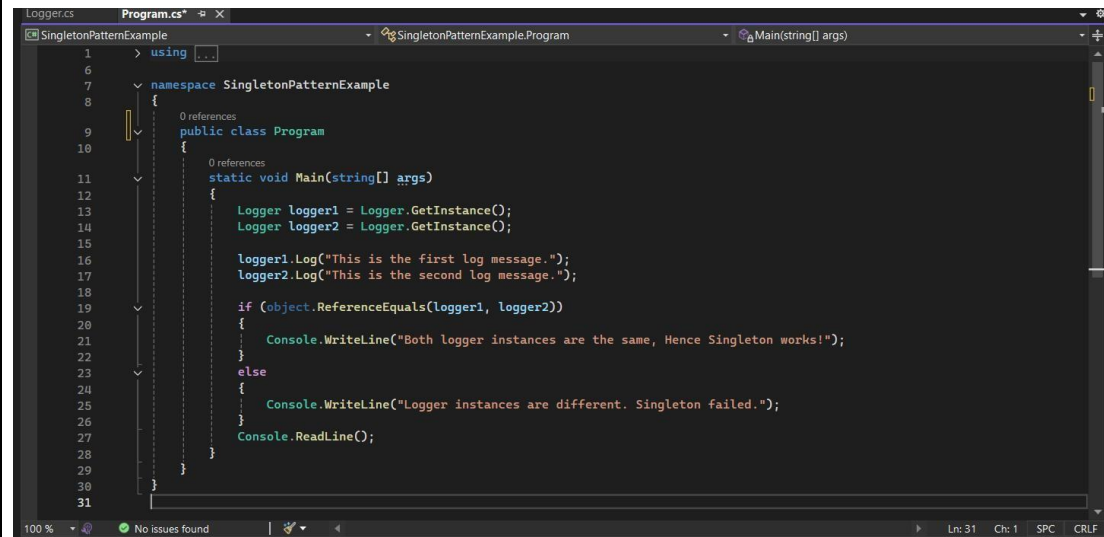


WEEK - 01 : HandsOn Solutions

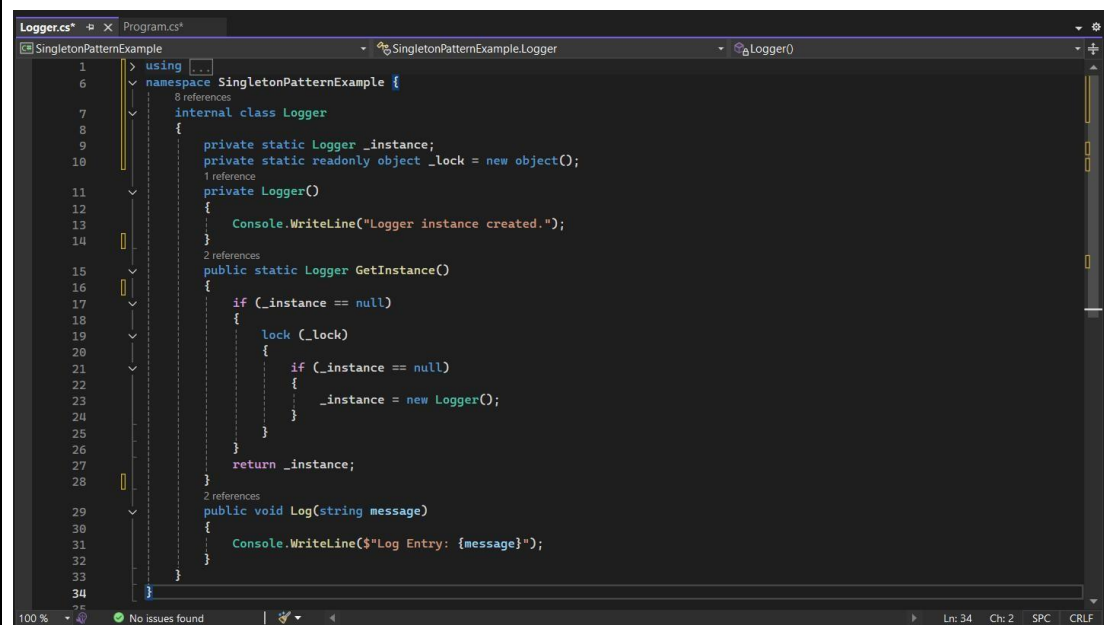
❖ Design Principles & Patterns :-

➤ Exercise - 1 : Implementing the Singleton Pattern.

