THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Algorithms and Data Structures

Lecture 1 What is an algorithm and why analyse it?

Jiamou Liu
The University of Auckland

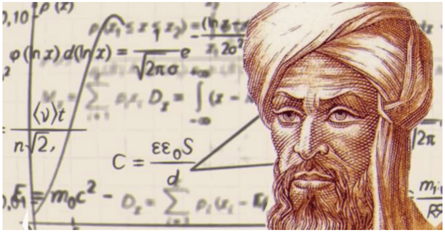
Algorithms and Data Structures

- Algorithms are sequences of clearly-stated rules that specify a step-by-step method for solving a given problem.
- Data structure are particular ways of storing and organising data in a computer system so that it can be used efficiently.
- This is a challenging course about algorithms and data structures.
- What you will acquire the abilities to:
    - analyse the efficiency of algorithms
    - choose and use algorithms
    - analyse data structures
    - choose and use data structures

Algorithms and Data Structures

Lecture 1 What is an algorithm and why analyse it?

Jiamou Liu
The University of Auckland



Course Schedule

- Algorithm analysis
- Divide and conquer
- Graph traversal
- Greedy algorithms
- Dynamic programming
- Graph and matrices
- Other advanced topics

Resources: Algorithm (Dasgupta, Papadimitriou, Vazirani) `http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf`

About Me

- Jiamou Liu
- **Research Interest:** AI, multi-agent systems, representation learning, natural language
- Joined UoA in 2016
- Web presence: `https://www.liuailab.org/`

"We are an **AI research group** at the University of Auckland. We are engaged in artificial intelligence research and development from both the industrial and the academic side. Our research interests cover a wide range of topics across the modern AI world, including deep learning, reinforcement learning, multi-agent systems, natural language processing, and complex network analysis."

A Few Notes About My Slides

- Newly introduced keywords will be coloured in dark red.
- Important phrases will be stated in blue. This highlights key entities or properties.
- Important mathematical facts (definition, theorems, algorithms, etc.) will be given in special "boxes". e.g.,

Definition.

A set is an unordered collection of distinct objects, called elements of the set. We write $a \in X$ to mean that a is an element of X.

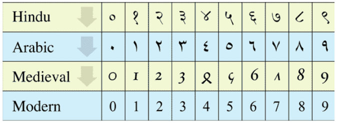
- **Examples**, **Remarks**, **Questions**, etc., will be notified in **bold**.

  e.g., **Examples.**
  - $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers
  - $\mathbb{R}$ is the set of real numbers
  - $\emptyset = \{\}$ is the empty set
- Sometimes other colours will also be used to enhance readability.

We are now starting our main story . . .

9th Century: al-Khwarizmi

“Algoritmi de numero Indorum” (al-Khwārizmī on the Hindu Art of Reckoning)



Hindu	०	१	२	३	४	५	६	७	८	९
Arabic	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Medieval	o	ɪ	2	3	ꝗ	ϥ	6	ʌ	8	9
Modern	0	1	2	3	4	5	6	7	8	9

- Introduce the Hindu/Arabic numeral system to Europe
- Describe arithmetic operations on numbers based on the Hindu system: +, -, ×, ÷
- These operations are specified by precise, unambiguous, mechanical procedures.

Mohammad ibn Musa
al-Khwarizmi
(780 - 850)

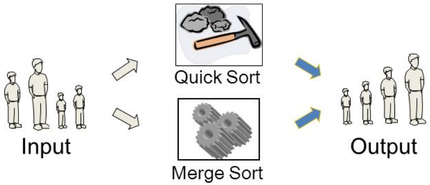
```
   359
+  276
   1 1
   635
```

```
   359
 x 276
  2154
 2513
 718
 99084
```

Two Central Questions

1. **What is an algorithm?**
   - A list of unambiguous and detailed rules that specify successive operations.
   - An idealised/abstracted version of a computer program.
2. **What is a good algorithm?**
   - It correctly produces the intended output.
   - It runs efficiently.

