fibonacci_report

January 31, 2026

1 Fibonacci Matrix Exponentiation

1.1 Divide and Conquer Algorithm

Course: Análisis y Diseño de Algoritmos

1.2 Abstract

This report presents an efficient algorithm for computing Fibonacci numbers using matrix exponentiation with divide and conquer. The method achieves O(log n) time complexity.

1.3 1. Mathematical Foundation

The Fibonacci sequence can be expressed using matrix exponentiation:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

Therefore: **F(n) = M^n[0,1]**

1.3.1 Divide and Conquer Strategy

$$M^n = \begin{cases} M & \text{if } n = 1 \\ (M^{n/2})^2 & \text{if } n \text{ is even} \\ (M^{n/2})^2 \times M & \text{if } n \text{ is odd} \end{cases}$$

[ ]:
```python
import numpy as np

M = np.array([[1, 1], [1, 0]], dtype=np.int64)
print("Base matrix M:")
print(M)
```

```
Base matrix M:
[[1 1]
 [1 0]]
```

1.4 2. Algorithm Implementation

[ ]:
```python
def power_matrix(matrix, n):
    """Recursively compute matrix power using divide and conquer.

    Args:
        matrix: 2x2 numpy array
        n: Positive integer exponent

    Returns:
        matrix^n as numpy array
    """
    if n == 1:
        return matrix

    half = power_matrix(matrix, n // 2)
    half = np.dot(half, half)
    return half if n % 2 == 0 else np.dot(half, matrix)


def fibonacci_matrix(n):
    """Calculate nth Fibonacci number using recursive matrix exponentiation.

    Args:
        n: Positive integer position in Fibonacci sequence

    Returns:
        The nth Fibonacci number

    Raises:
        ValueError: If n is not positive
    """
    if n <= 0:
        raise ValueError("n must be positive")

    base_matrix = np.array([[1, 1], [1, 0]], dtype=np.int64)
    return power_matrix(base_matrix, n)[0, 1]
```

1.5 3. Example Execution

For n=21 (binary: 10101):

```
M^21 = (M^10)² × M
  M^10 = (M^5)²
    M^5 = (M^2)² × M
      M^2 = (M^1)²
        M^1 = M
```

Only **5 recursive calls**

[ ]:
```python
print("Test Results:")
print("-" * 30)
for n in [5, 10, 21, 50]:
    print(f"F({n}) = {fibonacci_matrix(n)}")
```

```
Test Results:
------------------------------
F(5) = 5
F(10) = 55
F(21) = 10946
F(50) = 12586269025
```

1.6 4. Complexity Analysis

1.6.1 Time Complexity

Method	Complexity
Naive Recursion	O(2^n)
Iterative	O(n)
Matrix (Divide & Conquer)	**O(log n)**

1.6.2 Recurrence Relation

T(n) = T(n/2) + O(1)

By Master Theorem: T(n) = O(log n)

1.7 5. Conclusions

1. Matrix exponentiation achieves **O(log n)** time complexity
2. Divide and conquer reduces problem size by half at each step
3. Efficient for computing large Fibonacci numbers
4. Technique extends to other linear recurrence relations