递归数列及函数“增长”

离散数学教学组

回顾

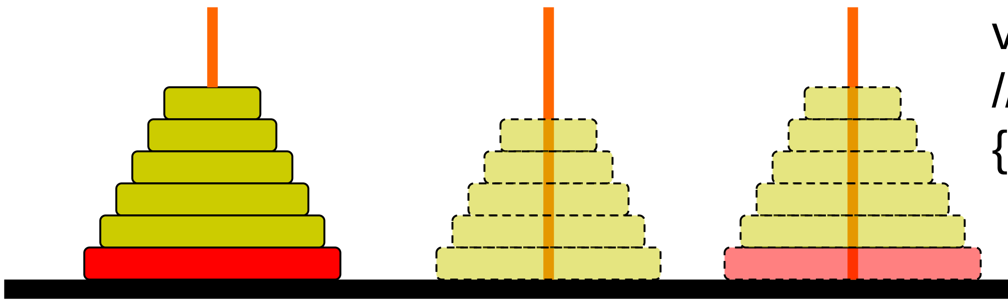
- 鸽笼原理
  - 基本的原理
  - 一般的鸽笼原理
  - 运用的例子
- 排列与组合
  - 基本的排列组合
  - 组合与二项式系数
  - 有重复的排列组合

提要

- 递归数列
- 函数“增长”

递归思维：例 1

- 汉诺塔问题: How many moves are need to move all the disks to the third peg by moving only one at a time and never placing a disk on top of a smaller one.



$$T(1) = 1$$
$$T(n) = 2T(n-1) + 1$$

```
void hanoi(int n,char one, two, three)
// 将n个盘从one座借助two座,移到three座
{
   void move(char x,char y);
   if(n==1)  then move(one,three);
      else  {
          hanoi(n-1,one,three,two);
          move(one,three);
          hanoi(n-1,two,one,three);
       }
}
```

汉诺塔问题的解

$$T(n) = 2T(n-1) + 1$$

$$2T(n-1) = 4T(n-2) + 2$$

$$4T(n-2) = 8T(n-3) + 4$$

$$\ldots\ldots\ldots$$

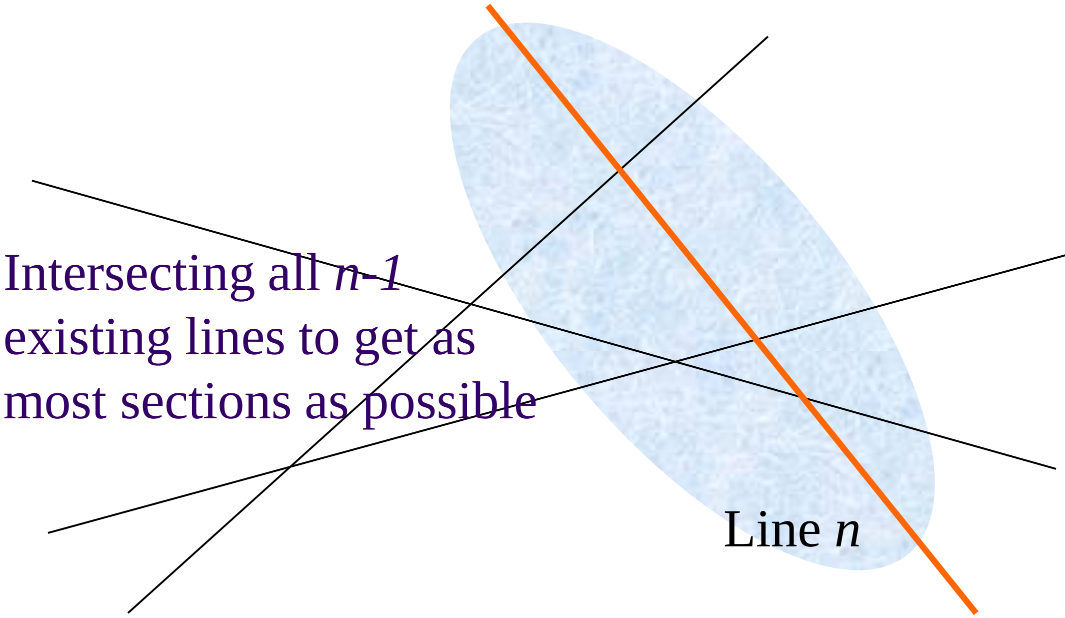
$$2^{n-2}T(2) = 2^{n-1}T(1) + 2^{n-2}$$

$$\Rightarrow$$

$$\boldsymbol{T(n) = 2^n - 1}$$

递归思维：例 2

- Cutting the plane
  - How many sections can be generated at most by *n* straight lines with infinite length？



$$L(0) = 1$$
$$L(n) = L(n-1) + n$$

Solution of Cutting the Plane

$$
\begin{aligned}
L(n) &= L(n-1)+n \\
&= L(n-2)+(n-1)+n \\
&= L(n-3)+(n-2)+(n-1)+n \\
&= \ldots\ldots \\
&= L(0)+1+2+\ldots\ldots+(n-2)+(n-1)+n
\end{aligned}
$$

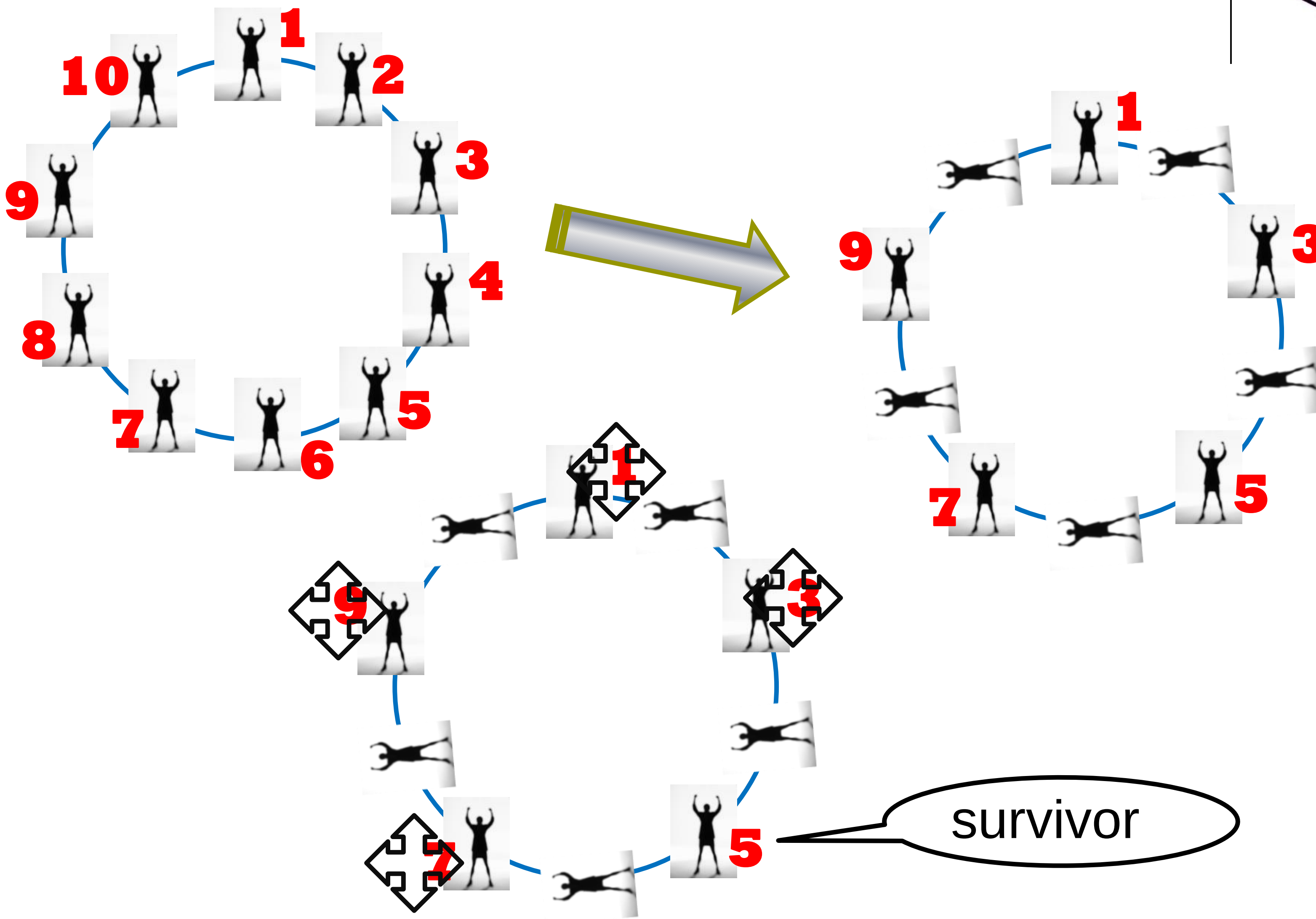
$$L(n) = n(n+1)/2 + 1$$

递归思维：例 3
Josephus Problem

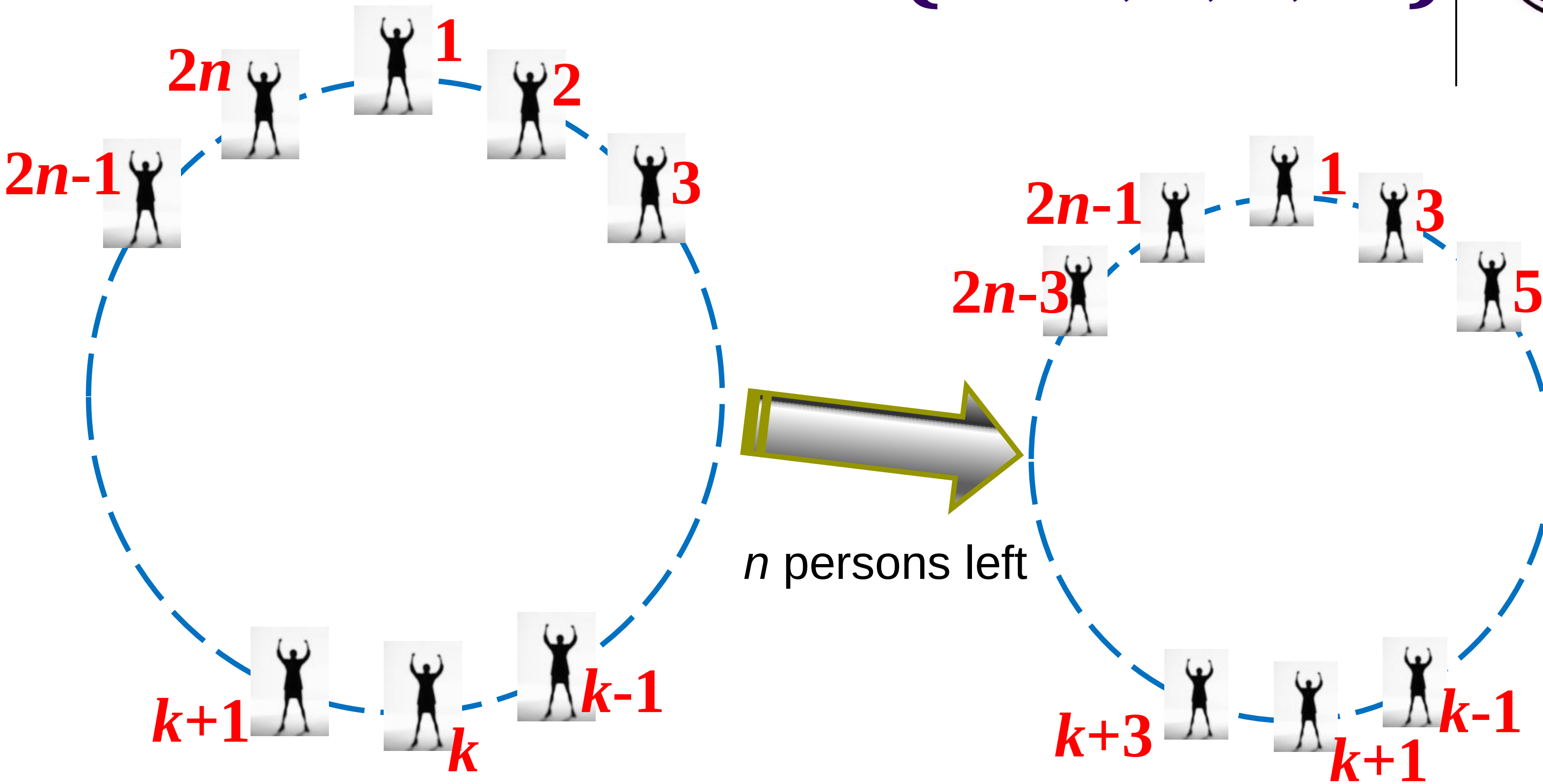
- Live or die, it’s a problem!
- Legend has it that Josephus wouldn't have lived to become famous without his mathematical talents. During the Jewish Roman war, he was among a band of 41 Jewish rebels trapped in a cave by the Romans. Preferring suicide to capture, the rebels decided to form a circle and, proceeding around it, to kill every third remaining person until no one was left. But Josephus, along with an unindicted co-conspirator, wanted none of this suicide nonsense; so he quickly calculated where he and his friend should stand in the vicious circle.

