Luiss
Libera Università Internazionale degli Studi Sociali Guido Carli

Algorithms A.Y. 2022/2023

Lab – Fibonacci Series

Irene Finocchi, Flavio Giorgi, Bardh Prenkaj
finocchi@luiss.it, fgiorgi@luiss.it, bprenkaj@luiss.it©

10 February 2023

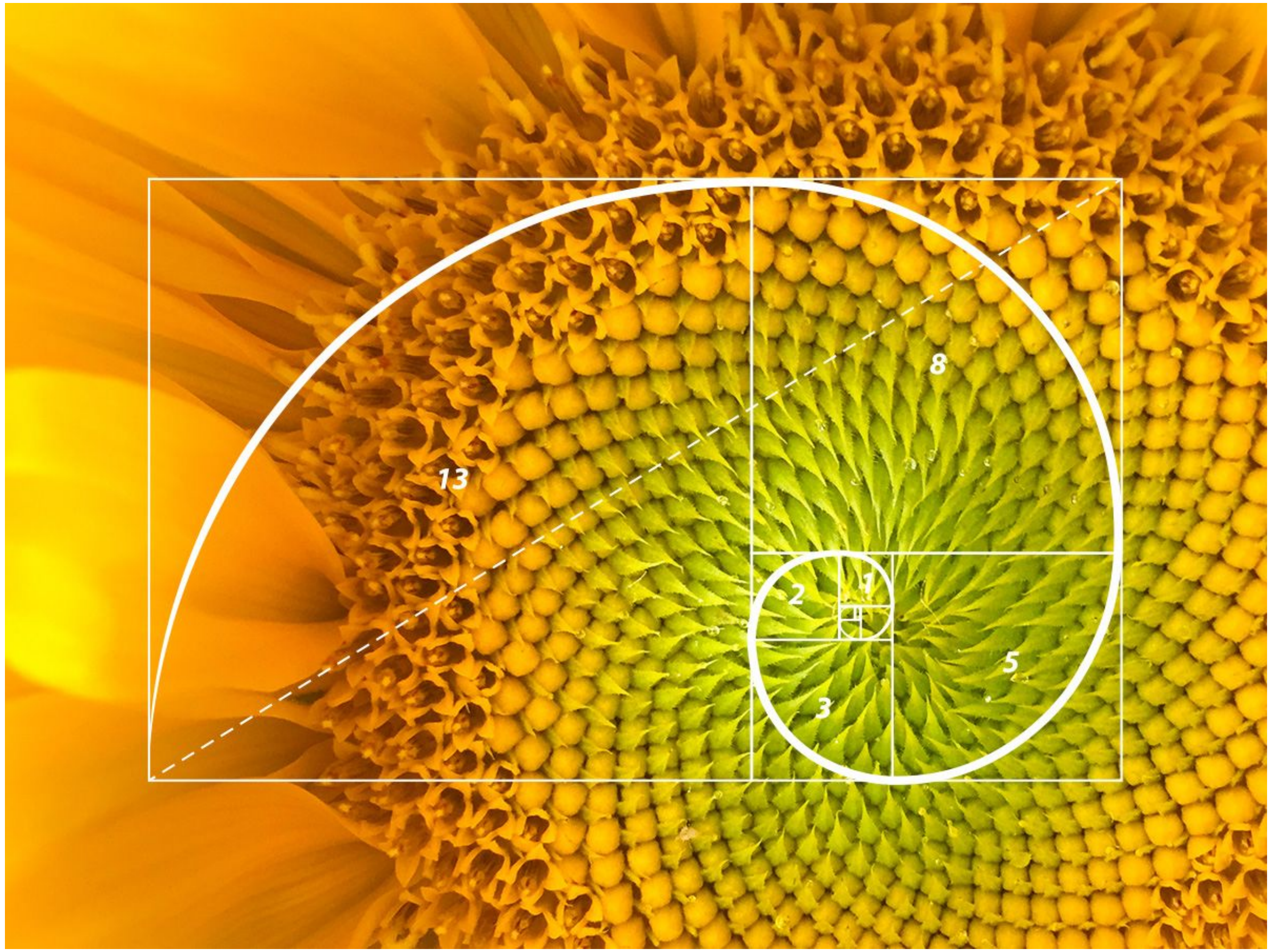
courtesy of: *Andrea Coletta*

LUISS

Dipartimento di Impresa e Management

Lab lecture 2 overview:

- We implement three different algorithms to compute Fibonacci number
- We compare their performance (time and memory)
- We solve Exercises
- Q/A project

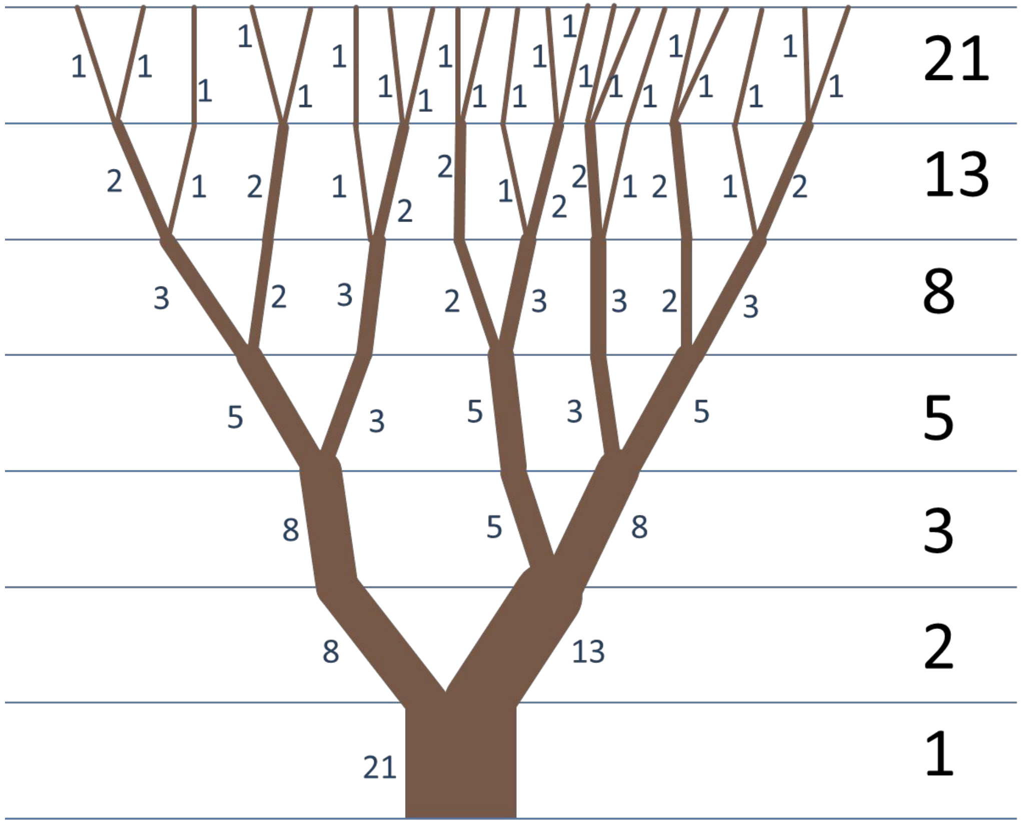


What is the Fibonacci sequence?