Code :

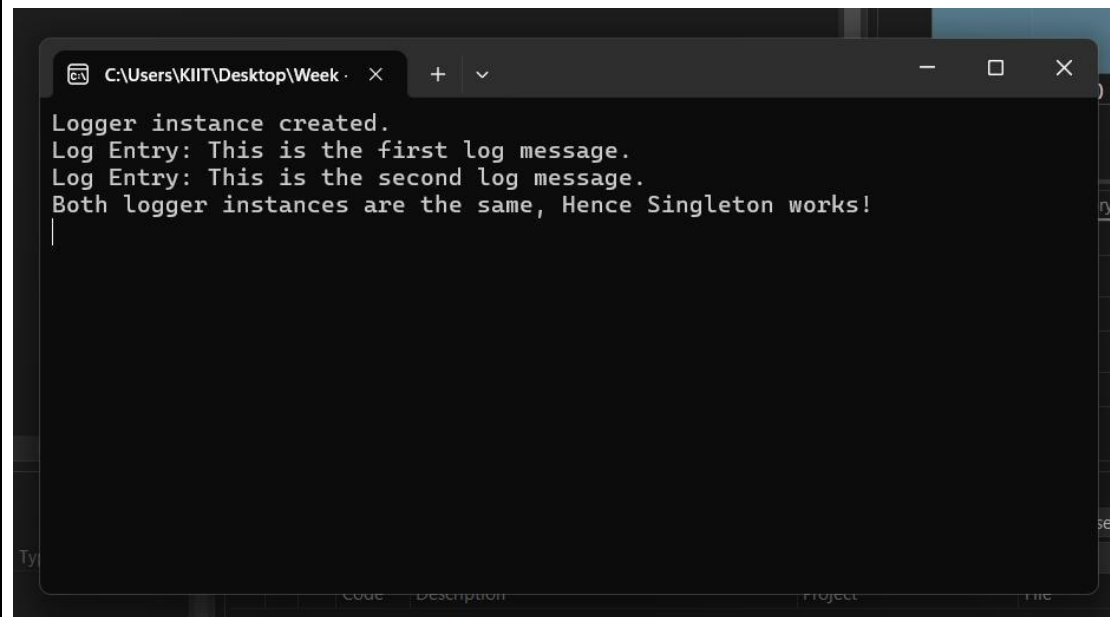


```
1  > using ...
6
7  namespace SingletonPatternExample
8  {
9      0 references
10     public class Program
11     {
12         0 references
13         static void Main(string[] args)
14         {
15             Logger logger1 = Logger.GetInstance();
16             Logger logger2 = Logger.GetInstance();
17
18             logger1.Log("This is the first log message.");
19             logger2.Log("This is the second log message.");
20
21             if (Object.ReferenceEquals(logger1, logger2))
22             {
23                 Console.WriteLine("Both logger instances are the same, Hence Singleton works!");
24             }
25             else
26             {
27                 Console.WriteLine("Logger instances are different. Singleton failed.");
28             }
29             Console.ReadLine();
30         }
31     }
```



```
1  > using ...
6  namespace SingletonPatternExample {
7      8 references
8      internal class Logger
9      {
10         private static Logger _instance;
11         private static readonly object _lock = new object();
12         1 reference
13         private Logger()
14         {
15             Console.WriteLine("Logger instance created.");
16         }
17         2 references
18         public static Logger GetInstance()
19         {
20             if (_instance == null)
21             {
22                 lock (_lock)
23                 {
24                     if (_instance == null)
25                     {
26                         _instance = new Logger();
27                     }
28                 }
29             }
30             return _instance;
31         }
32         2 references
33         public void Log(string message)
34         {
35             Console.WriteLine($"Log Entry: {message}");
36         }
37     }
```

