Python 进阶

- 模块，包，程序
- 系统操作
- 算法基础

大纲

- 模块，包，程序(module, packages, program)
- 系统操作(File I/O, Systems)
- 算法基础(Complexity and BigO,
- Divide & Conquer, Sorting)

程序，功能的组合

第 5 章

Python盒子：模块、包和程序

5.3 模块和import语句

继续进入下一个阶段：在多个文件之间创建和使用 Python 代码。一个模块仅仅是 Python 代码的一个文件。

本书的内容按照这样的层次组织：单词、句子、段落以及章。否则，超过一两页后就没有很好的可读性了。代码也有类似的自底向上的组织层次：数据类型类似于单词，语句类似于句子，函数类似于段落，模块类似于章。以此类推，当我说某个内容会在第 8 章中说明时，就像是在其他模块中引用代码。

引用其他模块的代码时使用 import 语句，被引用模块中的代码和变量对该程序可见。

继续进入下一个阶段：在多个文件之间创建和使用 Python 代码。一个模块仅仅是 Python 代码的一个文件。

- 数据类型 像 单词
- 语句 像 句子
- 函数 像 段落
- 包 像 章节

```
sound/
        __init__.py
        formats/
                __init__.py
                wavread.py
                wavwrite.py
                aiffread.py
                aiffwrite.py
                auread.py
                auwrite.py
                ...
        effects/
                __init__.py
                echo.py
                surround.py
                reverse.py
                ...
        filters/
                __init__.py
                equalizer.py
                vocoder.py
                karaoke.py
```

系统操作

- 文件，目录
- 程序，进程
- 日期 和 时间

Data Scientist Interview

- Resume
- Machine Learning
- Probability and Statistics
- Algorithm and Coding
- SQL
- Case Study
- Behavior Question

Algorithm and Coding

- Big O, Time Complexity, Space Complexity
- Searching and Sorting
- Array, List, String, Set, Dictionary
- Divide and Conquer
- Easy to Medium
- Advanced:
  - Various Data Structure
  - Medium

算法基础

- 什么是算法
- 如何评估算法
- 算法入门

算法 Algorithms

1) **x > 1,** 求 x的平方根y, 0 < y < x,
设 Low 为 0, High 为 x

2) 假设 **Guess** 是 **(Low+High) / 2,** 如果**Guess**的平方非常接近**x,** 那么 **y = g**

3) 若, **g*g < x,** **L**设定为**Guess ,** 然后重复第二步

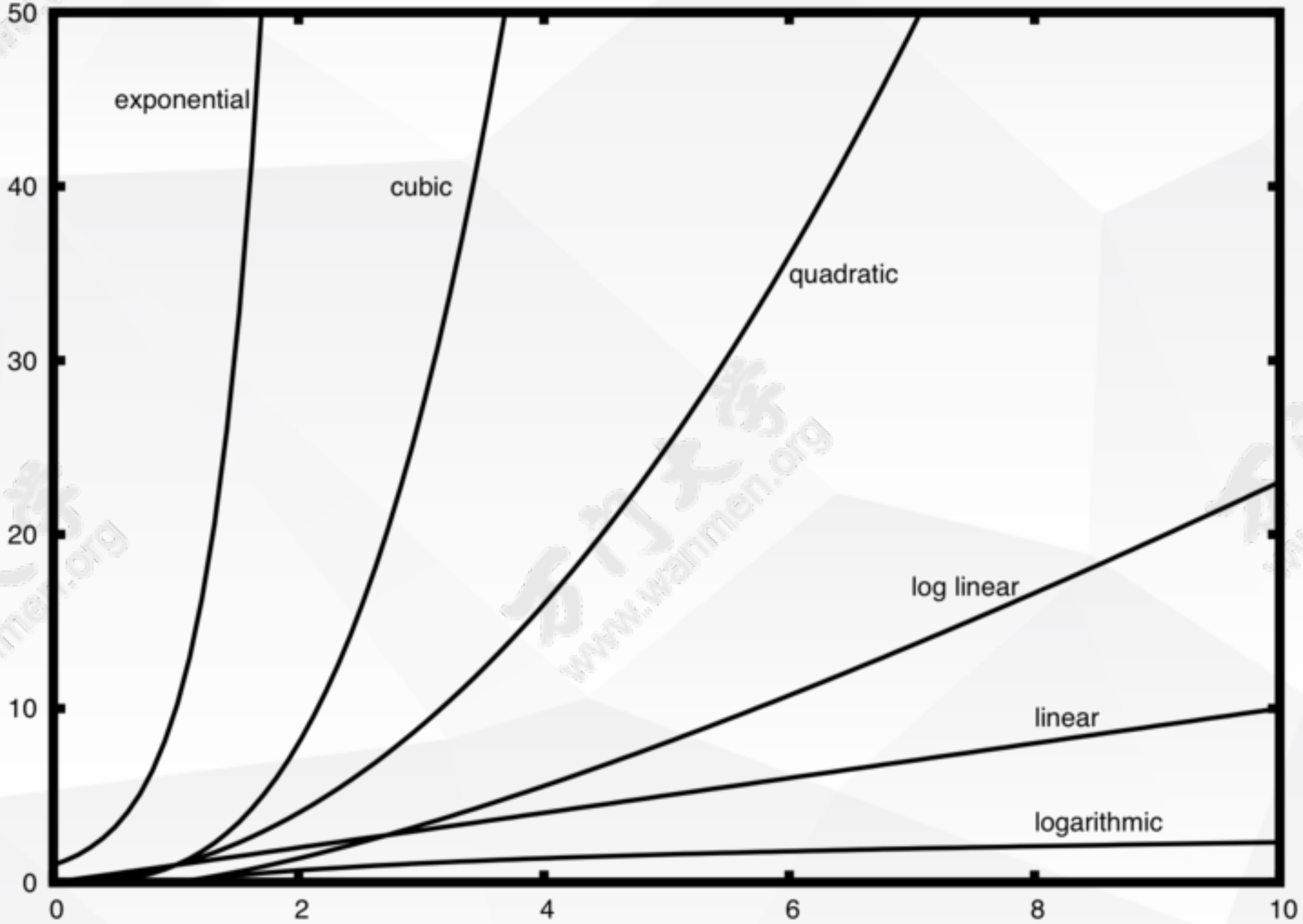
4) 否则, **g*g > x,** **H** 设定为**Guess ,** 然后重复第二步

