Algorithm Design and Analysis

David N. JANSEN, Bohua ZHAN
名 姓

算法设计与分析

詹博华，杨大卫

This week's content

- Today Wednesday:
  - Introduction of teachers
  - Grading rules
  - Why study algorithms?
    Chapter 3: Growth of functions
  - Exercises
- Tomorrow Thursday:
  - Exercise solutions
  - Chapter 4: Divide and Conquer
    Introduction of the remaining content

这周的内容

- 今天周三：
  - 老师介绍
  - 课程评分规则
  - 为什么学习算法?
    第三章：函数的增长
  - 函数的增长的练习
- 明天周四：
  - 练习题解答
  - 第四章：分治策略
    以后内容的介绍

David N. Jansen

名 姓

杨大卫

- Swiss and Dutch
- Mathematics Master in Switzerland
- Computer Science Ph. D. in the Netherlands
- worked later in Germany, Netherlands, China (ISCAS)
- interested in formal verification. also interested in philosophy of s

prove correctness of algorithms with mathematical proof

Can science answer all questions?

"Will you marry me?"

- 瑞士人和同时荷兰人
- 在瑞士获得数学硕士学位
- 在荷兰获得计算机科学博士学位
- 以后工作在德国，荷兰，中国（中科院软件研究所）
- 研究方向：形式化验证，也对科学的哲学感兴趣

用数学证明算法的正确性

科学有能力解决所有的问题吗?

你愿意嫁给我吗?

詹博华的介绍

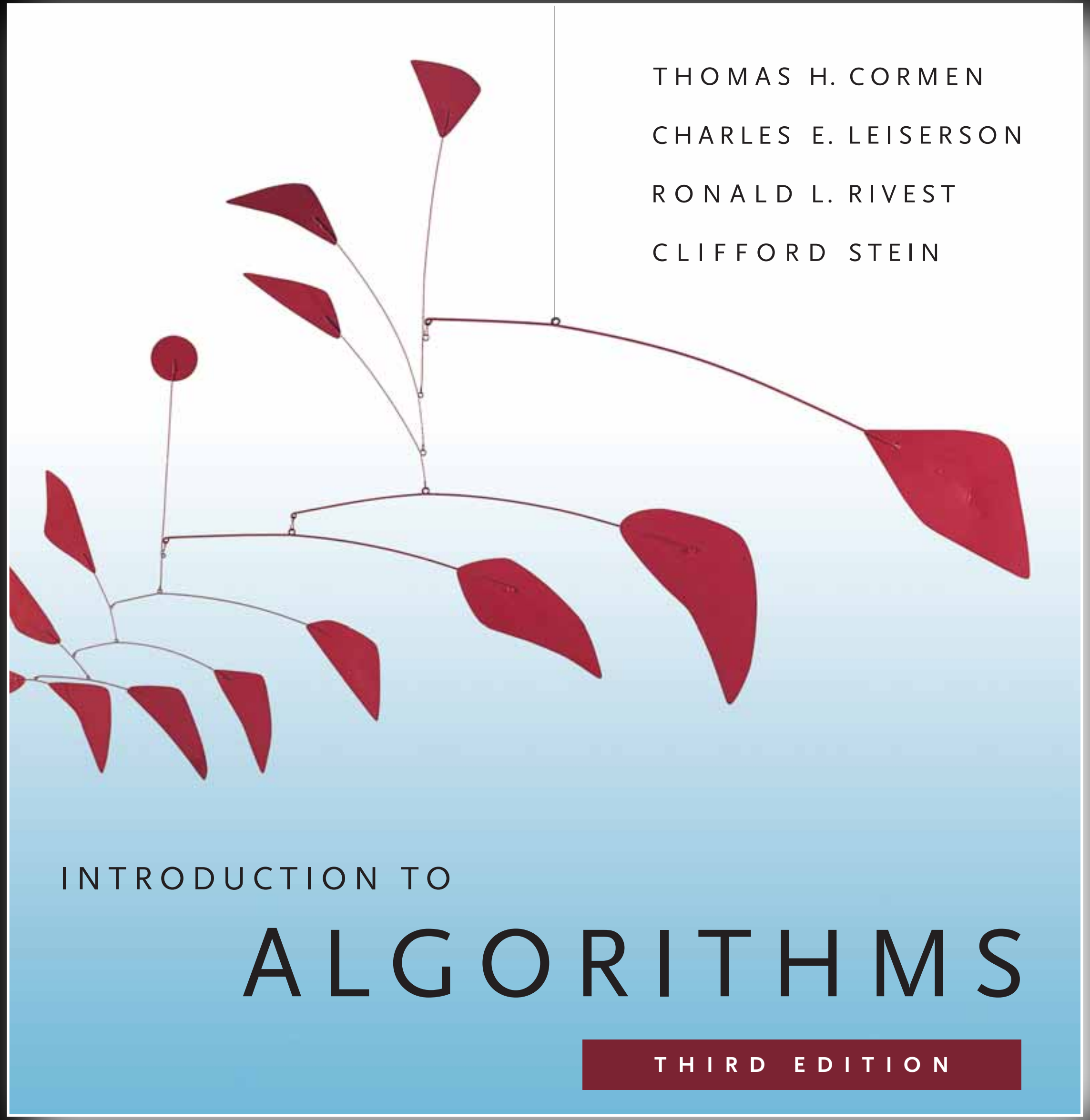
- 在美国获得数学学士和博士学位
- 2018年回国，加入中科院软件所，现任副研究员
- 研究方向：形式化方法、交互式定理证明、嵌入式系统的建模与验证