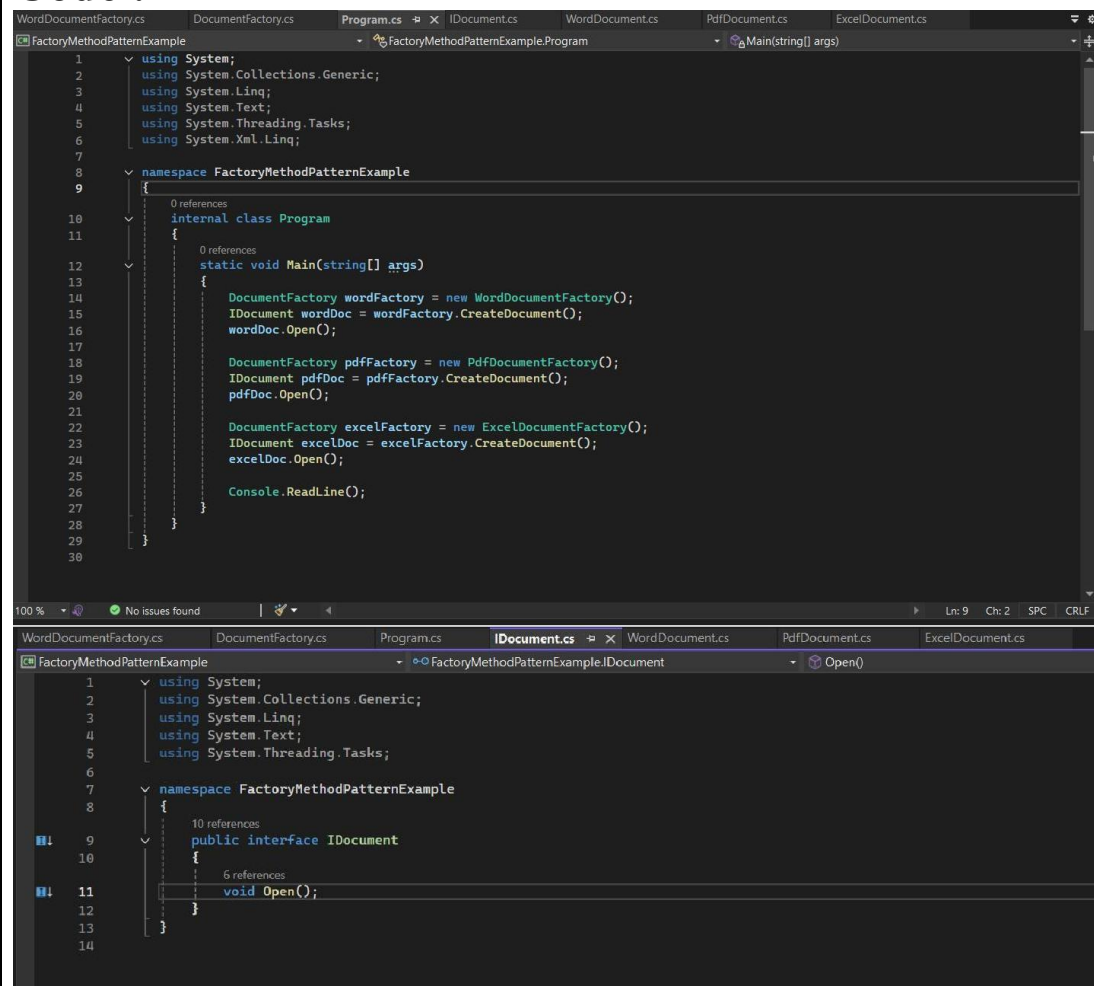
Output:



```
C:\Users\KIIT\Desktop\Week · X + v
Logger instance created.
Log Entry: This is the first log message.
Log Entry: This is the second log message.
Both logger instances are the same, Hence Singleton works!
```

