

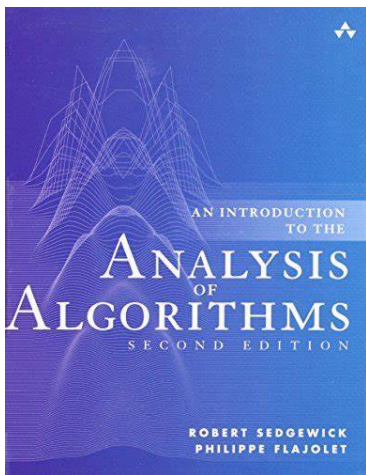
## 2-2 The Efficiency of Algorithms

Hengfeng Wei

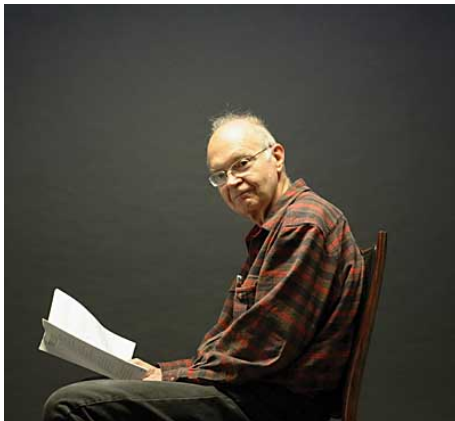
hfwei@nju.edu.cn

March 05, 2020





## The Analysis of Algorithms



Donald E. Knuth (1938 ~)



Donald E. Knuth (1974)

*“For his major contributions to **the analysis of algorithms** and **the design of programming languages**, and in particular for his contributions to the **“art of computer programming”** through his well-known books in a continuous series by this title.”*

Fibonacci numbers in the analysis of Euclid's GCD algorithm  
 $H_n$  in the analysis of FIND-MAX @ Stanford Lecture by Knuth

*“People who **analyze algorithms** have **double happiness**.*

*First of all they experience the sheer beauty of elegant **mathematical patterns** that surround elegant **computational procedures**.*

*Then they receive a **practical payoff** when their theories make it possible to get other jobs done **more quickly and more economically**.”*

## How Fast is It?



Time (and Space) Complexity of Algorithms

$O$   $\Omega$   $\Theta$

$o$   $\omega$

## Space Complexity of Algorithms

We only care about the **extra** space caused by the algorithm.

The space for **inputs** is not part of space complexity of algorithms.

INSERTION-SORT( $A, n$ ) :  $O(1)$  (constant)

Is it the Fastest?



Complexity of Problems

This is much harder and is not our focus today.





Whenever you design an algorithm,  
you provide an **upper bound** for the **complexity of the problem**.

Whenever you encounter a “hardcore” of the problem,  
you obtain a **lower bound** for **all possible algorithms**.

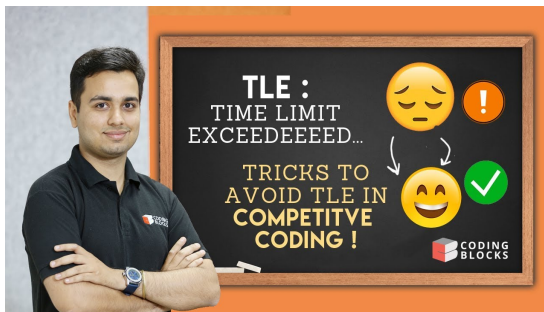
Often, there is an “**algorithmic gap**” between them.

When the gap is gone, you get the **optimal** algorithm.

$$\text{sorting}(A, n) : \Theta(n \log n) = O(n \log n) \cap \Omega(n \log n)$$

Q : How fast is your algorithm?

A : It runs 3.1415926 seconds.