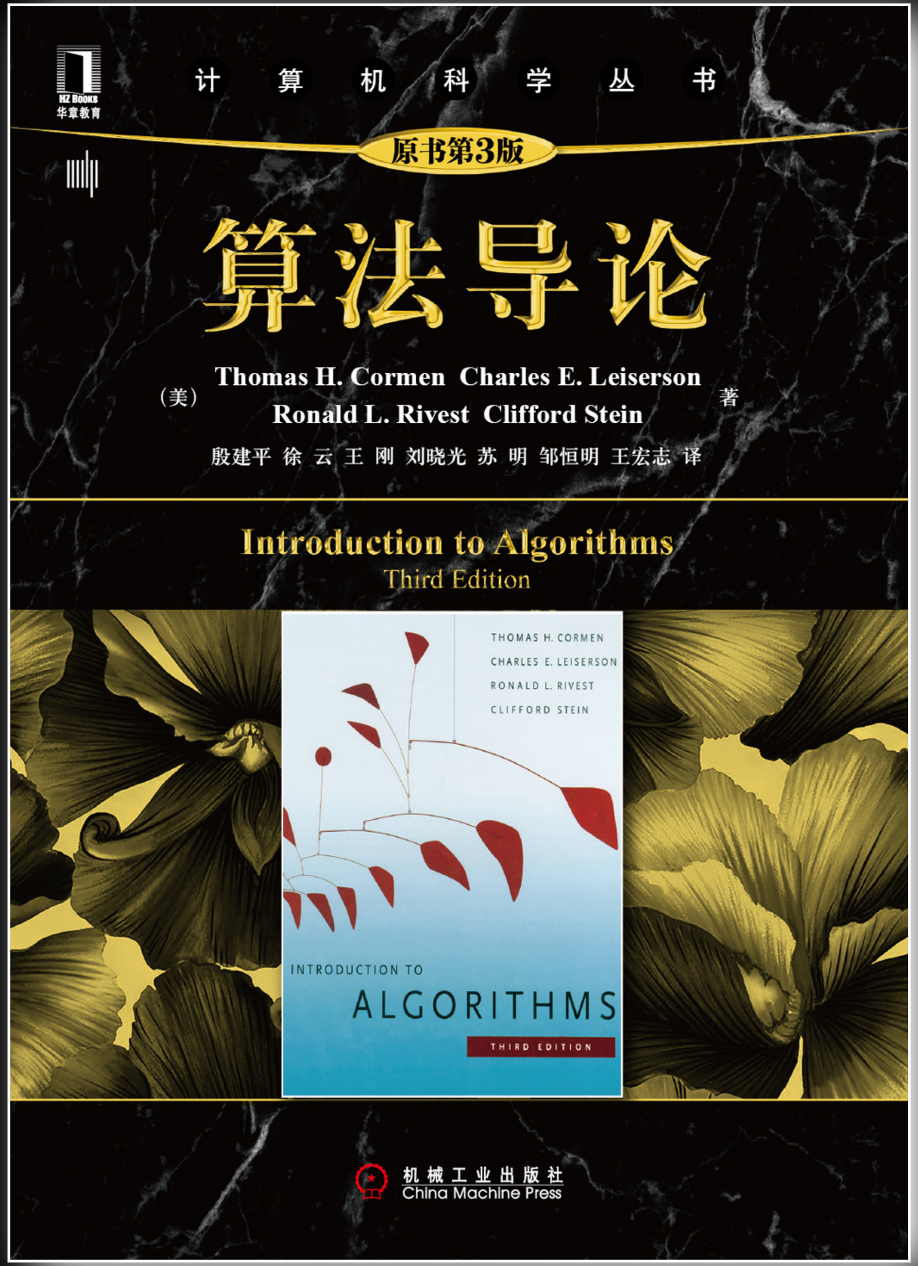
课程评分规则

- 40%大作业、60%期末考试
- 定期布置（不计分的）作业，用于熟悉考试内容和题目形式

Book

- We will follow this book:
  Cormen / Leiserson / Rivest / Stein: Introduction to Algorithms.
  3rd edition. MIT Press 2009.





Open office

- I want to offer a time to ask questions every week.
  Suggested time: Thursday after the lecture.

开放时间

- 每周有时间可以问我问题。
  建议：周四下课后

This week's content

- Today Wednesday:
  - Introduction of teachers
  - Grading rules
  - Why study algorithms?
    Chapter 3: Growth of functions
  - Exercises
- Tomorrow Thursday:
  - Exercise solutions
  - Chapter 4: Divide and Conquer
    Introduction of the remaining content

这周的内容

- 今天周三：
  - 老师介绍
  - 课程评分规则
  - 为什么学习算法?
    第三章：函数的增长
  - 函数的增长的练习
- 明天周四：
  - 练习题解答
  - 第四章：分治策略
    以后内容的介绍

Why study Algorithms?

为什么学习算法?

What is an algorithm?

算法是什么？

- a precise sequence of instructions that transform an input into an output
- satisfies some specification / goal
- The word “algorithm” is derived from the Persian mathematician Al Khwārizmī
- can be communicated in natural language,
but often written as pseudocode

- 把输入转换成输出的计算步骤的序列
- 满足需求/目标
- “algorithm”词从波斯的数学家Al Khwārizmī
- 可以使用自然语言沟通算法，
但通常也使用伪代码

Example: maximum 例子：最大元素

- specification: find the largest number in a sequence of integers
- 需求:
  找到一个整数的序列的最大的元素

MAXIMUM($\overbrace{a_1, a_2, \ldots, a_n}^{\text{input 输入}}$: integers)

sequence of instructions
计算步骤的序列

max = a_1
for i = 2 **to** n
    if $max < a_i$
        max = a_i
return $\underbrace{max}_{\text{output 输出}}$

Properties of Algorithms

- definite: every step of the algorithm is defined precisely
