Lab 3: Fibonacci Numbers and Tiling Numbers Revisit

Algorithms and Computation

Name: Robert Rice
Date: December 4, 2025
Student ID: 657588340
Language: Java

---

1. Problem Statement

In this lab I:

- Implemented successive squaring (binary exponentiation) for modular powers.
- Used matrix exponentiation to get $O(\log n)$ algorithms for:
  - LeetCode 509: Fibonacci Number.
  - LeetCode 790: Domino and Tromino Tiling.
- Connected these with general linear recurrences.

2. Successive Squaring for Modular Exponentiation

Goal: compute $a^e \bmod m$ without performing e multiplications.

Idea:

- Write e in binary.
- Loop while $e > 0$:
  - If the current bit of e is 1, set `result = (result * base) % m`.
  - Always set `base = (base * base) % m`.
  - Shift e to the right by 1 bit.

This uses $O(\log e)$ multiplications.

Test case from the lab:

$$7^{327} \bmod 853 = 286$$

My Java method `modPow(7, 327, 853)` returns 286, which matches the expected output.

3. Binary Encoding and Ternary Idea (EC Sketch)

Binary drives the algorithm because:

$$e = \sum b_i 2^i, \quad b_i \in \{0, 1\}.$$

At each bit:

- Square the base.
- Multiply into the answer only if $b_i = 1$.

In ternary (base 3), the digits would be $0, 1, 2$. Conceptually:

- Repeatedly replace a with a^3.
- At each ternary digit $d_i \in \{0, 1, 2\}$, multiply by a^{d_i}.

I only explored this idea conceptually and did not finish a coded ternary version, so I am **not** claiming this extra credit.

4. Fibonacci in $O(\log n)$ via Matrix Exponentiation (LeetCode 509)

Definition:

$$F(0) = 0,\ F(1) = 1,\ F(n) = F(n-1) + F(n-2).$$

Naive recursion is exponential. The iterative DP is $O(n)$. We can do better with the Fibonacci Q-matrix:

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

So $F(n) = (Q^n)_{0,1}$. I compute Q^n with successive squaring on the 2×2 matrices:

- Start with the identity matrix.
- While $n > 0$, if bit is 1 multiply the answer by current matrix; always square current matrix; shift n.

This gives a $O(\log n)$ solution in Java that passes all LeetCode 509 tests.

5. Domino and Tromino Tiling in $O(\log n)$ (LeetCode 790)

We count the ways to tile a $2 \times n$ board with dominoes and trominoes. A known recurrence is

$$f(0) = 1,\ f(1) = 1,\ f(2) = 2, \quad f(n) = 2f(n-1) + f(n-3)\ (n \geq 3),$$

with all results taken modulo $10^9 + 7$.

Define the state vector:

$$v(n) = \begin{bmatrix} f(n) \\ f(n-1) \\ f(n-2) \end{bmatrix}.$$

Then

$$v(n+1) = A\,v(n), \quad A = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

From the base vector $v(2) = [2, 1, 1]^T$, we have:

$$v(n) = A^{n-2}v(2).$$

I compute A^{n-2} using successive squaring on 3×3 matrices, modulo $10^9 + 7$. The first entry of $v(n)$ is $f(n)$. This gives an $O(\log n)$ Java solution that passes all LeetCode 790 tests.

6. General Linear Recurrence (EC Sketch)

A general k-th order linear recurrence:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}.$$

I did not implement a fully generic solver, so I am **not** claiming this additional credit.

7. Algorithm Design and Time Complexity

This lab showed how choosing a better algorithm changes the time complexity:

- Modular exponent: $O(e)$ naive $\to O(\log e)$ with successive squaring.
- Fibonacci: $O(2^n)$ naive recursion $\to O(n)$ DP $\to O(\log n)$ matrix exponentiation.
- Domino/ Romino tiling: $O(n)$ DP $\to O(\log n)$ matrix exponentiation.

Even when $O(n)$ is "fast enough" for the given constraints, knowing the $O(\log n)$ method is important for larger inputs and for understanding more advanced algorithms.

8. Passing All Tests and Extra Credit Status

- **LeetCode 509 (Fibonacci):** My Java matrix exponentiation solution passes all online judge tests.
- **LeetCode 790 (Domino and Tromino Tiling):** My Java matrix exponentiation solution passes all online judge tests.
- **Extra Credit:** I did *not* complete full coded solutions for the ternary exponentiation or a fully generic linear recurrence solver, so I am **not** claiming any extra credit.