算法设计与分析

Lecture 2: Growth of Functions

曹刘娟

厦门大学信息学院计算机系

caoliujuan@xmu.edu.cn

Review（复习）

什么是算法?

- An algorithm is a sequence of computational steps that transform the input into the output to solve a given problem.
- Example: The sorting （排序问题） problem.
  - Input: A sequence of n numbers $A = \langle a_1, a_2, \dots, a_n \rangle$.
  - Output: A permutation (reordering) $A' = \langle a_1', a_2', \dots, a_n' \rangle$ of the input sequence such that $a_1' \le a_2' \le \dots \le a_n'$.

定义

- Problem (问题): A question to which we seek an answer.
  - Find the non-decreasing order of a sequence of n numbers A.
- Algorithm (算法): A step-by-step procedure applying a technique for solving the problem.
  - Insertion sort
  - Quicksort
  - Mergesort
  - …
- Parameters (参数): Variables that are not assigned specific values in the statement of the problem.
  - A.
- Instance (实例): Specific assignment of values to the parameters.
  - $A = \langle 3,6,1,7,2 \rangle$.
- Solution (答案): The answer to the problem in that instance.
  - $A' = \langle 1,2,3,6,7 \rangle$.

算法的性质

- Finite (有穷性): An algorithm consists of finite number of operations.
- Feasible (可行性): Every operation is executable.
- Deterministic (确定性): Every operation must not be ambiguous (generate random results).
- Input (输入): 0 or more, describe the initial state.
- Output (输出): 1 or more, describe the result process from the input.

PSEUDOCODE （伪代码）

- Pseudocode (伪代码) is a plain language description of the steps in an algorithm, irrelevant to specific programming language.
- We have the following conventions:
  - "←" represents variable assignment (变量赋值).
  - Indentation (缩进) indicates block structure. "{}" 不是必须
  - While, for, repeat, if, then, and else have the same interpretation as in most programming language.
  - // indicates comments.
  - Composite data type (复合类型) are typically organized into objects, which are comprised of attributes or fields, e.g. T.Lchild.
  - Array elements (数组) are accessed by specifying the array name followed by the index in square brackets, e.g. A[i].
    - Most of the time, we start the index i from 1, instead of 0.

LOOP INVARIANTS （循环不变量）

- At the start of each iteration of the for loop, the subarray $A[1\ldots j-1]$ consists of the elements originally in $A[1\ldots j-1]$ but in sorted order.
- We state these properties of $A[1\ldots j-1]$ formally as a loop invariants (循环不变量).
- 使用loop invariants 来证明算法的正确性 why an algorithm is correct.
- 很多时候采用数学归纳法

插入排序算法的正确性

The loop invariants are:

$A[1...j-1]$ is sorted before each iteration.

The proof is similar to mathematical induction (数学归纳法):

- Initiation 初始化 (归纳基础): It is true prior to the first iteration of the loop.
- Maintenance (归纳步骤): If it is true before an iteration of the loop, it remains true before the next iteration.
- Termination(终止): When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

EFFICIENCY OF ALGORITHM

- Is correctness of an algorithm enough? How good is an algorithm?
- Time complexity (时间复杂度)
  - Indicates how fast an algorithm runs.
  - How many CPU cycles needed.
- Space complexity (空间复杂度)
  - Amount of memory units required by an algorithm.
- Does there exist a better algorithm?
- How to compare algorithms?

TIME COMPLEXITY (时间复杂度)

- Running time of an algorithm on a particular input generally be the number of primitive operations or "steps" executed.
- Define the notion of step so that it is as machine independent as possible. （比较与具体的计算机配置无关）
  - 一个算法在一台586 CPU 上的运行时间和在i9 CPU上的运行时间比较是没有意义的
- The comparison should be instance independent. （比较与具体的数据输入无关）
  - 我们不是比较算法在某个特定输入数据的情况，而是比较一个问题的通用情况

- The worst-case time complexity (最坏情形时间复杂度) of an algorithm is the function defined by the maximum number of steps taken on any instance of size n.
- The best-case time complexity (最好情形时间复杂度) of an algorithm is the function defined by the minimum number of steps taken on any instance of size n.
- The average-case time complexity (平均情形时间复杂度) of an algorithm is the function defined by an average number of steps taken on any instance of size n.
- Which of these is the best to use?

WORST CASE OF INSERTION SORT

- The worst case occurs if the array is in reverse sorted order. In this case:
  - The key is always moving to the front in the while loop and thus $t_j = j$.
- Then, $T(n)$ becomes:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j$$
$$+ c_5 \sum_{j=2}^{n} (j-1) + c_6 \sum_{j=2}^{n} (j-1) + c_7(n-1) + c_8$$
$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) n$$
$$-(c_2 + c_3 + c_4 + c_7 - c_8)$$

- This running time is a quadratic function of n.