We use a simpler version:
“every second...”

Make a Try: for *n*=10



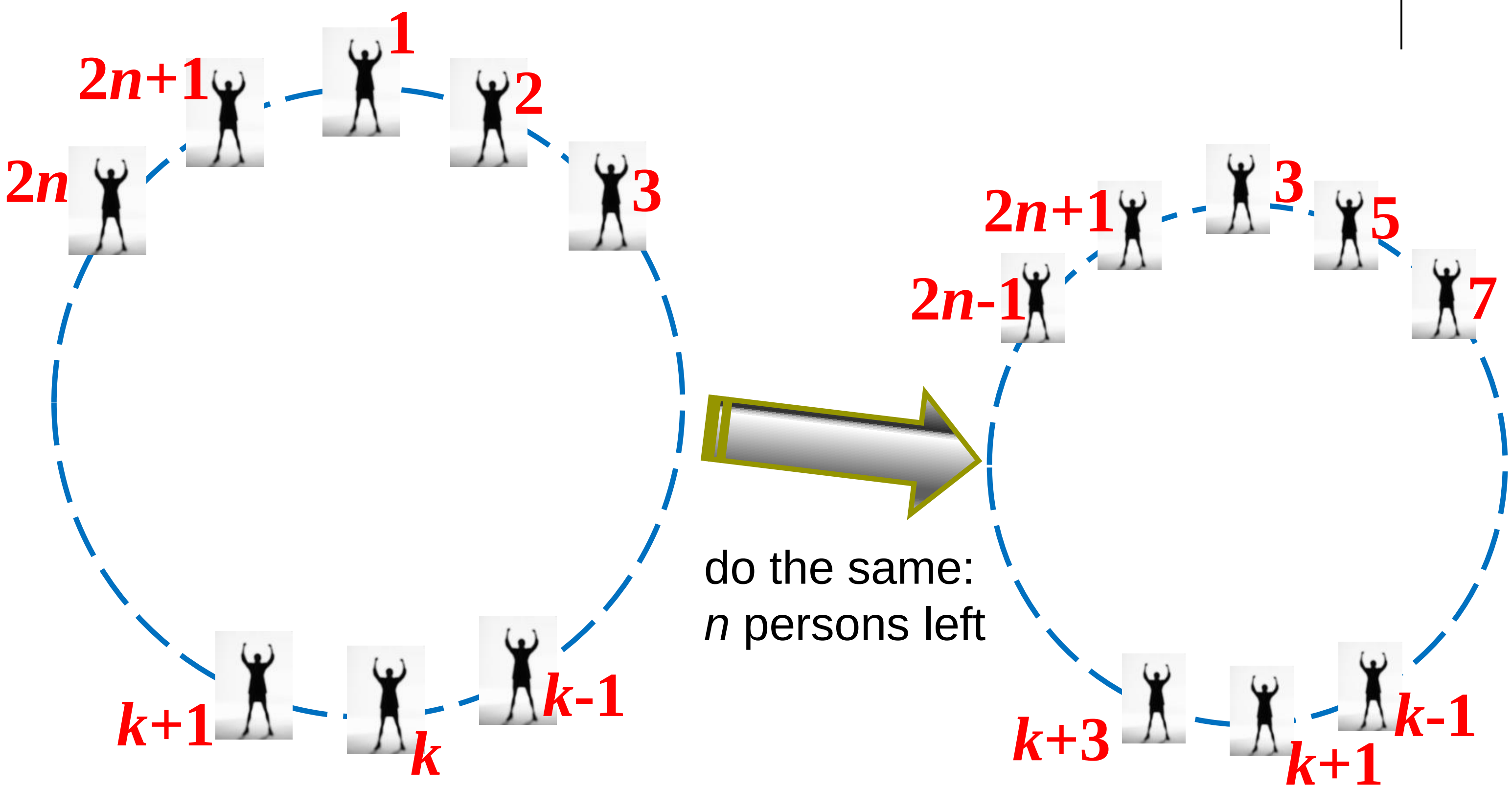
For 2*n* Persons (*n*=1,2,3,...)



The solution is: newnumber (J(*n*))

And the newnumber(*k*) is **2*k*-1**

And What about 2*n*+1 Persons (*n*=1,2,3,…)



The solution is: newnumber (J(*n*))

And for the time, the newnumber(*k*) is **2*k*+1**

Solution in Recursive Equations

$$
\begin{aligned}
J(1) &= 1\,; \\
J(2n) &= 2J(n) - 1\,, && \text{for } n \geq 1\,; \\
J(2n+1) &= 2J(n) + 1\,, && \text{for } n \geq 1\,.
\end{aligned}
$$

Explicit Solution for small *n*'s

n	1	2 3	4 5 6 7	8 9 10 11 12 13 14 15	16
$J(n)$	1	1 3	1 3 5 7	1 3 5 7 9 11 13 15	1

Look carefully ...

and, find the pattern...

and, prove it!

Eureka!

If we write n in the form $n = 2^m + l$,
(where 2^m is the largest power of 2 not exceeding n and where l is what's left),
the solution to our recurrence seems to be:

$$J(2^m + l) = 2l + 1, \qquad \text{for } m \geq 0 \text{ and } 0 \leq l < 2^m.$$

As an example: J(100) = J(64+36) = 36*2+1 = 73

Binary Representation

- Suppose *n*'s binary expansion is :

$$n = (b_m \, b_{m-1} \ldots b_1 \, b_0)_2$$

- then:

$$\begin{aligned}
n &= (1 \, b_{m-1} \, b_{m-2} \ldots b_1 \, b_0)_2 \,, \\
l &= (0 \, b_{m-1} \, b_{m-2} \ldots b_1 \, b_0)_2 \,, \\
2l &= (b_{m-1} \, b_{m-2} \ldots b_1 \, b_0 \, 0)_2 \,, \\
2l + 1 &= (b_{m-1} \, b_{m-2} \ldots b_1 \, b_0 \, 1)_2 \,, \\
J(n) &= (b_{m-1} \, b_{m-2} \ldots b_1 \, b_0 \, b_m)_2
\end{aligned}$$

递归（递推）

Reproducing pairs (at least two months old)	Young pairs (less than two months old)	Month	Reproducing pairs	Young pairs	Total pairs
		1	0	1	1
		2	0	1	1
		3	1	1	2
		4	1	2	3
		5	2	3	5
		6	3	5	8

Rabbits and the Fibonacci Numbers

递归（递推）数列

- 例子:
  - 4,7,10,13,16,......
  - 1,1,2,3,5,8,13,21,34,...... (a)
  - 0, 1, 2, 2, 6, 5,12,10, 20, 17, 30, 26, ...

- Recurrence relation: the recursive formula, e.g.:
  - for (a)
    - $f_n = f_{n-1}+f_{n-2}$ $(n>2)$, $f_1=f_2=1$
    - $f_1=f_2=1$: initial condition

寻找递推公式

- Let A={0,1}. C_n: the number of strings of length *n* in A* that do not contain adjacent 0's
  - C_1=?; C_2=?;
  - C_3=?
  - C_n=?
- $C_n = C_{n-1} + C_{n-2}$

寻找显式公式

- 如何为递归序列给出“显式”的公式
  - 即找到一个以自然数为定义域的函数
- Backtracking
  - E.g. 1:
    - $a_n = a_{n-1}+3$, $a_1 = 2$ =>recurrence relation
    - $a_n = 2+3(n-1)$ => explicit formula
  - E.g. 2
    - $b_n = 2b_{n-1}+1$, $b_1 = 7$
    - $b_n = 2^{n+2}-1$

Linear Homogeneous Relation

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_m a_{n-k}$$

is called linear homogeneous relation of degree k.

$$c_n = (-2)c_{n-1}$$

$$f_n = f_{n-1} + f_{n-2}$$

Yes

$$a_n = a_{n-1} + 3$$

$$g_n = g_{n-1}^2 + g_{n-2}$$

No

特征方程

- For a linear homogeneous recurrence relation of degree *k*

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_m a_{n-k}$$

the polynomial of degree *k*

$$x^k = r_1 x^{k-1} + r_2 x^{k-2} + \cdots + r_m$$

is called its characteristic equation.

- The characteristic equation of linear homogeneous recurrence relation of degree 2 is:

$$\boxed{x^2 - r_1 x - r_2 = 0}$$

Solution of Recurrence Relation

- If the characteristic equation $x^2 - r_1 x - r_2 = 0$ of the recurrence relation $a_n = r_1 a_{n-1} + r_2 a_{n-2}$ has two distinct roots s_1 and s_2, then

$$a_n = u s_1^n + v s_2^n$$

where *u* and *v* depend on the initial conditions, is the explicit formula for the sequence.

Proof of the Solution

Remember the equation : $x^2 - r_1 x - r_2 = 0$

We need prove that : $us_1^n + vs_2^n = r_1 a_{n-1} + r_2 a_{n-2}$

$$
\begin{aligned}
&us_1^n + vs_2^n = us_1^{n-2} s_1^2 + vs_2^{n-2} s_2^2 \\
&= us_1^{n-2}(r_1 s_1 + r_2) + vs_2^{n-2}(r_1 s_2 + r_2) \\
&= r_1 us_1^{n-1} + r_2 us_1^{n-2} + r_1 vs_2^{n-1} + r_2 vs_2^{n-2} \\
&= r_1(us_1^{n-1} + vs_2^{n-1}) + r_2(us_1^{n-2} + vs_2^{n-2}) \\
&= r_1 a_{n-1} + r_2 a_{n-2}
\end{aligned}
$$

Solution of Recurrence Relation

- If the equation has a single root s, then,

$$a_n = us^n + vns^n$$

Solution of Recurrence Relation

- $c_n = 3c_{n-1} - 2c_{n-2},\ c_1 = 5,\ c_2 = 3$
  - Characteristic equation:
    - $X^2 = 3x - 2$;
  - Get the root: 1,2
  - $C_n = u*1^n + v*2^n$
  - We have equations:
    - $C1 = u+2v = 5$
    - $C2 = u+4v = 3$
  - So: $u = 7,\ v = -1$
  - So: $C_n = 7-2^n$

Fibonacci Sequence

$f_1=1$

$f_2=1$

$f_n= f_{n-1}+ f_{n-2}$

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Explicit formula for Fibonacci Sequence

The characteristic equation is $x^2-x-1=0$, which has roots:

$$s_1=\frac{1+\sqrt{5}}{2} \quad and \quad s_2=\frac{1-\sqrt{5}}{2}$$

Note: (by initial conditions) $f_1=us_1+vs_2=1$ *and* $f_2=us_1^{\,2}+vs_2^{\,2}=1$

which results:

$$f_n=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n-\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

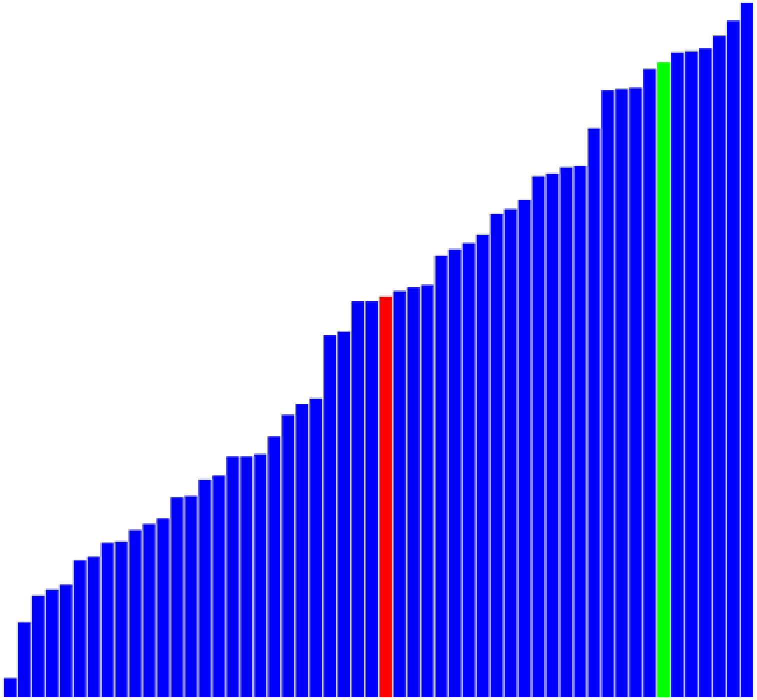
算法的执行步骤数

- 算法的正确性 VS 算法的效率
- 如何去评判一个算法的效率？
  - 时间开销: steps
  - 空间开销: memory
- 算法的执行步骤计数是主要手段
  - 算法的执行步骤数不是简单的算法语句条数！

从一个已排序的列表中查找

- 线性查找 Linear (sequential) search

- 折半查找 Binary search



插入排序法

6 5 3 1 8 7 2 4

遍历所有元素：

构造已排序的子序列；

将待排序元素插入子序列中的合适位置；

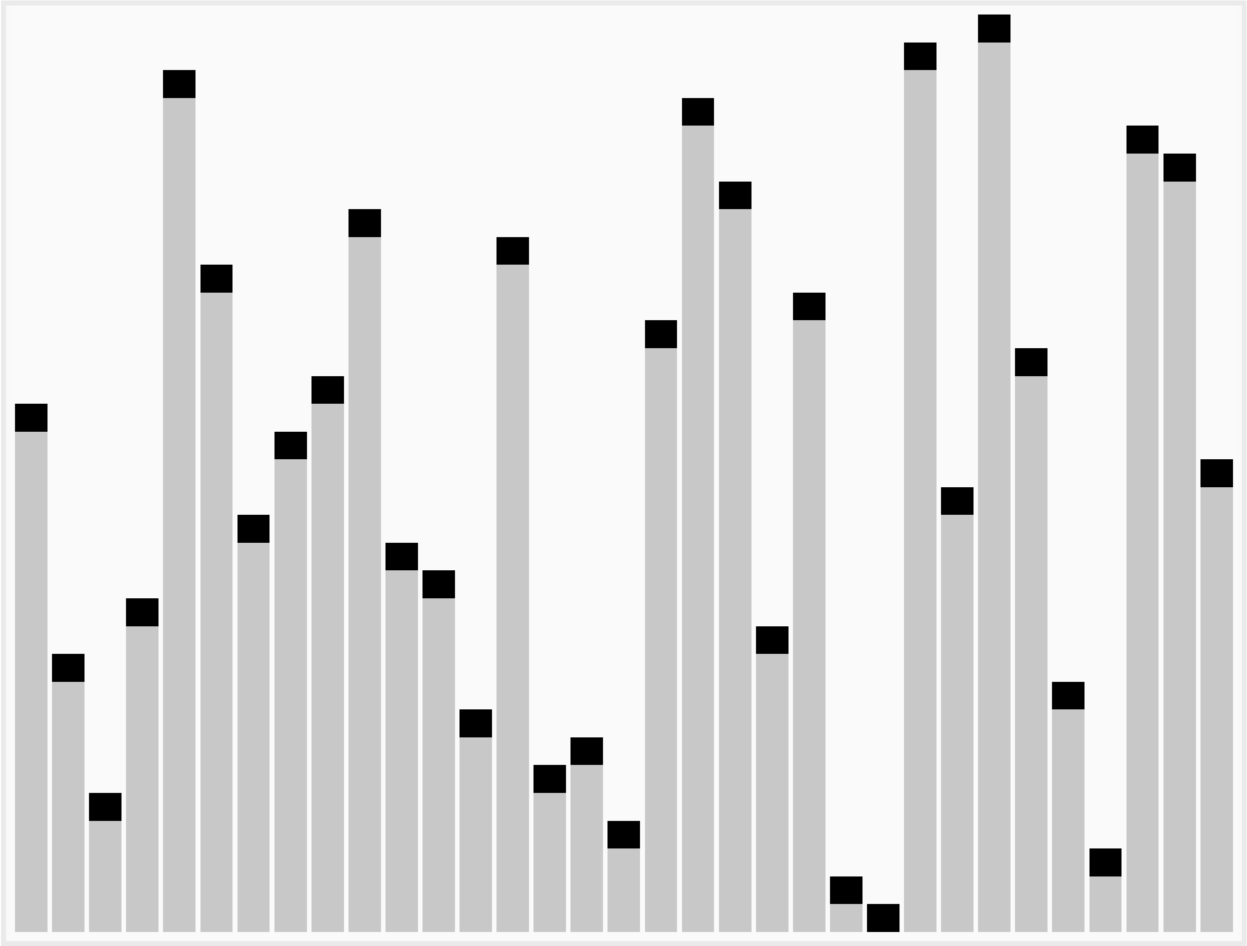
插入法伪代码

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

冒泡排序

6　5　3　1　8　7　2　4

快速排序



算法的执行步骤数

N (数据集规模)	T(n) (算法执行步数)
10	550
50	63750
100	505000

算法执行步数随着数据规模的变化而变化
不同的算法，变化的“剧烈程度”不同

引入一个数学工具
来刻画这种变化并
尝试判断其规律

算法执行步骤函数

- 针对每个算法，可以定义该算法的执行步骤函数 T:N->N：
  - 数据规模->算法执行步骤数

- 该函数：
  - 每个算法均有最佳情况、最差情况和平均情况下的函数
  - 基本代表一个算法的执行效率
  - 随着数据规模变化，可以考察该函数的“增长”速度

算法的效率分析-时间开销

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

对于每个待插入元素，插入已有序子序列时情况不一：

有不同时的比较次数、因插入而导致的子序列移动也不一

定义t_j为第j个插入数据所进行的比较次数

算法的最差性能：

最差性能：待排序元素完全逆序！

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

and

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

$$\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8) .
\end{aligned}$$

函数的增长-算法分析初步

- 集合A上的关系R，令|A| = n， |R| = $n^2/2$
- 求该关系的传递闭包算法有：S1算法,S2算法
- 如何去判断哪个算法更好一些？
  - 时间开销: steps
    - T_{S1}函数；T_{S2}函数
  - 如何比较时间开销？
    - 看谁“长得快”！

函数增长

N (数据集规模)	S1 (算法执行步数)	S2 (算法执行步数)
10	550	1250
50	63750	781250
100	505000	12500000

两个算法执行步数随着数据规模的变化而变化
不同的算法，变化的“剧烈程度”不同

需要一种数学工具通过执行步骤函数的处理来反映上述“剧烈程度”

函数的增长

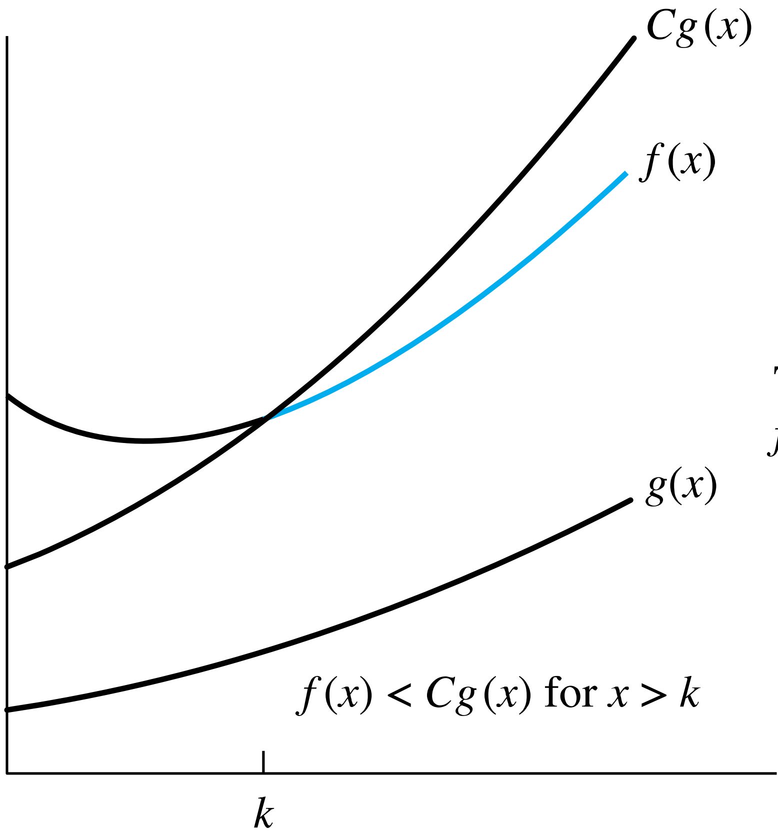
- 定义函数T:N->N：
  - 数据规模->算法执行步骤数
- 针对上述两个算法：
  - $T_{S1}(n) = n^3/2 + n^2/2$ for algorithm S1
  - $T_{S2}(n) = n^4/8$ for algorithm S2

哪个好一些？

函数的增长速度

- 给定f:$N \rightarrow N$, g:$N \rightarrow N$,（注：通常$N \rightarrow R$）
  - 如果存在常数 $C \in N$ 和 $k \in N$ 使得对于所有大于等于k的n，都有 $f(n) \leq C \times g(n)$
  - 我们称：
    - f is $O(g)$ $f \in O(g)$
    - f 增长速度不高于g

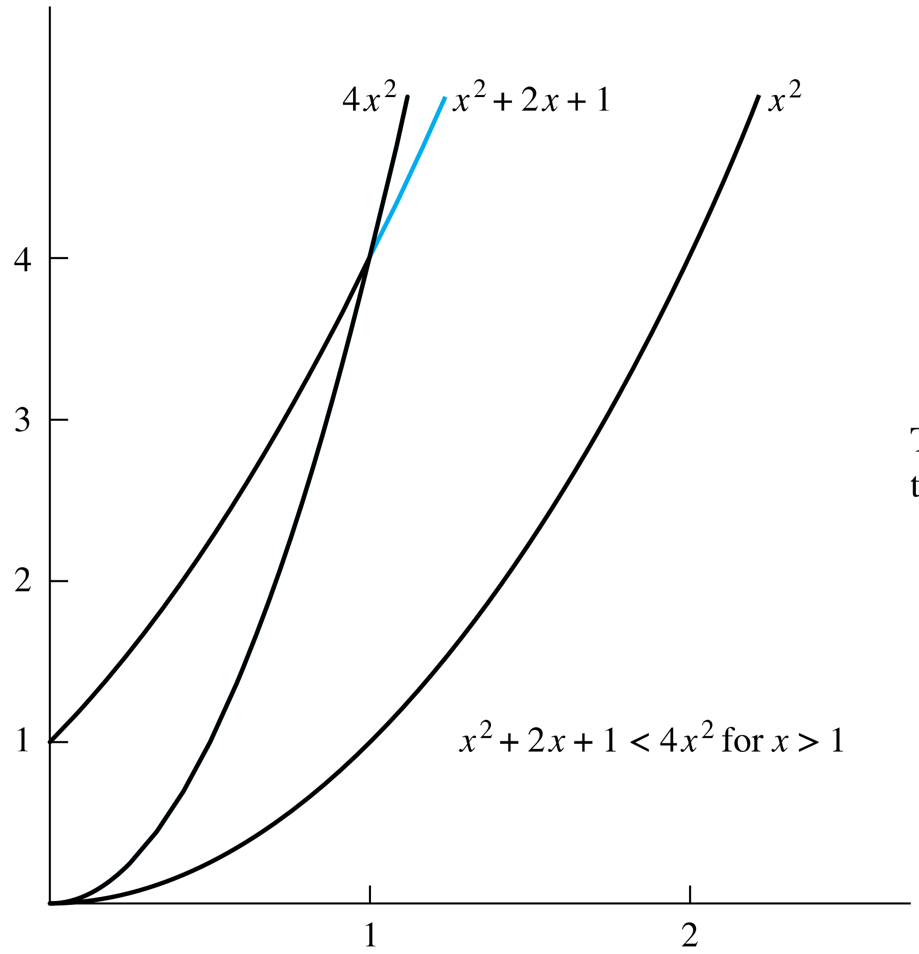
Big-O notation的含义



The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

FIGURE 2 **The Function $f(x)$ is $O(g(x))$.**

例子： $x^2 + 2x + 1$ is $O(x^2)$



The part of the graph of $f(x) = x^2 + 2x + 1$ that satisfies $f(x) < 4x^2$ is shown in blue.

实际上：

- 可以做如下判断：
  - 函数 f 是$O(g)$ if $\lim_{n\to\infty}[f(n)/g(n)]=C < \infty$
  - if there exists constants $C \in N$ and $k \in N$ such that for all $n \geq k$, $f(n) \leq Cg(n)$
- 例如: let $f(n)=n^2$, $g(n)=n\lg n$，则:
  - f 不是$O(g)$, 因为$\lim_{n\to\infty}[f(n)/g(n)]= \lim_{n\to\infty}[n^2/n\lg n]= \lim_{n\to\infty}[n/\lg n]= \lim_{n\to\infty}[1/(1/n\ln 2)]=\infty$
  - g 是$O(f)$, 因为$\lim_{n\to\infty}[g(n)/f(n)]=0$

再例：

- let $f(n)=n^2$, $g(n)=7n^2+9n-1$
  - $\lim_{n\to\infty}[f(n)/g(n)] = \lim_{n\to\infty}[n^2/(7n^2+9n-1)] = 1/7$
  - 所以： f 是$O(g)$

  - $\lim_{n\to\infty}[g(n)/f(n)] = \lim_{n\to\infty}[(7n^2+9n-1)/n^2] = 7$
  - 所以： g 是$O(f)$

- 我们称：f和g长得一样快(同阶)

Θ 关系

- $n^2/100+5n$ 是 $O(3n^4-5n^2)$, 它是$O(10n^4)$?
- $3n^4-5n^2$ 和$10n^4$ 长得一样快
- 实际上，n^4 是所有和$3n^4-5n^2$同阶的函数中的最简形式
- 定义N to R+函数集合上的关系Θ：
  - $f\ \Theta g$ iff f 和g同阶
  - $3n^4-5n^2\ \Theta\ 10n^4$, $(n^4, 3n^4-5n^2) \in \Theta$
- 定理: Θ 是等价关系