It is a sequence of **integer** numbers in which each number is the sum of the two preceding ones.

Fibonacci sequences appear often in nature:

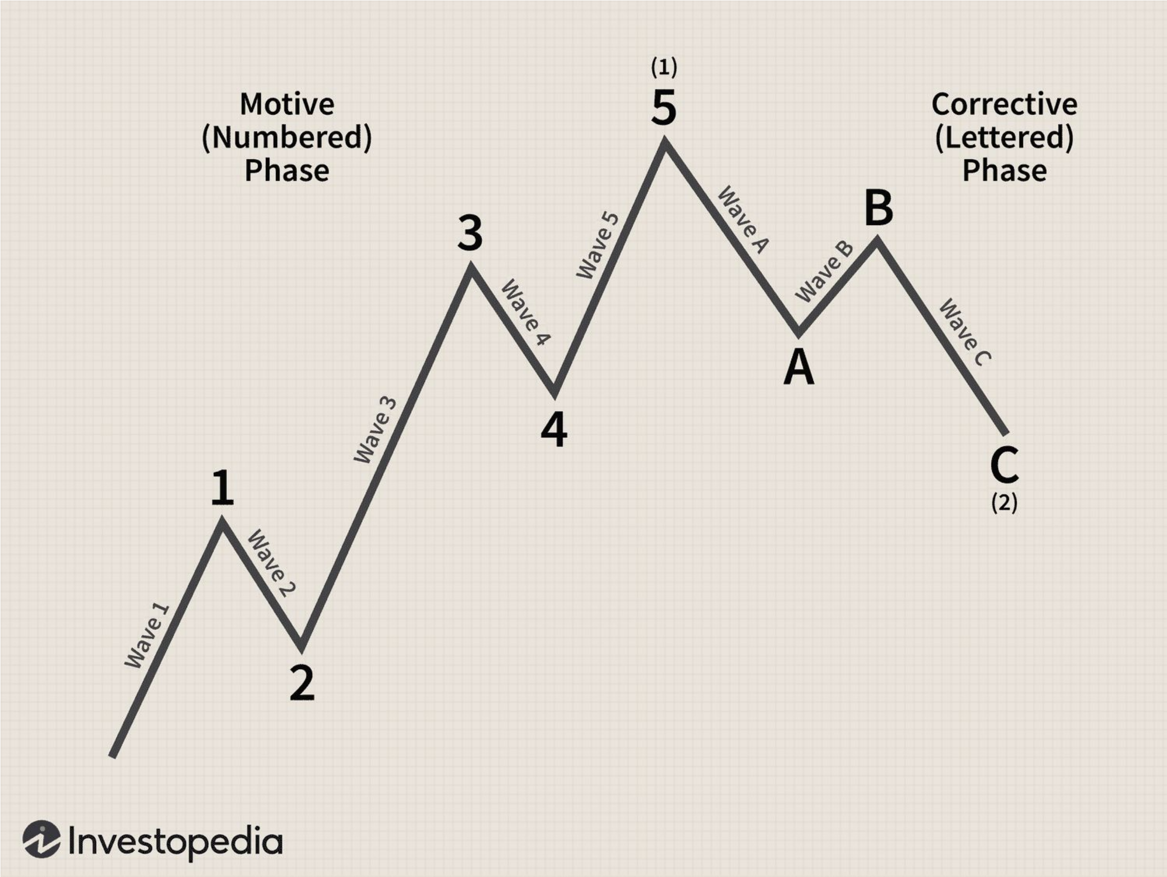
- ❖ Branching in trees
- ❖ Arrangement of leaves on a stem



Lab Lecture 2 - Fibonacci

Fibonacci Applications

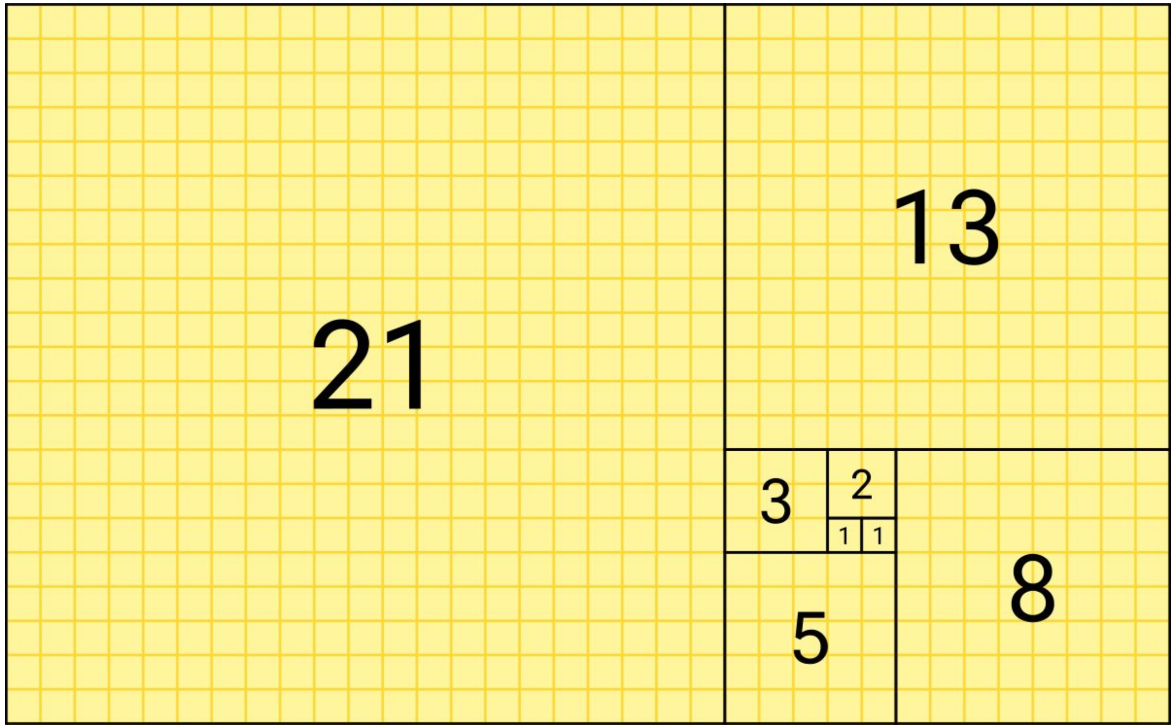
Fibonacci numbers are utilized to perform technical analysis on a stock's price action to forecast future trends in **Elliot Waves Theory**