- correct: produces the correct output for every input
- general: applicable to all problems of a certain class
- finite: for each input, it terminates in a finite number of steps

- 确定的
- 正确的
- 一般的
- 有限的

Why study algorithms?

- Need to solve problems correctly:
  good algorithm has proof of correctness

- Need to solve problems efficiently:
  good algorithm is fast /
  uses little memory

A little history

- algorithms existed before computers
  - Babylonian algorithm:
    2500 BC, division
  - Euclid’s algorithm:
    300 BC, greatest common divisor
- Many new algorithms with computers
  - bubble sort –> quick sort

- 算法计算机前已有
  - 巴比伦：2500 BC，出发
  - 欧几里得：300BC，最大公因子
  - 冒泡排序 –> 快速排序

Growth of Functions (Big-O Notation)

Chapter 3

函数的增长（大O记号）

第三章

Insertion Sort 插入排序

- animation with numbered cards

Insertion Sort

sequence $A[1 \ldots j{-}1]$ is sorted

select an element to be sorted

Insert the element into the sorted sequence $A[1 \ldots j{-}1]$

sequence $A[1 \ldots j]$ is sorted

INSERTION-SORT(array A)

for $j := 2$ **to** $A.length$
    $key = A[j]$
    $i = j - 1$
    while $i > 0$ **and** $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i + 1] = key$

Insertion Sort: Loop Invariant

- Correctness proof of an algorithm with loops:
  uses a loop invariant 环不变式.
  - Initialisation: When the loop starts, the loop invariant holds.
  - Maintenance: If the loop invariant holds at the beginning of the loop body, then it holds at the end of the loop body.
  - Termination: When the loop terminates, the property required after the loop holds.

