

分而治之的艺术

# 递归与分治算法

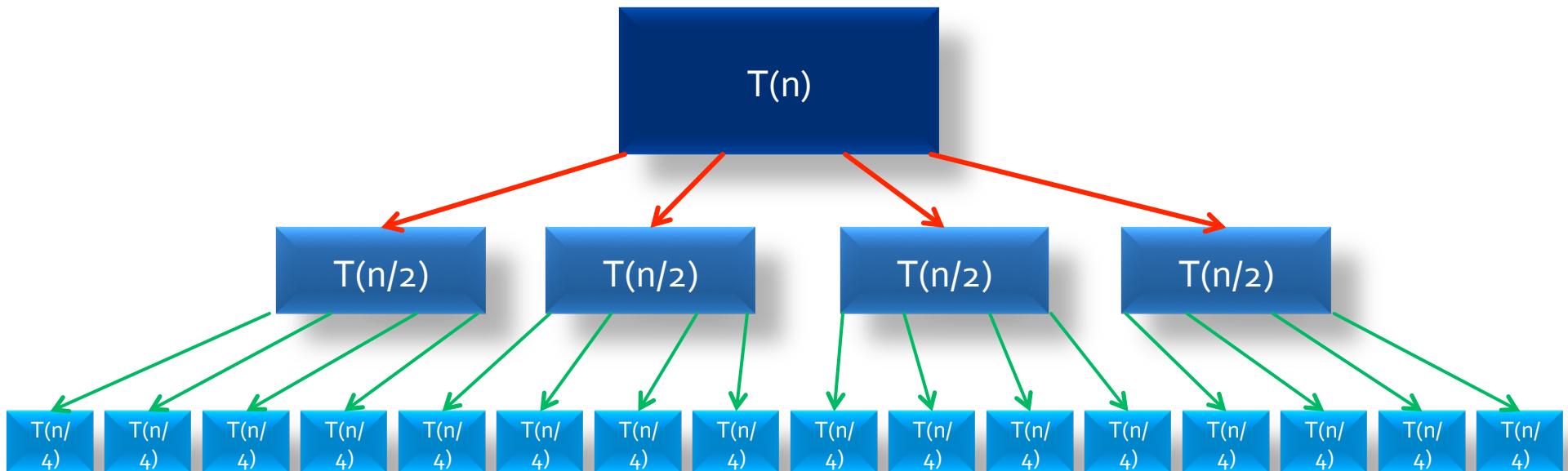
# 分治——分而治之



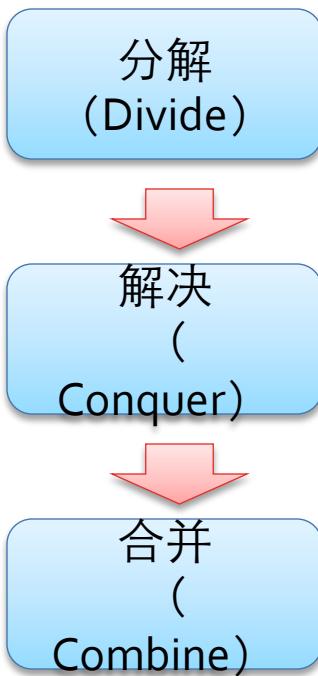
# 基本思想

将原问题划分成 $m$ 个规模较小，而结构与原问题相似的子问题，递归的解决这些子问题，然后再合并其结果，就得到原问题的解。

所谓递归就是对子问题也要进行划分，直到子问题容易的求解



# 基本步骤



将原问题分解为规模更小的子问题



如果问题足够小，可以直接解决；否则调用相同的方法求解



将子问题的结果合并为原问题的解



# 例子：二分查找

- 问题: 给定已按升序排好序的n个元素  $a[0:n-1]$ , 现要在这n个元素中查找一特定元素x
- 分析:
  - 当数组小到一个元素的时候就可以判断
  - 可以选取数组中的中间位置的值  $a[n/2]$ , 如果
    - $X < a[n/2]$ , 则在  $a[0 - (n/2-1)]$  中查找
    - $X > a[n/2]$ , 则在  $a[(n/2+1) - n-1]$  中查找
    - $X = a[n/2]$ , 已找到, 返回
  - 不需要最后的合并步骤

# 能否解决的问题

该问题可以分解为若干个规模较小的相同问题

该问题的规模缩小到一定的程度就可以容易地解决

利用该问题分解出的子问题的解可以合并为该问题的解

子问题独立性

该问题所分解出的各个子问题是相互独立的  
即子问题之间不包含公共的子问题

分解  
(Divide)

解决  
(  
Conquer)

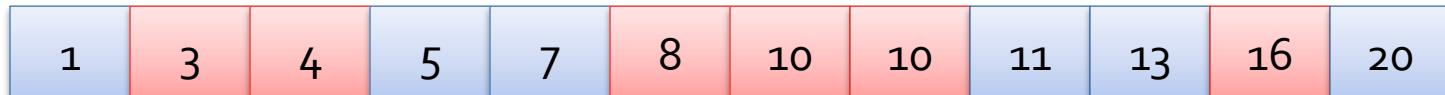
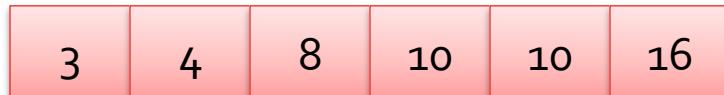
合并  
(  
Combine)

# 问题：归并排序

- 问题：对一列数字进行排序
- 分析
  - 分解：可以把一列数字分成多段进行排序
  - 解决：当每个序列只剩一个元素的时候自然有序
  - 合并：对于已经有序的两个序列，如何进行合并
  - 子问题独立性：每个问题的子问题是相互独立的
- 基本步骤
  - 分解：将 $n$ 个元素分成各含 $n/2$ 个元素的子序列
  - 解决：对每个子序列进行归并排序（当每个子列中只有一个元素时，那么这个序列已经是有序了）
  - 合并：合并两个已排序的子序列得到排列结果

