Financial Forecasting

Step 1: Understand Recursive Algorithms

In recursion, two main parts are defined:

- **Base Case:** The condition under which the recursion stops.
- **Recursive Case:** The function calls itself with updated parameters that move toward the base case.

Recursive algorithms can make complex problems simpler to implement logically, though they may lead to performance issues if not optimized.

Step 2: Setup

To forecast financial growth, assume the following:

- `P` = Principal (initial amount)
- `r` = Growth rate per period (e.g., 5% = 0.05)
- `n` = Number of years

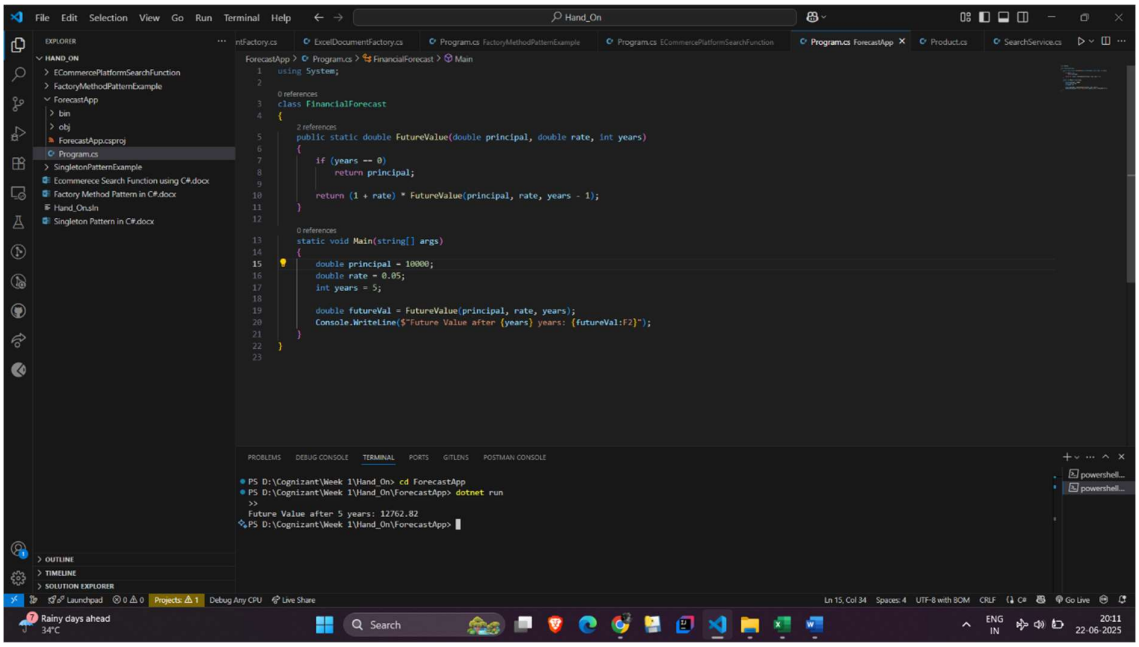
The standard formula for future value (FV) is:

```
FV = P * (1 + r)^n
```

Step 3: Implementation in C#

```
using System;

class FinancialForecast
{
    public static double FutureValue(double principal, double rate, int years)
    {
        if (years == 0)
            return principal;

        return (1 + rate) * FutureValue(principal, rate, years - 1);
    }

    static void Main(string[] args)
    {
        double principal = 10000;
        double rate = 0.05;
        int years = 5;

        double futureVal = FutureValue(principal, rate, years);
        Console.WriteLine($"Future Value after {years} years: {futureVal:F2}");
   }
}
```

Step 4:Output



Step 5: Analysis

Time Complexity:

- The recursive approach performs one function call per year.
- Thus, the time complexity is O(n), where `n` is the number of years.

Space Complexity:

- O(n) due to the recursion stack.

Optimization:

If the number of years (`n`) is very large, recursion can lead to a stack overflow. To avoid this, the recursive approach can be replaced with an iterative method:

```
public static double FutureValueIterative(double principal, double rate, int years)
{
    double result = principal;
    for (int i = 0; i < years; i++)
    {
        result *= (1 + rate);
    }
    return result;
}
```

The iterative method maintains O(n) time complexity but reduces space complexity to O(1), making it more efficient for large inputs.