分而治之的艺术

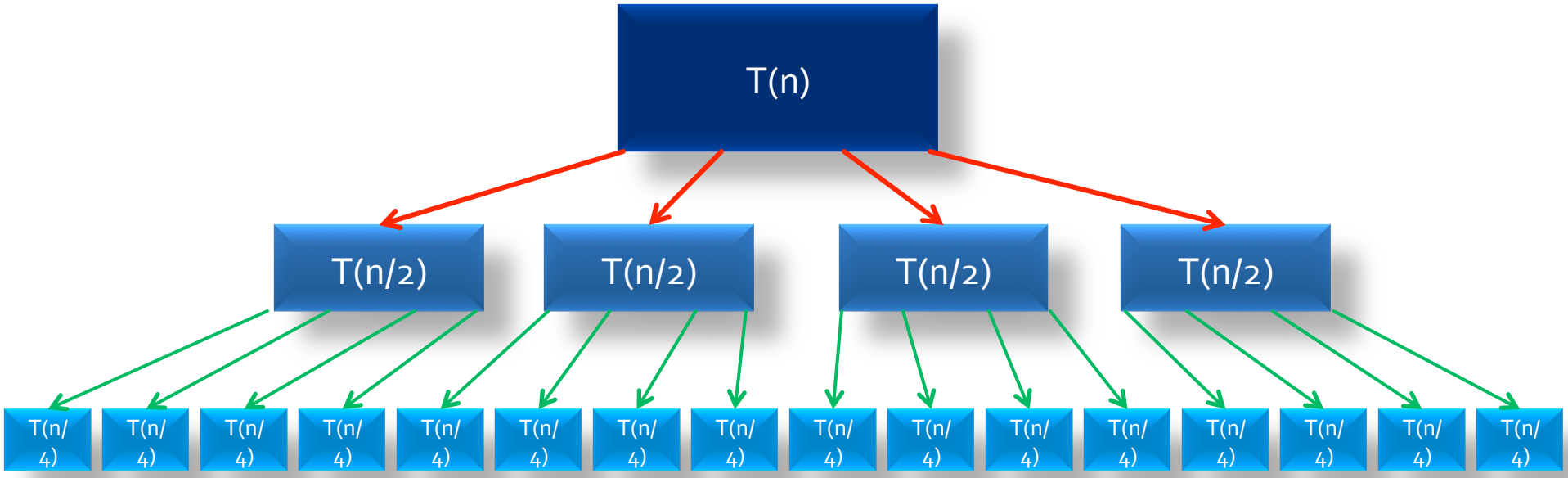
递归与分治算法

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

分解（Divide）

将原问题分解为规模更小的子问题

解决（Conquer）

如果问题足够小，可以直接解决；否则调用相同的方法求解

合并（Combine）

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

分解（Divide）

解决（Conquer）

合并（Combine）

子问题独立性

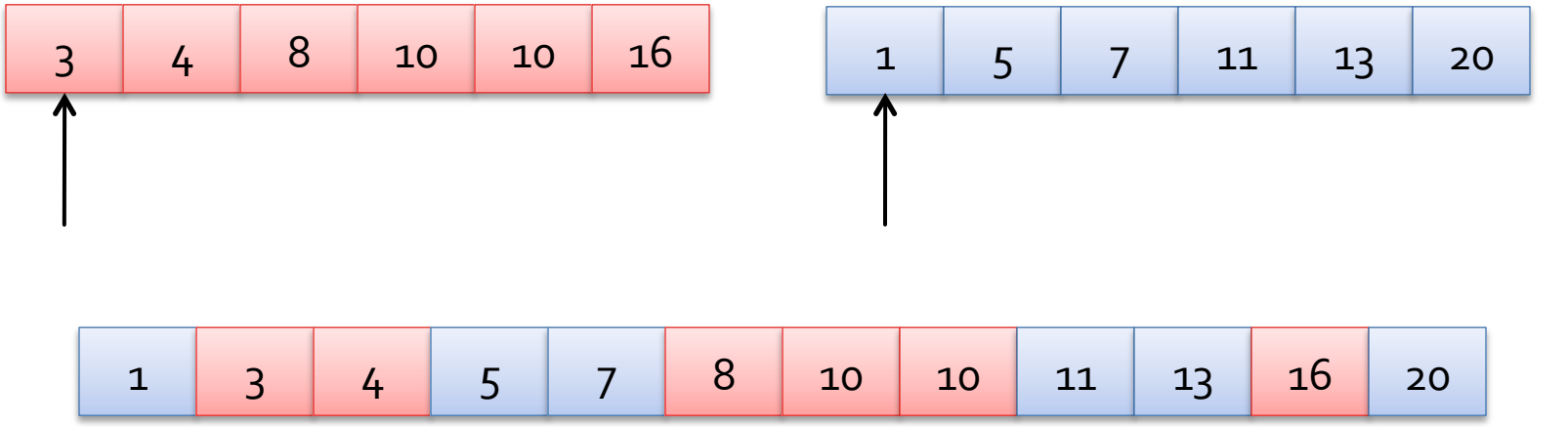
该问题所分解出的各个子问题是相互独立的
即子问题之间不包含公共的子问题

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：每个问题的子问题都是相互独立的
- 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

