Fibonacci Performance Monitoring Script

Salvador Gutierrez Portocarrero

August 5, 2025

Overview

This document describes a Python script designed to compute Fibonacci numbers and monitor system performance metrics such as execution time, CPU usage, and memory usage during the calculation.

The results are visualized using plots and saved to a CSV file with a name based on the current date. The file is saved to a predefined directory, and it is used as an overview of the computation time to evaluate processor and system performance.

Algorithm Description

The script uses the Fibonacci sequence defined recursively as:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

This version of the function is implemented recursively, which is computationally expensive for larger values of n. While inefficient, the recursive version is intentionally used here to stress test CPU and memory performance, allowing us to visualize resource scaling with increasing computation load.

Performance Monitoring

For each selected value of n, the following metrics are measured:

- **Execution Time:** Time taken to compute $F(n)$
- **CPU Usage:** CPU percentage usage immediately after the computation
- **Memory Usage:** System memory usage percentage after computation

Fibonacci Sequence Definition

The Fibonacci sequence is a series of numbers where each term is the sum of the two preceding ones. It begins with:

$$F(0) = 0, \quad F(1) = 1$$

and follows the recurrence relation:

$$F(n) = F(n-1) + F(n-2), \quad \text{for } n \geq 2$$

The sequence begins:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ \ldots$$

With detailed calculations:

$$\begin{aligned}
F(0) &= 0 \\
F(1) &= 1 \\
F(2) &= F(1) + F(0) = 1 + 0 = 1 \\
F(3) &= F(2) + F(1) = 1 + 1 = 2 \\
F(4) &= F(3) + F(2) = 2 + 1 = 3 \\
F(5) &= F(4) + F(3) = 3 + 2 = 5 \\
F(6) &= F(5) + F(4) = 5 + 3 = 8 \\
F(7) &= F(6) + F(5) = 8 + 5 = 13 \\
F(8) &= F(7) + F(6) = 13 + 8 = 21 \\
F(9) &= F(8) + F(7) = 21 + 13 = 34 \\
F(10) &= F(9) + F(8) = 34 + 21 = 55
\end{aligned}$$

Visual Representation

Below is a visual representation of the Fibonacci sequence (e.g., bar chart), Figure 1:

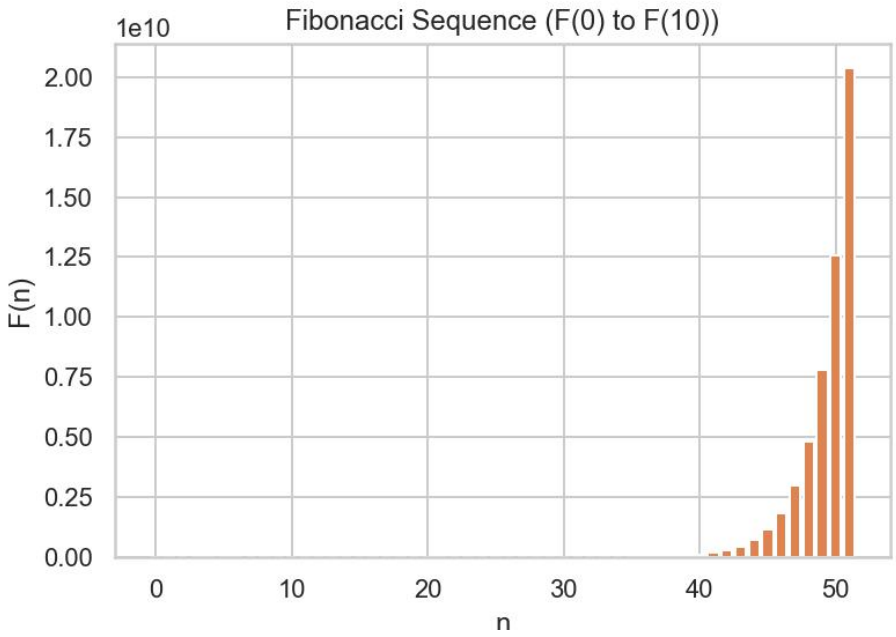

Figure 1: Fibonacci sequence visualized as a bar chart

Number of Additions

The number of additions required to compute $F(n)$ recursively is approximately:

$$\text{Additions}(n) \approx F(n+1) - 1$$

Therefore, to compute $F(50)$, the number of additions is:

$$\text{Additions}(50) = F(51) - 1 = 20{,}365{,}011{,}074 - 1 = \boxed{20{,}365{,}011{,}073}$$