本节课内容-Growth of Functions

引进渐进符号简化时间复杂度的计算和分析

- Intuitively, just look at the dominant term.

$$T(n) = \cancel{0.1}n^3 + \cancel{10n^2 + 5n + 25}$$

  - Drop lower-order terms $10n^2 + 5\text{n} + 25$.
  - Ignore constant 0.1.
- But we can't say that $T(n)$ equals to n^3.
  - It grows like n^3. But it doesn't equal to n^3.
- We define asymptotic notations (渐进符号) like $T(n) = \Theta(n^3)$ to describe the asymptotic running time of an algorithm.
  - "Asymptotic" here means "as something tends to infinity", as we want to compare algorithms for very large n.

为描述最坏情形时间复杂度以及上界的概念，引进BIG O

Definition 2.2

For a given complexity function $g(n)$, $O\left(g(n)\right)$ is the set of complexity functions $f(n)$ for which there exists some positive real constant c and some nonnegative integer n_0 such that for all $n \geq n_0$,

$$0 \leq f(n) \leq cg(n).$$

- *上面的定义要注意，是存在正整数c和n_0使得对所有的n ≥n0都成立*
- $O\left(g(n)\right)$ is a set of functions in terms of $g(n)$ that satisfy the definition.
- If $f(n) = O\left(g(n)\right)$, it represents that $f(n)$ is an element in $O\left(g(n)\right)$. We say that $f(n)$ is "big O (大O)" of $g(n)$.
  - Strictly, we should use "∈" instead of "=". However, it is conventional to use "=" for asymptotic notations.

- 我们关心的是当算法规模n非常大的情况
- No matter how large $f(n)$ is, it will eventually be smaller than $cg(n)$ for some c and some n_0.
- Big O notation describes an upper bound （上界）. We use it to bound the worst-case running time（最差运行时间）of an algorithm on arbitrary inputs.