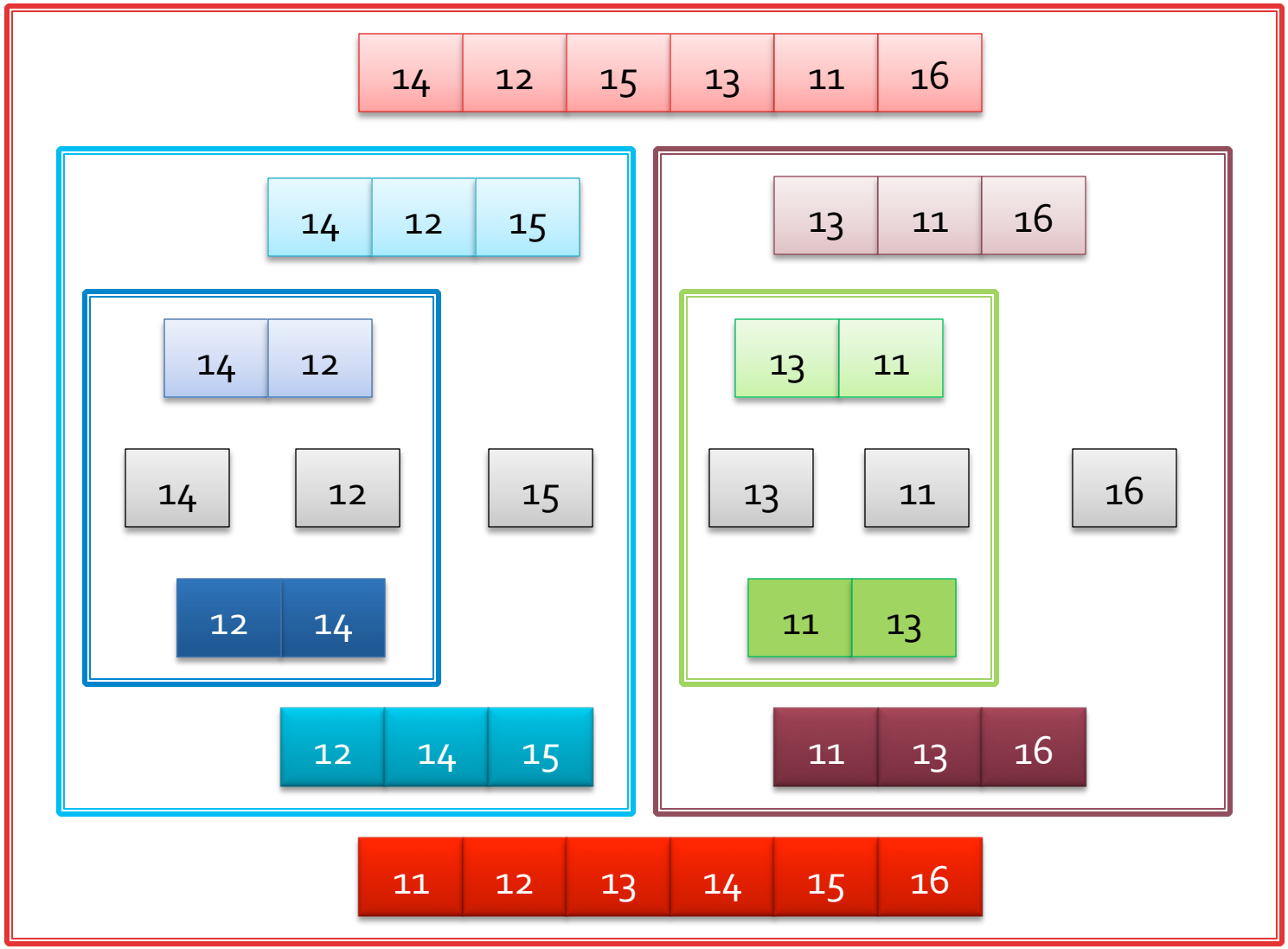
归并两个有序数列

将两个有序的数列合并为一个有序的数列



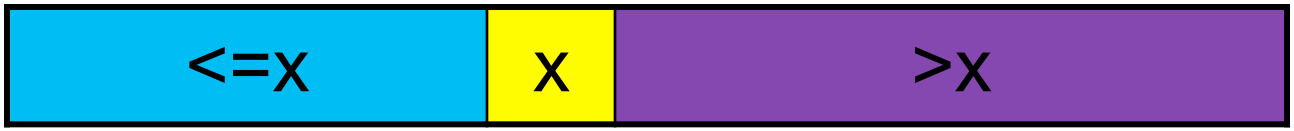
时间复杂度O(n)

递归过程


14 12 15 13 11 16
14 12 15
13 11 16
14 12
13 11
14 12 15
13 11 16
12 14
11 13
12 14 15