Insertion Sort: Loop Invariant

- Loop invariant for insertion sort: At the beginning of an iteration, the sequence $A[1 \ldots j{-}1]$ is sorted.

  - Initialisation: When the loop starts ($j = 2$), A[1 ... 1] is sorted.

  - Maintenance: If at the beginning of the loop body $A[1 \ldots j{-}1]$ is sorted, then $A[1 \ldots j]$ is sorted at the end of the loop body.

  - Termination: When the loop terminates ($j = n$), $\underbrace{\mathrm{A}[1 \ldots n] \text{ is sorted}}_{\text{property at the end of the loop body}}$.

Insertion Sort: Running Time 运行时间

- We want to know how long INSERTION-SORT takes, depending on *A.length*.
- Problem: operations on different computers take different time.
- Simplification: only count operations (we will see later that it's ok.)

Insertion Sort

INSERTION-SORT(array A)
for j := 2 **to** $A.length$
$key = A[j]$
$i = j - 1$
while $i > 0$ **and** $A[i] > key$
$A[i+1] = A[i]$
$i = i - 1$
$A[i + 1] = key$

This line is executed ≤ ... times
$n = A.length$
$n-1$
$n-1$
$2 + 3 + \ldots + n$
$1 + 2 + \ldots + (n-1)$
$1 + 2 + \ldots + (n-1)$
$n-1$

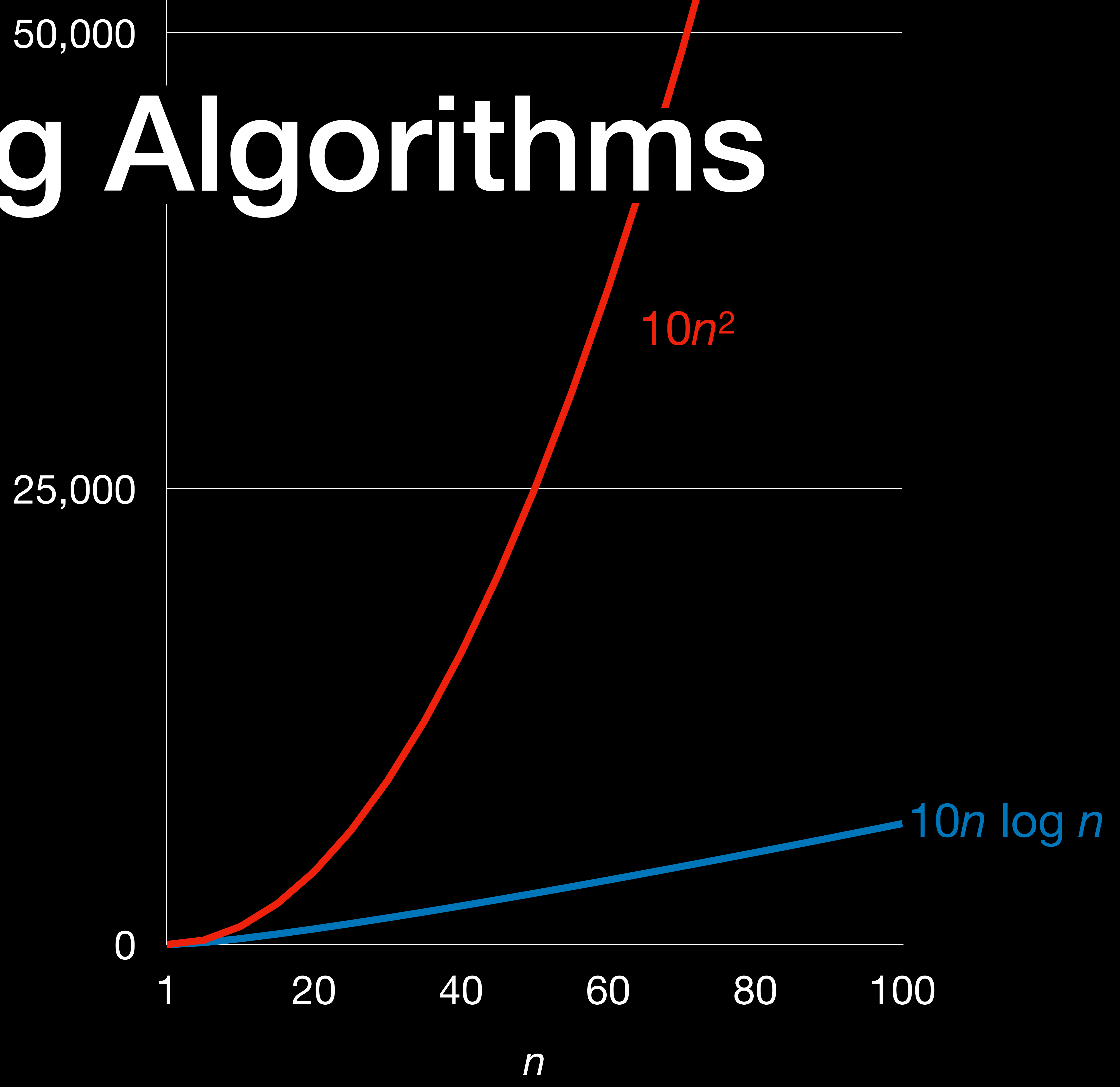
Running time is important if $n = A.length$ is large, but then $n \ll n^2$