➤ Exercise - 2 : Implementing the Factory Method Pattern

Code :



```
WordDocumentFactory.cs DocumentFactory.cs Program.cs IDocument.cs WordDocument.cs PdfDocument.cs ExcelDocument.cs
FactoryMethodPatternExample FactoryMethodPatternExample.Program Main(string[] args)
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.Xml.Linq;
7
8 namespace FactoryMethodPatternExample
9 {
10     0 references
11     internal class Program
12     {
13         0 references
14         static void Main(string[] args)
15         {
16             DocumentFactory wordFactory = new WordDocumentFactory();
17             IDocument wordDoc = wordFactory.CreateDocument();
18             wordDoc.Open();
19
20             DocumentFactory pdfFactory = new PdfDocumentFactory();
21             IDocument pdfDoc = pdfFactory.CreateDocument();
22             pdfDoc.Open();
23
24             DocumentFactory excelFactory = new ExcelDocumentFactory();
25             IDocument excelDoc = excelFactory.CreateDocument();
26             excelDoc.Open();
27
28             Console.ReadLine();
29         }
30     }
31 }
100 % No issues found Ln: 9 Ch: 2 SPC CRLF
WordDocumentFactory.cs DocumentFactory.cs Program.cs IDocument.cs WordDocument.cs PdfDocument.cs ExcelDocument.cs
FactoryMethodPatternExample FactoryMethodPatternExample.IDocument Open()
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace FactoryMethodPatternExample
8 {
9     10 references
10     public interface IDocument
11     {
12         6 references
13         void Open();
14     }
15 }
```

```
WordDocumentFactory.cs DocumentFactory.cs Program.cs IDocument.cs WordDocument.cs PdfDocument.cs ExcelDocument.cs
FactoryMethodPatternExample FactoryMethodPatternExample.DocumentFactory CreateDocument()
1 namespace FactoryMethodPatternExample
2 {
3     6 references
4     public abstract class DocumentFactory
5     {
6     6 references
7     public abstract IDocument CreateDocument();
8 }
9 }
```

```
PdfDocument.cs ExcelDocumentFactory.cs IDocument.cs PdfDocumentFactory.cs Program.cs WordDocument.cs WordDocumentFactory.cs
FactoryMethodPatternExample FactoryMethodPatternExample.PdfDocumentFactory CreateDocument()
1 namespace FactoryMethodPatternExample
2 {
3     1 reference
4     public class PdfDocumentFactory : DocumentFactory
5     {
6     4 references
7     public override IDocument CreateDocument()
8     {
9     return new PdfDocument();
10 }
11 }
```

```
WordDocumentFactory.cs DocumentFactory.cs Program.cs IDocument.cs WordDocument.cs PdfDocument.cs ExcelDocument.cs
FactoryMethodPatternExample FactoryMethodPatternExample.WordDocumentFactory CreateDocument()
1 namespace FactoryMethodPatternExample
2 {
3     1 reference
4     public class WordDocumentFactory : DocumentFactory
5     {
6     4 references
7     public override IDocument CreateDocument()
8     {
9     return new WordDocument();
10 }
11 }
```

```
PdfDocument.cs PdfDocumentFactory.cs Program.cs WordDocument.cs WordDocumentFactory.cs ExcelDocument.cs ExcelDocumentFactory.cs
FactoryMethodPatternExample FactoryMethodPatternExample.ExcelDocumentFactory CreateDocument()
1 namespace FactoryMethodPatternExample
2 {
3     1 reference
4     public class ExcelDocumentFactory : DocumentFactory
5     {
6     4 references
7     public override IDocument CreateDocument()
8     {
9     return new ExcelDocument();
10 }
11 }
```

```
WordDocumentFactory.cs DocumentFactory.cs Program.cs IDocument.cs WordDocument.cs PdfDocument.cs ExcelDocument.cs
FactoryMethodPatternExample FactoryMethodPatternExample.PdfDocument Open()
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace FactoryMethodPatternExample
8 {
9     1 reference
10    public class PdfDocument : IDocument
11    {
12    4 references
13    public void Open()
14    {
15    Console.WriteLine("Opening a PDF document.");
16    }
17 }
```