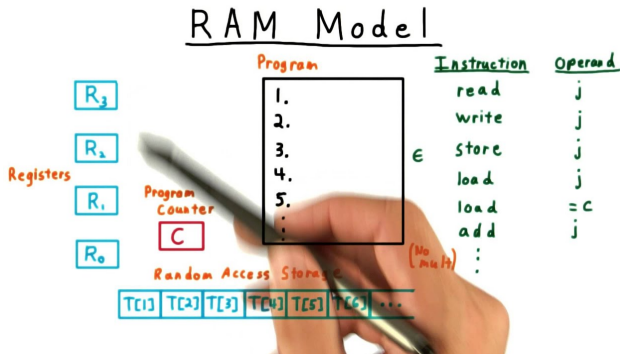
## Disadvantages:

- ▶ On different machines
- ▶ At different time
- ▶ On different inputs

No Standards.

We need a uniform **model of computation**.

## The RAM (Random Access Machine) Model of Computation



## The RAM (Random Access Machine) Model of Computation

- ▶ Each memory access takes constant time.
- ▶ Each “*primitive*” operation takes constant time.
- ▶ Compound operations should be decomposed.

Counting up the number of time units.

## Disadvantages:

- ▶ On different machines
- ▶ At different time
- ▶ On different inputs

Counting up the number of time units  
as a function of the input size  
in typical cases.

INSERTION-SORT( <i>A</i> )	<i>cost</i>	<i>times</i>
1 <b>for</b> <i>j</i> = 2 <b>to</b> <i>A.length</i>	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3       // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 <b>while</b> $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

$$\begin{aligned}
 T(n) = & c_1 n + c_2(n - 1) + c_4(n - 1) \\
 & + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)
 \end{aligned}$$

... as a function of the input size ...

INSERTION-SORT( $A$ )	<i>cost</i>	<i>times</i>
1 <b>for</b> $j = 2$ <b>to</b> $A.length$	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3       // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 <b>while</b> $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

$$\begin{aligned}
 T(n) = & c_1 n + c_2(n - 1) + c_4(n - 1) \\
 & + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)
 \end{aligned}$$

$T(n)$ : Depends on *which* input of size  $n$



... in typical cases.

Problem  $P$       Algorithm  $A$

Inputs:  $\mathcal{X}_n$  of size  $n$

$$W(n) = \max_{x \in \mathcal{X}_n} T(x)$$

$$B(n) = \min_{x \in \mathcal{X}_n} T(x)$$

$$A(n) = \boxed{\sum_{x \in \mathcal{X}_n} T(x) \cdot P(x)} = \mathbb{E}[T] = \boxed{\sum_{t \in T(\mathcal{X}_n)} t \cdot P(T = t)}$$

INSERTION-SORT( <i>A</i> )	<i>cost</i>	<i>times</i>
1 <b>for</b> <i>j</i> = 2 <b>to</b> <i>A.length</i>	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3       // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 <b>while</b> $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

$$B(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$$W(n) = \frac{c_5 + c_6 + c_7}{2}n^2 + (c_1 + c_2 + c_4 + c_8 - \frac{c_5 + c_6 + c_7}{2})n - (c_2 + c_4 + c_5 + c_8)$$

$$A(n) = 2.25n^2 + 7.75n - 3H_n - 6 \quad (H_n = \sum_{k=1}^n \frac{1}{k} \approx \ln n)$$

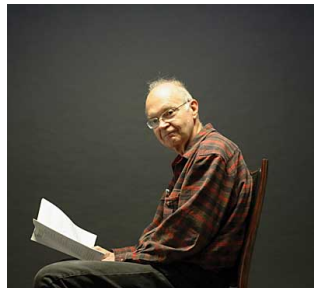
$Q$  : How fast is your algorithm?

listen carefully.

$$W(n) = \frac{c_5 + c_6 + c_7}{2}n^2 + (c_1 + c_2 + c_4 + c_8 - \frac{c_5 + c_6 + c_7}{2})n - (c_2 + c_4 + c_5 + c_8)$$

## BIG OMICRON AND BIG OMEGA AND BIG THETA

Donald E. Knuth  
Computer Science Department  
Stanford University  
Stanford, California 94305



## Reference:

*“Big Omicron and Big Omega and Big Theta”*, Donald E. Knuth, 1976.

