Algorithms and Data Structures

Course Introduction

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

February 18, 2020

General Information

- On-line course information
  - on iCorsi: ***INFO.ALGO20***
  - and on my web page: ***http://www.inf.usi.ch/carzaniga/edu/algo/***
  - last edition also on-line: ***http://www.inf.usi.ch/carzaniga/edu/algo19s/***

General Information

- On-line course information
  - on iCorsi: ***INFO.ALGO20***
  - and on my web page: ***http://www.inf.usi.ch/carzaniga/edu/algo/***
  - last edition also on-line: ***http://www.inf.usi.ch/carzaniga/edu/algo19s/***

- Announcements
  - ***you are responsible for reading the announcements page or the messages sent through iCorsi***

General Information

- On-line course information
  - on iCorsi: ***INFO.ALGO20***
  - and on my web page: ***http://www.inf.usi.ch/carzaniga/edu/algo/***
  - last edition also on-line: ***http://www.inf.usi.ch/carzaniga/edu/algo19s/***
- Announcements
  - ***you are responsible for reading the announcements page or the messages sent through iCorsi***
- Office hours
  - Antonio Carzaniga: *by appointment*
  - Ali Fattaholmanan: *by appointment*
  - Afrouz Jabalameli: *by appointment*
  - Ioannis Mantas: *by appointment*

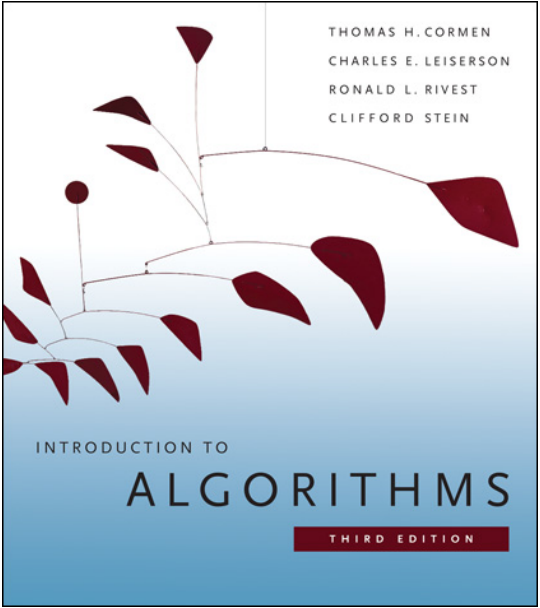
Textbook

Introduction to Algorithms

Third Edition

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

The MIT Press



Evaluation

- +30% homework
  - 3–5 assignments
  - grades added together, thus resulting in a weighted average
- +30% midterm exam
- +40% final exam
- ±10% instructor's discretionary evaluation
  - participation
  - extra credits
  - trajectory
  - ...

Evaluation

- +30% homework
  - 3–5 assignments
  - grades added together, thus resulting in a weighted average
- +30% midterm exam
- +40% final exam
- ±10% instructor's discretionary evaluation
  - participation
  - extra credits
  - trajectory
  - ...
- −100% plagiarism penalties

Plagiarism

Plagiarism

You should never take someone else’s material and present it as your own.

Plagiarism

You should never 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

You should never 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 of the Faculty of Informatics

Deadlines

Deadlines

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
- Each late day will reduce the assignment’s grade by *one third* of the total value of that assignment

Deadlines

Deadlines are firm.

- Exceptions may be granted
  - at the instructor’s discretion
  - for documented medical conditions or other documented emergencies
- Each late day will reduce the assignment’s grade by *one third* of the total value of that assignment
  - **Corollary 1:** The grade of an assignment turned in more than two days late is 0

(The proof of Corollary 1 is left as an exercise)

Now let’s move on to the real

interesting and fun stuff. . .

Fundamental Ideas

Fundamental Ideas

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

Example

- A sequence of numbers

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

- A sequence of numbers

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

- The well-known Fibonacci sequence

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

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

- 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))))))
```

- 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);
    } }
}
```

- 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);
  }
}
```

- 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:

Ruby

```
def F(n)
  case n
    when 0
      return 0
    when 1
      return 1
    else
      return F(n-1) + F(n-2)
  end
end
```

- 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)
```

- 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”

```
FIBONACCI(n)
if n == 0
      return 0
elseif n == 1
      return 1
else return FIBONACCI(n − 1) + FIBONACCI(n − 2)
```

Questions on Our First Algorithm

```
FIBONACCI(n)
if n == 0
    return 0
elseif n == 1
    return 1
else return FIBONACCI(n − 1) + FIBONACCI(n − 2)
```

Questions on Our First Algorithm

```
FIBONACCI(n)
if n == 0
        return 0
elseif n == 1
        return 1
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)
if n == 0
        return 0
elseif n == 1
        return 1
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)
if n == 0
      return 0
elseif n == 1
      return 1
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)
if n == 0
        return 0
elseif n == 1
        return 1
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)
if n == 0
      return 0
elseif n == 1
      return 1
else return FIBONACCI(n − 1) + FIBONACCI(n − 2)
```