❖ Data Structures & Algorithms :-

➤ Exercise - 2 : E-commerce Platform Search Function

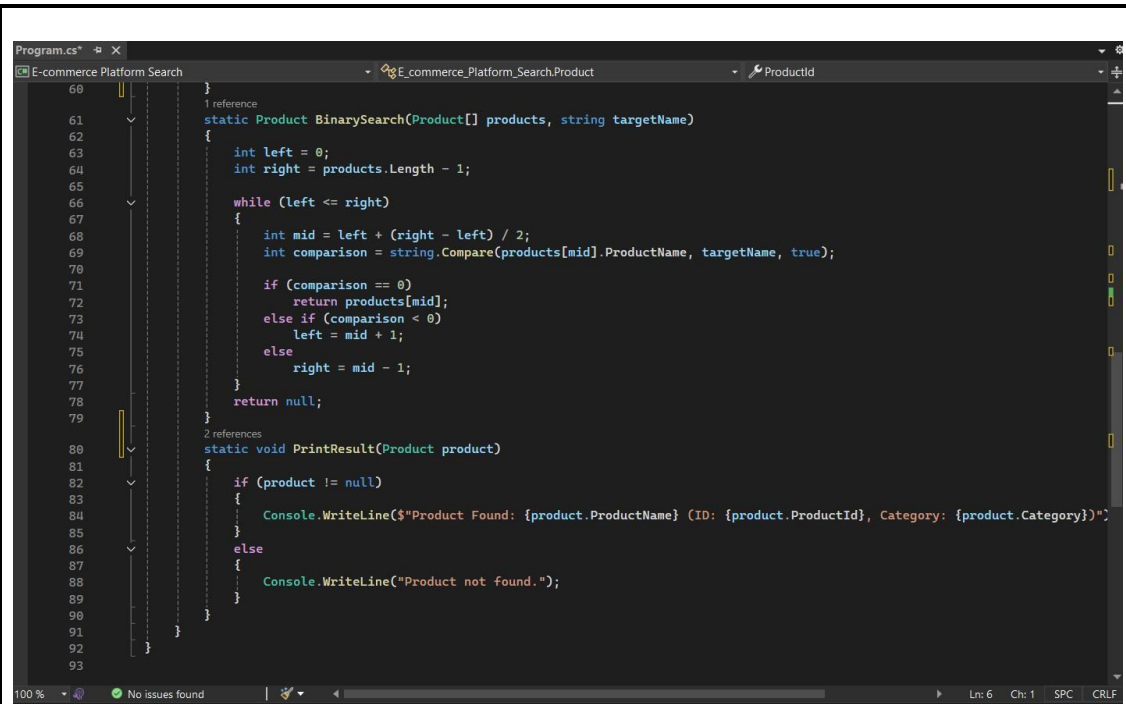
Code:

```
Program.cs x
E-commerce Platform Search  E_commerce_Platform_Search.Product  ProductId

1  > using ...
2
3
4
5
6  namespace E_commerce_Platform_Search
7  {
8
9      13 references
10     public class Product
11     {
12         2 references
13         public int ProductId { get; set; }
14         6 references
15         public string ProductName { get; set; }
16         2 references
17         public string Category { get; set; }
18
19         5 references
20         public Product(int productId, string productName, string category)
21         {
22             ProductId = productId;
23             ProductName = productName;
24             Category = category;
25         }
26     }
27
28     0 references
29     class Program
30     {
31         0 references
32         static void Main(string[] args)
33         {
34             Product[] products = new Product[]
35             {
36                 new Product(1, "Laptop", "Electronics"),
37                 new Product(2, "Phone", "Electronics"),
38                 new Product(3, "Shoes", "Fashion"),
39                 new Product(4, "Book", "Education"),
40                 new Product(5, "Tablet", "Electronics")
41             };
42         }
43     }
44
45 100 %  No issues found  Ln: 6  Ch: 1  SPC  CRLF
```

```
Program.cs x
E-commerce Platform Search  E_commerce_Platform_Search.Product  ProductId

34
35  Console.WriteLine("Enter product name to search: ");
36  string searchQuery = Console.ReadLine();
37
38  Console.WriteLine("\n=== Linear Search result ===");
39  var result1 = LinearSearch(products, searchQuery);
40  PrintResult(result1);
41
42  Array.Sort(products, (a, b) => string.Compare(a.ProductName, b.ProductName, StringComparison.OrdinalIgnoreCase));
43
44  Console.WriteLine("\n=== Binary Search result ===");
45  var result2 = BinarySearch(products, searchQuery);
46  PrintResult(result2);
47
48  Console.ReadKey();
49
50  1 reference
51  static Product LinearSearch(Product[] products, string targetName)
52  {
53      foreach (var product in products)
54      {
55          if (product.ProductName.Equals(targetName, StringComparison.OrdinalIgnoreCase))
56          {
57              return product;
58          }
59      }
60      return null;
61
62  1 reference
63  static Product BinarySearch(Product[] products, string targetName)
64  {
65      int left = 0;
66      int right = products.Length - 1;
67
68      while (left <= right)
69      {
70          int mid = left + (right - left) / 2;
71
72          if (products[mid].ProductName.Equals(targetName, StringComparison.OrdinalIgnoreCase))
73          {
74              return products[mid];
75          }
76          else if (products[mid].ProductName < targetName)
77          {
78              left = mid + 1;
79          }
80          else
81          {
82              right = mid - 1;
83          }
84      }
85      return null;
86  }
87
88 100 %  No issues found  Ln: 6  Ch: 1  SPC  CRLF
```

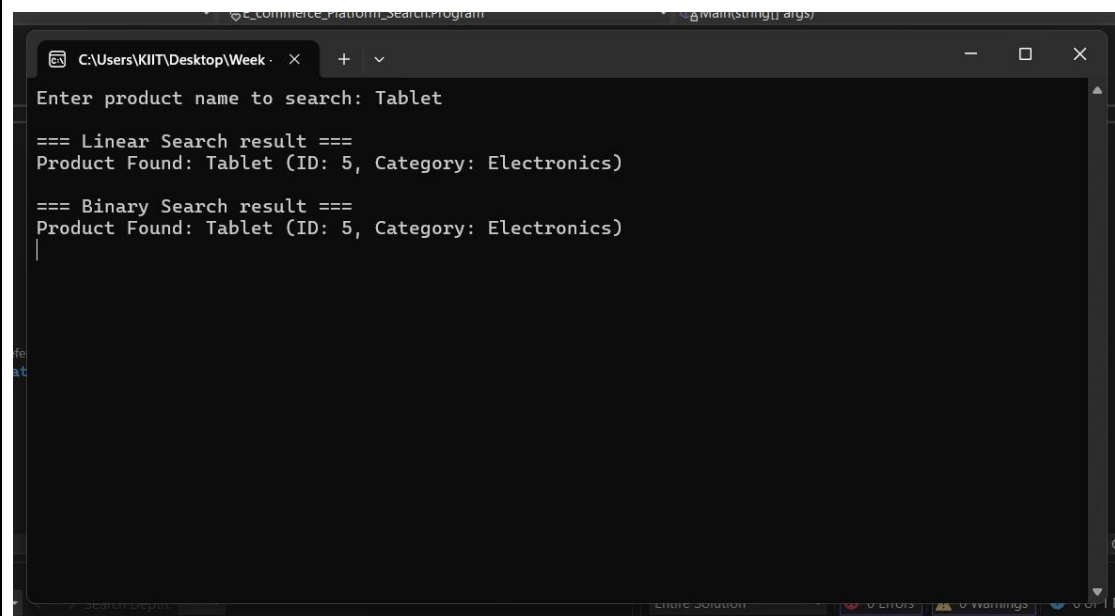


```
Program.cs - E-commerce Platform Search
1 reference
static Product BinarySearch(Product[] products, string targetName)
{
    int left = 0;
    int right = products.Length - 1;

    while (left <= right)
    {
        int mid = left + (right - left) / 2;
        int comparison = string.Compare(products[mid].ProductName, targetName, true);

        if (comparison == 0)
            return products[mid];
        else if (comparison < 0)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return null;
}
2 references
static void PrintResult(Product product)
{
    if (product != null)
    {
        Console.WriteLine($"Product Found: {product.ProductName} (ID: {product.ProductId}, Category: {product.Category})");
    }
    else
    {
        Console.WriteLine("Product not found.");
    }
}
```