Lab Lecture 2 - Fibonacci

Formal definition

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

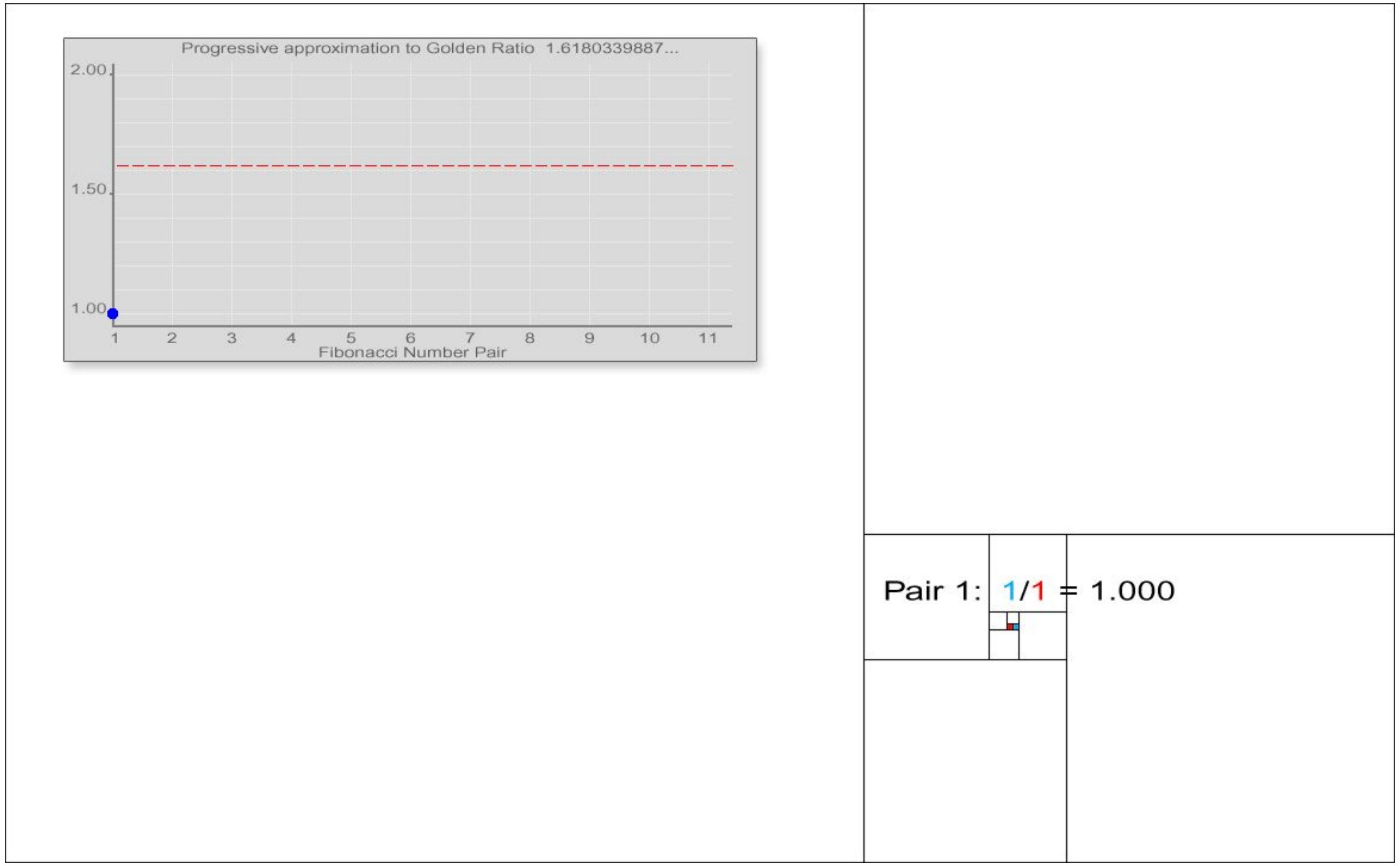


Lab Lecture 2 - Fibonacci

Interesting property

$$\lim_{n \to \infty} \frac{F_n}{F_{n-1}} = \varphi$$

$$\varphi = \frac{1+\sqrt{5}}{2} = 1{,}6180339887498 95\ldots$$



Fibonacci – A first recursive approach

algorithm `fibonacci2`(*integer n*) $\rightarrow$ *integer*

1. **if** ($n \leq 2$) **then return** 1
2. **else return** `fibonacci2`$(n-1)$ + `fibonacci2`$(n-2)$

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

Fibonacci – A first recursive approach

algorithm `fibonacci2`$(integer\ n) \rightarrow integer$

  if $(n \leq 2)$ **then return** 1

  else return `fibonacci2`$(n-1)$ + `fibonacci2`$(n-2)$

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

Exercise: Draw a tree representing the recursive calls to the function *fibonacci2* with *n=6*

LUISS

Fibonacci – A first recursive approach

algorithm `fibonacci2`$(integer\ n) \to integer$
1. **if** $(n \leq 2)$ **then return** 1
2. **else return** `fibonacci2`$(n-1)$ + `fibonacci2`$(n-2)$

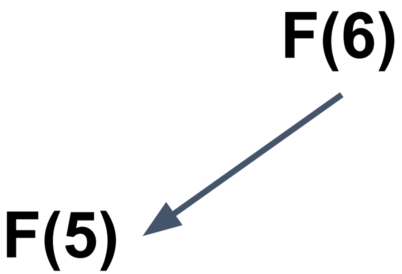
Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

F(6)

Fibonacci – A first recursive approach

```
algorithm fibonacci2(integer n) → integer
    if ( n ≤ 2 ) then return 1
    else return fibonacci2(n − 1) + fibonacci2(n − 2)
```

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

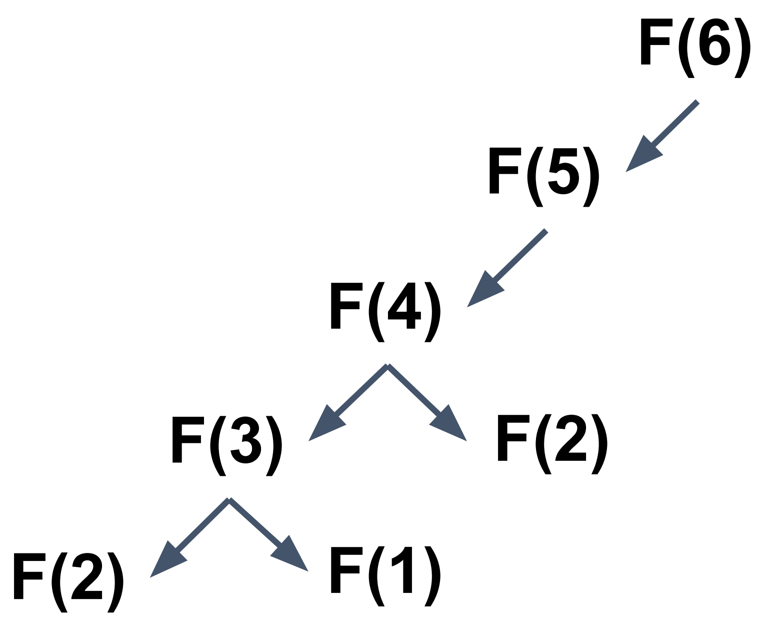


LUISS

Fibonacci – A first recursive approach

```
algorithm fibonacci2(integer n) → integer
    if ( n ≤ 2 ) then return 1
    else return fibonacci2(n − 1) + fibonacci2(n − 2)
```

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.



Fibonacci – A first recursive approach

```
algorithm fibonacci2(integer n) → integer
    if ( n ≤ 2 ) then return 1
    else return fibonacci2(n − 1) + fibonacci2(n − 2)
```

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

Question: How many recursive call the algorithm does approximately?

Fibonacci – A first recursive approach

---

algorithm `fibonacci2`$(integer\ n) \rightarrow integer$

1. **if** $(n \leq 2)$ **then return** 1
2. **else return** `fibonacci2`$(n-1)$ + `fibonacci2`$(n-2)$

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

Question: How many recursive call the algorithm does approximately?

Answer: $O(2^n)$

Fibonacci – A first recursive approach

---

algorithm `fibonacci2`$(integer\ n) \rightarrow integer$
1. **if** $(n \leq 2)$ **then return** 1
2. **else return** `fibonacci2`$(n-1)$ + `fibonacci2`$(n-2)$

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

Question: How many recursive call the algorithm does approximately?
Answer: $O(2^n)$
Question: Can we prove it?

LUISS

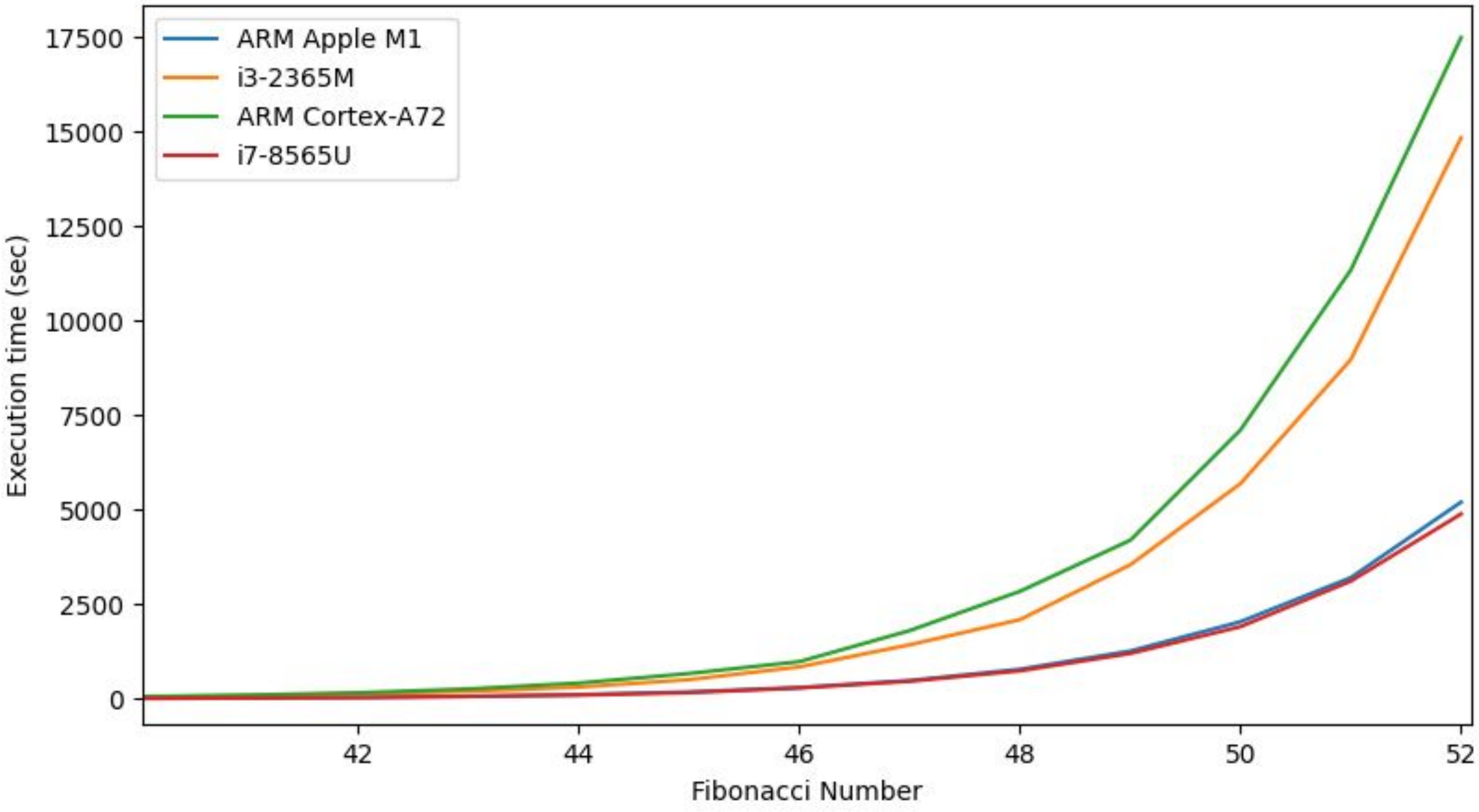
Fibonacci – A first recursive approach

algorithm $\texttt{fibonacci2}(integer\ n) \to integer$
if $(n \leq 2)$ **then return** 1
else return $\texttt{fibonacci2}(n-1) + \texttt{fibonacci2}(n-2)$

Figure 1.4 Algorithm `fibonacci2` to compute the n-th Fibonacci number.

Question: How many recursive call the algorithm does approximately?
Answer: $O(2^n)$
Question: Can we prove it?
Answer: ***YES!***

Fibonacci – Execution Time of *Fibonacci2*



LUISS

Fibonacci – An iterative solution

algorithm `fibonacci3`$(integer\ n) \rightarrow integer$

  Let Fib be an array of n integers

  $Fib[1] \leftarrow Fib[2] \leftarrow 1$

  for $i = 3$ **to** n **do**

    $Fib[i] \leftarrow Fib[i-1] + Fib[i-2]$

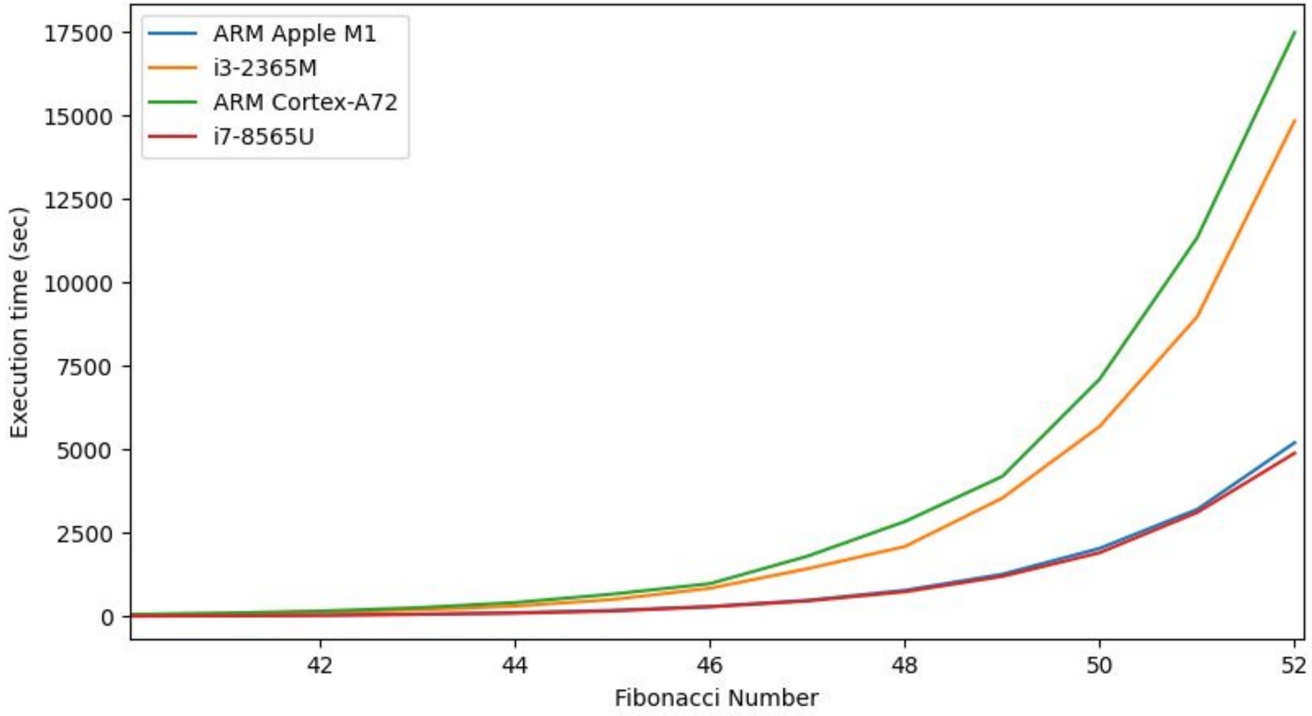
  return $Fib[n]$

Figure 1.6 Algorithm `fibonacci3` to compute the n-th Fibonacci number.

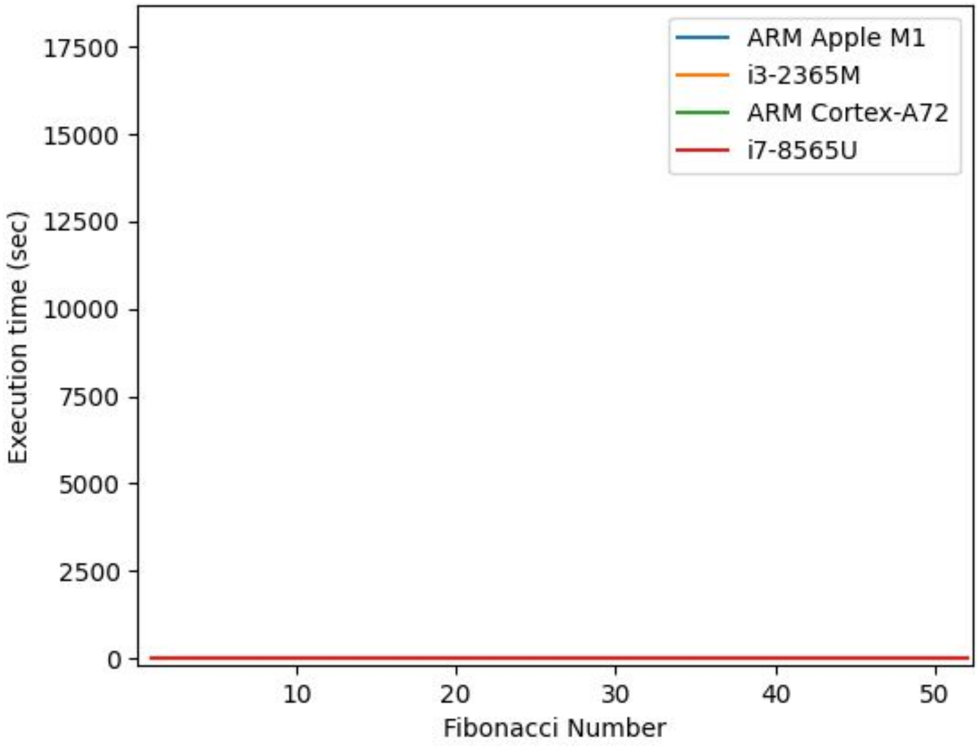
LUISS

Fibonacci – Execution Time: A comparison

Fibonacci2

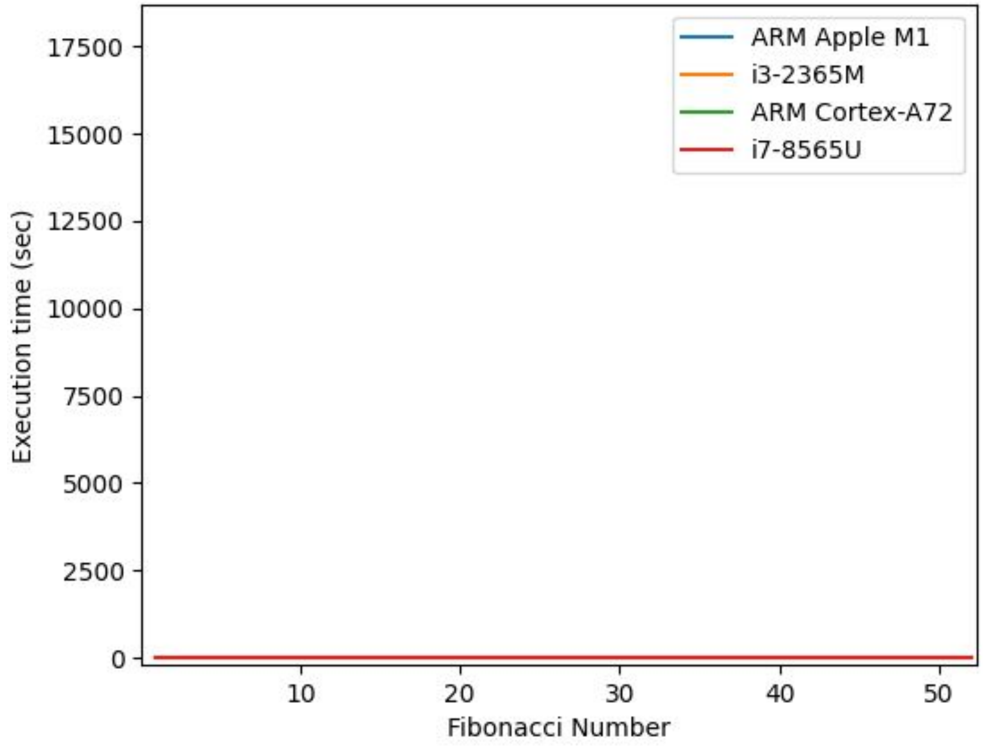


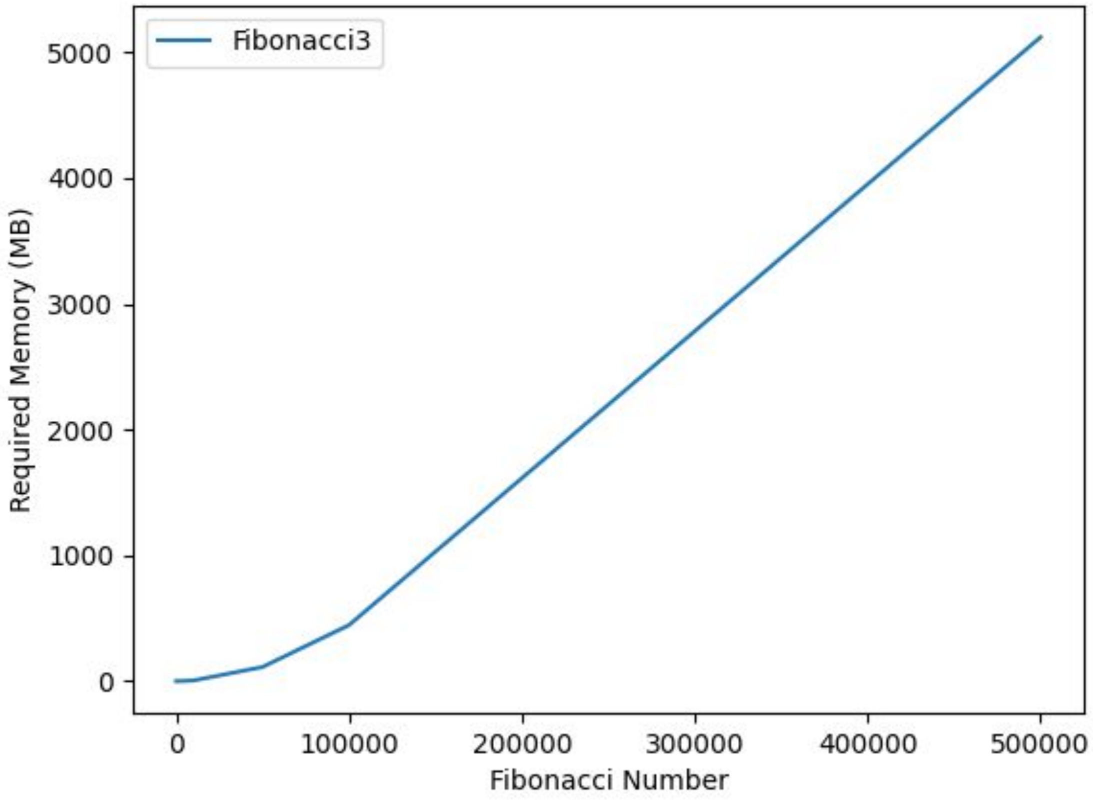
Fibonacci3



LUISS

Fibonacci3 - Execution time and memory required





LUISS

Fibonacci – A memory efficient solution

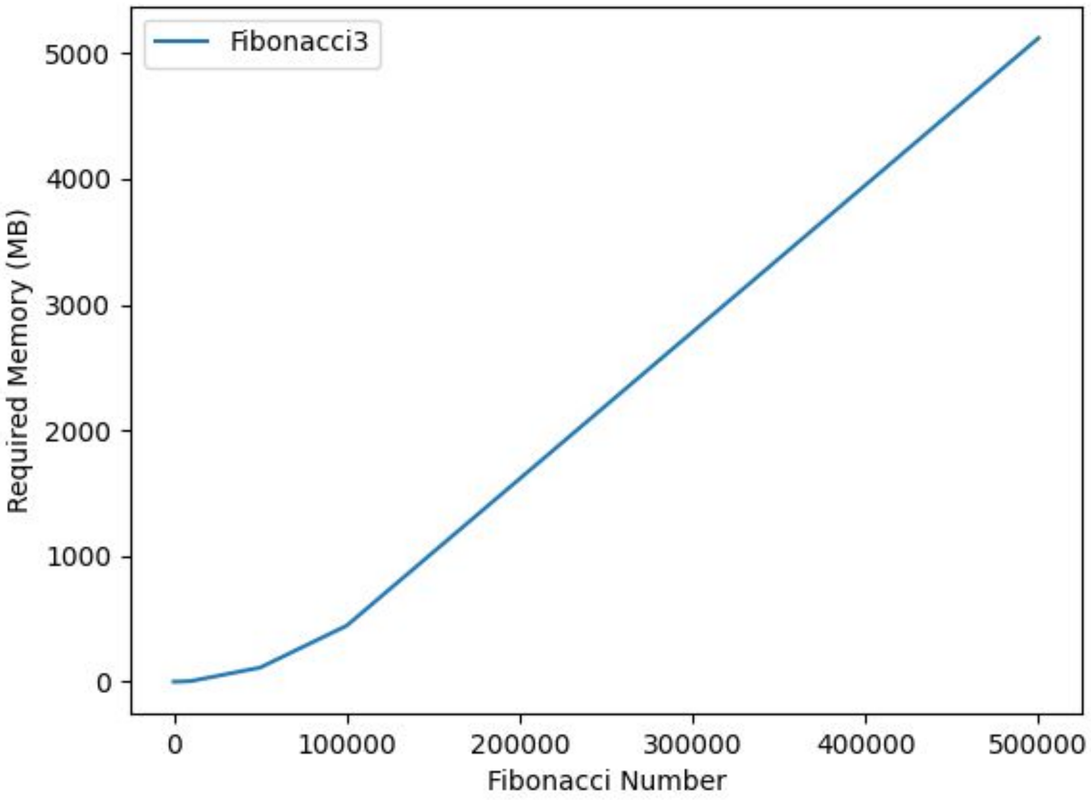
algorithm `fibonacci4`$(integer\ n) \rightarrow integer$

$a \leftarrow 1, b \leftarrow 1$
for $i = 3$ **to** n **do**
    $c \leftarrow a + b$
    $a \leftarrow b$
    $b \leftarrow c$
return b