Total ≤ $1\frac{1}{2}n^2 + 3\frac{1}{2}n - 4$ times

See next slide

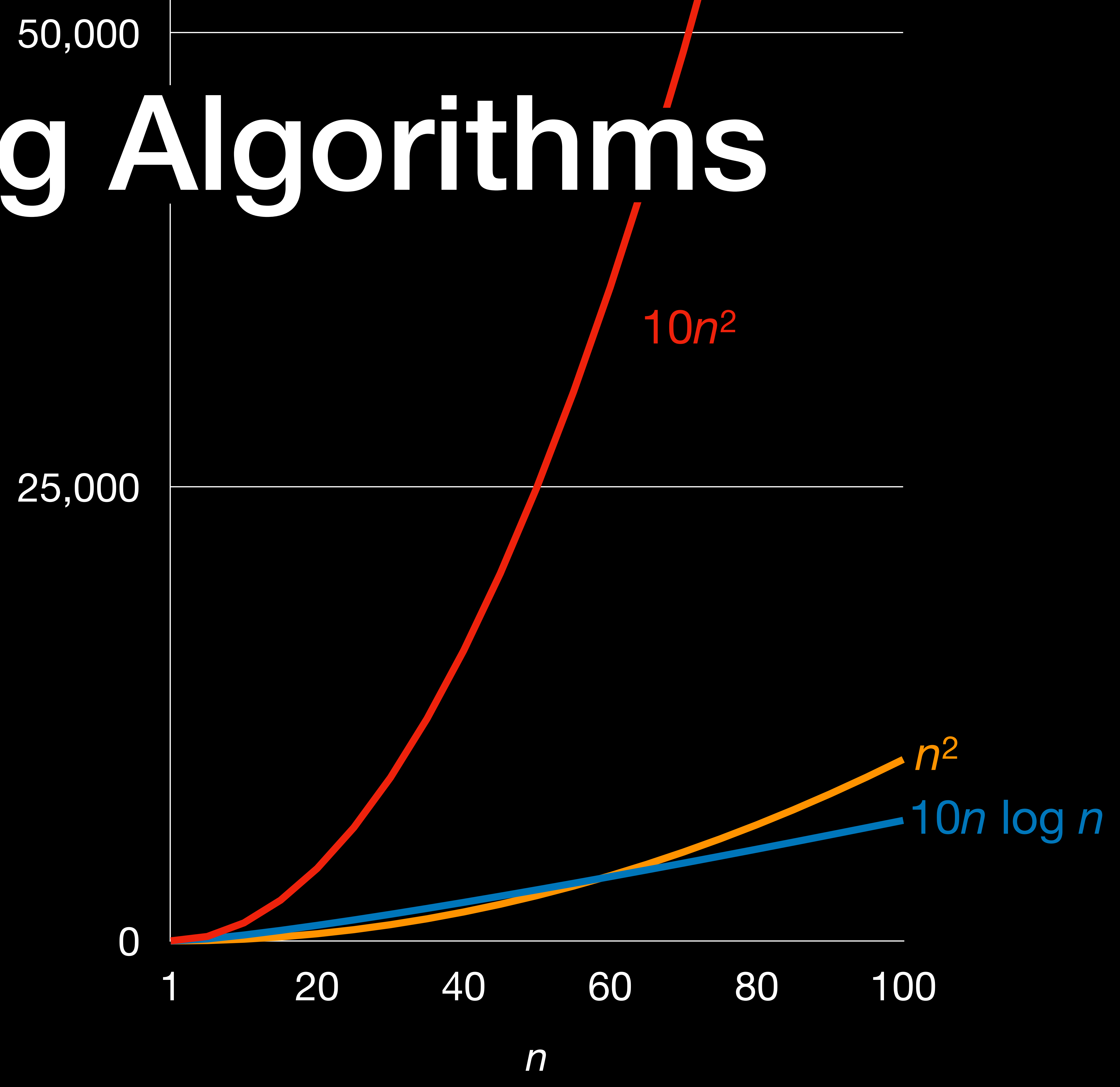
Comparing Algorithms

- Example:
  - algorithm A executes $\approx 10n^2$ instructions.
  - algorithm B executes $\approx 10n \log n$ instructions.
- Now let's make algorithm A 10× faster.



Comparing Algorithms

- Example:
  - algorithm A executes $\approx 10n^2$ instructions.
  - algorithm B executes $\approx 10n \log n$ instructions.
- Now let's make algorithm A 10× faster.
  ➥ For $n \geq 59$, algorithm B is still faster than the improved algorithm A.



Comparing Algorithms

- For large inputs, growth rate of the function is most important.
  (For small inputs, most algorithms are fast.)

- Big-O notation: brief notation to indicate the growth rate of a function.

Big-O Notation

- Notation to compare asymptotic growth rate of functions
  (Which function $\mathbb{N} \longrightarrow \mathbb{R}^{+}$ is larger, for large parameters?)

- used to describe order of growth of algorithms