Output :



```
C:\Users\KIIT\Desktop\Week...
Enter product name to search: Tablet

=== Linear Search result ===
Product Found: Tablet (ID: 5, Category: Electronics)

=== Binary Search result ===
Product Found: Tablet (ID: 5, Category: Electronics)
```

Time Complexity analysis :

Algorithm	Best Case	Average Case	Worst Case
Linear Search	$O(1)$	$O(n)$	$O(n)$
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$

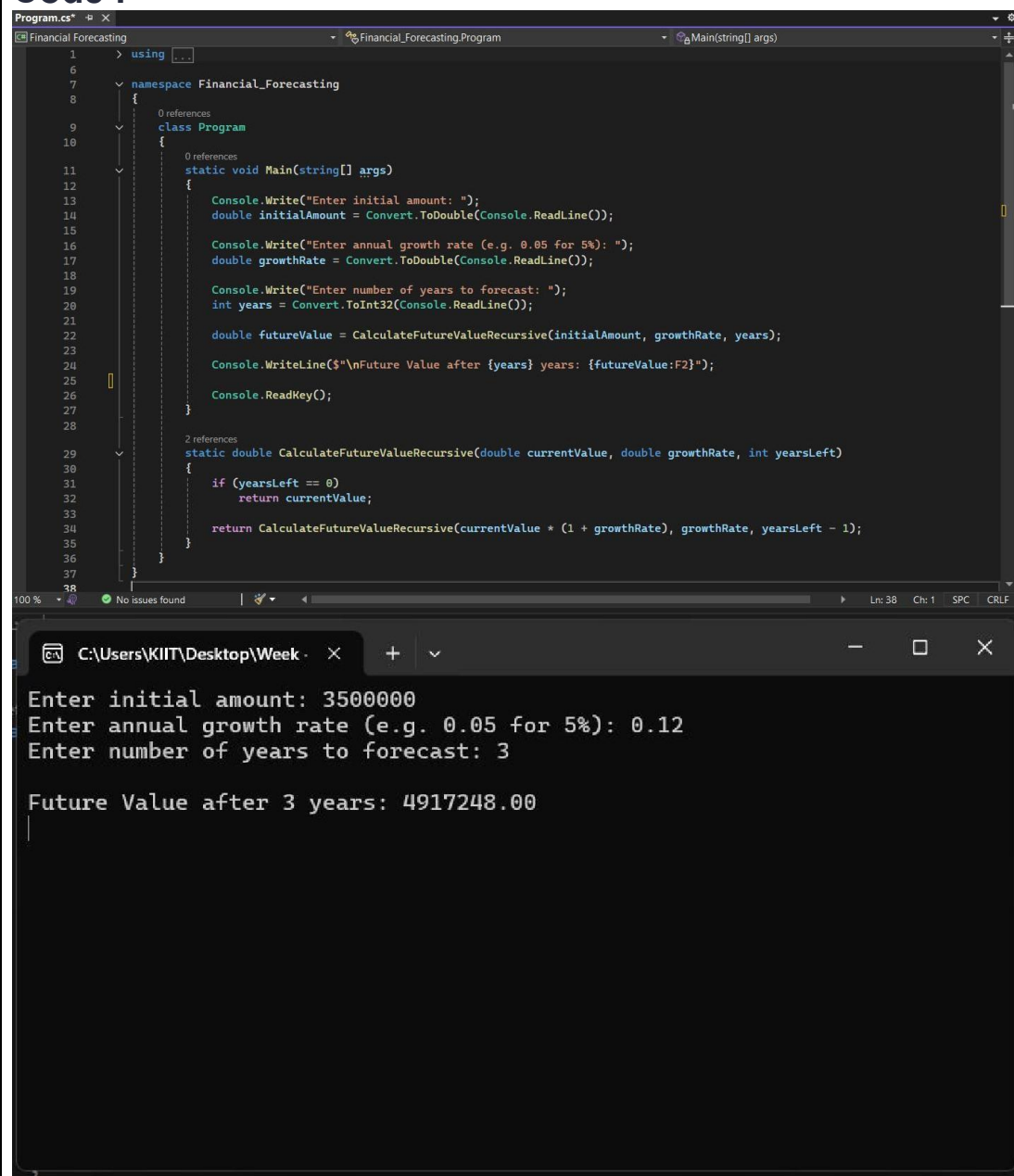
Which Algo is best suitable ?

The algorithm that is more suitable for our platform is Binary Search.

The main reason why binary search is more suitable for our e-commerce platform because , It provides logarithmic time performance ($O(\log n)$), which is essential for scaling to thousands or millions of products. It also works well with sorted, indexed product catalogs, which is typical in backend databases or caching systems. It's more aligned with the needs of a high-performance, real-time search system.

➤ Exercise - 7 : Financial Forecasting

Code :



The image shows a C# program in Visual Studio and its execution output. The program, named 'Financial_Forecasting', uses a recursive method to calculate the future value of an investment based on an initial amount, an annual growth rate, and a number of years. The console output shows the user inputting an initial amount of 3500000, a growth rate of 0.12, and 3 years, resulting in a future value of 4917248.00.

```
Program.cs* x
Financial_Forecasting Financial_Forecasting.Program Main(string[] args)
1 using ...
6
7 namespace Financial_Forecasting
8 {
9     0 references
10    class Program
11    {
12        0 references
13        static void Main(string[] args)
14        {
15            Console.WriteLine("Enter initial amount: ");
16            double initialAmount = Convert.ToDouble(Console.ReadLine());
17            Console.WriteLine("Enter annual growth rate (e.g. 0.05 for 5%): ");
18            double growthRate = Convert.ToDouble(Console.ReadLine());
19            Console.WriteLine("Enter number of years to forecast: ");
20            int years = Convert.ToInt32(Console.ReadLine());
21            double futureValue = CalculateFutureValueRecursive(initialAmount, growthRate, years);
22            Console.WriteLine($"Future Value after {years} years: {futureValue:F2}");
23            Console.ReadKey();
24        }
25    }
26    2 references
27    static double CalculateFutureValueRecursive(double currentValue, double growthRate, int yearsLeft)
28    {
29        if (yearsLeft == 0)
30            return currentValue;
31        return CalculateFutureValueRecursive(currentValue * (1 + growthRate), growthRate, yearsLeft - 1);
32    }
33 }
34
35
36
37
38
100 % No issues found Ln: 38 Ch: 1 SPC CRLF
C:\Users\KIIT\Desktop\Week - x + v
Enter initial amount: 3500000
Enter annual growth rate (e.g. 0.05 for 5%): 0.12
Enter number of years to forecast: 3
Future Value after 3 years: 4917248.00
```