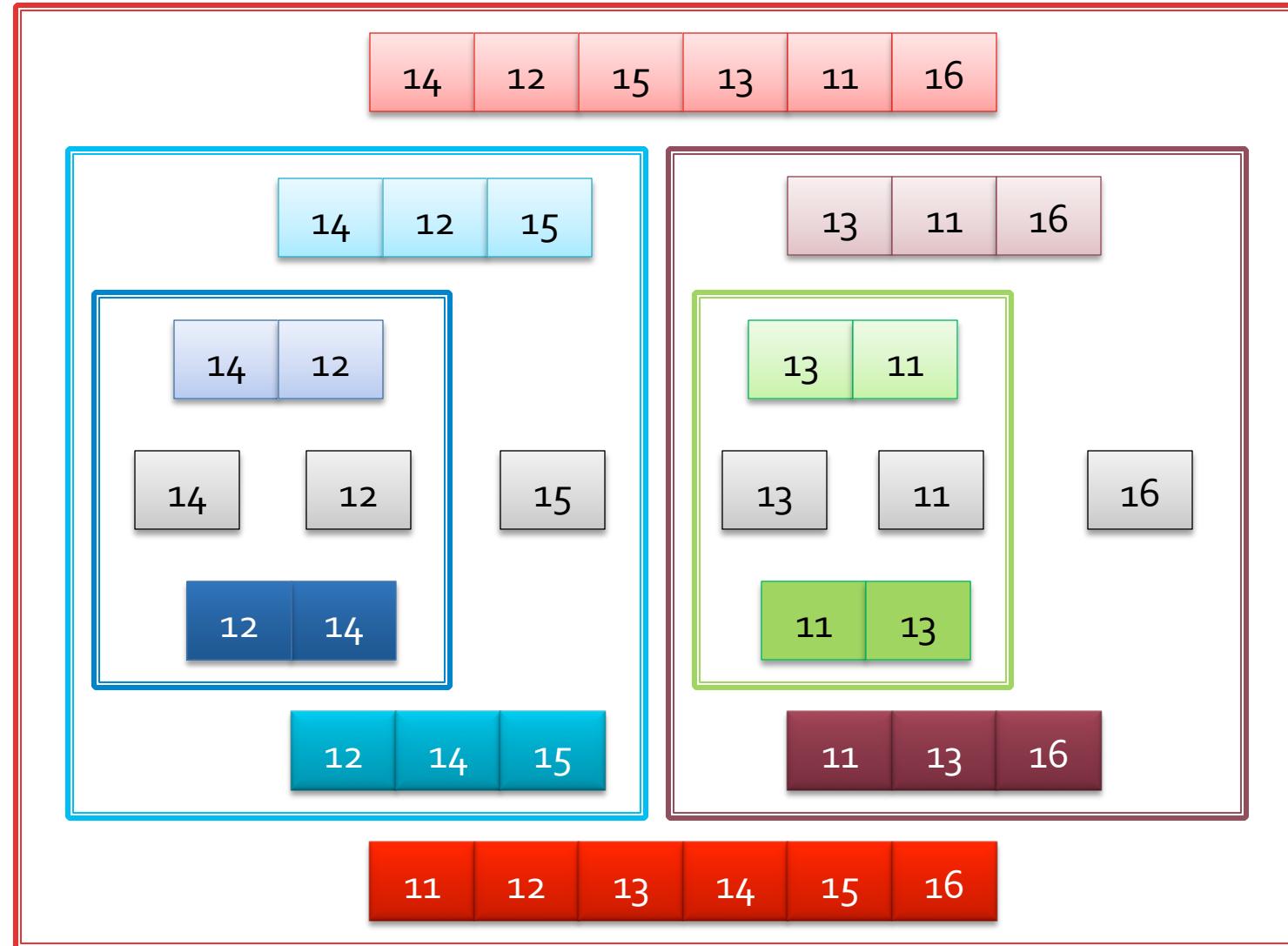
# 归并两个有序数列

将两个有序的数列合并为一个有序的数列



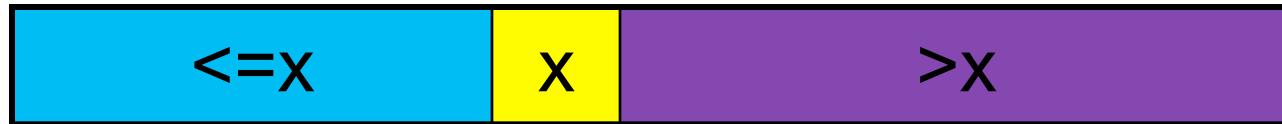
时间复杂度 $O(n)$

# 递归过程



# 问题：快速排序(QUICKSORT)

- 基本步骤
  - 分解：任取一个元素 $x$ ，将数组分为小于 $x$ 的部分和大于等于 $x$ 的部分
  - 解决：对每个子序列应用快速排序算法
  - 合并：无需合并，自然有序

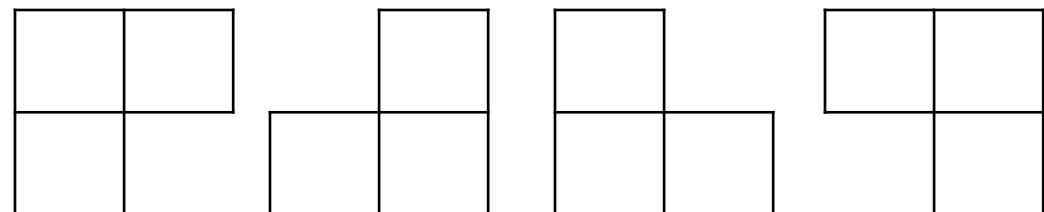
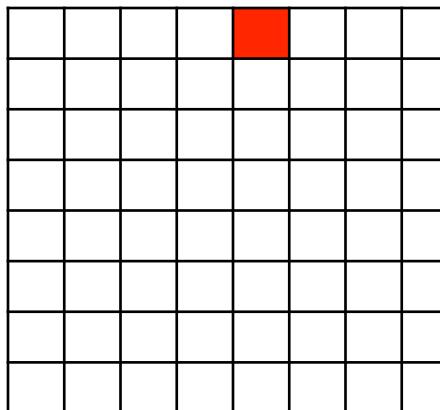


# 两种方法比较

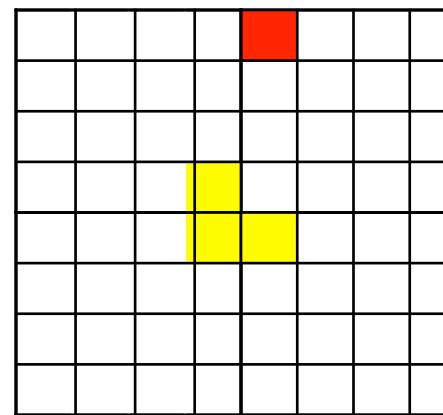
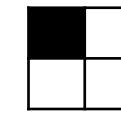
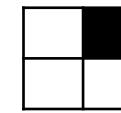
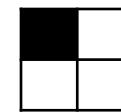
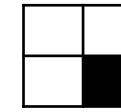
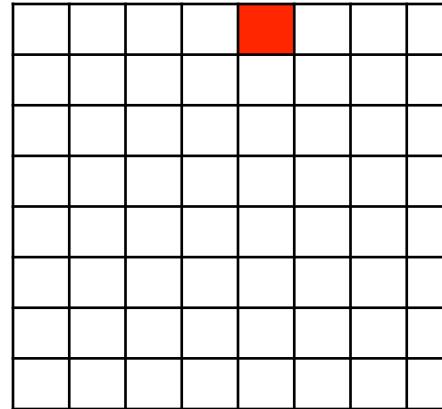


# 问题：棋盘覆盖

- 在一个 $2^k \times 2^k$ 个方格组成的棋盘中，恰有一个方格与其它方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。



$2^{k-1}X2^{k-1}$	$2^{k-1}X2^{k-1}$
$2^{k-1}X2^{k-1}$	$2^{k-1}X2^{k-1}$



# 再来说一下排序

冒泡排序：  $O(n^2)$

插入排序：  $O(n^2)$

快速排序：  $O(n \lg n)$

归并排序：  $O(n \lg n)$

Name ↗	Average ↗	Worst ↗	Memory ↗	Stable ↗
Binary tree sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	Yes
Bubble sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes
Cocktail sort	—	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes
Comb sort	—	—	$\mathcal{O}(1)$	No
Gnome sort	—	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes
Heapsort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$	No
In-place merge sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$	No
Insertion sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes
Introsort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(\log n)$	No
Library sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	Yes
Merge sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	Yes
Smoothsort	—	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$	No
Strand sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	Yes