## Asymptotics

$Q$  : How fast is your algorithm?

$$W(n) = \frac{c_5 + c_6 + c_7}{2}n^2 + (c_1 + c_2 + c_4 + c_8 - \frac{c_5 + c_6 + c_7}{2})n - (c_2 + c_4 + c_5 + c_8)$$

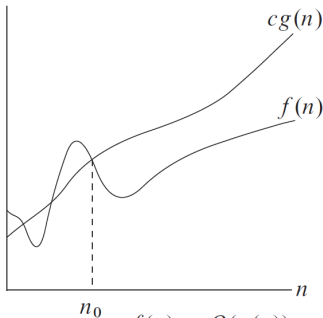
$$W(n) = O(n^2)$$

“Order at most  $n^2$ ”

“ $W(n)$  is a function whose **order of magnitude** is **upper-bounded** by a **constant times  $n^2$** , for all large  $n$ .”

$$f(n) = O(g(n))$$

“ $f(n)$  is a function whose **order of magnitude** is **upper-bounded** by a **constant times**  $g(n)$ , for all large  $n$ .”



$$O(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \right\}$$

$$\boxed{f(n) = O(g(n))}$$

$$O(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \right\}$$

$$\{ \}$$

It is a tradition to write  $f(n) = O(g(n))$  instead of  $f(n) \in O(g(n))$ .

$$42n^2 + 2020n = O(n^2) = O(n^3)$$

$$42n^2 + 2020n \in O(n^2) \subseteq O(n^3)$$



$$O(f(n)) + O(g(n)) \triangleq \{h + l \mid h \in O(f(n)), l \in O(g(n))\}$$

$$O(f(n))O(g(n)) \triangleq \{hl \mid h \in O(f(n)), l \in O(g(n))\}$$

$$O(f(n)) - O(g(n)) \triangleq$$

$$O(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \right\}$$

$$42n = O(0.50n^2) \quad 42n^2 = O(0.50n^2)$$

*Q* : What does  $O(1)$  mean?

*A* : It means constants.

$$\Omega(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq cg(n) \leq f(n) \right\}$$

$$0.50n^2 = \Omega(42n) \quad 0.50n^2 = \Omega(42n^2)$$

$$\Theta(g(n)) = \left\{ f(n) \mid \exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0, \forall n \geq n_0 : \right. \\ \left. 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \right\}$$

$$0.50n^2 = \Theta(42n^2)$$

$$o(g(n)) = \left\{ f(n) \mid \forall c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) < cg(n) \right\}$$

$$42n = o(0.50n^2)$$

$$\omega(g(n)) = \left\{ f(n) \mid \forall c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq cg(n) < f(n) \right\}$$

$$0.50n^2 = \omega(42n)$$

$O \quad \Omega \quad \Theta$

$o \quad \omega \quad \theta$

$$f(n) \sim g(n) \iff \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$$

$$42n^2 + 2020n \sim 42n^2 + 2019n$$

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$$

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \iff g(n) = \omega(f(n))$$

$$O(f(n)) + O(g(n)) = O(f(n) + g(n))$$

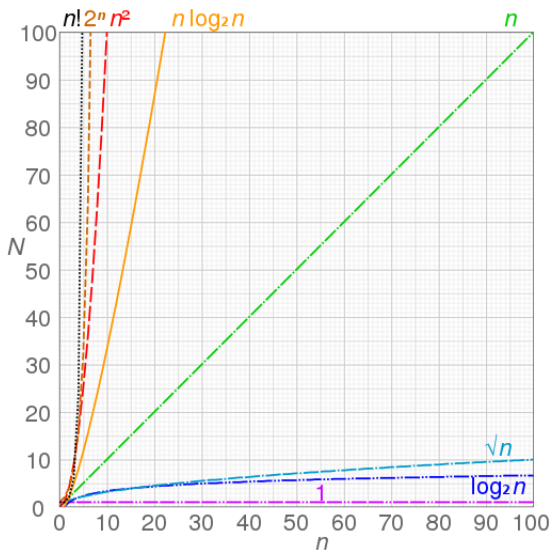
$$O(f(n))O(g(n)) = O(f(n)g(n))$$

*Q* : How to compare functions in terms of  $O/\Omega/\Theta$ ?

$$\begin{aligned} O(1) &= O(\log \log n) = O(\log n) = O((\log n)^c) \\ &= O(n^\epsilon) = O(n^c) \\ &= O(n^c \log n) = O(n^{\log n}) = O(c^n) = O(n^n) \end{aligned}$$

$(0 < \epsilon < 1 < c)$





Stirling Formula (by *James Stirling*):

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$



$$\log(n!) = \Theta(n \log n)$$

$$H_n = \sum_{k=1}^n \frac{1}{k} = \Theta(\log n)$$



$$A[0, \dots, n-1] \quad 1 \leq l \leq n$$

ROTATE( $A, n, l$ ) : Rotate  $A$  left by  $l$  places

0	1	2	3	4
---	---	---	---	---

3	4	0	1	2
---	---	---	---	---

Critical Operation: copy

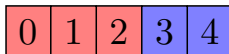
---

```

1: procedure ROTATE( $A, n, l$ )
2:   for  $i = 1 \dots l$  do
3:     ROTATE-BY-ONE( $A, n$ )

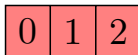
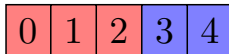
```

---



Algorithm	Time	Space
rotate-one-by-one	$nl = O(n^2)$	$O(1)$

- 
- 1: **procedure** ROTATE( $A, n, l$ )
  - 2:     copy  $A[0 \dots l - 1]$  into  $v$
  - 3:     move  $A[l \dots n - 1]$  left  $l$  places
  - 4:     copy  $v$  to  $A[l \dots n - 1]$
- 



Algorithm	Time	Space
rotate-copy	$O(n)$	$l = O(n)$

$$n = 5, \quad l = 3$$

0	1	2	3	4
---	---	---	---	---

3	4	0	1	2
---	---	---	---	---

(0, 2, 4, 1, 3)

$$n = 9, \quad l = 6$$

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

6	7	8	0	1	2	3	4	5
---	---	---	---	---	---	---	---	---

(0, 3, 6)      (1, 4, 7)      (2, 5, 8)

## Correctness Proof?

Permutations as **Product** of  
**Disjoint Cycles**



Algorithm	Time	Space
rotate-cyclic	$O(n)$	$O(1)$



$$B \cdot A = (A^R \cdot B^R)^R$$

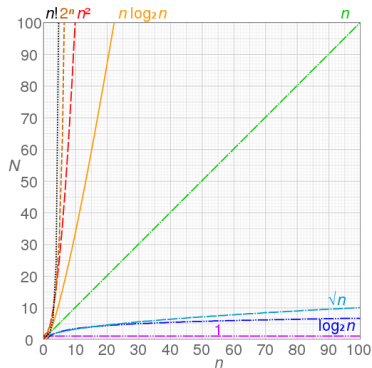
0	1	2	3	4
---	---	---	---	---

2	1	0	4	3
---	---	---	---	---

3	4	0	1	2
---	---	---	---	---

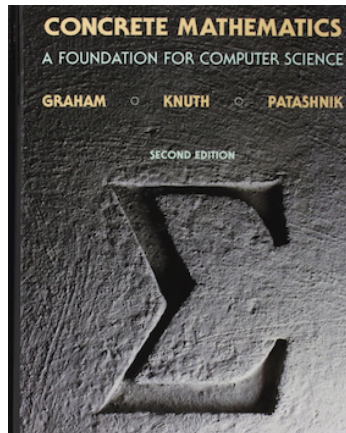
Algorithm	Time	Space
rotate-reverse	$O(n)$	$O(1)$

Algorithm	Time	Space
rotate-one-by-one	$O(n^2)$	$O(1)$
rotate-copy	$O(n)$	$O(n)$
rotate-cyclic	$O(n)$	$O(1)$
rotate-reverse	$O(n)$	$O(1)$



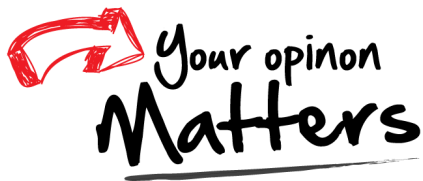
$O \quad \Omega \quad \Theta$

$o \quad \omega$



## Chapter 9: Asymptotics

Thank  
You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn