Algorithm Analysis Part 1

Data Structures for Computer Professionals

Patiwet Wuttisarnwattana, Ph.D.

patiwet@eng.cmu.ac.th

Computer Engineering, Chiang Mai University

Fibonacci Numbers

- Learning Objectives
  - Understand the definition of **Fibonacci numbers**
  - Show that Fibonacci number becomes **very large**
  - Understand that Fibonacci number is originally defined as **recursive function**

Fibonacci numbers

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

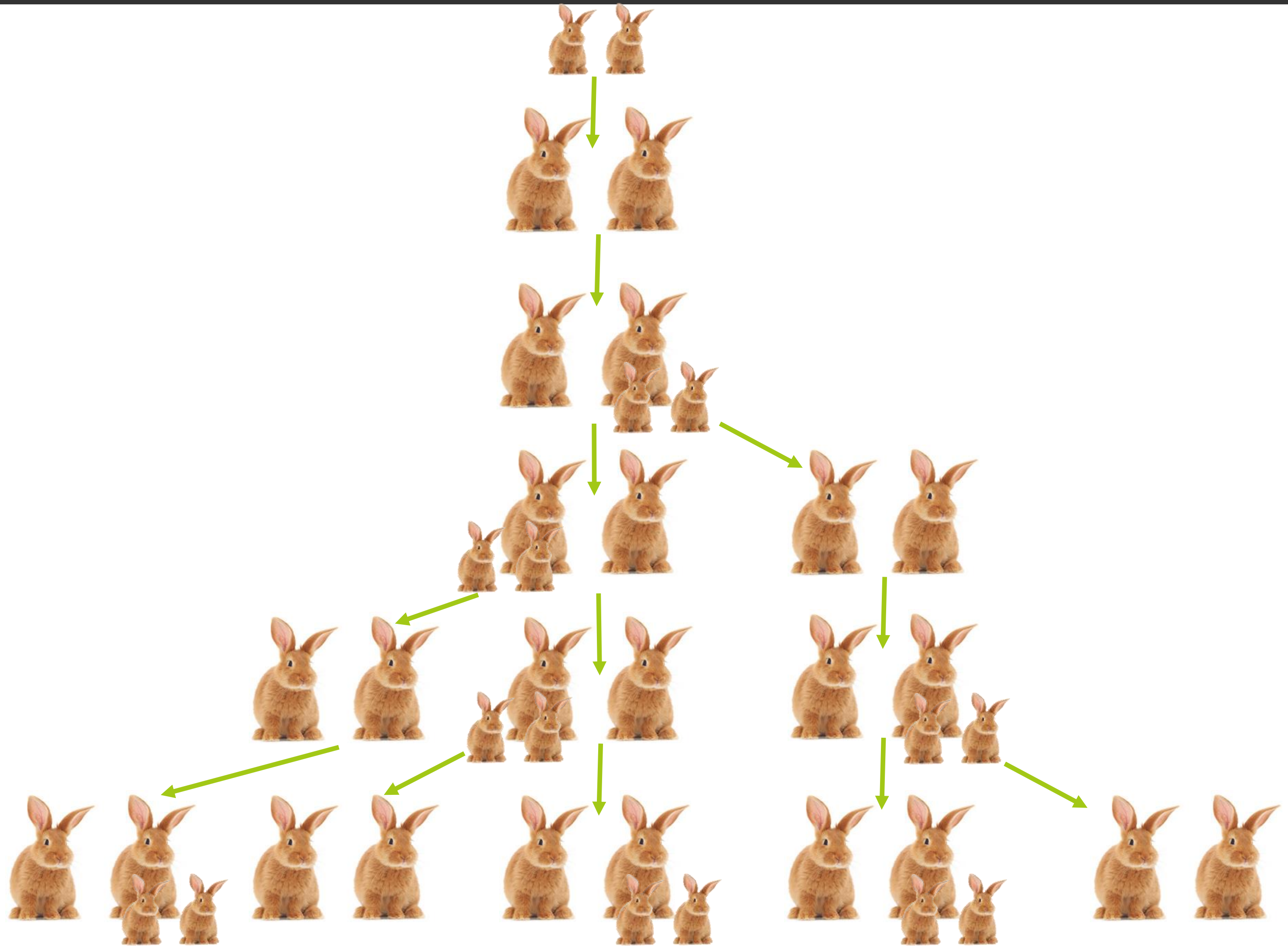
Fibonacci numbers

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

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Developed to Study Rabbit Populations