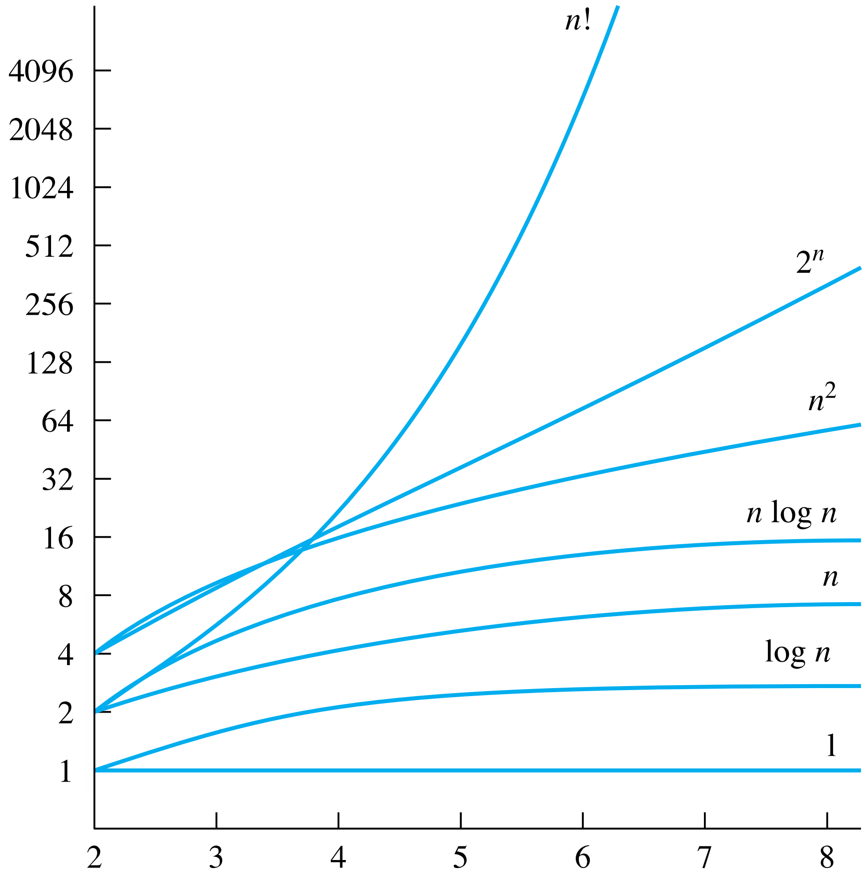
常见阶

- Θ等价类:
  - Let A: {f | f:N->R+}, **let $s \in A/\Theta$**
  - $\forall$ **f, g** $\in$ **s, f is O(g) and g** is O(f)
- 一些常见的代表性阶:
  - $\Theta(1)$, $\Theta(n)$, $\Theta(n^2)$, $\Theta(n^3)$, $\Theta(\lg(n))$, $\Theta(n\lg(n))$, and $\Theta(2^n)$

范例:

- 从低到高重新排列一下阶：
- $\Theta(1000n^2\text{-}n)$, $\Theta(n^{0.2})$, $\Theta(1000000)$, $\Theta(1.3^n)$, $\Theta(n+10^7)$,$\Theta(n\lg(n))$
  - $\Theta(1000000)$
  - $\Theta(n^{0.2})$
  - $\Theta(n+10^7)$
  - $\Theta(n\lg(n))$
  - $\Theta(1000n^2\text{-}n)$
  - $\Theta(1.3^n)$,

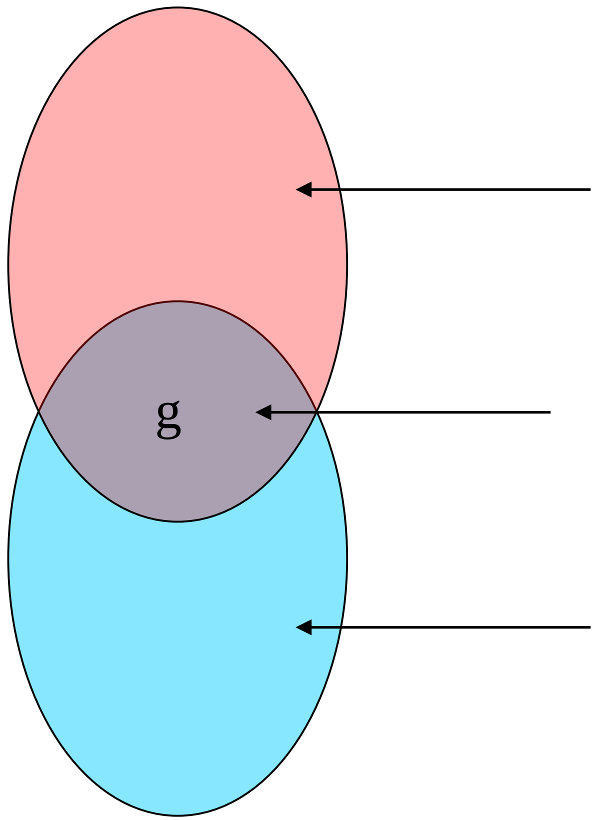
函数增长速度



注意纵坐标乃是对数刻度

相对增长速度

给定函数g:



Ω(g):functions that grow at least as fast as g

Θ(g):functions that grow at the same rate as g

O(g):functions that grow no faster as g

教材和练习

- 练习：
  - 第六版
    - P349：24；36
    - P360：4(a,b,c,d)
  - 第七版
    - P433：8；20
    - P442：4(a,b,c,d)