This exponential growth makes the naive recursive method impractical for large n.

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones. It starts from 0 and 1:

$$F(n) = \begin{cases} 0 & \text{if } n \leq 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

This recursive formula can be implemented directly in Python using a function. The following code snippet shows the implementation used in this script:

```python
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Listing 1: Recursive Fibonacci Function

This recursive approach is simple but inefficient for large n, due to repeated recalculations of the same subproblems. For higher values, an iterative or memoized version is recommended. However, this script is used to computed repeated and costly summations.

Main Loop Code

The following Python code iterates over values of n, runs the Fibonacci function, collects system metrics, and stores the results:

```python
if __name__ == "__main__":
    ns = range(1, 3)  # You can increase the range for more tests
    times = []
    cpu_usages = []
    memory_usages = []

    for n in ns:
        print(f"Running Fibonacci({n})...")
        time_taken, cpu_usage, memory_usage =
           run_algorithm_and_monitor(n)
        times.append(time_taken)
        cpu_usages.append(cpu_usage)
        memory_usages.append(memory_usage)

    plot_results(ns, times, cpu_usages, memory_usages)
```

Listing 2: Main loop for running Fibonacci and monitoring resources

Output and Saving

The results are stored in a Pandas DataFrame and saved as a CSV file. The filename includes the current date and is saved in a specified directory, Figure 2.

Dependencies

The script uses the following Python libraries:

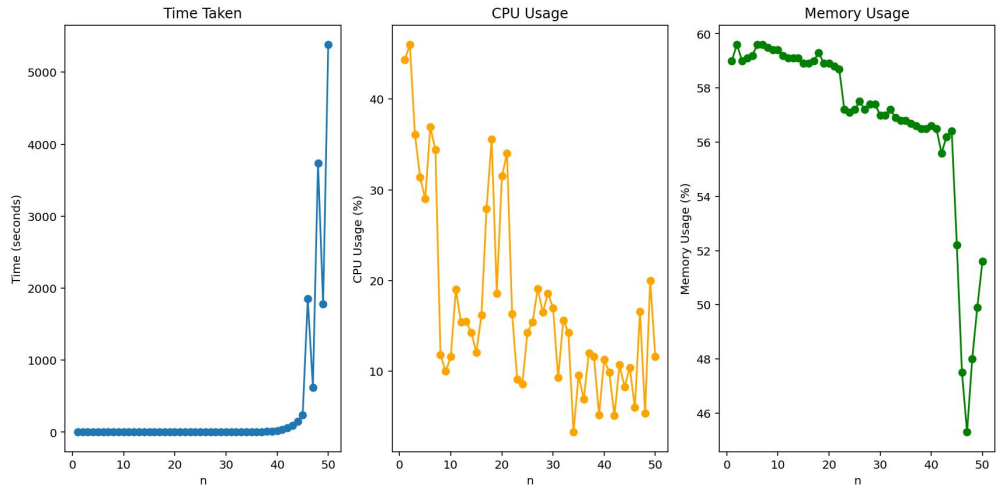

Figure 2: Plot of Execution Time, CPU Usage, and Memory Usage vs. Fibonacci n

- `psutil` for system monitoring
- `time` and `datetime` for timing and file naming
- `matplotlib` for plotting
- `pandas` for tabular data and CSV export
- `os` for file path management

Computational Cost: Calculating $F(50)$

The recursive Fibonacci function defined as:

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Listing 3: Recursive Fibonacci function