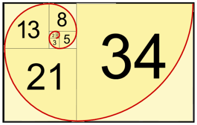
   We will focus on this question in this course.



12th Century: Leonardo of Pisa

“Liber Abaci” (The book of calculation) (1202)

Leonardo of Pisa (Fibonacci)
1170 - 1250



Fibonacci sequence: 0,1,1,2,3,5,8,13,21,33,54,. . .

- Golden ratio: $\frac{a}{b} = \frac{a+b}{a}$
- Fibonacci sequence approximates the golden ratio (≈ 1.618):

$$\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \frac{34}{21}, \frac{55}{34}, \ldots$$

Case Study: Calculating Fibonacci Sequence

Definition

The Fibonacci sequence is defined as

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

Note. $F(n)$ can be very large for small n

- $F(n) \approx 1.618^n$.
- $F(30) = 832040$.
- $F(100) = 3.54 \times 10^{20}$.

Fibonacci problem

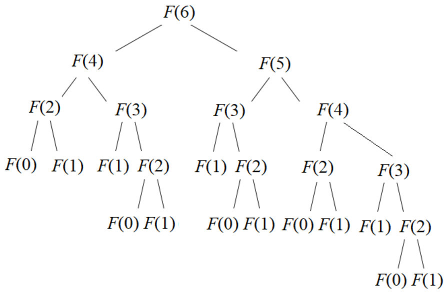
- INPUT: A number n
- OUTPUT: The value of $F(n)$

We will express algorithms using pseudocode in this course.

Algorithm 1 SLOWFIB

```
1: function SLOWFIB(integer $n$)
2:     if $n < 0$ then return 0
3:     else if $n = 0$ then return 0
4:     else if $n = 1$ then return 1
5:     else return SLOWFIB($n - 1$) + SLOWFIB($n - 2$)
```

Note. This algorithm may make a lot of recursive calls.



To compute $F(6)$, we will make 25 calls to **SLOWFIB**.

A second try: Working from the bottom-up instead of top-down.

0	1						
b	a						

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                    ▹ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                    ▹ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1					
b	a	a+b					

Algorithm 2 FASTFIB

```
 1: function FASTFIB(integer $n$)
 2:     if $n < 0$ then return 0
 3:     else if $n = 0$ then return 0
 4:     else if $n = 1$ then return 1
 5:     else
 6:         $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
 7:         $b \leftarrow 0$                    ▷ stores $F(i-1)$ at bottom of loop
 8:         for $i \leftarrow 2$ to $n$ do
 9:             $t \leftarrow a$
10:             $a \leftarrow a + b$
11:             $b \leftarrow t$
12:     return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1					
	b	a					

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                ▷ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2			
	b	a	a+b			

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                    ▷ stores $F(i - 1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2				
		b	a				

Algorithm 2 FASTFIB

```
1: function FASTFIB(integer $n$)
2:     if $n < 0$ then return 0
3:     else if $n = 0$ then return 0
4:     else if $n = 1$ then return 1
5:     else
6:         $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
7:         $b \leftarrow 0$                    ▷ stores $F(i-1)$ at bottom of loop
8:         for $i \leftarrow 2$ to $n$ do
9:             $t \leftarrow a$
10:            $a \leftarrow a + b$
11:            $b \leftarrow t$
12:    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3			
		b	a	a+b			

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                    ▷ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3			
			b	a			

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                    ▷ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3	5		
			b	a	a+b		

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                    ▷ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3	5		
				b	a		

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                    ▷ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3	5	8	
				b	a	a+b	

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                    ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                    ▷ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3	5	8	
					b	a	

Algorithm 2 FASTFIB

```
function FASTFIB(integer $n$)
    if $n < 0$ then return 0
    else if $n = 0$ then return 0
    else if $n = 1$ then return 1
    else
        $a \leftarrow 1$                          ▷ stores $F(i)$ at bottom of loop
        $b \leftarrow 0$                          ▷ stores $F(i-1)$ at bottom of loop
        for $i \leftarrow 2$ to $n$ do
            $t \leftarrow a$
            $a \leftarrow a + b$
            $b \leftarrow t$
    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3	5	8	13
					b	a	a+b

Algorithm 2 FASTFIB