使用分治思想设计的排序  
算法的时间复杂度降低了

# 时间复杂度的意义？

$\pi = 3.14159265358979323846264338327950288419716939937510\dots$

$e = 2.71828182845904523536028747135266249775724709369995\dots$

$$\begin{matrix} 2^n \\ e^{\pi} \end{matrix}$$

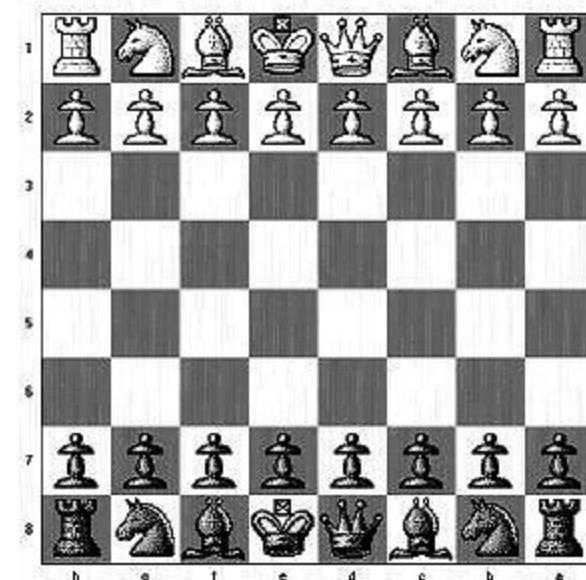
VS  
VS

$$\begin{matrix} n^2 \\ \pi^e \end{matrix}$$

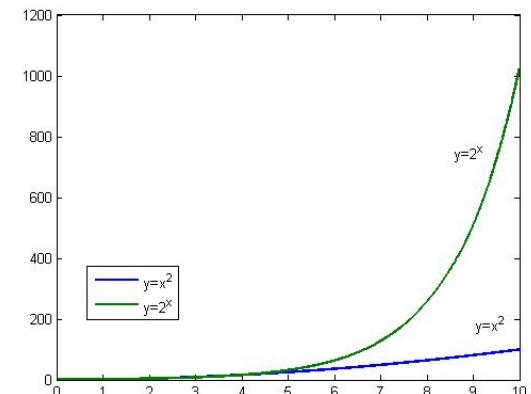
18446744073709551616

$1.844674407 \times 10^{19}$

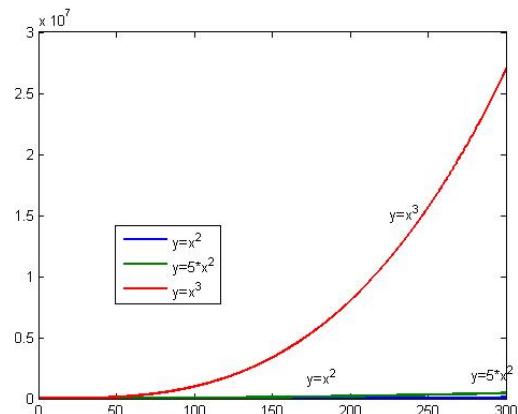
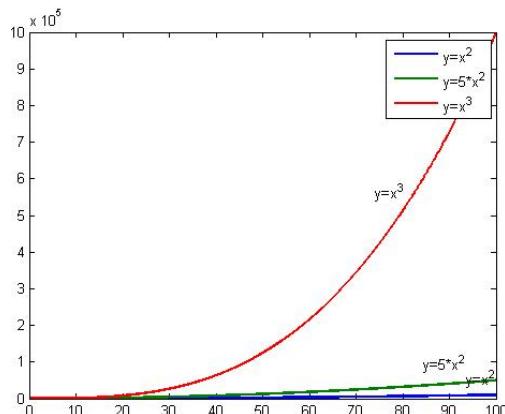
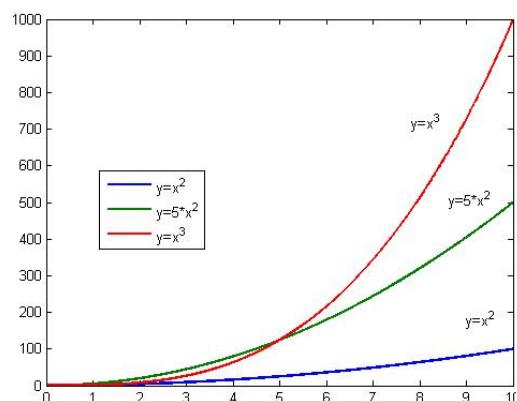
$$2^{64\dagger}$$



	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
$x^2$	4	9	16	25	36	49	64	81	100
$2^x$	4	8	16	32	64	128	256	512	1024



	<b>2</b>	<b>5</b>	<b>10</b>
$x^2$	4	25	100
$5x^2$	20	125	500
$x^3$	8	125	$10^3$



时间复杂度反应的是随着问题规模的增长，计算量增长的速度的快慢

有前面的讨论可以发现，之后次数最高的项才能真正体现速度的快慢，非最高项以及所有的系数在反应增长速度方面意义都不大

# 算法计算复杂度的度量

$f(N)$ 和 $g(N)$ 是定义在正数集上的正函数

如果存在  
正常数C和自然数 $N_0$

当 $N > N_0$ 时，  
有 $f(N) \leq Cg(N)$

当 $N > N_0$ 时，  
有 $f(N) \geq Cg(N)$

$g(N)$ 是 $f(N)$ 的上界

$g(N)$ 是 $f(N)$ 的下界

记做  
 $f(N) = O(g(N))$

记做  
 $f(N) = \Omega(g(N))$

$$f(N) = O(g(N))$$



$$f(N) = \Theta(g(N))$$

$$f(N) = \Omega(g(N))$$

# 举例

- $3n = O(n)$
- $n+1024 = O(n)$
- $2n^2+11n+5 = O(n^2)$
- $n^2 = O(n^3)$
- $3n = \Omega(n)$
- $2n^2+11n+5 = \Omega(n^2)$
- $n^3 = \Omega(n^2)$
- $2n^2+11n+5 = \Theta(n^2)$
- $3n = \Theta(n)$