按步骤，告诉计算机解决问题的方法

算法就是解决问题

- 问题是什么 Problem
- 我们有什么 Input
- 我们想要得到什么 Output
- 尝试最简单的方法 Simple Solution
- 看看如何改进 Develop Incrementally

Algorithm Analysis

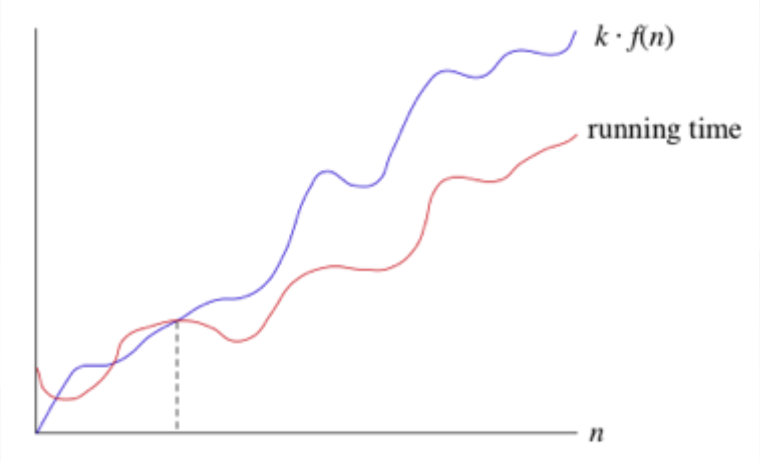


Big O

- 描述算法性能及复杂度的注解
- O(1), O($\log n$), O(n), O($n \log n$), O(n^2), O(2^n)

Big Oh Notation

- **Big-O notation**
- We use *Θ(n)* notation to asymptotically bound the growth of a running time to within constant factors above and below. Sometimes we want to bound from only above.
- Although the worst-case running time of binary search is *Θ(lgn)*, it would be incorrect to say that binary search runs in *Θ(lgn)* time in all cases.
- The running time of binary search is never worse than *Θ(lgn)*, but it's sometimes better.



Notation Summary

notation	provides	example	shorthand for	used to
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$ $\vdots$	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$ $\vdots$	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$	develop lower bounds

Algorithms (4th Edition)
Robert Sedgewick, Kevin Wayne

Master Theorem

The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$. Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Master Theorem

$$T(n) = 3T(n/2) + n^2 \implies T(n) = \Theta(n^2) \text{ (Case 3)}$$

$$T(n) = 4T(n/2) + n^2 \implies T(n) = \Theta(n^2 \log n) \text{ (Case 2)}$$

$$T(n) = T(n/2) + 2^n \implies \Theta(2^n) \text{ (Case 3)}$$

$$T(n) = 16T(n/4) + n \implies T(n) = \Theta(n^2) \text{ (Case 1)}$$

$$T(n) = 2T(n/2) + n \log n \implies T(n) = n \log^2 n \text{ (Case 2)}$$

https://math.dartmouth.edu/archive/m19w03/public_html/Section5-2.pdf
http://www.cs.cornell.edu/courses/cs3110/2011sp/lectures/lec19-master/mm-proof.pdf

排序

- Bubble Sort
- Insertion Sort
- Merge Sort
- Quick Sort

Sort and Search

- Sort
- Binary Search
- Divide and Conquer
- Two Pointers
- Sliding Window
- Others
- Greedy
- Dynamic Programming *

Coding Time

beijing@dataapplab.com