How many pairs of rabbits over time



1

1

2

3

5

8

Rapid growth

- Which one is the fastest growing function?
  - $G1(n) = 2n$
  - $G2(n) = n^2$
  - $G3(n) = 2^n$
  - Linear -> { 2 4 6 8 10 12 14 16 18 20 … }
  - Polynomial -> { 1 4 9 16 25 36 49 64 81 100 … }
  - Exponential -> {2 4 8 16 32 64 128 256 512 1024 … }

$F(n) \geq 2^{n/2}$ for $n \geq 6$

Formula

Theorem

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right).$$

ไม่ต้องจำ

Example

- F(20) = 6765
- F(50) = 12586269025
- F(100) = 354224848179261915075
- F(500) = 139423224561697880139724382870407283950070256587697307264108962948325571622863290691557658876222521294125

Computing Fibonacci numbers

Definition

$$F_n = \begin{cases} 0, & n = 0\,, \\ 1, & n = 1\,, \\ F_{n-1} + F_{n-2}, & n > 1\,. \end{cases}$$

Compute F_n

Input: An integer $n \geq 0$.

Output: F_n.

Naïve Algorithm

FibRecurs(n)

```
if  n ≤ 1:
      return  n
else:
      return  FibRecurs(n − 1) + FibRecurs(n − 2)
```

Running Time

Let $T(n)$ denote the number of lines of code executed by FibRecurs(n).

If $n \leq 1$

FibRecurs(n)

```
if n ≤ 1:
    return n
else:
    return FibRecurs(n − 1) + FibRecurs(n − 2)
```

$$T(n) = 2$$

If $n \geq 2$

FibRecurs(n)

```
if  n ≤ 1:
    return  n
else:
    return  FibRecurs(n − 1) + FibRecurs(n − 2)
```