# 分治算法的时间复杂度

把原规模为n的问题分解为a个规模为n/b子问题， $T(n)$ 代表求解规模为n的问题所需的时间， $f(n)$ 代表分解和合并所需的时间为

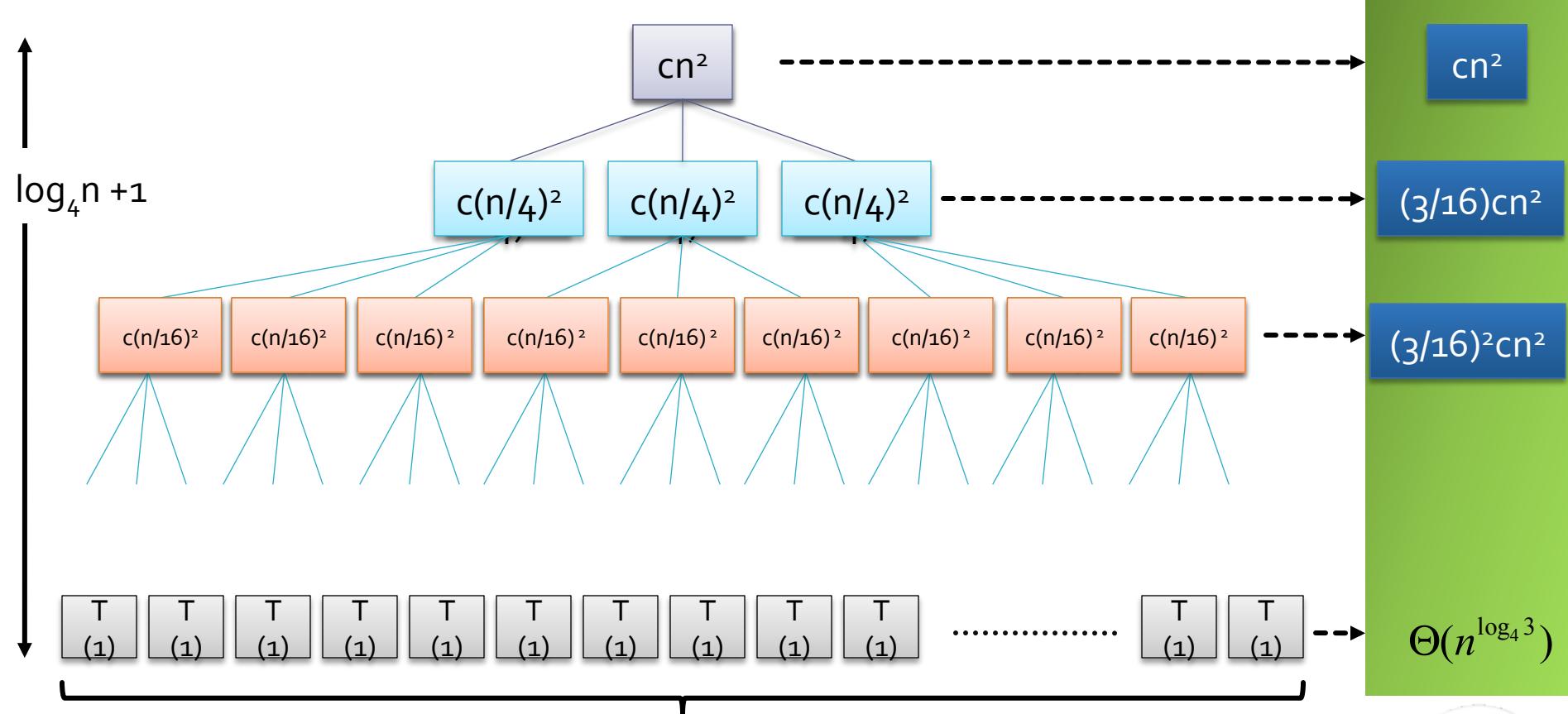
$$T(n) = aT(n/b) + f(n)$$

如何计算  
 $T(n)$

归并排序：  $T(n) = 2T(n/2) + cn$

# 递归树法

$$T(n) = 3T(n/4) + cn^2$$



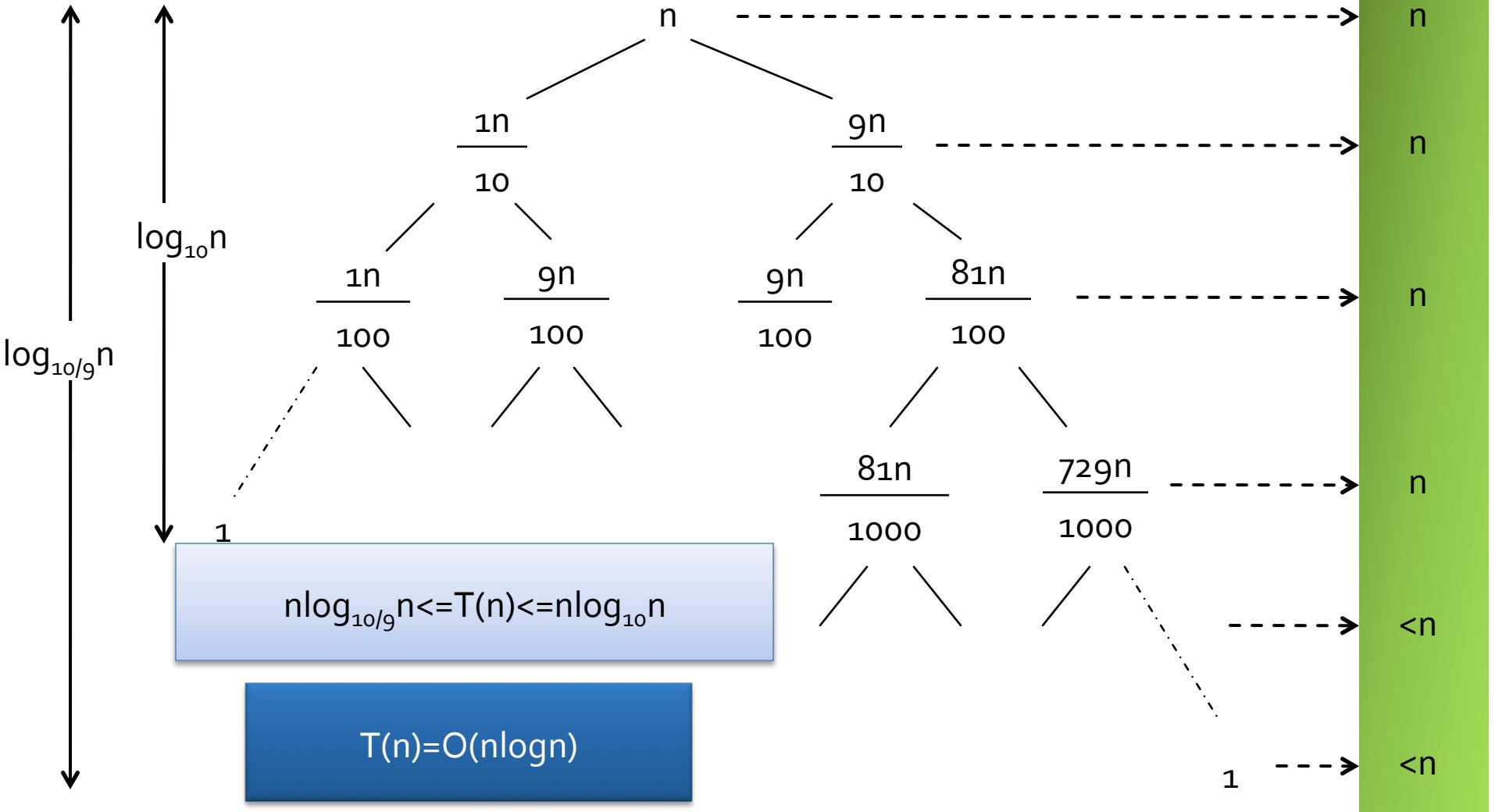
$$3^{\log_4 n} = n^{\log_4 3}$$



$$\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \dots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1 - \frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}$$