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

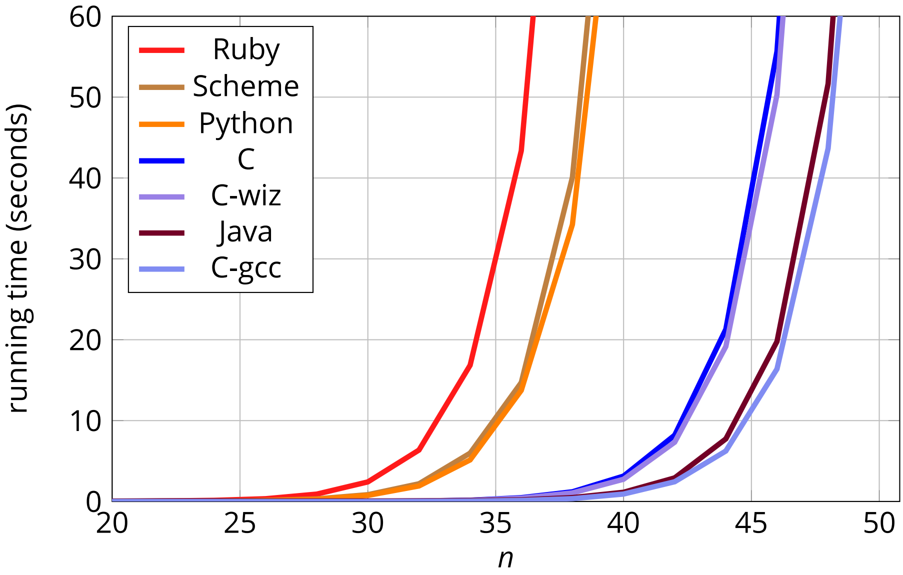
- The algorithm is clearly correct
  - assuming $n \geq 0$

Performance

- How long does it take?

- How long does it take?

  Let's try it out. . .


Ruby
Scheme
Python
C
C-wiz
Java
C-gcc
running time (seconds)
60
50
40
30
20
10
0
20
25
30
35
40
45
50
n

Comments

Comments

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

Comments

- 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. . .

Comments

- 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)
if n == 0
        return 0
elseif n == 1
        return 1
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)

if $n == 0$
        return 0
elseif $n == 1$
        return 1
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)
if n == 0
        return 0
elseif n == 1
        return 1
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)
if n == 0
        return 0
elseif n == 1
        return 1
else return FIBONACCI(n − 1) + FIBONACCI(n − 2)
```

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

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

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 $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

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

- 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 $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

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

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 $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

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

- 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 $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

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

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 $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

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

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 $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

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

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

- Again, the sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

A Better Algorithm

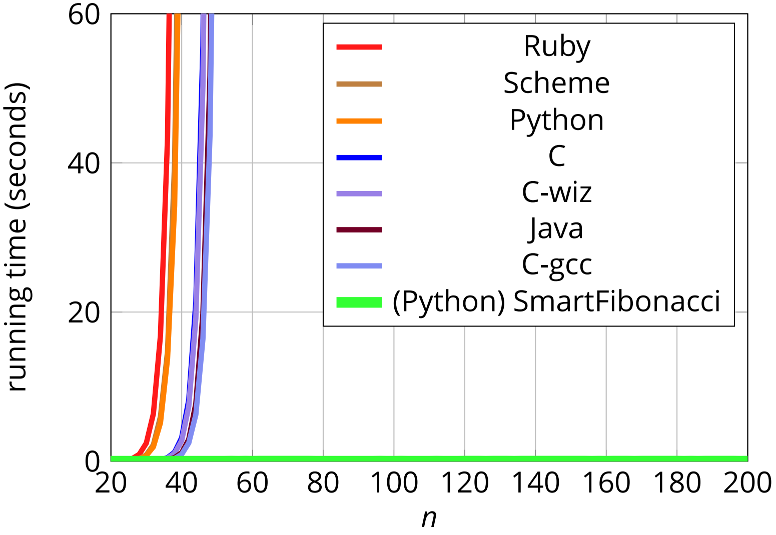
- Again, the sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .
- **Idea:** we can build F_n from the ground up!

A Better Algorithm

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

```
SMARTFIBONACCI(n)
if n == 0
        return 0
elseif n == 1
        return 1
else pprev = 0
        prev = 1
        for i = 2 to n
                f = prev + pprev
                pprev = prev
                prev = f
return f
```

Results



running time (seconds)
60
40
20
0
20
40
60
80
100
120
140
160
180
200
n
Ruby
Scheme
Python
C
C-wiz
Java
C-gcc
(Python) SmartFibonacci

Complexity of SmartFibonacci

```
SmartFibonacci(n)
if n == 0
        return 0
elseif n == 1
        return 1
else prev = 0
        pprev = 1
        for i = 2 to n
                f = prev + pprev
                pprev = prev
                prev = f
return f
```

Complexity of SMARTFIBONACCI

```
SMARTFIBONACCI(n)
if n == 0
        return 0
elseif n == 1
        return 1
else prev = 0
        pprev = 1
        for i = 2 to n
                f = prev + pprev
                pprev = prev
                prev = f
return f
```

$T(n) =$

Complexity of SmartFibonacci

```
SmartFibonacci(n)
if n == 0
        return 0
elseif n == 1
        return 1
else prev = 0
        pprev = 1
        for i = 2 to n
                f = prev + pprev
                pprev = prev
                prev = f
return f
```

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

Complexity of SmartFibonacci

```
SmartFibonacci(n)
if n == 0
        return 0
elseif n == 1
        return 1
else prev = 0
        pprev = 1
        for i = 2 to n
                f = prev + pprev
                pprev = prev
                prev = f
return f
```

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

Complexity of SmartFibonacci

```
SmartFibonacci(n)
if n == 0
        return 0
elseif n == 1
        return 1
else prev = 0
        pprev = 1
        for i = 2 to n
                f = prev + pprev
                pprev = prev
                prev = f
return f
```

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

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