```
1: function FASTFIB(integer $n$)
2:     if $n < 0$ then return 0
3:     else if $n = 0$ then return 0
4:     else if $n = 1$ then return 1
5:     else
6:         $a \leftarrow 1$                          ▷ stores $F(i)$ at bottom of loop
7:         $b \leftarrow 0$                          ▷ stores $F(i-1)$ at bottom of loop
8:         for $i \leftarrow 2$ to $n$ do
9:             $t \leftarrow a$
10:            $a \leftarrow a + b$
11:            $b \leftarrow t$
12:    return $a$
```

A second try: Working from the bottom-up instead of top-down.

0	1	1	2	3	5	8	13
					b	a	a+b

Algorithm 2 FASTFIB

```
1:  function FASTFIB(integer $n$)
2:      if $n < 0$ then return 0
3:      else if $n = 0$ then return 0
4:      else if $n = 1$ then return 1
5:      else
6:          $a \leftarrow 1$                ▷ stores $F(i)$ at bottom of loop
7:          $b \leftarrow 0$                ▷ stores $F(i-1)$ at bottom of loop
8:          for $i \leftarrow 2$ to $n$ do
9:              $t \leftarrow a$
10:             $a \leftarrow a + b$
11:             $b \leftarrow t$
12:     return $a$
```

To compute $F(6)$, **FASTFIB** will make $3 \times 5 + 2 = 17$ assignment operations to a, b, t.

Algorithm Analysis

Question 1. How many assignment operations[1] are performed by **FASTFIB**

(a) to compute $F(7)$?

- Each iteration of the **for**-loop makes 3 assignment operations.
- To compute 7 requires 6 iterations of the **for**-loop.
- The total number of assignment operations is $2 + 3 \times 7 = 23$

(b) to compute $F(n)$ for $n \geq 2$?

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

[1]We can consider the number of assignment operations as an indicator of the number of instructions ran by the algorithm.

Question 2. How many recursive calls[2] are made by **SLOWFIB**

(a) to compute $F(7)$?

- $F(5)$ made 15 recursive calls.
- $F(6)$ made 25 recursive calls.
- $F(7)$ requires $15 + 25 + 1 = 41$ recursive calls.

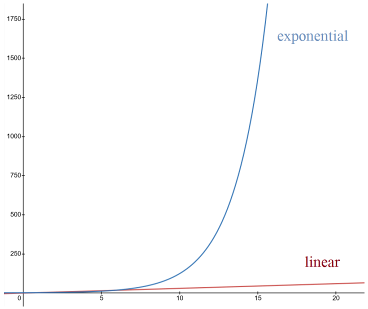
(b) to compute $F(n)$ for $n \geq 2$?

- Let $T(n)$ be the number of recursive calls to compute $F(n)$.
- If $n \geq 2$, $T(n) = T(n-1) + T(n-2) + 1 > T(n-1) + T(n-2)$.
- Thus
$$T(n) > F(n) \approx 1.618^n$$

[2]We can consider the number of recursive calls as an indicator of the number of instructions ran by the algorithm.

Question 3. How would you compare the numbers of instructions ran by **SLOWFIB** and by **FASTFIB**?

- The number of instructions ran by **FASTFIB** is of the form $An + B$ for constants $A > 0$ and B. This is called a linear function.
- The number of instructions ran by **SLOWFIB** is of the form C^n for constant $C > 1$. This is called an exponential function.



Thus **FASTFIB** wins over **SLOWFIB** in calculating $F(n)$ for sufficiently large n!

Summary

In this lecture, we covered the following:

- What this course is about.
- What an algorithm is.
- How to express an algorithm: pseudocode.
- Fibonacci sequence and the golden ratio.
- Two algorithms for computing $F(n)$, **SLOWFIB** and **FASTFIB**.
- The number of instructions executed by the algorithms:
    - **SLOWFIB**: exponential function
    - **FASTFIB**: linear function

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND