Algorithms and Data Structures

Course Introduction

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

February 20, 2024

General Information

- On-line course information
  - on iCorsi: ***INF.B.SP 2024.324***
  - and on my web page: ***https://www.inf.usi.ch/carzaniga/edu/algo/***
  - previous edition also on-line: ***https://www.inf.usi.ch/carzaniga/edu/algo23s/***

General Information

- On-line course information
  - on iCorsi: ***INF.B.SP 2024.324***
  - and on my web page: ***https://www.inf.usi.ch/carzaniga/edu/algo/***
  - previous edition also on-line: ***https://www.inf.usi.ch/carzaniga/edu/algo23s/***
- Announcements
  - ***you are responsible for reading the announcements (posted through iCorsi)***

General Information

- On-line course information
  - on iCorsi: ***INF.B.SP 2024.324***
  - and on my web page: ***https://www.inf.usi.ch/carzaniga/edu/algo/***
  - previous edition also on-line: ***https://www.inf.usi.ch/carzaniga/edu/algo23s/***
- Announcements
  - ***you are responsible for reading the announcements (posted through iCorsi)***
- Personal consultations: ***by appointment***
  - Antonio Carzaniga (yours, truly)
  - Thomas Bertini
  - Michal Burgunder
  - Fabio Di Lauro
  - Koppány Encz
  - Claudio Milanesi

Assessment

Assessment

- +40% midterm exam *18 April*, 10:30-12:30, Aula Polivalente
- +60% final exam
- ±10% instructor's discretionary evaluation
  - participation
  - extra credits
  - trajectory
  - …

Assessment

- +40% midterm exam — ***18 April***, 10:30-12:30, Aula Polivalente
- +60% final exam
- ±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!

Plagiarism

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

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

A note on learning Algorithms

A note on learning Algorithms

. . . or anything else, really

A note on learning Algorithms

. . . or anything else, really

(Yes, you are here to learn!)

A note on learning Algorithms
. . . or anything else, really

(Yes, you are here to learn!)

I can not *make you* learn

A note on learning Algorithms

. . . or anything else, really

(Yes, you are here to learn!)

I can not *make you* learn—***learning is indirect!***

A note on learning Algorithms
. . . or anything else, really

(Yes, you are here to learn!)

I can not *make you* learn—***learning is indirect!***

I will do my best to present ideas, show their beauty, stimulate your interest

A note on learning Algorithms
. . . or anything else, really

(Yes, you are here to learn!)

I can not *make you* learn—***learning is indirect!***

I will do my best to present ideas, show their beauty, stimulate your interest

You have to put in enough time!—studying and exercising

A note on learning Algorithms
. . . or anything else, really

(Yes, you are here to learn!)

I can not *make you* learn—***learning is indirect!***

I will do my best to present ideas, show their beauty, stimulate your interest

You have to put in enough time!—studying and exercising

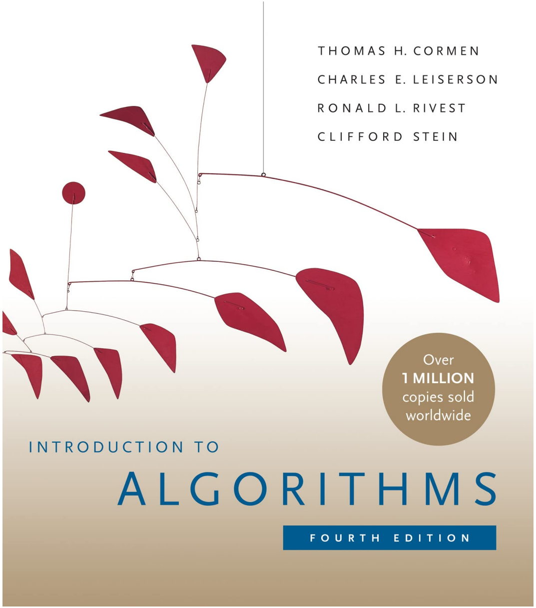
I will give you all the resources and all the help I can provide