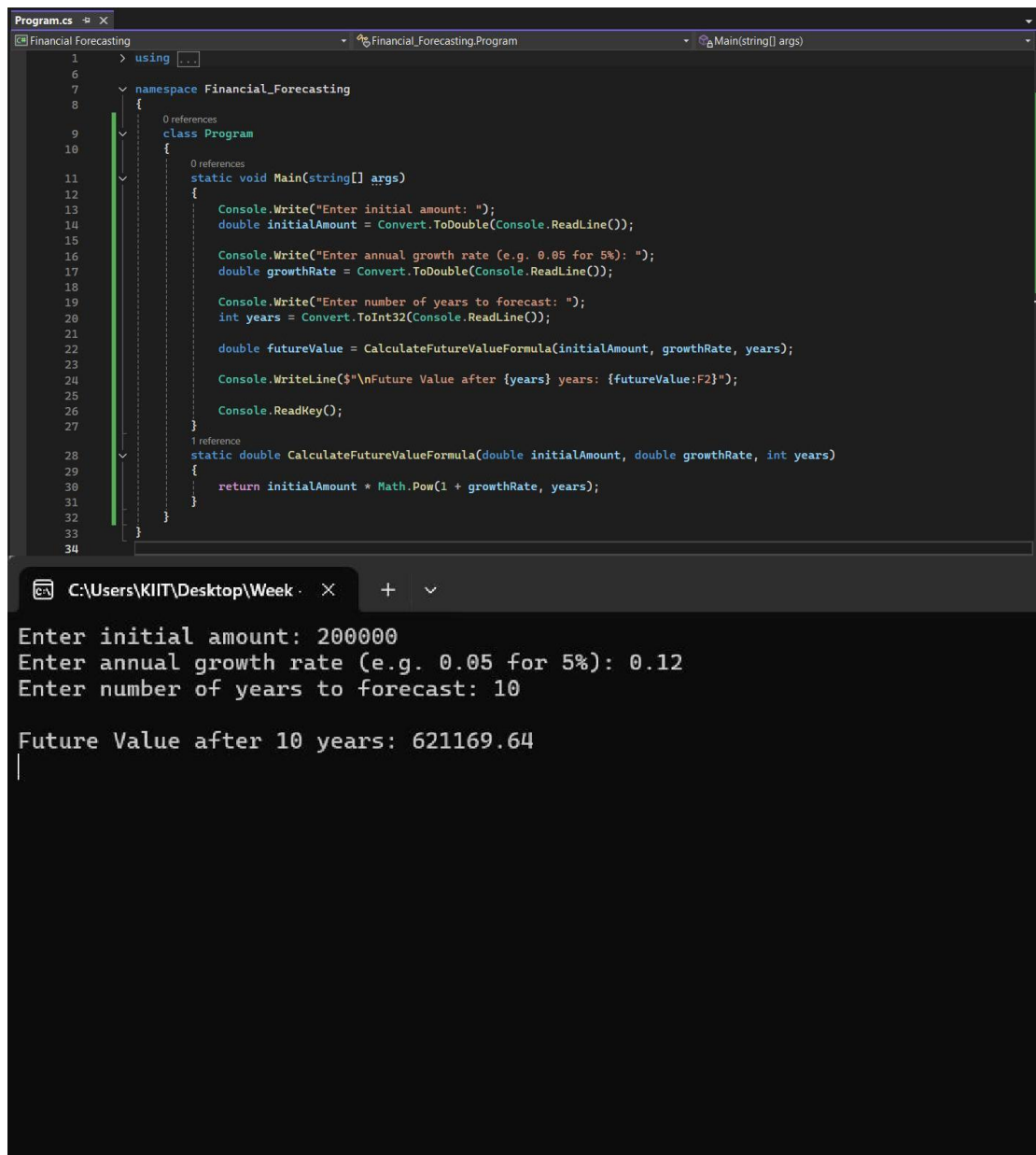

➤ Analysis :

Time Complexity = $O(n)$

Space Complexity = $O(n)$

Problem with above code is stack over flow for larger n value .

➤ Optimization :



The screenshot shows a C# program in Visual Studio. The code defines a namespace `Financial_Forecasting` with a class `Program`. Inside `Program`, there is a `Main` method that prompts the user for an initial amount, growth rate, and number of years. It then calls a static method `CalculateFutureValueFormula` to compute the future value. The `CalculateFutureValueFormula` method uses the formula $initialAmount * Math.Pow(1 + growthRate, years)$ to calculate the future value. The console output shows the program running with inputs: initial amount 200000, growth rate 0.12, and years 10, resulting in a future value of 621169.64.

```
1  > using ...
6
7  namespace Financial_Forecasting
8  {
9      0 references
10     class Program
11     {
12         0 references
13         static void Main(string[] args)
14         {
15             Console.WriteLine("Enter initial amount: ");
16             double initialAmount = Convert.ToDouble(Console.ReadLine());
17             Console.WriteLine("Enter annual growth rate (e.g. 0.05 for 5%): ");
18             double growthRate = Convert.ToDouble(Console.ReadLine());
19             Console.WriteLine("Enter number of years to forecast: ");
20             int years = Convert.ToInt32(Console.ReadLine());
21
22             double futureValue = CalculateFutureValueFormula(initialAmount, growthRate, years);
23
24             Console.WriteLine($"Future Value after {years} years: {futureValue:F2}");
25
26             Console.ReadKey();
27         }
28     }
29     1 reference
30     static double CalculateFutureValueFormula(double initialAmount, double growthRate, int years)
31     {
32         return initialAmount * Math.Pow(1 + growthRate, years);
33     }
34 }
```

```
C:\Users\KIIT\Desktop\Week . x + v
Enter initial amount: 200000
Enter annual growth rate (e.g. 0.05 for 5%): 0.12
Enter number of years to forecast: 10

Future Value after 10 years: 621169.64
|
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

No extra space is used and it is the Fastest and most efficient and best for production.