

Algorithms and Data Structures

Course Introduction

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

February 22, 2022

■ On-line course information

- ▶ on iCorsi: ***INF.B.SP22.05***
- ▶ and on my web page: ***<http://www.inf.usi.ch/carzaniga/edu/algo/>***
- ▶ previous edition also on-line: ***<http://www.inf.usi.ch/carzaniga/edu/algo21s/>***

■ On-line course information

- ▶ on iCorsi: ***INF.B.SP22.05***
- ▶ and on my web page: ***<http://www.inf.usi.ch/carzaniga/edu/algo/>***
- ▶ previous edition also on-line: ***<http://www.inf.usi.ch/carzaniga/edu/algo21s/>***

■ Announcements

- ▶ ***you are responsible for reading the announcements (posted through iCorsi)***

■ On-line course information

- ▶ on iCorsi: ***INF.B.SP22.05***
- ▶ and on my web page: ***<http://www.inf.usi.ch/carzaniga/edu/algo/>***
- ▶ previous edition also on-line: ***<http://www.inf.usi.ch/carzaniga/edu/algo21s/>***

■ Announcements

- ▶ ***you are responsible for reading the announcements (posted through iCorsi)***

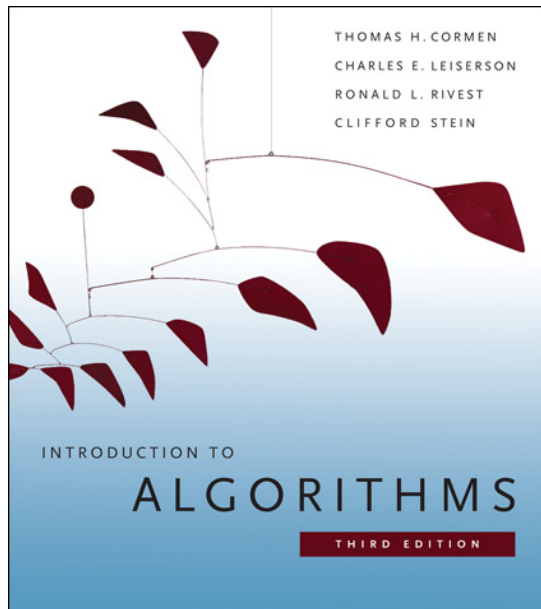
■ Office hours

- ▶ Antonio Carzaniga: *by appointment*
- ▶ Dylan Robert Ashley: *by appointment*
- ▶ Aditya Ramesh: *by appointment*
- ▶ Morteza Rezaalipour: *by appointment*

Introduction to Algorithms

Thomas H. Cormen
Charles E. Leiserson
Ronald L. Rivest
Clifford Stein

The MIT Press



- +20% homework
 - ▶ 3–5 assignments
 - ▶ grades added together, thus resulting in a weighted average
- +30% midterm exam
- +50% final exam
- $\pm 10\%$ instructor's discretionary evaluation
 - ▶ participation
 - ▶ extra credits
 - ▶ trajectory
 - ▶ ...

- +20% homework
 - ▶ 3–5 assignments
 - ▶ grades added together, thus resulting in a weighted average
- +30% midterm exam
- +50% final exam
- $\pm 10\%$ instructor's discretionary evaluation
 - ▶ participation
 - ▶ extra credits
 - ▶ trajectory
 - ▶ ...
- -100% plagiarism penalties

Plagiarism

Do NOT take someone else's material and present it as your own!

Do NOT take someone else's material and present it as your own!

- “material” means ideas, words, code, suggestions, corrections on one's work, etc.
- Using someone else's material may be appropriate
 - ▶ e.g., software libraries
 - ▶ ***always clearly identify the external material, and acknowledge its source!***
Failing to do so means committing plagiarism.
 - ▶ the work will be evaluated based on its *added value*

Do NOT take someone else's material and present it as your own!

- “material” means ideas, words, code, suggestions, corrections on one’s work, etc.
- Using someone else’s material may be appropriate
 - ▶ e.g., software libraries
 - ▶ ***always clearly identify the external material, and acknowledge its source!***
Failing to do so means committing plagiarism.
 - ▶ the work will be evaluated based on its *added value*
- Plagiarism or cheating on an assignment or an exam may result in
 - ▶ failing that assignment or that exam
 - ▶ losing one or more points *in the final note!*
- Penalties may be escalated in accordance with the regulations

Deadlines are firm!

