# Análise de Algoritmos de Fibonacci

## Solução Recursiva - $o(2^n)$

Clássica, facil de compreender e implementar, definida por: $F(n) = F(n-1) + F(n-2)$, considerando: $F(1) = 1$ e $F(0) = 0$

Contudo demanda muito tempo e memória visto que uma chamada realiza outras duas, e um mesmo valor pode ser calculado inumeras vezes desnecessáriamente.

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

---

## Solução Iterativa - $o(n)$

Também é simples de ser implementada. Além de ser mais eficiênte, também consome menos memória.

```
def fibonacci(n: int) → int:
    if n <= 0: return 0
    if n <= 2: return 1
    a, b = 0, 1
    for _ in range(1, n):
        a, b = b, a + b
    return b
```

---

## Solução pela Expressão Matricial - $o(\log n)$

Se tratando de uma recorrência linear, a expressão recursiva do termo geral pode ser escrita de forma matricial.

Considerando uma transformação $A$ que traduz $F_{n+1} = F_n + F_{n-1}$

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

A transformação $A$ pode ser definida como:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Considerando os valores iniciais chegamos em:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

Nesse ponto cabe análisar que a complexidade do problema foi reduzida a exponenciação da matriz $A$, que pode ser implementada em $o(\log n)$ a partir da tecnica de **Exponentiation by Squaring**.

```
def multiply_matrices(A: list, B: list) → list:
    return [[A[0][0] * B[0][0] + A[0][1] * B[1][0], A[0][0] * B[0][1] + A[0][1] * B[1]
[1]],
            [A[1][0] * B[0][0] + A[1][1] * B[1][0], A[1][0] * B[0][1] + A[1][1] * B[1][1]]]

def matrix_exponentiation(A: list, n: int) → list:
    if n == 1:
        return A
    if n % 2 == 0:
        half_power = matrix_exponentiation(A, n // 2)
        return multiply_matrices(half_power, half_power)
    else:
        return multiply_matrices(A, matrix_exponentiation(A, n-1))

def fibonacci(n: int) → int:
```

```
if n <= 0: return 0
if n <= 2: return 1
A = [[1, 1], [1, 0]]
result = matrix_exponentiation(A, n-1)
return result[0][0]
```

---

## Solução pela Fórmula de Binet - $o(1)$

Desenvolvida a partir da expressão matricial.

### 1. Função Característica

O objetivo é diagonalizar $A$, para isso encontramos primeiro o **polinômio característico**:

$$det(A - \lambda I) = \begin{vmatrix} 1-\lambda & 1 \\ 1 & -\lambda \end{vmatrix}$$

$$\lambda^2 - \lambda - 1 = 0$$

Que nos fornece os **autovalores** (raizes):

$$\lambda = \frac{1 \pm \sqrt{5}}{2}$$

Ou então:

$$\phi = \frac{1+\sqrt{5}}{2} \qquad \psi = \frac{1-\sqrt{5}}{2}$$

Necessários para encontrar os **autovetores** relevantes para a diagonalização.

Resolvendo o sistema $(A - \lambda I)v = 0$:

$$\begin{pmatrix} 1-\lambda & 1 \\ 1 & -\lambda \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$(1-\lambda)x + y = 0 \quad \Rightarrow \quad y = (\lambda - 1)x$$

Considerando $x = 1$, encontramos:

$$v_\phi = \begin{pmatrix} 1 \\ \phi - 1 \end{pmatrix} \qquad\qquad v_\psi = \begin{pmatrix} 1 \\ \psi - 1 \end{pmatrix}$$

---

## 2. Diagonalização de $A$

Devemos escrever $A = P\,D\,P^{-1}$

Em que as **colunas** de $P$ são os **autovetores**:

$$P = \begin{pmatrix} 1 & 1 \\ \phi - 1 & \psi - 1 \end{pmatrix}$$

A **diagonal principal** de $D$ são os **autovalores**:

$$D = \begin{pmatrix} \phi & 0 \\ 0 & \psi \end{pmatrix}$$

E a **inversa** de $P$ é dada por $\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$

$$ad - bc = 1 \cdot (\psi - 1) - 1 \cdot (\phi - 1) = \psi - \phi$$

$$P^{-1} = \frac{1}{\psi - \phi} \begin{pmatrix} \psi - 1 & -1 \\ -(\phi - 1) & 1 \end{pmatrix}$$