  (Which algorithm is faster for large input sizes?)

Big-*O* Notation

- $O(g(n))$ is a set of functions
  all functions that do not grow faster than $g(n)$
  $\{ f(n) \mid \exists c, n_0 : \forall n \geq n_0 : 0 \leq f(n) \leq c g(n) \}$

- $O(\,)$ gives an upper bound on a function.

Big-O Notation

- $n = \mathrm{O}(n^2)$.
- $2n = \mathrm{O}(n)$.
- $n = \mathrm{O}(2n - 4)$.
- The worst-case running time of insertion sort: $1\frac{1}{2}n^2 + 3\frac{1}{2}n - 4$.
  This time is in $O(n^2)$.
  It is also in $O(n^3)$.
  It is not in $O(n \log n)$.

Big-Ω Notation

- $\Omega(g(n))$ is a set of functions
  all functions that do not grow slower than $g(n)$
  $\{ f(n) \mid \exists c, n_0 \colon \forall n \geq n_0 \colon 0 \leq cg(n) \leq f(n) \}$

- $\Omega(\ )$ gives a lower bound on a function.

Big-Ω Notation

- $n^2 = \Omega(n)$.
- $n = \Omega(2n + 5)$.
- The best-case running time of insertion sort is $5n - 4$.
  It is in $\Omega(n)$.
  It is not in $\Omega(n^2)$.