Deadlines are firm!

- Exceptions may be granted
 - ▶ at the instructor's discretion
 - ▶ for documented medical conditions or other documented emergencies

Deadlines are firm!

- Exceptions may be granted
 - ▶ at the instructor's discretion
 - ▶ for documented medical conditions or other documented emergencies
- A late submission loses *one third* of the total value of the assignment
- Each full day of delay reduces the grade by an additional *one third*

Deadlines are firm!

- Exceptions may be granted
 - ▶ at the instructor's discretion
 - ▶ for documented medical conditions or other documented emergencies
- A late submission loses *one third* of the total value of the assignment
- Each full day of delay reduces the grade by an additional *one third*
 - ▶ The grade of an assignment turned in more than two days late is 0

an introductory example...

Fundamental Ideas

Fundamental Ideas



Johannes Gutenberg invents movable type and the printing press in Mainz, circa 1450 (already known in China and Korea, circa 1200 CE)

Maybe More Fundamental Ideas

Maybe More Fundamental Ideas

- The decimal numbering system (India, circa 600)

Maybe More Fundamental Ideas

- The decimal numbering system (India, circa 600)
- Persian mathematician Khwārizmī writes a book (Baghdad, circa 830)



Muhammad ibn Musa
al-Khwārizmī

Maybe More Fundamental Ideas

- The decimal numbering system (India, circa 600)
- Persian mathematician Khwārizmī writes a book (Baghdad, circa 830)
 - ▶ methods for adding, multiplying, and dividing numbers (and more)



Muhammad ibn Musa
al-Khwārizmī

Maybe More Fundamental Ideas

- The decimal numbering system (India, circa 600)
- Persian mathematician Khwārizmī writes a book (Baghdad, circa 830)
 - ▶ methods for adding, multiplying, and dividing numbers (and more)
 - ▶ these procedures were **precise**, **unambiguous**, **mechanical**, **efficient**, and **correct**



Muhammad ibn Musa
al-Khwārizmī

Maybe More Fundamental Ideas

- The decimal numbering system (India, circa 600)
- Persian mathematician Khwārizmī writes a book (Baghdad, circa 830)
 - ▶ methods for adding, multiplying, and dividing numbers (and more)
 - ▶ these procedures were **precise**, **unambiguous**, **mechanical**, **efficient**, and **correct**
 - ▶ *they were **algorithms!***



Muhammad ibn Musa
al-Khwārizmī

Algorithms are

the essence

of computer programs

Algorithms are

the essence

of computer programs

Algorithms are

the essence

of computer programs

Algorithms are

the essence

of computer programs

Algorithms are

the essence

of computer programs

- A sequence of numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

- A sequence of numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

- The well-known Fibonacci sequence



Leonardo da Pisa (ca. 1170–ca. 1250)
son of Guglielmo “Bonaccio”
a.k.a. *Leonardo Fibonacci*

The Fibonacci Sequence

Mathematical definition:

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

The Fibonacci Sequence

Mathematical definition:

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

Implementation on a computer:

Racket

```
(define (F n)
  (cond
    ((= n 0) 0)
    ((= n 1) 1)
    (else (+ (F (- n 1)) (F (- n 2))))))
```

The Fibonacci Sequence

Mathematical definition:

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

Implementation on a computer:

Java

```
public class Fibonacci {
    public static int F(int n) {
        if (n == 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        } else {
            return F(n-1) + F(n-2);
        }
    }
}
```

The Fibonacci Sequence

Mathematical definition:

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

Implementation on a computer:

C or C++

```
int F(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return F(n-1) + F(n-2);
    }
}
```

The Fibonacci Sequence

Mathematical definition:

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

Implementation on a computer:

Python

```
def F(n):  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return F(n-1) + F(n-2)
```

The Fibonacci Sequence

Mathematical definition:

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

Implementation on a computer:

very concise C/C++ (or Java)

```
int F(int n) {  
    return (n<2)?n:F(n-1)+F(n-2);  
}
```

The Fibonacci Sequence

Mathematical definition:

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

Implementation on a computer:

“pseudo-code”

```
F(n)  
1  if n == 0  
2      return 0  
3  elseif n == 1  
4      return 1  
5  else return F(n - 1) + F(n - 2)
```

Questions on Our First Algorithm

FIBONACCI(n)

1 **if** $n == 0$

2 **return** 0

3 **elseif** $n == 1$

4 **return** 1

5 **else return** **FIBONACCI**($n - 1$) + **FIBONACCI**($n - 2$)

Questions on Our First Algorithm

FIBONACCI(n)

1 **if** $n == 0$

2 **return** 0

3 **elseif** $n == 1$

4 **return** 1

5 **else return** **FIBONACCI**($n - 1$) + **FIBONACCI**($n - 2$)

1. Is the algorithm *correct*?

- ▶ for every valid input, does it terminate?
- ▶ if so, does it do the right thing?

Questions on Our First Algorithm

FIBONACCI(n)

1 **if** $n == 0$

2 **return** 0

3 **elseif** $n == 1$

4 **return** 1

5 **else return** **FIBONACCI**($n - 1$) + **FIBONACCI**($n - 2$)

1. Is the algorithm *correct*?
 - ▶ for every valid input, does it terminate?
 - ▶ if so, does it do the right thing?
2. How much *time* does it take to complete?

Questions on Our First Algorithm

FIBONACCI(n)

1 **if** $n == 0$

2 **return** 0

3 **elseif** $n == 1$

4 **return** 1

5 **else return** **FIBONACCI**($n - 1$) + **FIBONACCI**($n - 2$)

1. Is the algorithm *correct*?
 - ▶ for every valid input, does it terminate?
 - ▶ if so, does it do the right thing?
2. How much *time* does it take to complete?
3. Can we do better?

FIBONACCI(n)

1 **if** $n == 0$

2 **return** 0

3 **elseif** $n == 1$

4 **return** 1

5 **else return** **FIBONACCI**($n - 1$) + **FIBONACCI**($n - 2$)

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

FIBONACCI(n)

1 **if** $n == 0$

2 **return** 0

3 **elseif** $n == 1$

4 **return** 1

5 **else return** **FIBONACCI**($n - 1$) + **FIBONACCI**($n - 2$)

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

■ The algorithm is clearly correct

▶ assuming $n \geq 0$

- How long does it take?

- How long does it take?

Let's try it out...

- Different implementations perform differently
 - ▶ it is better to let the compiler do the optimization
 - ▶ simple language tricks don't seem to pay off

- Different implementations perform differently
 - ▶ it is better to let the compiler do the optimization
 - ▶ simple language tricks don't seem to pay off
- However, the differences are not substantial
 - ▶ *all* implementations sooner or later seem to hit a wall...

- Different implementations perform differently
 - ▶ it is better to let the compiler do the optimization
 - ▶ simple language tricks don't seem to pay off
- However, the differences are not substantial
 - ▶ *all* implementations sooner or later seem to hit a wall...
- Conclusion: ***the problem is with the algorithm***

Complexity of Our First Algorithm

- We need a mathematical characterization of the performance of the algorithm

We'll call it the algorithm's *computational complexity*

Complexity of Our First Algorithm

- We need a mathematical characterization of the performance of the algorithm

We'll call it the algorithm's *computational complexity*

- Let $T(n)$ be the number of *basic steps* needed to compute **FIBONACCI**(n)

Complexity of Our First Algorithm

- We need a mathematical characterization of the performance of the algorithm

We'll call it the algorithm's *computational complexity*

- Let $T(n)$ be the number of *basic steps* needed to compute **FIBONACCI**(n)

```
FIBONACCI( $n$ )
```

```
1  if  $n == 0$ 
```

```
2      return 0
```

```
3  elseif  $n == 1$ 
```

```
4      return 1
```

```
5  else return FIBONACCI( $n - 1$ ) + FIBONACCI( $n - 2$ )
```

Complexity of Our First Algorithm

- We need a mathematical characterization of the performance of the algorithm

We'll call it the algorithm's *computational complexity*

- Let $T(n)$ be the number of *basic steps* needed to compute **FIBONACCI**(n)

```
FIBONACCI( $n$ )  
1  if  $n == 0$   
2      return 0  
3  elseif  $n == 1$   
4      return 1  
5  else return FIBONACCI( $n - 1$ ) + FIBONACCI( $n - 2$ )
```

