Example (II): Fibonacci Number

Warning: You're not supposed to understand everything in this example.

Def (Fibonacci number): A seq of integers $F_0, F_1, F_2, \cdots, F_N, F_{N+1}, \cdots$

$$F_N = \begin{cases} 1, & \text{if } N = 0, 1 \\ F_{N-1} + F_{N-2}, & \text{if } N \geq 2 \end{cases}$$

Exmp: $1, 1, 2, 3, 5, 8, 13, 21, \cdots$

$F_0 \quad F_1 \quad F_2 \quad F_3 \quad F_4 \quad F_5 \quad F_6 \quad F_7$

Problem Def:

Input: N
Output: F_N

① Input size n: Is it $n = N$?

No: $n = O(\log N)$.

N

$\underbrace{1\,0\,1\,\ldots\ldots}_{\log_2 N}$

1 : 1
2 : 10
3 : 11
4 : 100
5 : 101
6 : 110
⋮

▷ First trial: Apply the definition directly

- Running time of Fib_1?

$T(N) :=$ running time of $Fib_1(N)$

$$T(N) = T(N-1) + T(N-2) + O(1)$$
$$\geq T(N-2) + T(N-2)$$
$$= 2 \cdot T(N-2)$$
$$\geq 2 \cdot 2\, T(N-4)$$
$$\geq 2 \cdot 2 \cdot 2 \cdot T(N-6)$$
$$\vdots$$
$$\geq \underbrace{(2 \cdot 2 \cdot 2 \cdots 2)}_{(N/2)\text{-many}} \cdot \underbrace{T(0)}_{O(1)} = \Omega(2^{N/2})$$

Algorithm 1: $Fib_1(N)$

```
Fib_1(N):
    If N = 0 or 1:
        return 1
    else:
        return Fib_1(N-1) + Fib_1(N-2)
```

$$T(N) = \Omega(2^{N/2})$$

- $n = \log N \iff N = 2^n$

$$T(n) = \Omega\left(2^{2^{n/2}}\right) = \Omega\left(2^{2^{n-1}}\right)$$

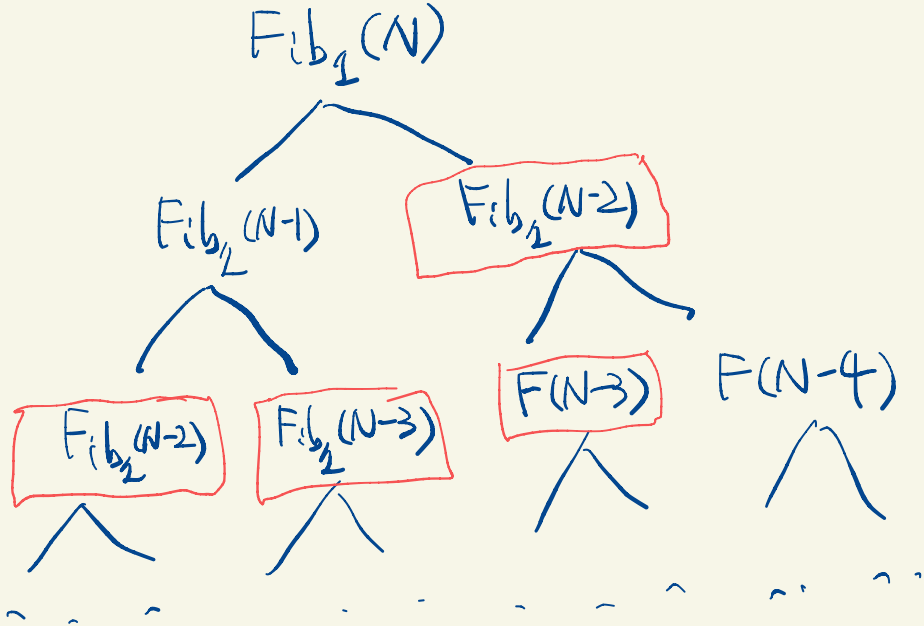
- $N = 128$, want F_{128}.

$$T(N) \geq 2^{N/2} = 2^{64}$$

- unit operation takes 1 nanosecond, 10^{-9} s or $\frac{1}{10^9}$ s

$$T(N) \geq 2^{64} \cdot 10^{-9}\,s \approx 10^{19} \cdot 10^{-9}\,s = 10^{10}\,s \approx 317 \text{ yrs.}$$

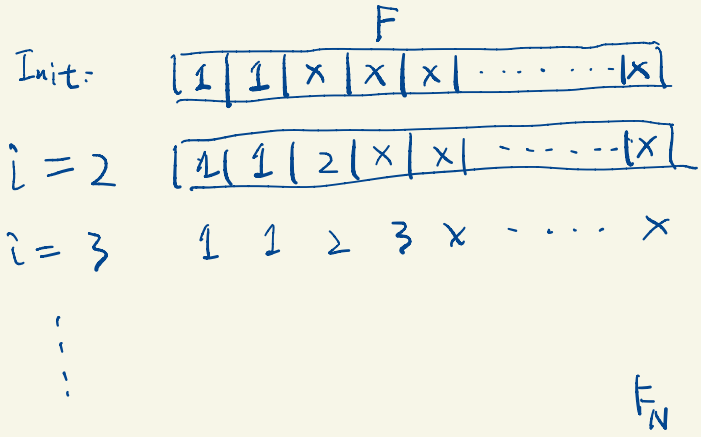
▷ Why is Fib_1 so slow?



We're repeating ourselves!

▷ First improvement : trade space for time

- Idea: remember the computation that have been done to avoid repeated effort.



Dynamic Programming.

```
Fib_2(N):
    F ← array of length N+1
    F[0], F[1] ← 1.
    for i = 2 to N:
        F[i] = F[i-1] + F[i-2]
    return F[N]
```

O(1) time

- Running time : $O(N)$

  Extra space : $\Theta(N)$ ⟹ can be reduced to $O(1)$

▷ Can we get faster?

▷ Prob (Compute power): How many multiplications you need to conduct to compute a^n, where a is a fixed const, and n is the input.

$a^n = \underbrace{a \cdot a \cdot a \cdot \cdots \cdot a}_{(n-1) \text{ muls.}}$

$2^8 = \overbrace{\underbrace{2 \times 2 \times 2 \times 2}_{2^4} \times 2 \times 2 \times 2 \times 2}^{2^8}$

$= (2^4)^2 = \left((2^2)^2\right)^2$

$\Rightarrow$ Only 3 muls suffice.

$\left((2 \times 2)^2\right)^2$

$\Rightarrow$ for general n : $O(\log n)$ muls suffice.

Alg power(a, n)

tmp = power(a, n/2)

return (tmp)^2.

running time: $f(n) = f(n/2) + O(1)$

$= f(n/4) + O(1) + O(1)$

$\vdots$

$= f(1) + \underbrace{O(1) + \cdots + O(1)}_{(\log n)\text{-many}}$

$\underbrace{\quad}_{O(1)}$

$= O(\log n)$

▷ Linear algebra.

Def: (2x2 matrix) $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$

Matrix × Matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix} := \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix} \Rightarrow O(1)$$

Matrix × Vector

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e \\ g \end{pmatrix} = \begin{pmatrix} ae+bg \\ ce+dg \end{pmatrix} \Rightarrow O(1)$$

Exmp:

$$\begin{cases} 2x+3y=a \\ 4x-y=b \end{cases} \iff \begin{pmatrix} 2 & 3 \\ 4 & -1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

Alg $Fib_3(N)$:

$$\begin{cases} F_{N-1} + F_{N-2} = F_N \\ F_{N-1} + 0 \cdot F_{N-2} = F_{N-1} \end{cases} \Rightarrow \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{N-1} \\ F_{N-2} \end{pmatrix} = \begin{pmatrix} F_N \\ F_{N-1} \end{pmatrix}$$

similarly: $\begin{pmatrix} F_{N-1} \\ F_{N-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{N-2} \\ F_{N-3} \end{pmatrix}$

$$\Rightarrow \begin{pmatrix} F_N \\ F_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{N-2} \\ F_{N-3} \end{pmatrix}$$

$$\vdots$$

$$= \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}}_{(N-1)\text{-many}} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} F_N \\ F_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{N-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$O(\log N)$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$\therefore$ Running time $O(\log N)$

- Can we do better?
  - ▷ recall input size $n = O(\log N)$

  $\Rightarrow$ Seems we have gotten the best possible.

Caveat:

Known: $F_N = \phi^N$

$\Rightarrow$ size of F_N = #bits used to store F_N

$= \log_2 \phi^N = N \log_2 \phi = O(N)$