Tendo em vista que $\psi - \phi = \frac{1-\sqrt{5}}{2} - \frac{1+\sqrt{5}}{2} = -\sqrt{5}$

$$P^{-1} = \frac{1}{-\sqrt{5}} \begin{pmatrix} \psi - 1 & -1 \\ -(\phi - 1) & 1 \end{pmatrix} = \frac{1}{\sqrt{5}} \begin{pmatrix} 1 - \psi & 1 \\ \phi - 1 & -1 \end{pmatrix}$$

---

## 3. Expressão de $A^n$

Sabemos que $A^n = P\,D^n\,P^{-1}$, e pretendemos substituir na expressão matricial:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix} = P\,D^n\,P^{-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$P\,D^n\,\frac{1}{\sqrt{5}} \begin{pmatrix} 1-\psi & 1 \\ \phi-1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow P\,D^n\,\frac{1}{\sqrt{5}} \begin{pmatrix} 1-\psi \\ \phi-1 \end{pmatrix}$$

$$P \begin{pmatrix} \phi^n & 0 \\ 0 & \psi^n \end{pmatrix} \frac{1}{\sqrt{5}} \begin{pmatrix} 1-\psi \\ \phi-1 \end{pmatrix} \Rightarrow P\,\frac{1}{\sqrt{5}} \begin{pmatrix} \phi^n(1-\psi) \\ \psi^n(\phi-1) \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ \phi-1 & \psi-1 \end{pmatrix} \frac{1}{\sqrt{5}} \begin{pmatrix} \phi^n(1-\psi) \\ \psi^n(\phi-1) \end{pmatrix}$$

$$\frac{1}{\sqrt{5}} \begin{pmatrix} 1\cdot\phi^n(1-\psi) + 1\cdot\psi^n(\phi-1) \\ (\phi-1)\cdot\phi^n(1-\psi) + (\psi-1)\cdot\psi^n(\phi-1) \end{pmatrix}$$

Concluímos que o vetor resultante é:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \frac{1}{\sqrt{5}} \begin{pmatrix} \phi^n(1-\psi) + \psi^n(\phi-1) \\ *** \end{pmatrix}$$

O qual vamos considerar apenas o primeiro termo para simplificar.

---

## 4. Simplificando

Considerando:

$$1-\psi = 1 - \frac{1-\sqrt{5}}{2} = \frac{2-(1-\sqrt{5})}{2} = \frac{1+\sqrt{5}}{2} = \phi$$

$$\phi - 1 = \frac{1+\sqrt{5}}{2} - 1 = \frac{(1+\sqrt{5})-2}{2} = \frac{\sqrt{5}-1}{2} = -\psi$$

Podemos substituir:

$$F_{n+1} = \frac{1}{\sqrt{5}} \left[\phi^n \cdot \phi - \psi^n \cdot \psi\right]$$

$$F_{n+1} = \frac{\phi^{n+1} - \psi^{n+1}}{\sqrt{5}}$$

Por fim, substituimos $n$ por $n-1$ para ajustar a função para o N-ésimo termo, chegando então na fórmula de Binet.

$$F_n = \frac{\phi^n - \psi^n}{\sqrt{5}}$$

Implementando o algoritmo:

```
def fibonacci(n: int) → int:
    SQRT_FIVE = 5 ** (1/2)
    PHI = (1 + SQRT_FIVE) / 2
    PSI = (1 - SQRT_FIVE) / 2
    return int( (PHI ** n - PSI ** n) / SQRT_FIVE )
```

Contudo ainda cabe verificar que, apesar do menor tempo de execução (teórico), essa solução pode apresentar imprecisões devido operações com ponto flutuante. E mais um detalhe: na prática, o interpretador de python ainda traduz o operador de exponenciação (**) em instruções que não são executadas estritamente em $o(1)$. Assim, restando a solução $o(\log n)$, pela expressão matricial, como a mais eficiênte e precisa.