$$T(n) = 3 + T(n\text{-}1) + T(n\text{-}2)$$

Running Time

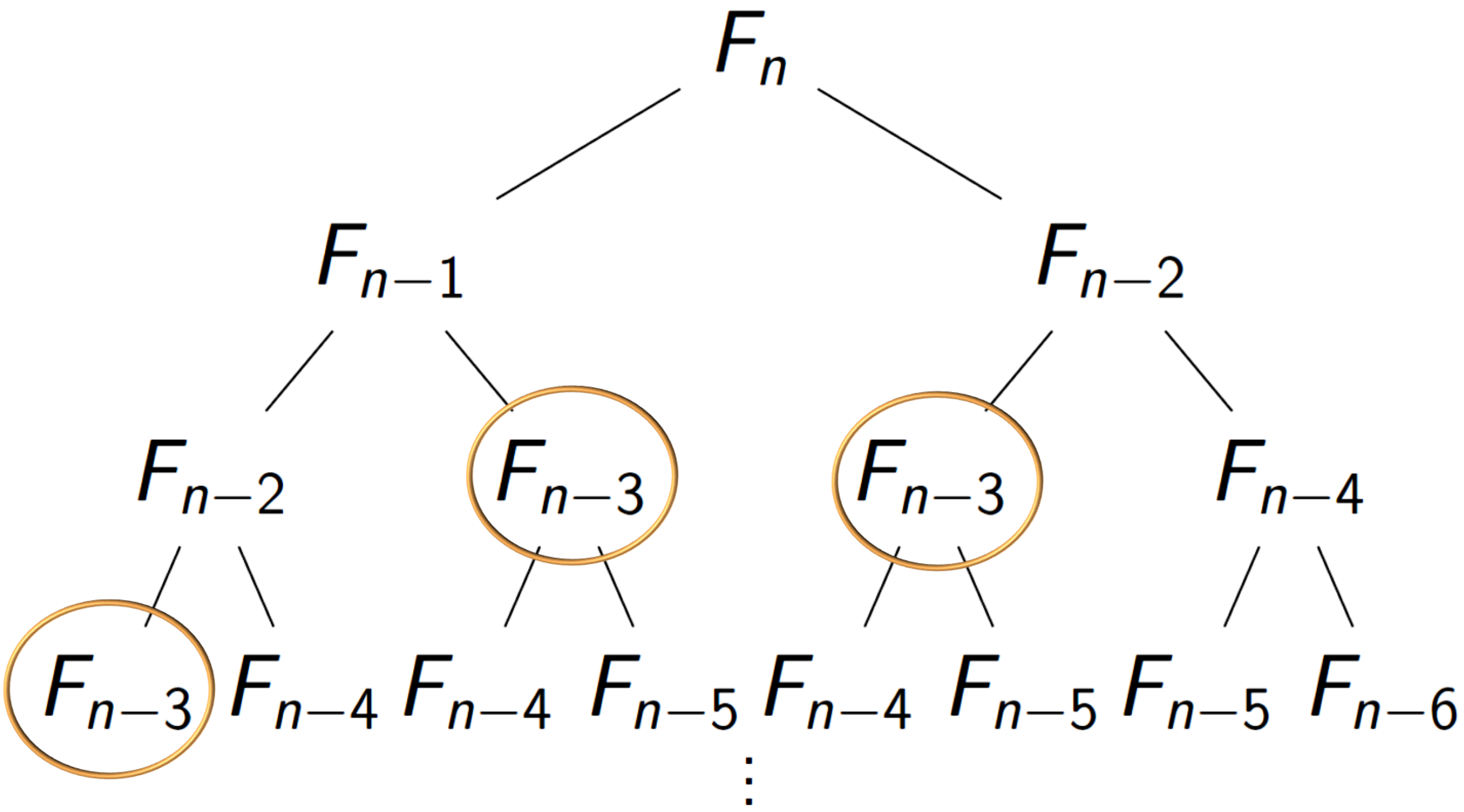
$$T(n) = \begin{cases} 2 & \text{if } n \leq 1 \\ T(n-1) + T(n-2) + 3 & \text{else.} \end{cases}$$