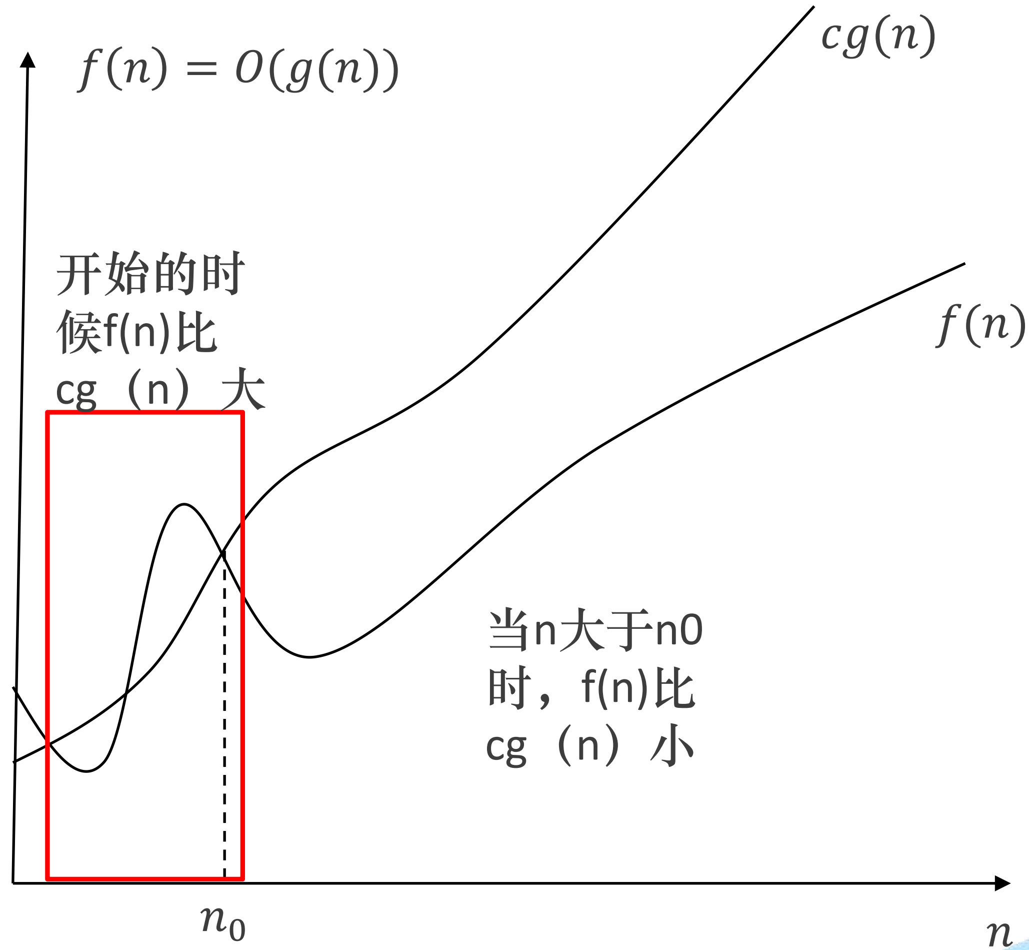


Image source: 图2.2, 张德富, 算法设计与分析, 国防工业出版社, 2009.

DISPLAY OF GROWTH OF FUNCTIONS

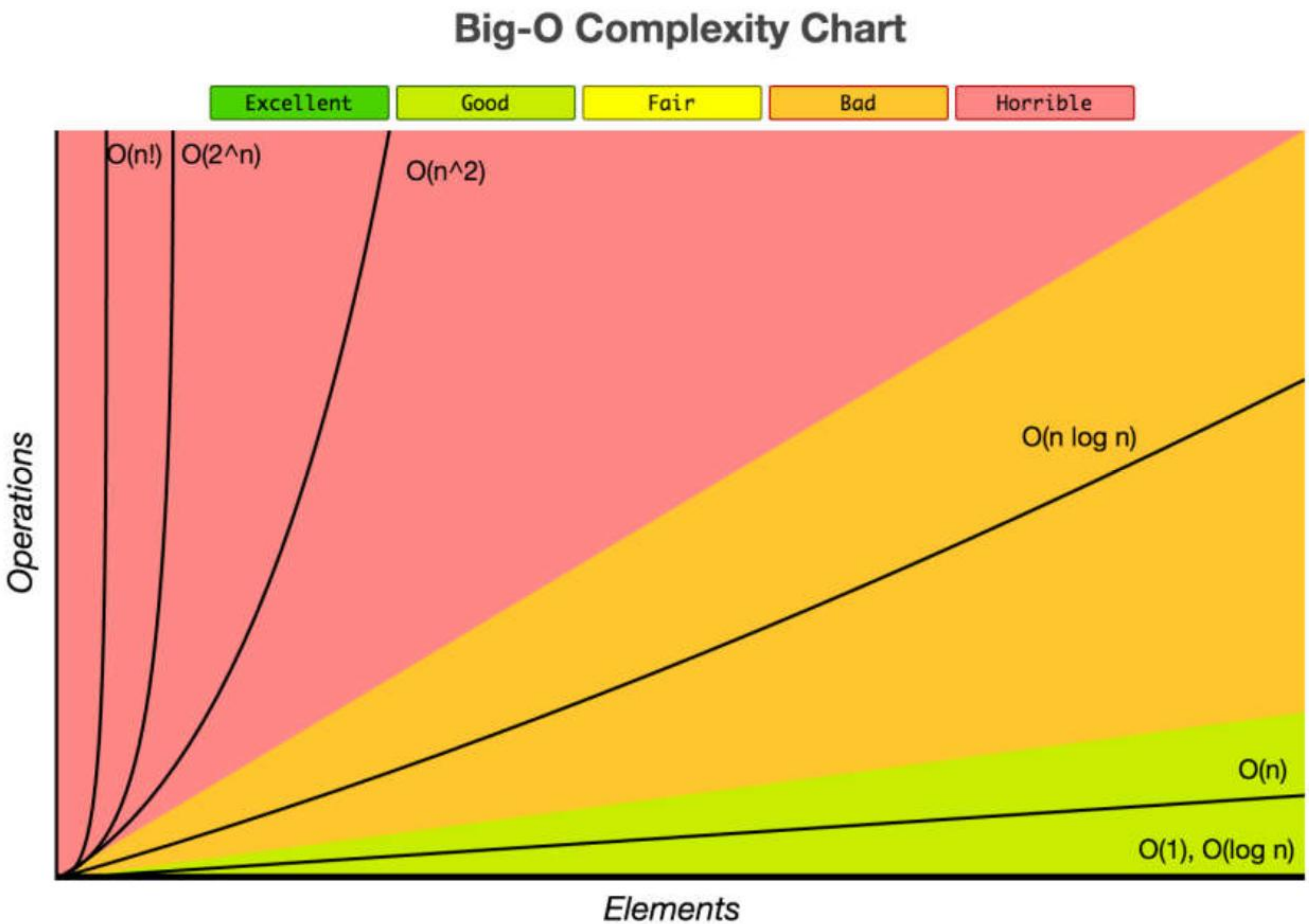


Image source: http://bigocheatsheet.com/img/big-o-complexity-chart.png

Example 1

We show that $n^2 + 10n = O(n^2)$. Because, for $n \geq 1$,

$$n^2 + 10n \leq n^2 + 10n^2 = 11n^2,$$

we can take $c = 11$ and $n_0 = 1$ to obtain our result.

- To show a function is in big *O* of another function, the key is to find a specific value of c and n_0 that make the inequality hold.
- More examples of functions in $O(n^2)$:
  - n^2, $n^2 + n$, $n^2 + 1000n$, $1000n^2 + 1000n$, n, $n/1000$, $n^{1.99999}$, $n^2/\lg\lg\lg n$.

Use the definition of Big O notation to show:

Is $2^{2n} = O(2^n)$?

Proof:

We prove it by contradiction （反证法）. Assume there exist constants $c > 0$ and $n_0 \geq 0$, such that

$$2^{2n} \leq c2^n,$$

for all $n \geq n_0$. Then

$$2^{2n} = 2^n 2^n \leq c2^n,$$
$$2^n \leq c.$$

But we can't find any constant c is greater than 2^n for all $n \geq n_0$. So the assumption leads to a contradiction. Then we can certify that $2^{2n} \neq O(2^n)$.

How about $2^{n+1} = O(2^n)$?

BIG Ω NOTATION （发音 OMEGA）

为描述最好情形时间复杂度以及下界的概念，引进BIG **Ω**

Definition 2.3

For a given complexity function $g(n)$, $\Omega\big(g(n)\big)$ is the set of complexity functions $f(n)$ for which there exists some positive real constant c and some nonnegative integer n_0 such that for all $n \geq n_0$,

$$0 \leq cg(n) \leq f(n).$$

- $\Omega\big(g(n)\big)$ is the opposite of $O\big(g(n)\big)$.
- If $f(n) = \Omega\big(g(n)\big)$, it represents that $f(n)$ is an element in $\Omega\big(g(n)\big)$. We say that $f(n)$ is "big Ω (大Ω)" of $g(n)$.

BIG Ω NOTATION

- No matter how small $f(n)$ is, it will eventually be larger than $cg(n)$ for some c and some n_0.
- Big Ω notation describes an lower bound（下界）. We use it to bound the best-case running time （最佳情况） of an algorithm on arbitrary inputs.

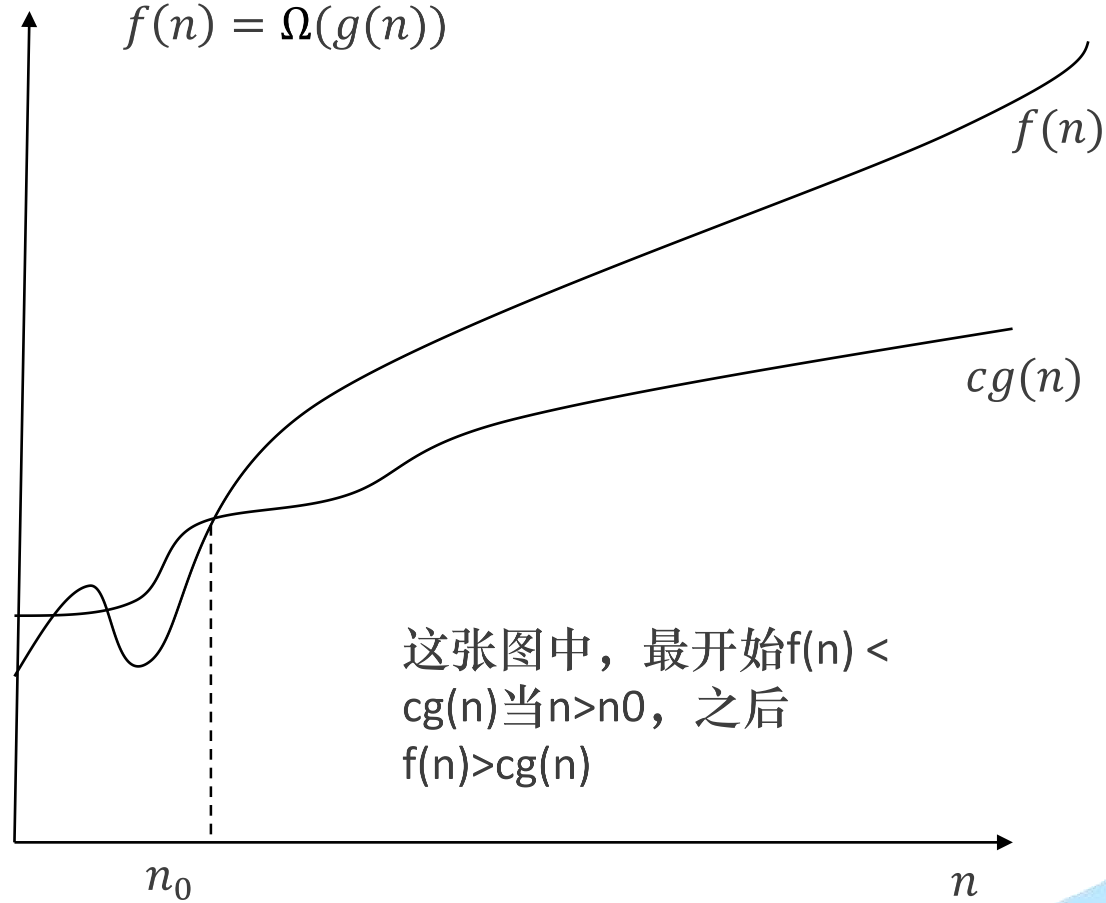


Image source: 图2.3, 张德富, 算法设计与分析, 国防工业出版社, 2009.

BIG Θ NOTATION (发音THETA)

为描述平均时间复杂度的概念，引进BIG Θ

Definition 2.1

For a given complexity function $g(n)$, $\Theta(g(n))$ is the set of complexity functions $f(n)$ for which there exists some positive real constants c_1 and c_2 and some nonnegative integer n_0 such that, for all $n \geq n_0$,

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n).$$

- If $f(n) = \Theta(g(n))$, we say that $f(n)$ is "big Θ (大Θ)" or has the same order (数量级) of $g(n)$.
- $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$.
- Θ 代表既是上界、又是下界

- Big Θ can also be used to bound the worst-case time complexity.
  - For insertion sort, the worst-case is both $\Theta(n^2)$ and $O(n^2)$.
- However, we usually use Big O notation because we don't care the best-case.

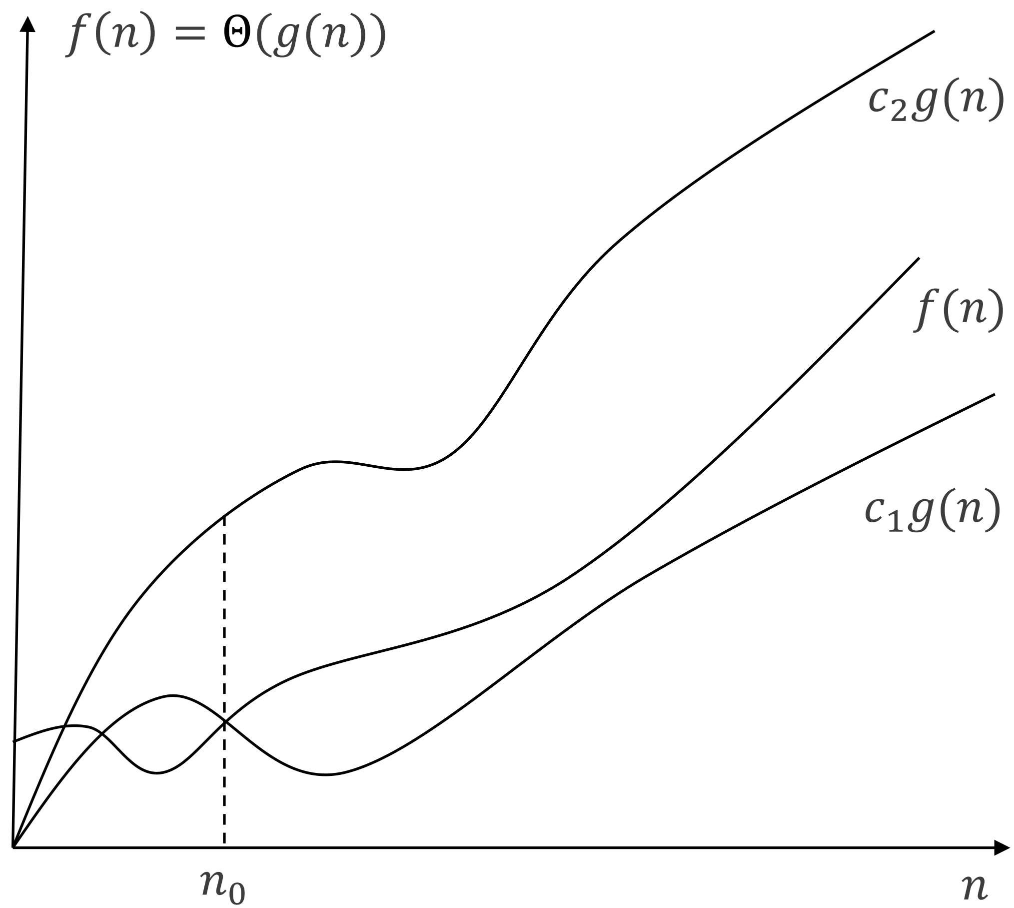


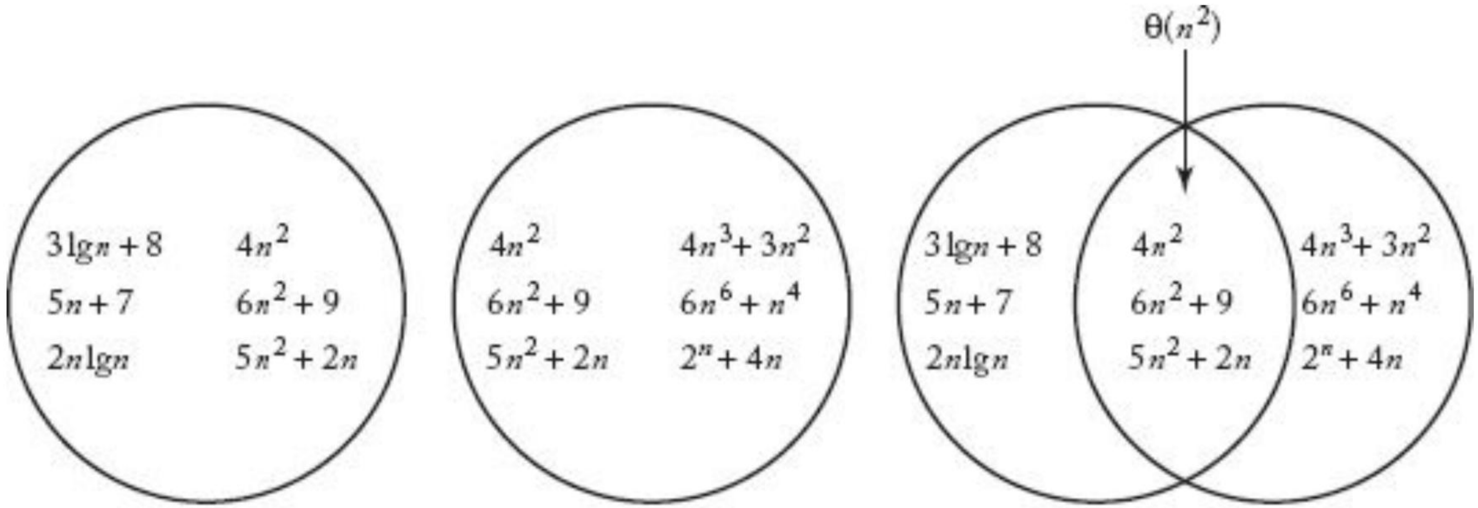
Image source: 图2.1, 张德富, 算法设计与分析, 国防工业出版社, 2009.

RELATION BETWEEN BIG O, BIG Ω AND BIG Θ (关系)

圆圈里面的函数的上界是O(n^2)

圆圈里面的函数的下界是Ω(n^2)

交集里面的函数的上确界和下确界是Θ(n^2)



(a) $O(n^2)$ (b) $\Omega(n^2)$ (c) $\Theta(n^2) = O(n^2) \cap \Omega(n^2)$

- Now we have O, Ω, and Θ. Intuitively, they just like "≤", "=", and "≥" for complexity functions.

Image source: Figure 1.6, Richard E. Neapolitan, Foundations of Algorithms (5th Edition), Jones & Bartlett Learning, 2014

Theorem 2.1 （定理）

For any two functions $f(n)$ and $g(n)$, $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

- $\Theta = \mathrm{O}$ and Ω.

$f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Theorem 2.2（定理）

For any two functions $f_1(n)$ and $f_2(n)$, if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, we have $f_1(n) + f_2(n) = O(\max\{g_1(n),\ g_2(n)\})$.

- Pick the larger one.

Transitivity (传递性)

- If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$.
- Same for O and Ω.

Additivity (可加性)

- If $f(n) = \Theta(h(n))$ and $g(n) = \Theta(h(n))$ then $f(n) + g(n) = \Theta(h(n))$.
- Same for O and Ω.

Reflexivity (自反性)

- $f(n) = \Theta(f(n))$.
- Same for O and Ω.

Symmetry (对称性)

- $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$.
- Not hold for O and Ω.

- 比较复杂度函数
- Consider the following ordering of complexity categories:
  $\Theta(\lg n)$ $\Theta(n)$ $\Theta(n \lg n)$ $\Theta(n^2)$ $\Theta(n^j)$ $\Theta(n^k)$ $\Theta(a^n)$ $\Theta(b^n)$ $\Theta(n!)$
  where $k \geq j \geq 2$ and $b \geq a \geq 1$.
- If $f(n)$ is to the left of $g(n)$ in the above sequence, then
  $$f(n) = O(g(n))$$
- Notice: Big Θ is a set of functions. We can't say $\Theta(\lg n) < \Theta(n)$.

Example 2

Given $f(n) = \frac{1}{2}n(n-1)$, prove that $f(n) = \Theta(n^2)$

Proof:

By the property, we first show that $f(n) = O(n^2)$:

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \text{ (for } c = \frac{1}{2} \text{ and } n_0 = 0).$$

Then we show that $f(n) = \Omega(n^2)$:

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n = \frac{1}{4}n^2 \text{ (for } c = \frac{1}{4} \text{ and } n_0 = 2).$$

Thus $f(n) = \Theta(n^2)$.

- In addition to proving by definition, we can also use limit to get asymptotic notations.

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}\begin{cases} = c & \text{implies } f(n) = \Theta\big(g(n)\big) \quad \text{if } 0 < c < \infty \\ = 0 & \text{implies } f(n) = O\big(g(n)\big) \\ = \infty & \text{implies } f(n) = \Omega\big(g(n)\big) \end{cases}$$

Example 3

Compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2.

$$\lim_{n\to\infty}\frac{\frac{1}{2}n(n-1)}{n^2}=\frac{1}{2}\lim_{n\to\infty}\frac{n^2-n}{n^2}=\frac{1}{2}\lim_{n\to\infty}(1-\frac{1}{n})=\frac{1}{2},$$

Thus, $\frac{1}{2}n(n-1)=\Theta(n^2)$.

Compare the orders of growth of a^n and b^n, when $b > a > 0$

Solution:

$$\lim_{n\to\infty}\frac{a^n}{b^n}=\lim_{n\to\infty}\left(\frac{a}{b}\right)^n=0.$$

The limit is 0 because $0<\frac{a}{b}<1$. Thus, $a^n=O(b^n)$.

- When calculating $\lim_{n\to\infty}\frac{f(n)}{g(n)}$, how to deal with the following cases?

$$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0 \text{ or } \pm\infty$$

L'Hôpital's Rule (洛必达法则)

If $f(x)$ and $g(x)$ are both differentiable with derivatives $f'(x)$ and $g'(x)$, respectively, and if

$$\lim_{x\to\infty} f(x) = \lim_{x\to\infty} g(x) = 0 \text{ or } \pm\infty,$$

then

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)},$$

whenever the limit on the right exists.

Example 4

$$\frac{d(\log_a x)}{dx} = \frac{1}{x \ln a}$$

$$\lg n = O(n)$$

because

$$\lim_{x\to\infty} \frac{\lg x}{x} = \lim_{x\to\infty} \frac{d(\lg x)/dx}{dx/dx} = \lim_{x\to\infty} \frac{1/(x \ln 2\,)}{1} = 0.$$

Show the correctness of the following statements.

- $\lg n = O(n)$
- $n = O(n \lg n)$
- $n \lg n = O(n^2)$

CONCLUSION

After this lecture, you should know:

- Why do we need asymptotic notation?
- What are the meaning of these asymptotic notations big O, big Θ, or big Ω?
- How to prove a complexity function is big O, big Θ, or big Ω?
- How to compare the order of two complexity function?

HOMEWORK

- Page 19

  2.1

  2.2

  2.3

  2.9

有问题欢迎随时跟我讨论

Definition

$\log_b a$ is the unique number c s.t. $b^c = a$.

- Notations:
  - $\lg n = \log_2 n$ (binary logarithm)
  - $\ln n = \log_e n$ (natural logarithm)
  - $\lg^k n = (\lg n)^k$ (exponentiation)
  - $\lg \lg n = \lg(\lg n)$ (composition)
- Derivative:
  - $\dfrac{d(\log_a x)}{dx} = \dfrac{1}{x \ln a}$
- Useful identities for all real $a > 0, b > 0, c > 0$, and n, and where logarithm bases are not 1 :
  - $\log_c(ab) = \log_c a + \log_c b$
  - $\log_b a^n = n \log_b a$
  - $\log_b \left(\frac{1}{a}\right) = -\log_b a$
  - $\log_b a = (\log_a b)^{-1}$

现代希腊语字母表

字体 / 序号	Times New Roman		Arial		Garamond		Monotype Corsiva		PMingLiu		Lucida Sans Unicode		英文注音	音标注音	中文注音
1	Α	α	Α	α	Α	α	Α	α	Α	α	Α	α	alpha	['ælfə]	阿尔法
2	Β	β	Β	β	Β	β	Β	β	Β	β	Β	β	beta	['beitə]	1贝塔 2比特
3	Γ	γ	Γ	γ	Γ	γ	Γ	γ	Γ	γ	Γ	γ	gamma	['gæmə]	伽马
4	Δ	δ	Δ	δ	Δ	δ	Δ	δ	Δ	δ	Δ	δ	delta	['deltə]	德尔塔
5	Ε	ε	Ε	ε	Ε	ε	Ε	ε	Ε	ε	Ε	ϵ	epsilon	[ep'sailən]	诶普西龙
6	Ζ	ζ	Ζ	ζ	Ζ	ζ	Ζ	ζ	Ζ	ζ	Ζ	ζ	zeta	['zi:tə]	1日-伊塔 2日-诶塔
7	Η	η	Η	η	Η	η	Η	η	Η	η	Η	η	eta	['i:tə]	伊塔
8	Θ	θ	Θ	θ	Θ	θ	Θ	θ	Θ	θ	Θ	θ	theta	['θi:tə]	西塔
9	Ι	ι	Ι	ι	Ι	ι	Ι	ι	Ι	ι	Ι	ι	iota	[ai'əutə]	1爱欧塔 2哟塔
10	Κ	κ	Κ	κ	Κ	ϰ	Κ	κ	Κ	κ	Κ	κ	kappa	[kæpə]	卡帕
11	Λ	λ	Λ	λ	Λ	λ	Λ	λ	Λ	λ	Λ	λ	lambda	['læmdə]	兰-布达
12	Μ	μ	Μ	μ	Μ	μ	Μ	μ	Μ	μ	Μ	μ	mu	[mju:]	1谬 2木
13	Ν	ν	Ν	ν	Ν	ν	Ν	ν	Ν	ν	Ν	ν	nu	[nju:]	1拗(niu) 2怒
14	Ξ	ξ	Ξ	ξ	Ξ	ξ	Ξ	ξ	Ξ	ξ	Ξ	ξ	xi	[ksai]	1克-赛2然-爱3可-西
15	Ο	ο	Ο	ο	Ο	ο	Ο	ο	Ο	ο	Ο	ο	omicron	[oumaik'rən]	1欧麦克荣2欧米克荣
16	Π	π	Π	π	Π	π	Π	π	Π	π	Π	π	pi	[pai]	派
17	Ρ	ρ	Ρ	ρ	Ρ	ρ	Ρ	ρ	Ρ	ρ	Ρ	ρ	rho	[rou]	楼
18	Σ	σ	Σ	σ	Σ	σ	Σ	σ	Σ	σ	Σ	σ	sigma	['sigmə]	西格玛
19	Τ	τ	Τ	τ	Τ	τ	Τ	τ	Τ	τ	Τ	τ	tau	[tau]	1套 2拓
20	Υ	υ	Υ	υ	Υ	υ	Υ	υ	Υ	υ	Υ	υ	upsilon	[ju:p'silən]	1宇普西龙2哦普斯龙
21	Φ	φ	Φ	φ	Φ	φ	Φ	φ	Φ	φ	Φ	ϕ	phi	[fai]	1佛-爱 2佛-伊
22	Χ	χ	Χ	χ	Χ	χ	Χ	χ	Χ	χ	Χ	χ	chi	[kai]	1恺 2可-亿
23	Ψ	ψ	Ψ	ψ	Ψ	ψ	Ψ	ψ	Ψ	ψ	Ψ	ψ	psi	[psai]	1普西 2普赛
24	Ω	ω	Ω	ω	Ω	ω	Ω	ω	Ω	ω	Ω	ω	omega	['oumigə]	1欧美伽 2欧米伽

注:所有的各种注音仅供参考，与标准的希腊语发音有区别。中文注音按照标准希腊语发音标注，并参考网上及教科书的中文注音，力求准确，但与标准的希腊语发音仍有出入。有些字母的读音不只一种，中文注音给出了不同的发音，需要连读为一个音节的，用“-”连结。 制作:MHL 2013/08/12