# 分析实例

- 用递归树法分析归并排序的时间复杂度
- 假设一个数列用快速排序，每次都能将其分解为10:1的两段，试用递归树法分析在此种情况下算法的时间复杂度



# 替换解法——数学归纳法

$$T(n) = 2T(n/2) + cn$$

我们对结果有一个猜想  $T(n)=O(n \lg n)$

假设  $T(m) \leq cm \lg m$   $m < n$   
要证  $T(n) \leq cn \lg n$

$$\begin{aligned} T(n) &= 2T(n/2) + cn \\ &< 2c(n/2) \lg(n/2) + cn \\ &= cn(\lg n - \lg 2) + cn \\ &= cn \lg n + (1 - \lg 2)cn \\ &< cn \lg n \end{aligned}$$



我们对结果有一个猜想  $T(n)=O(n \lg_2 n)$

假设  $T(m) \leq cm \lg_2 m$   $m < n$   
要证  $T(n) \leq cn \lg_2 n$

$$\begin{aligned} T(n) &= 2T(n/2) + cn \\ &< 2c(n/2) \lg_2(n/2) + cn \\ &= cn(\lg_2 n - \lg_2 2) + cn \\ &= cn \lg_2 n + (1 - 1)cn \\ &= cn \lg_2 n \end{aligned}$$



# 主方法

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

**CASE 1:**  $f(n) = O(n^{\log_b a - \varepsilon})$ , constant  $\varepsilon > 0$   
 $\Rightarrow T(n) = \Theta(n^{\log_b a})$ .

**CASE 2:**  $f(n) = \Theta(n^{\log_b a} \lg^k n)$ , constant  $k \geq 0$   
 $\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .

**CASE 3:**  $f(n) = \Omega(n^{\log_b a + \varepsilon})$ , constant  $\varepsilon > 0$ ,  
and regularity condition  
 $\Rightarrow T(n) = \Theta(f(n))$ .

# 应用主定理

将 $f(n)$ 与 $n^{\log_b a}$ 相比

**CASE 2:**  $f(n) = \Theta(n^{\log_b a} \lg^k n)$ , constant  $k \geq 0$   
 $\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .

归并排序: $T(n)=2T(n/2)+cn$

$a=2, b=2, \lg_b a=1$ , 相当于  
case2中 $k=0$

$T(n)=\Theta(n \lg n)$

二分搜索: $T(n)=T(n/2)+c$

$a=1, b=2, \lg_b a=0$ , 相当于  
case2中 $k=0$

$T(n)=\Theta(\lg n)$

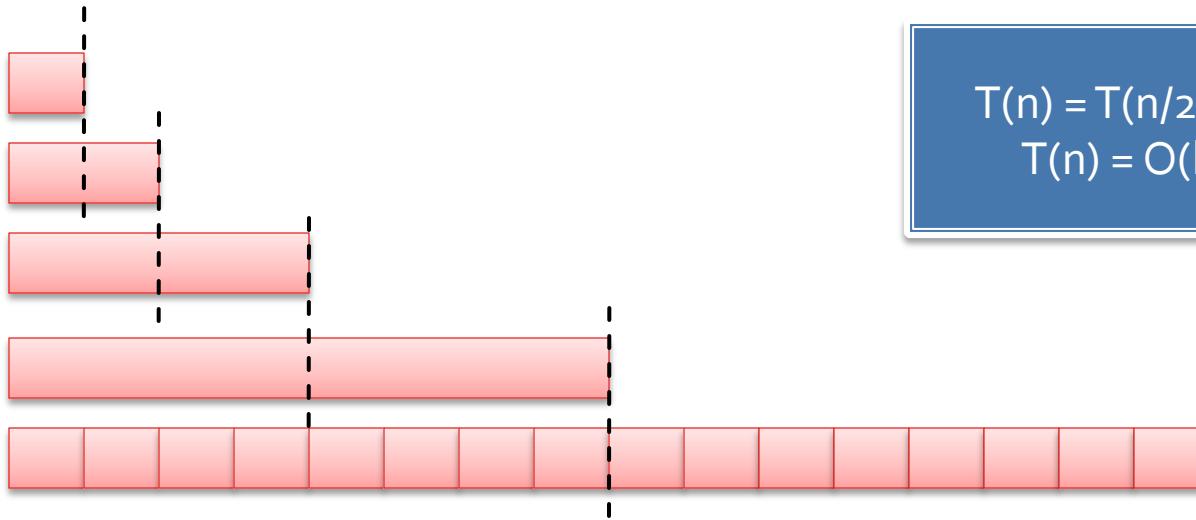
# 幂次计算

计算  $a^n$

$a*a*a*a*a*.....*a$

时间复杂度  $O(n)$

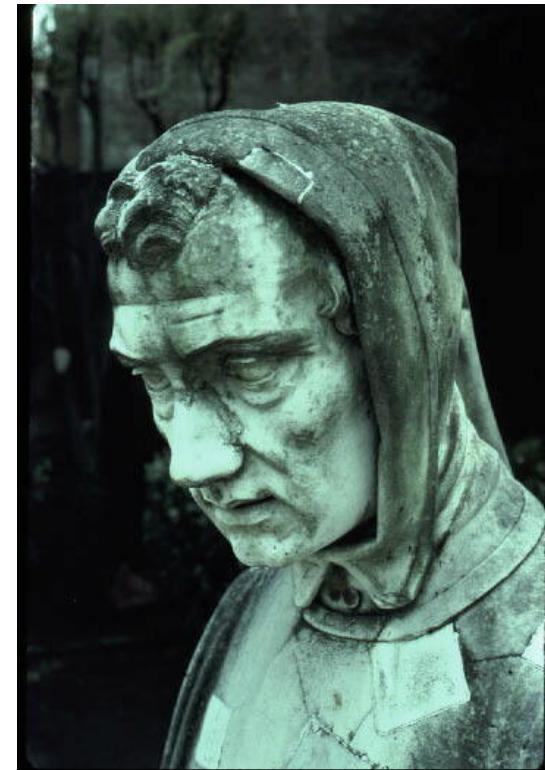
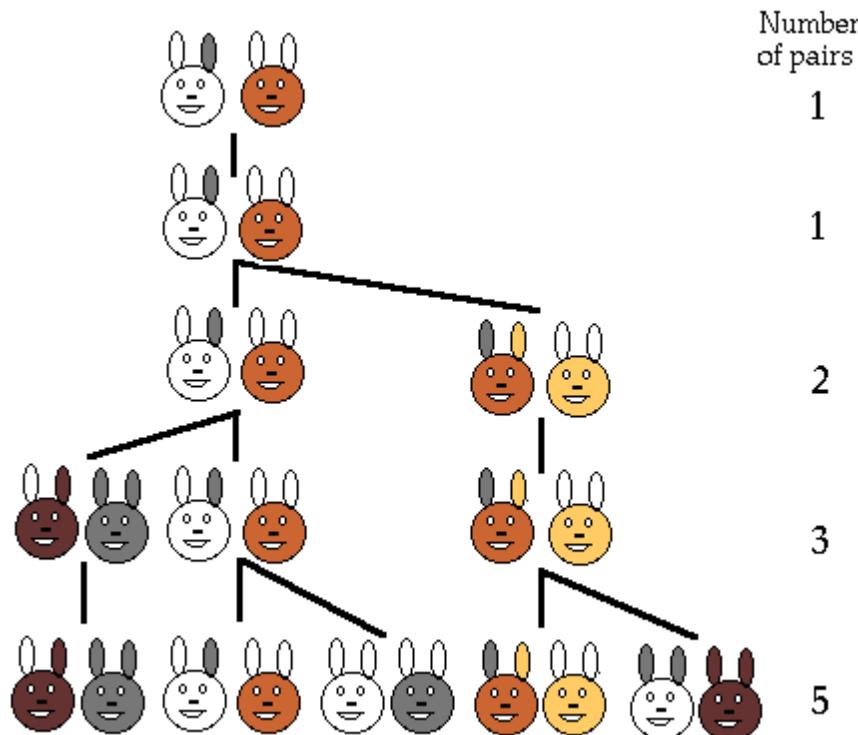
$$a^n = \begin{cases} a^{n/2}a^{n/2} & n \text{为偶数} \\ a^{(n-1)/2}a^{(n-1)/2}a & n \text{为奇数} \end{cases}$$