$O, \Omega, \Theta, o, \omega$

Nota-tion	**English Pronunciation**	**Meaning**	**Formula**
$O(g(n))$	(big) Oh of $g(n)$	functions that don't grow faster than g	$\{ f(n) \mid \exists c,n_0 : \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \}$
$\Omega(g(n))$	(big) Omega of $g(n)$	functions that grow at least as fast as g	$\{ f(n) \mid \exists c,n_0 : \forall n \geq n_0 : 0 \leq cg(n) \leq f(n) \}$
$\Theta(g(n))$	Theta of $g(n)$	functions that grow equally fast as g	$O(g(n)) \cap \Omega(g(n))$
$o(g(n))$	small oh of $g(n)$	functions that grow slower than g	$\{f(n) \mid \forall c>0 : \exists n_0 : \forall n \geq n_0 : 0 \leq f(n) < cg(n)\}$
$\omega(g(n))$	small omega of $g(n)$	functions that grow faster than g	$\{f(n) \mid \forall c>0 : \exists n_0 : \forall n \geq n_0 : 0 \leq cg(n) < f(n)\}$

Further Examples

- $n = \Theta(2n)$.
- $4n^3 + 2n^2 + 5 = \Theta(n^3)$.
- $n = o(n^2)$. $n \notin o(2n + 5)$.
- $n = \omega(\log n)$.

Facts about Big-O Notation

- Transitivity: $f(n) = O(g(n))$ and $g(n) = O(h(n)) \quad \Rightarrow \quad f(n) = O(h(n))$

  $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n)) \quad \Rightarrow \quad f(n) = \Omega(h(n))$

  also for Θ, o and ω

- Symmetry of Θ: $f(n) = \Theta(g(n)) \quad \Leftrightarrow \quad g(n) = \Theta(f(n))$

- Duality: $f(n) = O(g(n)) \quad \Leftrightarrow \quad g(n) = \Omega(f(n))$

- Be careful: $f(n) = O(g(n))$ and $f(n) \notin \Theta(g(n)) \quad \not\Rightarrow \quad f(n) = o(g(n))$

  Example: $f(n) = n$ if n is odd 奇数 $\qquad g(n) = n^2$

  $\qquad = n^2$ if n is even 偶数

Big-O Notation for Functions

- When approximating a function, also use these notations.
- Example: number of comparisons in a certain sort algorithm
  $= 2n \log n + O(n)$.
  Meaning: $= 2n \log n + f(n)$, for some function $f(n) = O(n)$.
- Other uses: memory consumption of an algorithm
  general approximation of functions

Exercises 练习

- 1.2-2
  Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

- 假设我们正比较插入排序与合并排序在相同机器上的实现。对规模为n的输入，插入排序运行$8n^2$步，而合并排序运行$64n \lg n$步。问对哪些n值，插入排序优于合并排序?

Exercises 练习

Problem 1-1: Comparison of running times
For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds (μsec).

思考题 1-1: 运行时间的比较
假设求解问题的算法需要f(n)微妙，对下表中的每个函数$f(n)$和时间t，确定可以在时间t内求解的问题的最大规模n。

t = 1 sec, 1 min, 1 hour, 1 day, 1month/月, 1 year, 100 years

$f(n) = \lg n, \sqrt{n}, n, n \lg n, n^2, n^3, 2^n, n!$ (lg $n = \log_2 n$)

Exercises 练习

	1秒	1分	1小时	1天	1月	1年	1世纪
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Exercises 练习

- 2.1-3
  Consider the searching problem:
  Input: A sequence of n numbers $A = (a_1, a_2, ..., a_n)$ and a value v.
  Output: An index i such that $v = A[i]$, or the special value NIL if v does not appear in A.
  Write pseudocode for linear search, which scans through the sequence, looking for v. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

- 考虑以下查找问题：
  输入：n个数的一个序列$A = (a_1, a_2, ..., a_n)$和一个值v。
  输出：下标i使得$v = A[i]$或者当v不在A中出现时，特殊值NIL。
  写出线性查找的伪代码，它扫描整个序列来查找v。使用一个循环不变式来证明你的算法的正确性。确保你的循环不变式满足三条必要的性质。

Exercises 练习

- 3-2.3
  Prove / 证明
  - $n! = \omega(2^n)$
  - $n! = o(n^n)$
  - $\log(n!) = \Theta(n \log n)$
- You may use Stirling's approximation / 可以使用斯特林近似公式:

$$n! = \sqrt{2\pi n}\,(n/e)^n\,(1 + \Theta(1/n))$$