Therefore $T(n) \geq F_n$

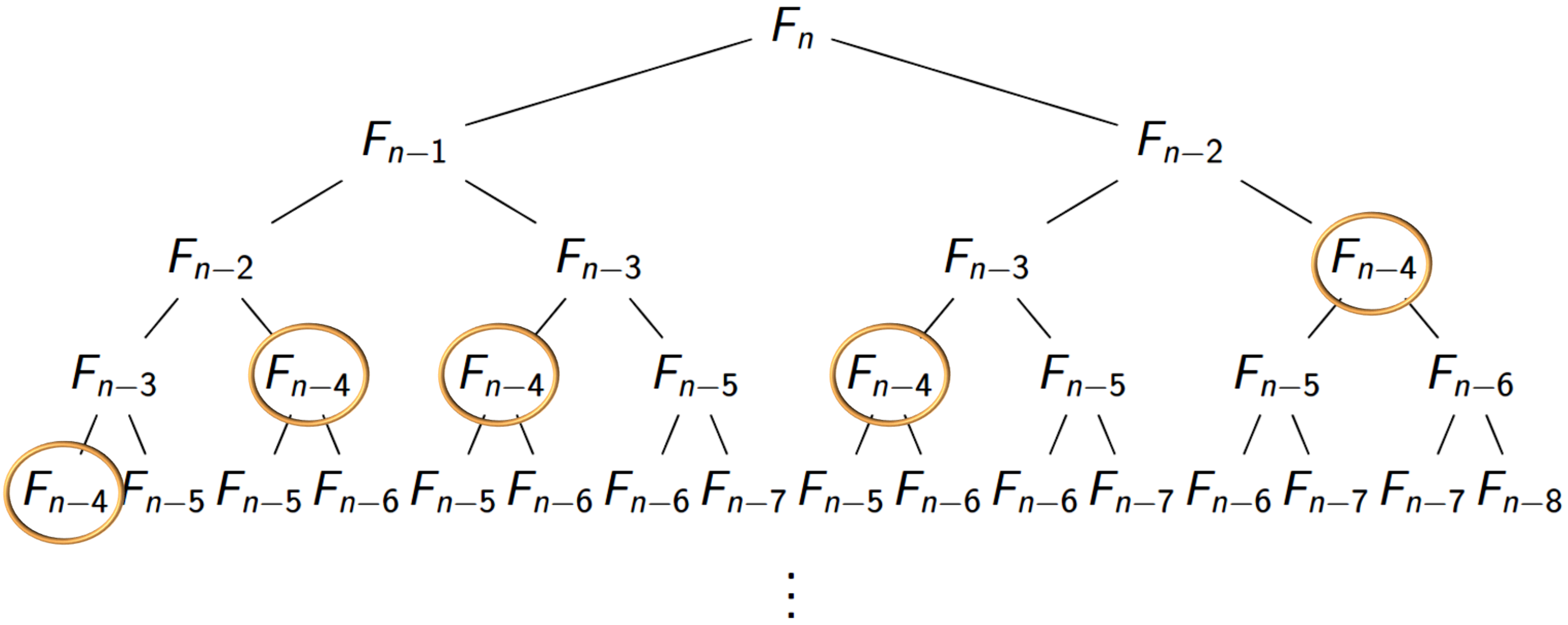
$T(100) \approx 1.77 \cdot 10^{21}$ (1.77 sextillion)

Takes 56,000 years at 1GHz.

Why so slow?



Why so slow?



F_n
F_{n-1} F_{n-2}
F_{n-2} F_{n-3} F_{n-3} F_{n-4}
F_{n-3} F_{n-4} F_{n-4} F_{n-5} F_{n-4} F_{n-5} F_{n-5} F_{n-6}
F_{n-4} F_{n-5} F_{n-5} F_{n-6} F_{n-5} F_{n-6} F_{n-6} F_{n-7} F_{n-5} F_{n-6} F_{n-6} F_{n-7} F_{n-6} F_{n-7} F_{n-7} F_{n-8}
$\vdots$

Naïve algorithm is bad, we need an efficient one

- Imitate hand computation
- 0, 1, 1
- 0 + 1 = 1

Naïve algorithm is bad, we need an efficient one

- Imitate hand computation
- 0, 1, 1, 2
- 0 + 1 = 1
- 1 + 1 = 2

Naïve algorithm is bad, we need an efficient one

- Imitate hand computation
- 0, 1, 1, 2, 3
- 0 + 1 = 1
- 1 + 1 = 2
- 1 + 2 = 3

Naïve algorithm is bad, we need an efficient one

- Imitate hand computation
- 0, 1, 1, 2, 3, 5
- 0 + 1 = 1
- 1 + 1 = 2
- 1 + 2 = 3
- 2 + 3 = 5

Naïve algorithm is bad, we need an efficient one

- Imitate hand computation
- 0, 1, 1, 2, 3, 5, 8
- 0 + 1 = 1
- 1 + 1 = 2
- 1 + 2 = 3
- 2 + 3 = 5
- 3 + 5 = 8

New Algorithm

FibList(n)

create an array $F[0 \ldots n]$
$F[0] \leftarrow 0$
$F[1] \leftarrow 1$
for i from 2 to n:
    $F[i] \leftarrow F[i-1] + F[i-2]$
return $F[n]$

- $T(n) = 2n + 2$. So $T(100) = 202$.
- Easy to compute.

Summary

- Introduced Fibonacci numbers.
- Naive algorithm takes ridiculously long time on small examples.
- Improved algorithm incredibly fast.