$$\begin{aligned} T(n) &= T(n/2) + O(1) \\ T(n) &= O(\log n) \end{aligned}$$

# 斐波那契 (Fibonacci) 数列

《珠算原理》：某人把一对兔子放入一个四面被高墙围住的地方。假设每对兔子每月能生下一对小兔，而每对新生小兔从第二个月开始又具备生育能力，请问：一年后应有多少对兔子

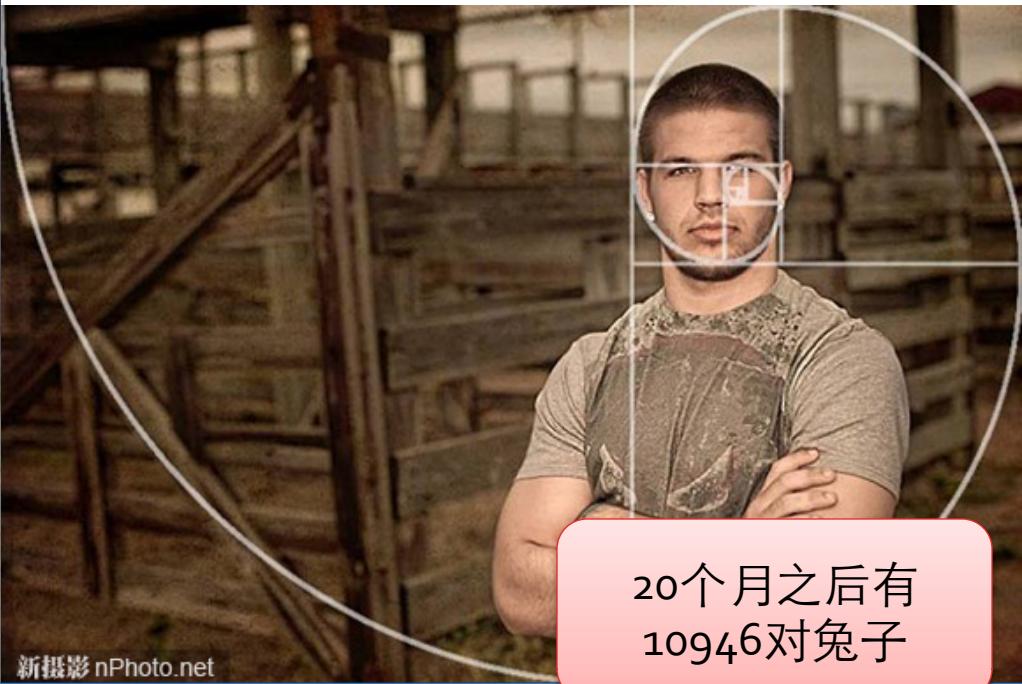
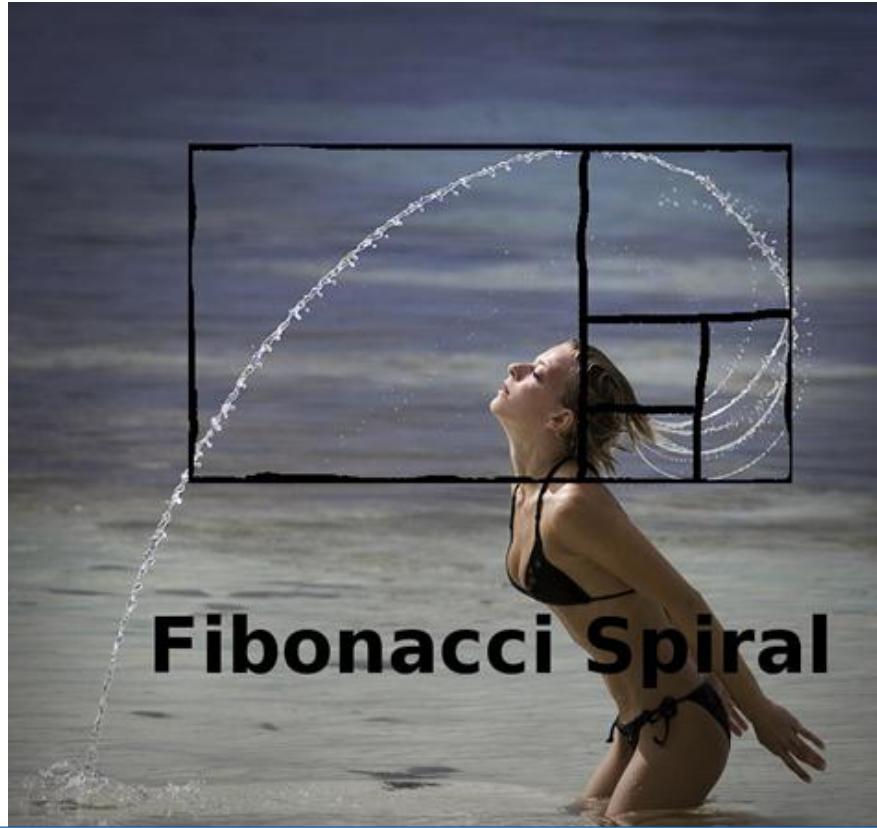
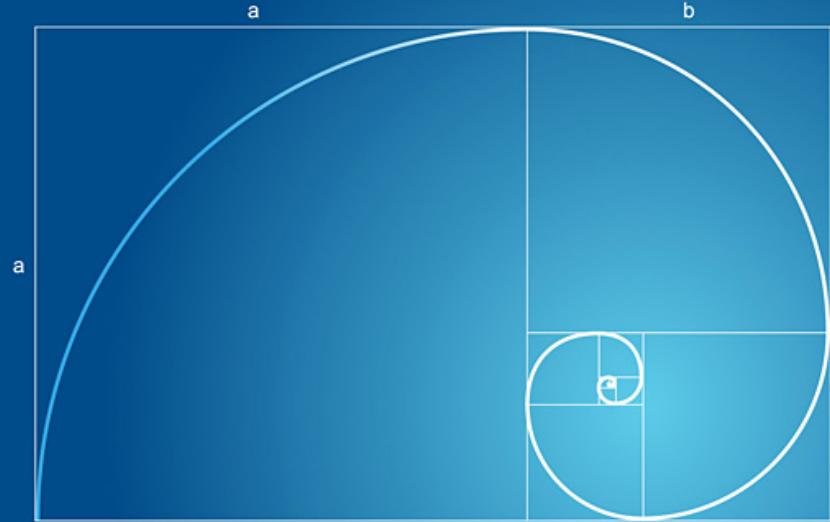