Textbook

Introduction to Algorithms

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

The MIT Press

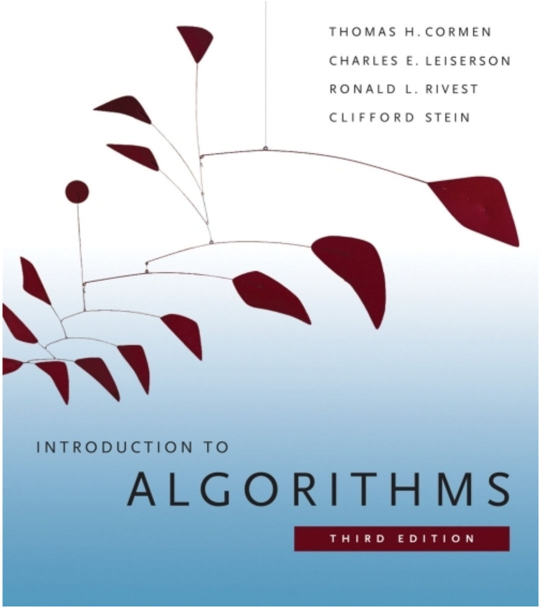


Textbook

Introduction to Algorithms

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

The MIT Press



Exercises and Other Material

Exercises and Other Material

- *Notes on Elementary Algorithmic Programming in Python*
  https://www.inf.usi.ch/carzaniga/edu/algo/programming.html

Exercises and Other Material

- *Notes on Elementary Algorithmic Programming in Python*
  https://www.inf.usi.ch/carzaniga/edu/algo/programming.html

- Exercises for Elementary Algorithmic Programming in Python
  https://www.inf.usi.ch/carzaniga/edu/algo/python_exercises.html

Exercises and Other Material

- *Notes on Elementary Algorithmic Programming in Python*
  https://www.inf.usi.ch/carzaniga/edu/algo/programming.html

- Exercises for Elementary Algorithmic Programming in Python
  https://www.inf.usi.ch/carzaniga/edu/algo/python_exercises.html

- Other exercises (a bit more involved) in Python, *with solutions*
  https://www.inf.usi.ch/carzaniga/edu/python/index.html

Exercises and Other Material

- *Notes on Elementary Algorithmic Programming in Python*
  https://www.inf.usi.ch/carzaniga/edu/algo/programming.html

- Exercises for Elementary Algorithmic Programming in Python
  https://www.inf.usi.ch/carzaniga/edu/algo/python_exercises.html

- Other exercises (a bit more involved) in Python, *with solutions*
  https://www.inf.usi.ch/carzaniga/edu/python/index.html

- A collection of 298 exam exercises, many of them with solutions
  https://www.inf.usi.ch/carzaniga/edu/algo/exercises.pdf

Our Time and Energy

Our Time and Energy

- Personal meetings
  - extemporaneous, *any time I have time!*
  - individually or in small groups
  - questions, exercises, discussions, . . .

Our Time and Energy

- Personal meetings
  - extemporaneous, *any time I have time!*
  - individually or in small groups
  - questions, exercises, discussions, …
- ***Exercise sessions***
  - every ***Wednesday 15:30–17:00 in C1.04***
  - supervised exercises, analysis of solutions, discussions

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

Example: Poetic Rhythms

Imagine you are a poet, perhaps a bit of a musician, and also a mathematician…

Example: Poetic Rhythms

Imagine you are a poet, perhaps a bit of a musician, and also a mathematician…

- The rhythm of your musical poetry is based on a regular *beat*
  - a “beat” is the basic unit of time

Example: Poetic Rhythms

Imagine you are a poet, perhaps a bit of a musician, and also a mathematician…

- The rhythm of your musical poetry is based on a regular *beat*
  - a “beat” is the basic unit of time

- You compose your rhythms with one- and two-beat intervals
  - a rhythm is a sequence of elements (words, syllables, notes) of 1 or 2 time units

Example: Poetic Rhythms

Imagine you are a poet, perhaps a bit of a musician, and also a mathematician…

- The rhythm of your musical poetry is based on a regular *beat*
  - a "beat" is the basic unit of time

- You compose your rhythms with one- and two-beat intervals
  - a rhythm is a sequence of elements (words, syllables, notes) of 1 or 2 time units

How many 1,2-rhythms can you compose over a total of* n *beats?

Example: Poetic Rhythms

How many 1,2-rhythms can you compose over a total of n beats?

Example: Poetic Rhythms

How many 1,2-rhythms can you compose over a total of n beats?

Let's call this function **PINGALA**(n), or $P(n)$ for short, in honor of the ancient Indian poet and mathematician who is the first person known to have studied these things

Example: Poetic Rhythms

How many 1,2-rhythms can you compose over a total of n beats?

Let's call this function **Pingala**(n), or $P(n)$ for short, in honor of the ancient Indian poet and mathematician who is the first person known to have studied these things

Example:

We have $n = 4$ total beats. How many different rhythms can we have?

Example: Poetic Rhythms

How many 1,2-rhythms can you compose over a total of n beats?

Let's call this function **PINGALA**(n), or $P(n)$ for short, in honor of the ancient Indian poet and mathematician who is the first person known to have studied these things

Example:

We have $n = 4$ total beats. How many different rhythms can we have?

1–1–1–1	*Ta–Ta–Ta–Ta–*
1–1–2	*Ta–Ta–Ta-a–*
1–2–1	*Ta–Ta-a–Ta–*
2–1–1	*Ta-a–Ta–Ta–*
2–2	*Ta-a–Ta-a–*

Example: Poetic Rhythms

How many 1,2-rhythms can you compose over a total of n beats?

Let's call this function **PINGALA**(n), or $P(n)$ for short, in honor of the ancient Indian poet and mathematician who is the first person known to have studied these things

Example:

We have $n = 4$ total beats. How many different rhythms can we have?

1–1–1–1	*Ta–Ta–Ta–Ta–*
1–1–2	*Ta–Ta–Ta-a–*
1–2–1	*Ta–Ta-a–Ta–*
2–1–1	*Ta-a–Ta–Ta–*
2–2	*Ta-a–Ta-a–*

$$P(4) = 5$$

Example: Poetic Rhythms

How many rhythms can you compose over a total of n beats?

Example:

$$P(4) = 5$$

Example: Poetic Rhythms

How many rhythms can you compose over a total of n beats?

Example:

$$P(4) = 5$$

$$P(3) = ?$$

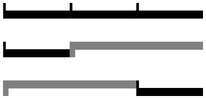
Example: Poetic Rhythms

How many rhythms can you compose over a total of n beats?

Example:

$$P(4) = 5$$

$$P(3) = 3$$



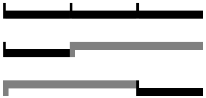
Example: Poetic Rhythms

How many rhythms can you compose over a total of n beats?

Example:

$$P(4) = 5$$

$$P(3) = 3$$



$$P(8) = ?$$

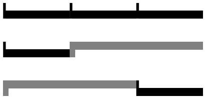
Example: Poetic Rhythms

How many rhythms can you compose over a total of n beats?

Example:

$$P(4) = 5$$

$$P(3) = 3$$



$$P(8) = ?$$

We want a general *algorithm* to compute $P(n)$

$n = 5$:

A First Algorithm

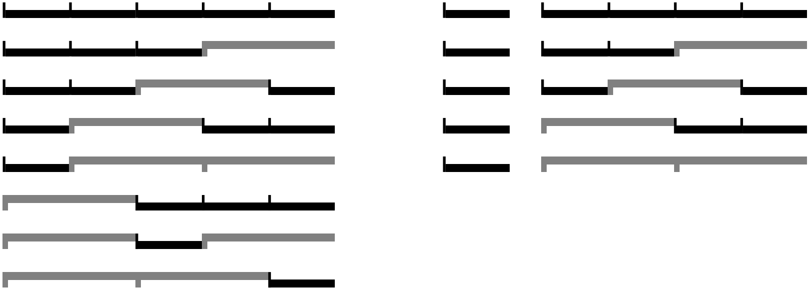
$n = 5$:

A First Algorithm

$n = 5$:

$\textsc{Pingala}(5) = 8$

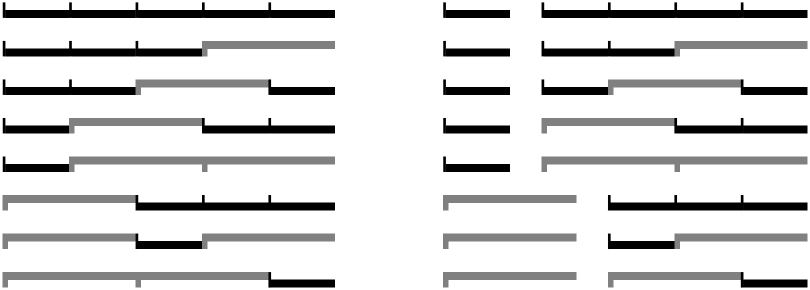
$n = 5$:



$\textsc{Pingala}(5) = 8$

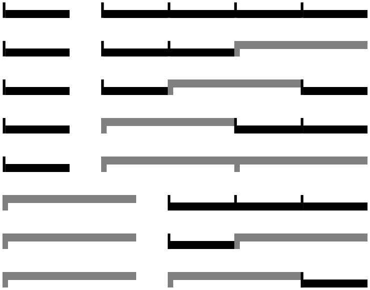
A First Algorithm

$n = 5$:



$\textbf{Pingala}(5) = 8$

$n = 5$:



PINGALA(5) = 8

PINGALA(5) = PINGALA(4) + PINGALA(3)

A First Algorithm

$n = 5$:

$\textsc{Pingala}(5) = 8$

$\textsc{Pingala}(5) = \textsc{Pingala}(4) + \textsc{Pingala}(3)$

```
PINGALA(n)
if n ≤ 2
        return n
return PINGALA(n − 1) + PINGALA(n − 2)
```

Questions on Our First Algorithm

```
PINGALA(n)
if n ≤ 2
    return n
return PINGALA(n − 1) + PINGALA(n − 2)
```

Questions on Our First Algorithm

```
PINGALA(n)
if n ≤ 2
    return n
return PINGALA(n − 1) + PINGALA(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

```
PINGALA(n)
if n ≤ 2
        return n
return PINGALA(n − 1) + PINGALA(n − 2)
```

1. Is the algorithm ***correct?***
   - for every valid input, does it terminate?
   - if so, does it do the right thing?

2. Is the algorithm ***efficient?***
   - How much ***time*** does it take to complete?

Questions on Our First Algorithm

```
PINGALA(n)
  if n ≤ 2
      return n
  return PINGALA(n − 1) + PINGALA(n − 2)
```

1. Is the algorithm ***correct?***
   - for every valid input, does it terminate?
   - if so, does it do the right thing?

2. Is the algorithm ***efficient?***
   - How much ***time*** does it take to complete?

3. Can we do better?

Correctness

```
PINGALA(n)
if n ≤ 2
    return n
return PINGALA(n − 1) + PINGALA(n − 2)
```

Correctness

