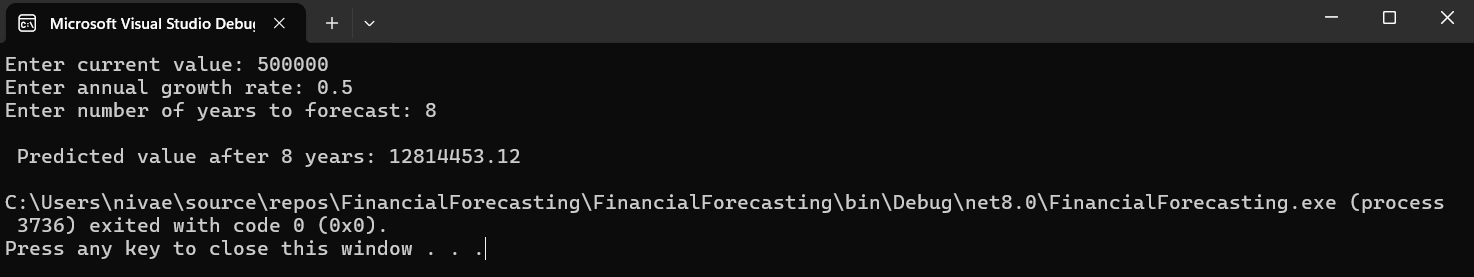
**Exercise 7: Financial Forecasting**

**RECURSION:**

A function that calls itself again and again till it meets a base case. The recursion stops to work when the condition in base case is satisfied.

It can be used repeat something again and again till some condition instead of writing individual function for each one.





**TIME COMPLEXITY : O(n)**