列昂纳多·斐波那契  
意大利数学家

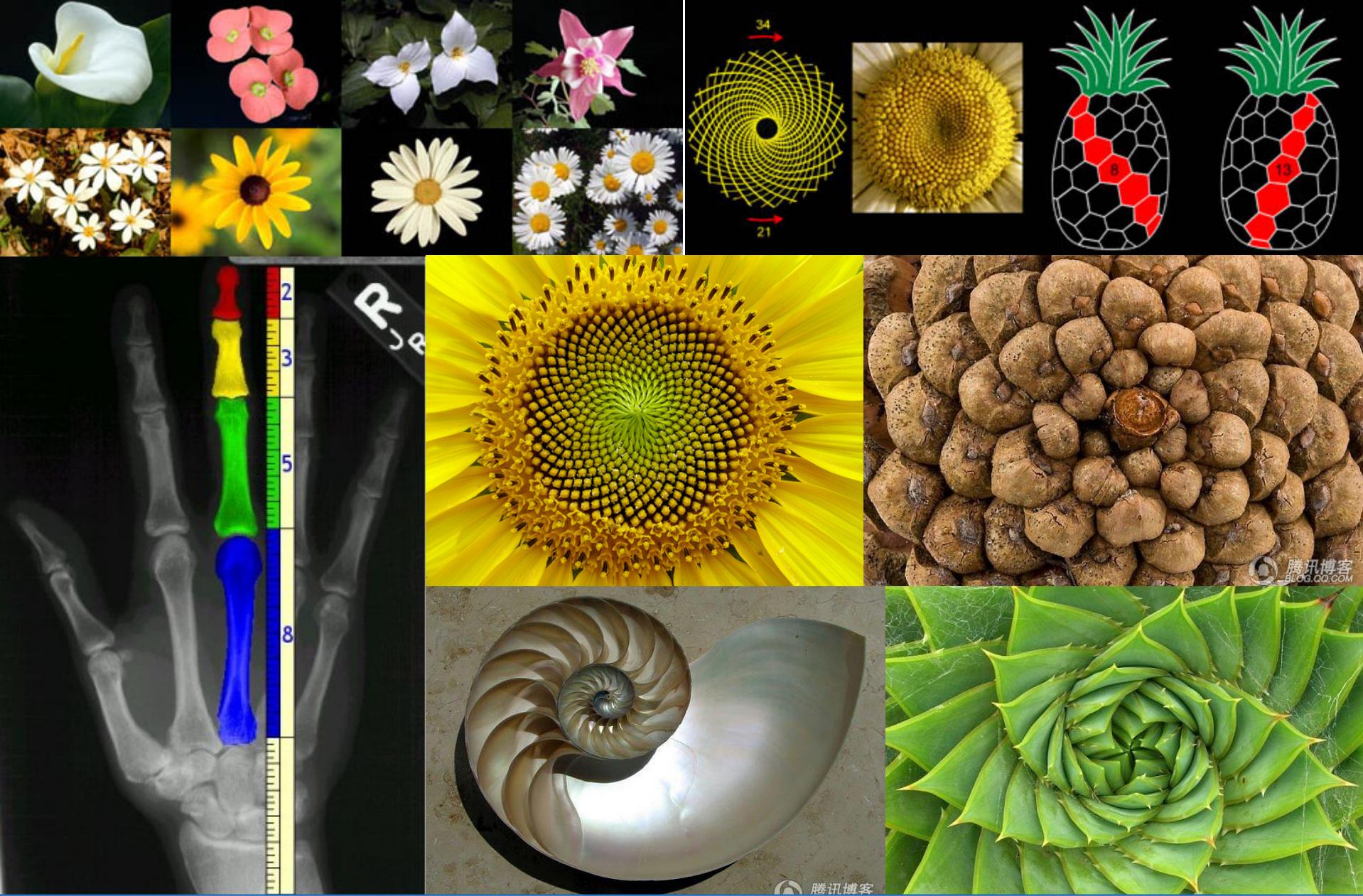
$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$



1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, ....



1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946,.....

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946,.....

1, 0.5, 0.67, 0.6, 0.625, 0.615, 0.619, 0.6176, 0.61818, 0.625, 0.6181, 0.61802, 0.61803,.....

$$\frac{a}{b} = \frac{b}{a+b} \quad \frac{b}{a} = \frac{\sqrt{5}+1}{2} \text{ 或者 } \frac{a}{b} = \frac{\sqrt{5}-1}{2}$$

0.6180339887 4989484820 4586834365 6381177203 0917980576 2862135448 6227052604  
6281890244 9707207204 1893911374 8475408807 5386891752 1266338622 2353693179  
3180060766 7263544333 8908659593 9582905638 3226613199 2829026788 0675208766  
8925017116 9620703222 1043216269 5486262963 1361443814 9758701220 3408058879  
5445474924 6185695364 8644492410 4432077134 4947049565 8467885098 7433944221  
2544877066 4780915884 6074998871 2400765217 0575179788 3416625624 9407589069  
7040002812 1042762177 1117778053 1531714101 1704666599 1466979873 1761356006  
7087480710 1317952368 9427521948 4353056783 0022878569 9782977834 7845878228  
9110976250 0302696156 1700250464 3382437764 8610283831 2683303724 2926752631.....

# 问题：如何计算它的第1M项

$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

依次计算

$F_0, F_1, F_2, F_3, F_4, F_5, \dots, F_{1000000}$

时间复杂度  $O(n)$

$$(F_{n+2}, F_{n+1}) = (F_{n+1}, F_n) \times A, \quad \text{其中 } A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{aligned} (F_{n+2}, F_{n+1}) &= (F_{n+1}, F_n) \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = (F_n, F_{n-1}) \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \\ &= \dots = (F_1, F_0) \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1} \end{aligned}$$

求矩阵的  $n$  次方，矩阵相乘满足结合律，解决方法与整数  $n$  次方类似

时间复杂度  $O(\lg n)$

# 问题

新华社东京2009年8月18日电（记者钱铮）日本筑波大学17日宣布，筑波大学研究人员借助最新的超级计算机系统，将圆周率计算到小数点后 $25769.8037$ 亿位，打破了东京大学和日立制作所于2002年创下的小数点后12411亿位的世界纪录。

