计算机问题求解 — 论题2-03

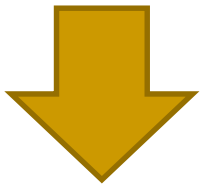
- 算法的效率

2016年03月09日

问题1：你如何理解这里的“优化”？算法级？程序级？

(2) **for** I **from** 1 **to** N **do**:

(2.1) $L(I) \leftarrow L(I) \times 100/MAX$



(1) compute the maximum score in MAX;

(2) $FACTOR \leftarrow 100/MAX$;

(3) **for** I **from** 1 **to** N **do**:

(3.1) $L(I) \leftarrow L(I) \times FACTOR$.

问题2：以下两个程序，在执行效率上有什么区别？

- 对一个二维数组A[m,n]进行遍历：
- 程序1：
  - for *I* from 1 to M do
    - for *J* from 1 to N do
      - do something with A[*I*,*J*]
- 程序2：
  - for *J* from 1 to N do
    - for *I* from 1 to M do
      - do something with A[*I*,*J*]

问题3：

你能说出如何用Linear Search算法搜索一个未排序的序列吗？书中给出的优化方法是什么？这种优化是量上的优化还是质上的优化？？

给定一个算法，什么样的优化算是质上的优化？

问题3：

“算法分析”主要是干什么？

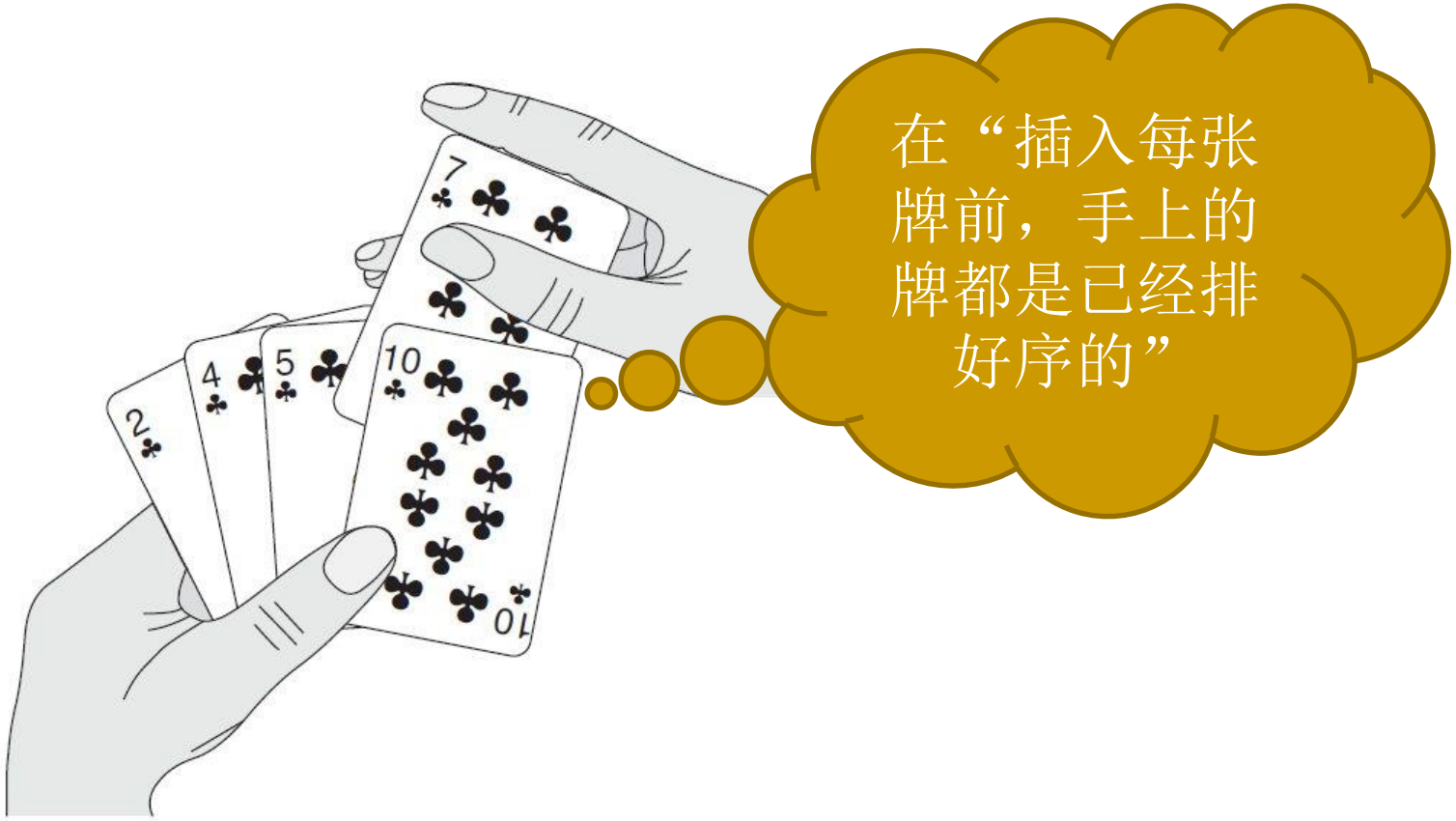
一个插入排序方法

Our first algorithm, insertion sort, solves the ***sorting problem*** introduced in Chapter 1:

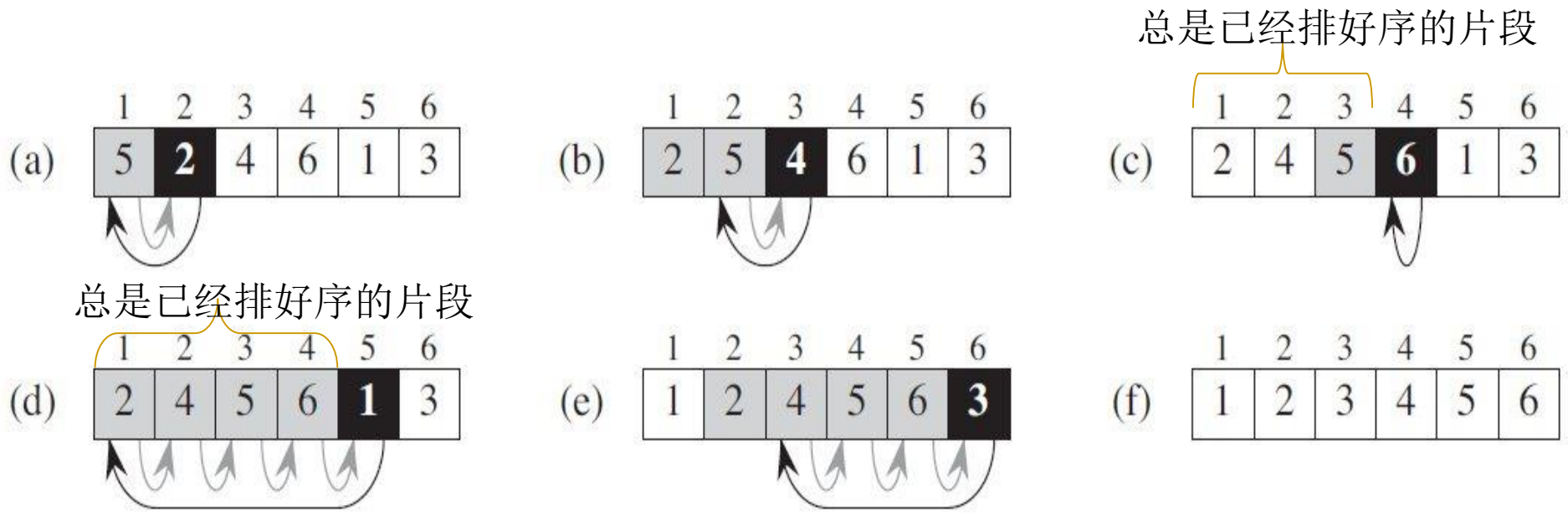
Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

插入法：



范例


总是已经排好序的片段
(a)
1 2 3 4 5 6
5 2 4 6 1 3
(b)
1 2 3 4 5 6
2 5 4 6 1 3
(c)
1 2 3 4 5 6
2 4 5 6 1 3
总是已经排好序的片段
(d)
1 2 3 4 5 6
2 4 5 6 1 3
(e)
1 2 3 4 5 6
1 2 4 5 6 3
(f)
1 2 3 4 5 6
1 2 3 4 5 6

伪代码

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

用哪些数值来标定算法的性能？

- 时间性能（时间复杂度）
  - 算法在给定一个输入的情况下，要执行多少条指令

没错，数数字！

数数字！

```
INSERTION-SORT(A)                                   cost    times
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted
           sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

性能评估函数

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

问题4：我们能用一个具体的数值来表示算法的效率吗？如果不能，我们该用什么？

一个N上的函数

对于任意一个算法，我们都能找到一个N上的函数来表示算法的效率吗？

Best case

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \, .
\end{aligned}
$$

Worst case

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum$$

e these summations)
-SORT is

$$\left(\frac{n(n+1)}{2} - 1\right)$$

$$\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

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

$$-(c_2 + c_4 + c_5 + c_8) .$$

best case和worst case两种结果中，更重要的是哪个?

既然有best case和worst case，有average case吗？如果有，会如何进行average case的分析?

我们需要对一个算法的语句的执行条数进行统计，但我们需要如此精确地进行统计吗?

往往是：我们可以容忍我们的某种程度上的“粗心”：

- 我们往往忽略不同语句的执行开销
- 我们选择“代表性”语句，进行统计
  - 哪些是代表性的语句？
- 我们往往忽略代表性语句执行条数的“系数”而只保留其指数
  - 系数的忽略会导致算法性能评估的失效吗？

插入排序：best case是n级别的；worst case是n^2级别的。
归并排序：worst case是nlgn级别的
哪个算法“好”呢？

问题5:
什么是“Big-O”？

Average case

The "average case" is often roughly as bad as the worst case. Suppose that we randomly choose n numbers and apply insertion sort. How long does it take to determine where in subarray $A[1 \,..\, j-1]$ to insert element $A[j]$? On average, half the elements in $A[1 \,..\, j-1]$ are less than $A[j]$, and half the elements are greater. On average, therefore, we check half of the subarray $A[1 \,..\, j-1]$, and so t_j is about $j/2$. The resulting average-case running time turns out to be a quadratic function of the input size, just like the worst-case running time.

两个不同算法的优劣关键是”增长率”

Algorithm	1	2	3	4	
Time function(ms)	$33n$	$46n \lg n$	$13n^2$	$3.4n^3$	2^n
Input size(n)	Solution time				
10	0.00033 sec.	0.0015 sec.	0.0013 sec.	0.0034 sec.	0.001 sec.
100	0.0033 sec.	0.03 sec.	0.13 sec.	3.4 sec.	4×10^{16} yr.
1,000	0.033 sec.	0.45 sec.	13 sec.	0.94 hr.	
10,000	0.33 sec.	6.1 sec.	22 min.	39 days	
100,000	3.3 sec.	1.3 min.	1.5 days	108 yr.	
Time allowed	Maximum solvable input size (approx.)				
1 second	30,000	2,000	280	67	20
1 minute	1,800,000	82,000	2,200	260	26

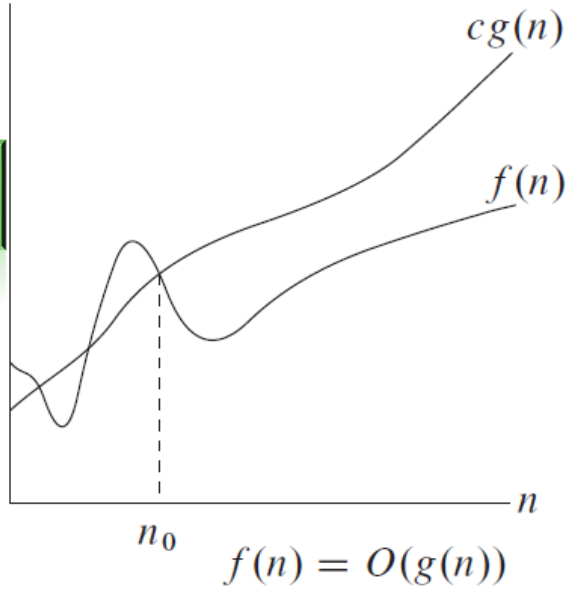
通常情况下：

算法	1	2	3	4	5
Big-O	O(n)	O(nlgn)	$O(n^2)$	$O(n^3)$	$O(2^n)$

集合 “Big Oh”

- Definition
  - Giving $g:N\rightarrow R^+$, then $O(g)$ is the set of f:N$\rightarrow$R$^+$, such that for some c$\in$R$^+$ and some $n_0\in$N, $f(n)\leq cg(n)$ for all $n\geq n_0$.

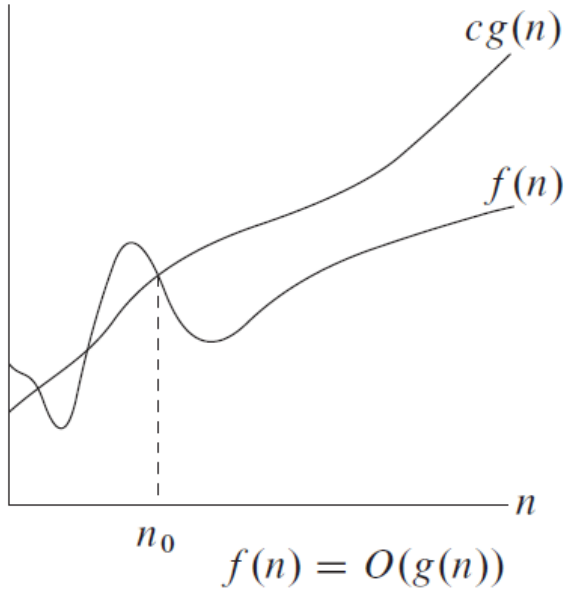
问题：你能结合右图解释c和n_0的含义吗？



集合 “Big Oh”

- Definition
  - Giving $g:N\rightarrow R^+$, then $O(g)$ is the set of $f:N\rightarrow R^+$, such that for some $c\in R^+$ and some $n_0\in N$, $f(n)\leq cg(n)$ for all $n\geq n_0$.
- If $f\in O(g)$,we say that f grows not fast than g
- At the same time if $g\in O(f)$, we say that they grow at the same speed.

问题：你能结合下图解释f和g的增长性吗？



实际上，我们可以用下式来判断：

- 函数 f 是O(g) if $\lim_{n\to\infty}[f(n)/g(n)]=c<\infty$
- if there exists constants $c \in R^+$ and $n_0 \in N$ such that for all $n(n>=n_0)$, $f(n)\le cg(n)$

- Let $f(n)=n^2$, $g(n)=n\lg n$, then:
  - $f\notin O(g)$, since $\lim_{n\to\infty}\frac{n^2}{n\lg n}=\lim_{n\to\infty}\frac{n}{\lg n}=\infty$
  - $g\in O(f)$, since $\lim_{n\to\infty}\frac{n\log n}{n^2}=\lim_{n\to\infty}\frac{\log n}{n}=0$

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

对数函数与幂函数

Which grows faster?

$\log_2 n$ or $\sqrt{n}$?

So, $\log_2 n \in O(\sqrt{n})$

$$\lim_{n\to\infty}\frac{\log_2 n}{\sqrt{n}}=\lim_{n\to\infty}\frac{\frac{\log_2 e}{n}}{\frac{1}{2\sqrt{n}}}=(2\log_2 e)\lim_{n\to\infty}\frac{\sqrt{n}}{n}=0$$

一般性结论

- The log function grows more slowly than ***any*** positive power of n

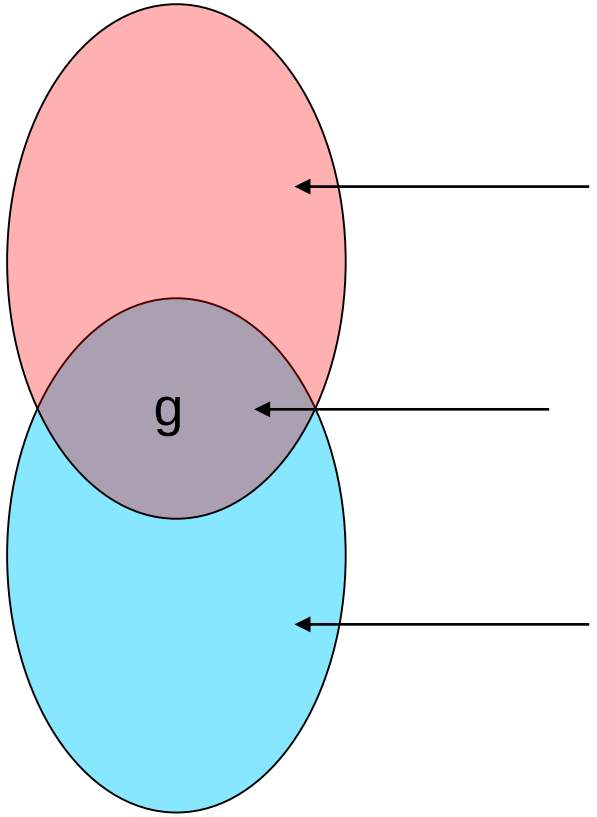
$$\lg n \in o(n^{\alpha}) \text{ for any } \alpha>0$$

By the way:

The power of n grows more slowly than any exponential function with base greater than 1

$$n^{k} \in o(c^{n}) \text{ for any } c>1$$

函数增长率的比较



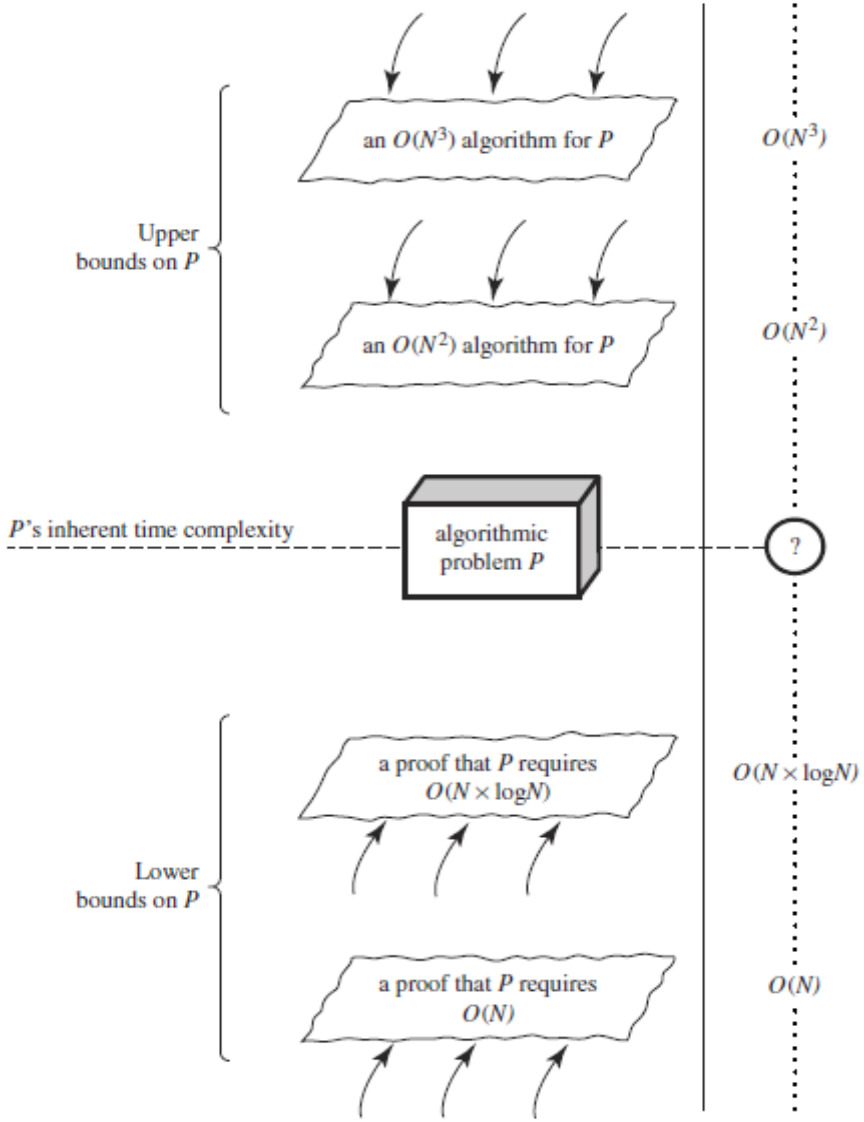
Ω(g): 该集合中任一函数的增长率不低于*g*的增长率（至少与*g*增长得一样快！）

Θ(g): 这里的函数与g具有“相同”的增长率。

O(g): 该集合中任一函数的增长率不高于*g*的增长率（最多与*g*增长得一样快！）

问题6：

给定一个算法，什么样的
优化算是质上的优化？



问题7：什么是“Algorithmic Gap”？

Open topic:

- What we talk about when we talk about the robustness of Big O?
  - 建议以二分搜索算法为例

- 证明：“在一个长度为N的有序列表L中搜索对象Y”问题的难度是$\log_2 N$

课外作业

- DH: pp.153-
  - 6.1 – 6.3
  - 6.10 , 6.11
  - 6.15, 6.16