$$T(0) = 2; T(1) = 3$$

Complexity of Our First Algorithm

- We need a mathematical characterization of the performance of the algorithm

We'll call it the algorithm's *computational complexity*

- Let $T(n)$ be the number of *basic steps* needed to compute **FIBONACCI**(n)

```
FIBONACCI( $n$ )  
1  if  $n == 0$   
2      return 0  
3  elseif  $n == 1$   
4      return 1  
5  else return FIBONACCI( $n - 1$ ) + FIBONACCI( $n - 2$ )
```

$$T(0) = 2; T(1) = 3$$

$$T(n) = T(n - 1) + T(n - 2) + 3$$

Complexity of Our First Algorithm

- We need a mathematical characterization of the performance of the algorithm

We'll call it the algorithm's *computational complexity*

- Let $T(n)$ be the number of *basic steps* needed to compute **FIBONACCI**(n)

```
FIBONACCI( $n$ )  
1  if  $n == 0$   
2      return 0  
3  elseif  $n == 1$   
4      return 1  
5  else return FIBONACCI( $n - 1$ ) + FIBONACCI( $n - 2$ )
```

$$T(0) = 2; T(1) = 3$$

$$T(n) = T(n - 1) + T(n - 2) + 3 \Rightarrow T(n) \geq F_n$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2}$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4})$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6}))$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \dots$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \dots \geq 2^{\frac{n}{2}}$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \dots \geq 2^{\frac{n}{2}}$$

This means that

$$T(n) \geq (\sqrt{2})^n \approx (1.4)^n$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \dots \geq 2^{\frac{n}{2}}$$

This means that

$$T(n) \geq (\sqrt{2})^n \approx (1.4)^n$$

- $T(n)$ **grows exponentially** with n

Complexity of Our First Algorithm (2)

- So, let's try to understand how F_n grows with n

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Now, since $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \dots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \dots \geq 2^{\frac{n}{2}}$$

This means that

$$T(n) \geq (\sqrt{2})^n \approx (1.4)^n$$

- $T(n)$ **grows exponentially** with n
- Can we do better?

A Better Algorithm

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

A Better Algorithm

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . **Idea:** we can build F_n from the ground up!

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . **Idea:** we can build F_n from the ground up!

```
SMARTFIBONACCI(n)
1  if n == 0
2      return 0
3  elseif n == 1
4      return 1
5  else pprev = 0
6      prev = 1
7      for i = 2 to n
8          f = prev + pprev
9          pprev = prev
10         prev = f
11 return f
```


Complexity of SMARTFIBONACCI

SMARTFIBONACCI(n)

```
1  if  $n == 0$ 
2      return 0
3  elseif  $n == 1$ 
4      return 1
5  else  $prev = 0$ 
6       $pprev = 1$ 
7      for  $i = 2$  to  $n$ 
8           $f = prev + pprev$ 
9           $pprev = prev$ 
10          $prev = f$ 
11 return  $f$ 
```


Complexity of SMARTFIBONACCI

```
SMARTFIBONACCI(n)
1  if n == 0
2      return 0
3  elseif n == 1
4      return 1
5  else prev = 0
6      pprev = 1
7      for i = 2 to n
8          f = prev + pprev
9          pprev = prev
10         prev = f
11 return f
```

$T(n) =$

Complexity of SMARTFIBONACCI

```
SMARTFIBONACCI(n)  
1  if n == 0  
2      return 0  
3  elseif n == 1  
4      return 1  
5  else prev = 0  
6      pprev = 1  
7      for i = 2 to n  
8          f = prev + pprev  
9          pprev = prev  
10         prev = f  
11 return f
```

$$T(n) = 6 + 6(n - 1)$$

Complexity of SMARTFIBONACCI

```
SMARTFIBONACCI(n)  
1  if n == 0  
2      return 0  
3  elseif n == 1  
4      return 1  
5  else prev = 0  
6      pprev = 1  
7      for i = 2 to n  
8          f = prev + pprev  
9          pprev = prev  
10         prev = f  
11 return f
```

$$T(n) = 6 + 6(n - 1) = 6n$$

Complexity of SMARTFIBONACCI

```
SMARTFIBONACCI(n)
1  if n == 0
2      return 0
3  elseif n == 1
4      return 1
5  else prev = 0
6      pprev = 1
7      for i = 2 to n
8          f = prev + pprev
9          pprev = prev
10         prev = f
11 return f
```

$$T(n) = 6 + 6(n - 1) = 6n$$

The complexity of **SMARTFIBONACCI**(*n*) is *linear* in *n*