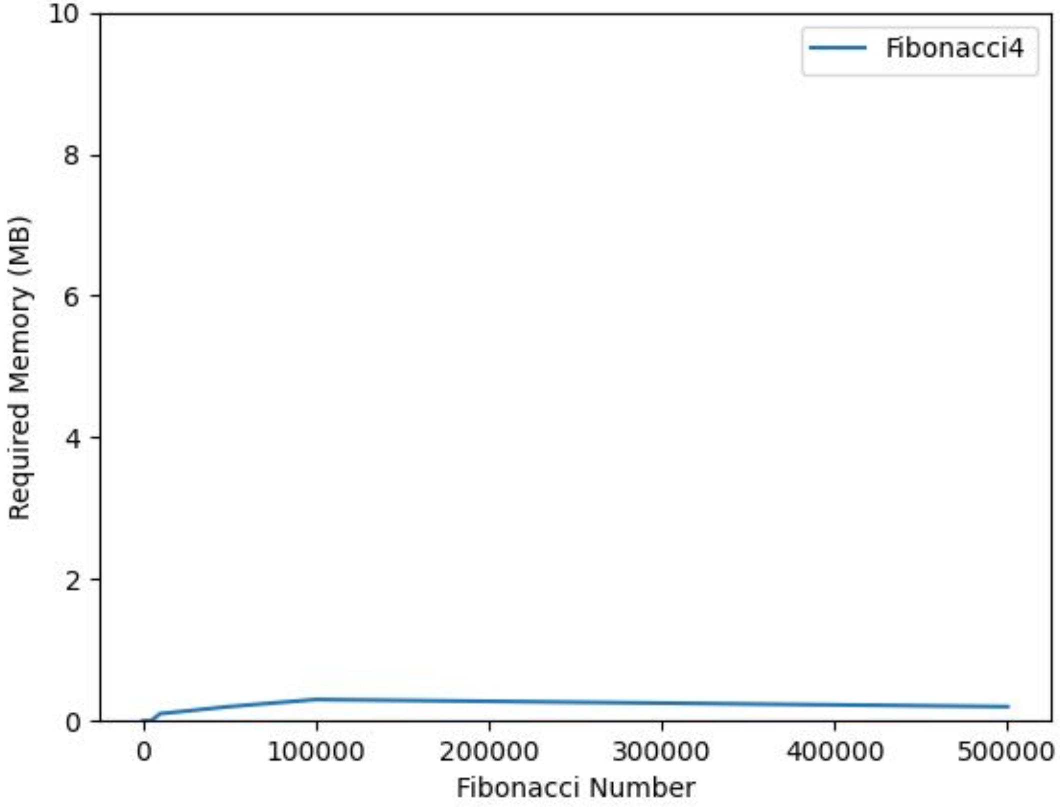
Figure 1.8 Algorithm `fibonacci4` to compute the n-th Fibonacci number.

Fibonacci - Memory Usage

Fibonacci3



Fibonacci4



LUISS

Fibonacci - Execution time

2021

2018

2012

	fibonacci2(52)	fibonacci3(52)
ARM Apple M1 3200Mhz	5210 sec. ($\simeq$ 1.5 hours)	1.7 Nanoseconds
Intel i7-8565U 4600MHz	4896 sec. ($\simeq$ 1.5 hours)	1.6 Nanoseconds
Intel i3-2365M 1400MHz	14850 sec. ($\simeq$ 4 hours)	3.2 Nanoseconds
ARM Cortex-A72 1500 MHz	17500 sec. ($\simeq$ 5 hours)	3.8 Nanoseconds

Table 1: Running time for a Python implementation of *fibonacci2 and fibonacci3*

2001

1999

	fibonacci2(58)	fibonacci3(58)
Pentium IV 1700MHz	15820 sec. ($\simeq$ 4 hours)	0.7 Nanoseconds
Pentium III 450MHz	43518 sec. ($\simeq$ 12 hours)	2.4 Nanoseconds
PowerPC G4 500MHz	58321 sec. ($\simeq$ 16 hours)	2.8 Nanoseconds

Table 2: Running time for a C implementation of *fibonacci2 and fibonacci3*

Fibonacci - Execution time

Why the Intel i3 architecture, just to compute ***fibonacci(52)***, needs the same time of an older architecture (*Pentium IV*) to compute ***fibonacci(58)***?

2021

2018

2012

	fibonacci2(52)	fibonacci3(52)
ARM Apple M1 3200Mhz	5210 sec. ($\simeq$ 1.5 hours)	1.7 Nanoseconds
Intel i7-8565U 4600MHz	4896 sec. ($\simeq$ 1.5 hours)	1.6 Nanoseconds
Intel i3-2365M 1400MHz	14850 sec. ($\simeq$ 4 hours)	3.2 Nanoseconds
ARM Cortex-A72 1500 MHz	17500 sec. ($\simeq$ 5 hours)	3.8 Nanoseconds

Table 1: Running time for a Python implementation of *fibonacci2 and fibonacci3*

2001

1999

	fibonacci2(58)	fibonacci3(58)
Pentium IV 1700MHz	15820 sec. ($\simeq$ 4 hours)	0.7 Nanoseconds
Pentium III 450MHz	43518 sec. ($\simeq$ 12 hours)	2.4 Nanoseconds
PowerPC G4 500MHz	58321 sec. ($\simeq$ 16 hours)	2.8 Nanoseconds

Table 2: Running time for a C implementation of *fibonacci2 and fibonacci3*

Fibonacci - Execution time

Why the Intel i3 architecture, just to compute ***fibonacci(52)***, needs the same time of an older architecture (*Pentium IV*) to compute ***fibonacci(58)***?

The algorithms in table 1 are implemented in **Python**, which we will see is **40 -70 times slower than C**!!

		fibonacci2(52)	fibonacci3(52)
2021	ARM Apple M1 3200Mhz	5210 sec. ($\simeq$ 1.5 hours)	1.7 Nanoseconds
2018	Intel i7-8565U 4600MHz	4896 sec. ($\simeq$ 1.5 hours)	1.6 Nanoseconds
2012	Intel i3-2365M 1400MHz	14850 sec. ($\simeq$ 4 hours)	3.2 Nanoseconds
	ARM Cortex-A72 1500 MHz	17500 sec. ($\simeq$ 5 hours)	3.8 Nanoseconds

Table 1: Running time for a Python implementation of *fibonacci2 and fibonacci3*

		fibonacci2(58)	fibonacci3(58)
2001	Pentium IV 1700MHz	15820 sec. ($\simeq$ 4 hours)	0.7 Nanoseconds
	Pentium III 450MHz	43518 sec. ($\simeq$ 12 hours)	2.4 Nanoseconds
1999	PowerPC G4 500MHz	58321 sec. ($\simeq$ 16 hours)	2.8 Nanoseconds

Table 2: Running time for a C implementation of *fibonacci2 and fibonacci3*

Fibonacci - Execution time

Why the Intel i3 architecture, just to compute ***fibonacci(52)***, needs the same time of an older architecture (*Pentium IV*) to compute ***fibonacci(58)***?

Around **2008** processors companies stopped doubling the single cpu performance, and started focusing more on parallel executions!