```
PINGALA(n)
if n ≤ 2
        return n
return PINGALA(n − 1) + PINGALA(n − 2)
```

- For now we wave our hands…
  - “the algorithm is clearly correct!”
  - assuming $n > 0$

Performance

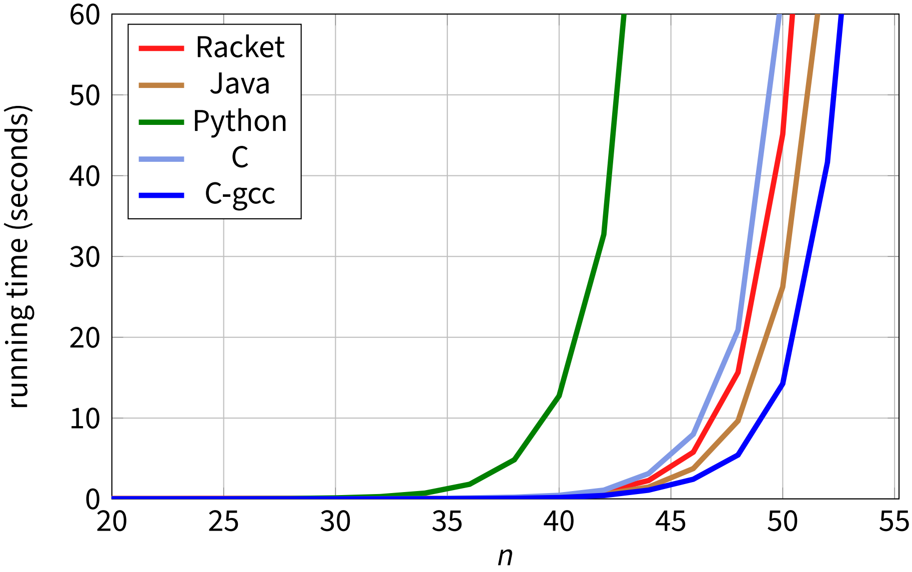
- How long does it take?

Performance

- How long does it take?

  Let's try it out…

Results



Racket
Java
Python
C
C-gcc
running time (seconds)
0
10
20
30
40
50
60
20
25
30
35
40
45
50
55
n

Comments

Comments

- Different implementations perform differently
  - with different languages you get different performances
  - compiler optimizations can make a difference

Comments

- Different implementations perform differently
  - with different languages you get different performances
  - compiler optimizations can make a difference
- However, the differences are not substantial
  - *all* implementations sooner or later seem to hit a wall…

Comments

- Different implementations perform differently
  - with different languages you get different performances
  - compiler optimizations can make a difference
- 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 **PINGALA**(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 **PINGALA**(n)

```
PINGALA(n)
if n ≤ 2
    return n
return PINGALA(n − 1) + PINGALA(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 **PINGALA**(n)

```
PINGALA(n)
1  if n ≤ 2
2      return n
3  return PINGALA(n − 1) + PINGALA(n − 2)
```

$T(1) = T(2) = 2$

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 **PINGALA**(n)

```
PINGALA(n)
if n ≤ 2
        return n
return PINGALA(n − 1) + PINGALA(n − 2)
```

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

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 **PINGALA**(n)

```
PINGALA(n)
if n ≤ 2
      return n
return PINGALA(n − 1) + PINGALA(n − 2)
```

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

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \ldots$

$$T(n) \geq 2T(n-2)$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \dots$

$$T(n) \geq 2T(n-2) \geq 2(2T(n-4))$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \ldots$

$$T(n) \geq 2T(n-2) \geq 2(2T(n-4)) \geq 2(2(2T(n-6)))$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \ldots$

$$T(n) \geq 2T(n-2) \geq 2(2T(n-4)) \geq 2(2(2T(n-6))) \geq \ldots$$

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n)$ = grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \ldots$

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

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \ldots$

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

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 $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \ldots$

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

This means that

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

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

Complexity of Our First Algorithm (2)

- So, let's try to understand how $T(n) =$ grows with n

$$T(n) \geq T(n-1) + T(n-2)$$

Now, since $T(n) \geq T(n-1) \geq T(n-2) \geq T(n-3) \geq \ldots$

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

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

A Better Algorithm

Idea: we can avoid repeating the same computations over and over again

A Better Algorithm

Idea: we can avoid repeating the same computations over and over again

```
PINGALA-MEM(n, M)
if n ≤ 2
    return n
if M == ∅
    M = array of n NIL elements
if M[n] == NIL
    M[n] = PINGALA-MEM(n − 1, M) + PINGALA-MEM(n − 2, M)
return M[n]
```