如何在计算过程中存储这些数值呢？

如何在计算中存储任意位的整数呢？

如何进行任意位整数的加减乘除运算呢？

C语言中，int的范围是 $-2^{31} \sim 2^{31}-1$

可以表示的整数位数不超过10位

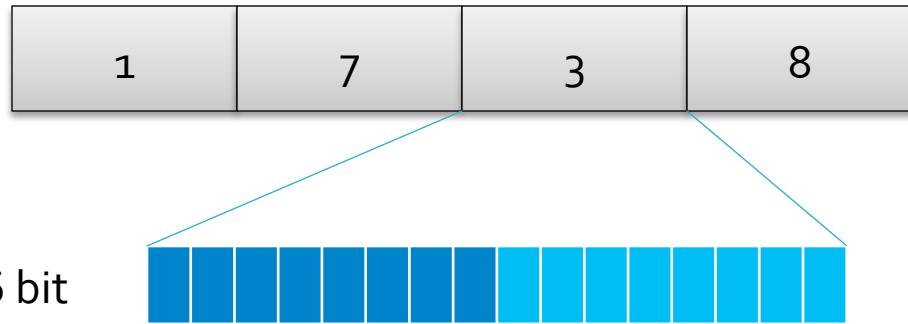


# 直接方法

用短整形数组存放，每个short int存放1个整数

short int a[4]

2 byte = 16 bit



元素之间的运算容易实现。

如何将整数输入到这个数组中。

scanf

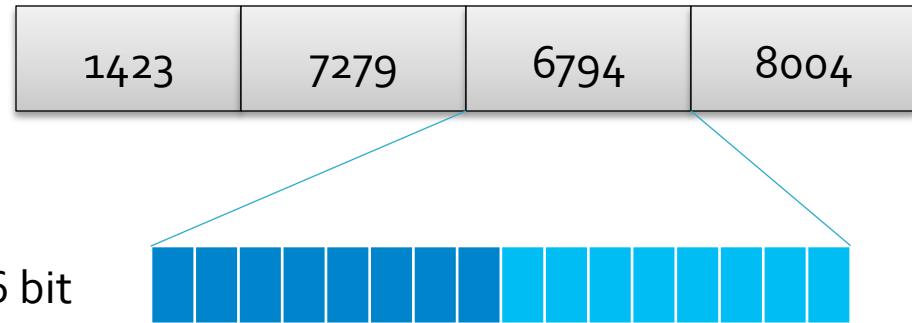
short int的范围是-32768~32767，有很大的浪费

只需要4个bit就可以表示一个十进制数

# 节省空间

用整形数组存放，每个  
short int存放4个整数

short int a[4]



2 byte = 16 bit

元素之间的运算容易实现。

如何将整数输入到这个数组中。

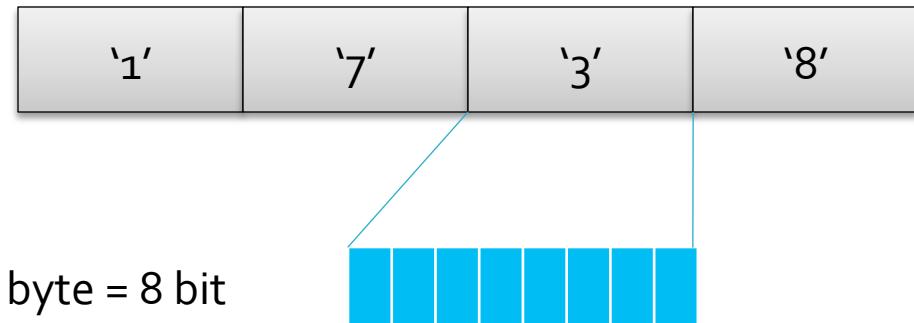
scanf

short int的范围是-32768~32767，用来保存4位整数

# 方便输入

用字符型数组存放，每个char存放1个整数字符

char a[4]



元素之间的运算需要另外的转化

方便作为字符串输入到数组中

scanf("%s",&a); a="1738";

空间使用率介于两者之间

short int a[4]	1	7	3	8
short int a[4]	1423	7279	6794	8004

# 大整数表示（空间最省）

1 byte



范围0 ~ 255

把整数看作是 $256$  ( $2^8$ ) 进制，一个byte代表这个整数的一位

4 byte 大整数



$$124 * 2^{32} + 212 * 2^{16} + 117 * 2^8 + 61$$

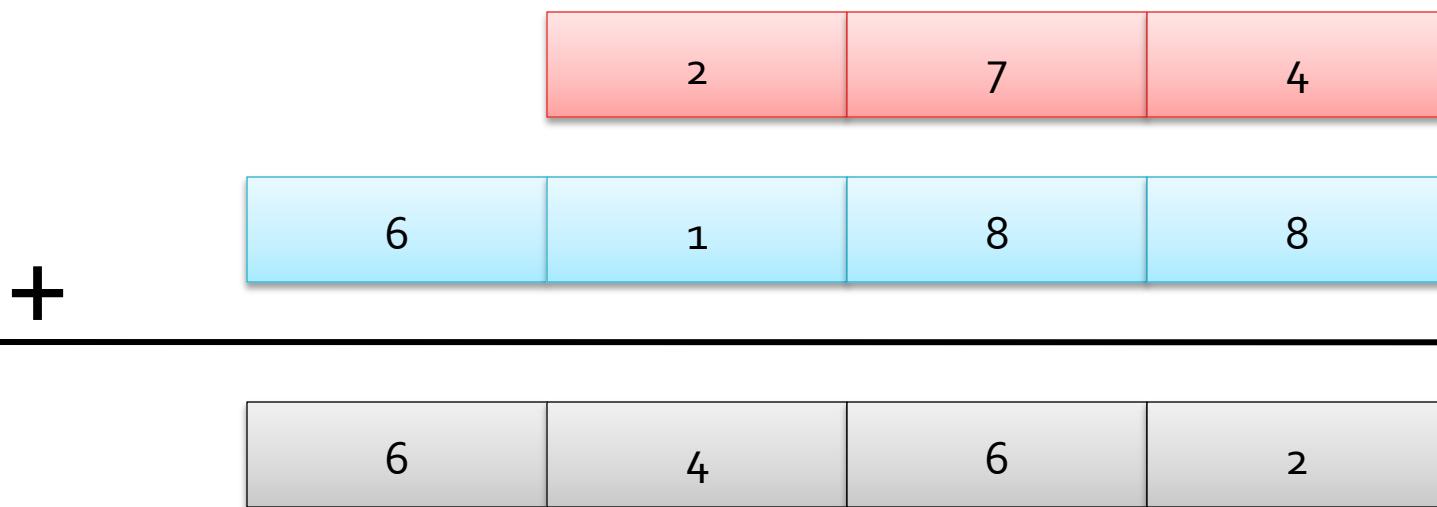
元素之间的运算需要另外的转化

字符串输入需要转化

空间最省

# 大整数加法

数组对应的位置分别作加法



时间复杂度  $O(n)$

# 大整数乘法

$$\begin{array}{r} & \boxed{3} & \boxed{4} \\ \times & \boxed{7} & \boxed{2} & \boxed{9} \\ \hline \end{array}$$

$$\begin{array}{r} \boxed{3} & \boxed{0} & \boxed{6} \end{array}$$

$$\begin{array}{r} \boxed{6} & \boxed{8} \end{array}$$

$$\begin{array}{r} \boxed{2} & \boxed{3} & \boxed{8} \end{array}$$

$$\begin{array}{r} \boxed{2} & \boxed{4} & \boxed{7} & \boxed{8} & \boxed{6} \end{array}$$

时间复杂度  $O(n^2)$