2021

2018

2012

	fibonacci2(52)	fibonacci3(52)
ARM Apple M1 3200Mhz	5210 sec. ($\simeq$ 1.5 hours)	1.7 Nanoseconds
Intel i7-8565U 4600MHz	4896 sec. ($\simeq$ 1.5 hours)	1.6 Nanoseconds
Intel i3-2365M 1400MHz	14850 sec. ($\simeq$ 4 hours)	3.2 Nanoseconds
ARM Cortex-A72 1500 MHz	17500 sec. ($\simeq$ 5 hours)	3.8 Nanoseconds

Table 1: Running time for a Python implementation of *fibonacci2 and fibonacci3*

2001

1999

	fibonacci2(58)	fibonacci3(58)
Pentium IV 1700MHz	15820 sec. ($\simeq$ 4 hours)	0.7 Nanoseconds
Pentium III 450MHz	43518 sec. ($\simeq$ 12 hours)	2.4 Nanoseconds
PowerPC G4 500MHz	58321 sec. ($\simeq$ 16 hours)	2.8 Nanoseconds

Table 2: Running time for a C implementation of *fibonacci2 and fibonacci3*

Memoization

What is memoization?

Memoization is an **optimization** technique for improving the performance of recursive algorithms.
It is based on the idea of **storing the results** of expensive function calls and returning the stored result when the same input occurs again.

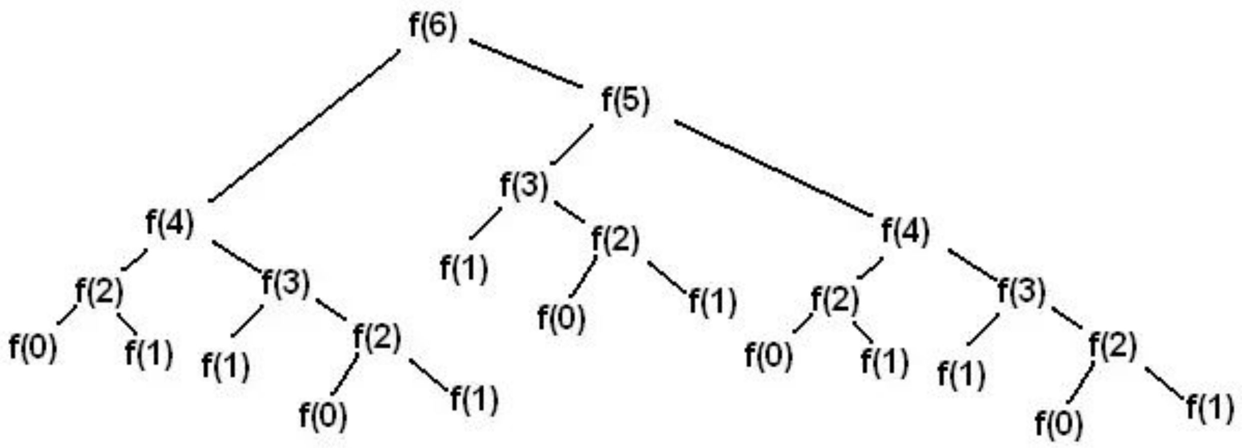


Figure 1: Functions calls to compute *F(6)* using *fibonacci2*, the recursive approach

Memoization

What is memoization?

Memoization is an **optimization** technique for improving the performance of recursive algorithms.
It is based on the idea of **storing the results** of expensive function calls and returning the stored result when the same input occurs again.

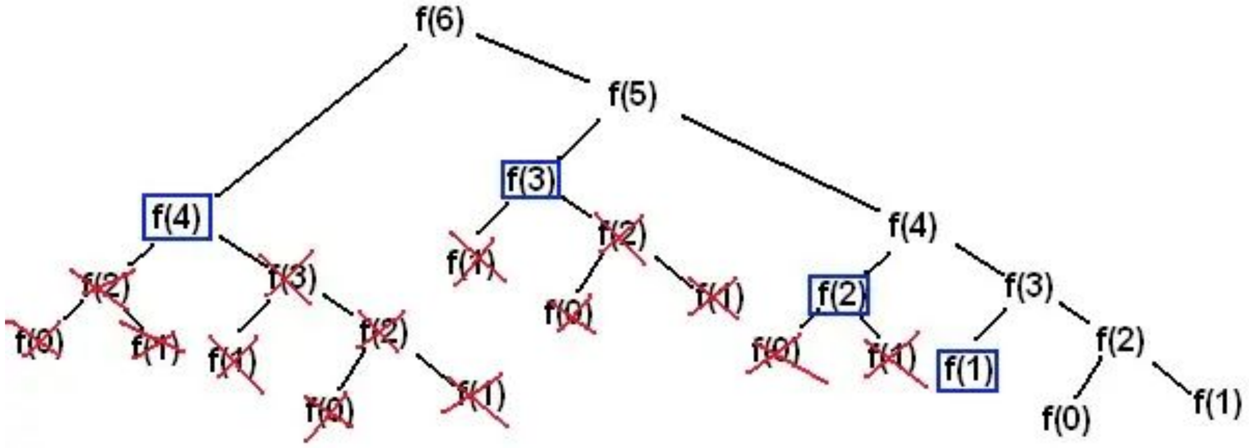


Figure 2: Functions calls to compute *F(6)* using *fibonacci2*, the recursive approach including **Memoization**

Memoization

Computational complexity for *fibonacci2* algorithm is 2^n, thus, virtually impossible to use.

Using memoization we can lower a function's **time** cost in exchange for **space** cost

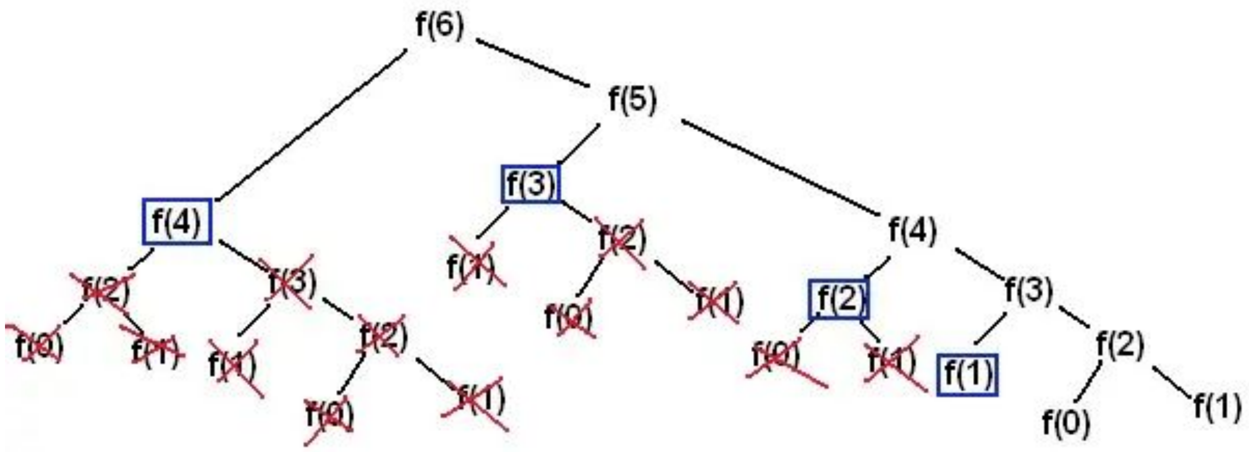


Figure 2: Functions calls to compute *F(6)* using *fibonacci2,* the recursive approach including **Memoization**