An Even Better Algorithm

An Even Better Algorithm

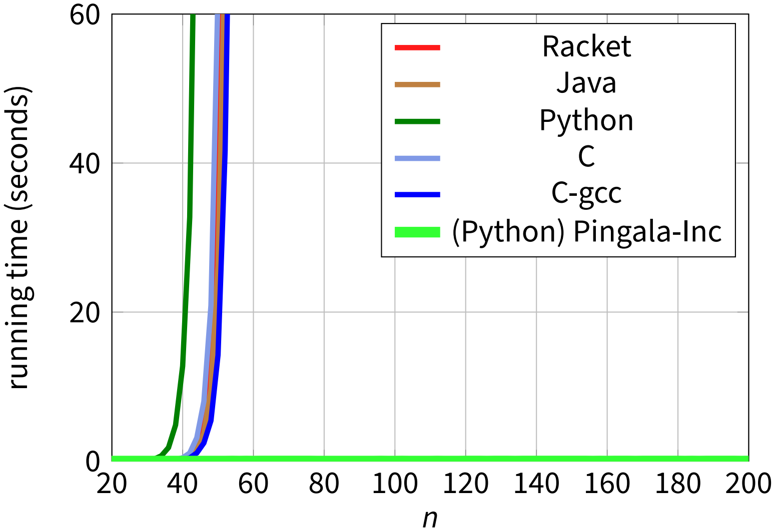
Idea: we can build $P(n)$ from the ground up, with just a couple of extra variables!

An Even Better Algorithm

Idea: we can build $P(n)$ from the ground up, with just a couple of extra variables!

```
PINGALA-INC(n)
if n ≤ 2
    return n
pprev = 1
prev = 2
for i = 3 to n
    P = prev + pprev
    pprev = prev
    prev = P
return P
```

Results



Racket
Java
Python
C
C-gcc
(Python) Pingala-Inc
running time (seconds)
60
40
20
0
20
40
60
80
100
120
140
160
180
200
n

Complexity of Pingala-Inc

```
PINGALA-INC(n)
if n ≤ 2
    return n
pprev = 1
prev = 2
for i = 3 to n
    P = prev + pprev
    pprev = prev
    prev = P
return P
```

Complexity of PINGALA-INC

```
PINGALA-INC(n)
if n ≤ 2
        return n
pprev = 1
prev = 2
for i = 3 to n
        P = prev + pprev
        pprev = prev
        prev = P
return P
```

$T(n) =$

Complexity of PINGALA-INC

```
PINGALA-INC(n)
if n ≤ 2
        return n
pprev = 1
prev = 2
for i = 3 to n
        P = prev + pprev
        pprev = prev
        prev = P
return P
```

$T(n) = 4 + 5(n - 2)$

Complexity of PINGALA-INC

```
PINGALA-INC(n)
if n ≤ 2
        return n
pprev = 1
prev = 2
for i = 3 to n
        P = prev + pprev
        pprev = prev
        prev = P
return P
```

$$T(n) = 4 + 5(n - 2) = 5n + \ldots$$

Complexity of PINGALA-INC

```
PINGALA-INC(n)
if n ≤ 2
        return n
pprev = 1
prev = 2
for i = 3 to n
        P = prev + pprev
        pprev = prev
        prev = P
return P
```

$$T(n) = 4 + 5(n-2) = 5n + \ldots = O(n)$$

Complexity of PINGALA-INC

```
PINGALA-INC(n)
if n ≤ 2
        return n
pprev = 1
prev = 2
for i = 3 to n
        P = prev + pprev
        pprev = prev
        prev = P
return P
```

$T(n) = 4 + 5(n - 2) = 5n + \ldots = O(n)$

The *complexity* of **PINGALA-INC**(n) is ***linear*** in n