11 13 16
11 12 13 14 15 16

问题：快速排序(QUICKSORT)

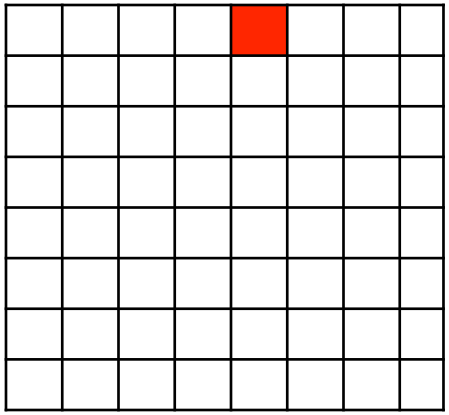
- 基本步骤
  - 分解：任取一个元素x，将数组分为小于x的部分和大于等于x的部分
  - 解决：对每个子序列应用快速排序算法
  - 合并：无需合并，自然有序

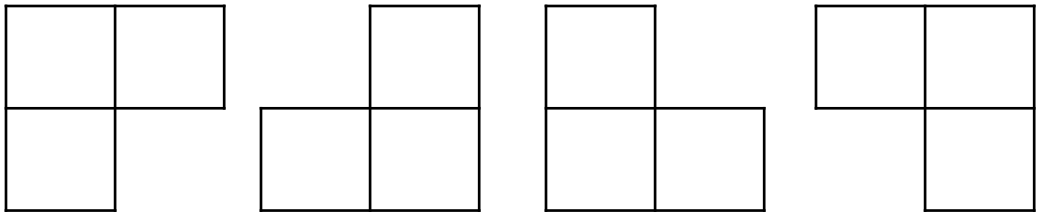


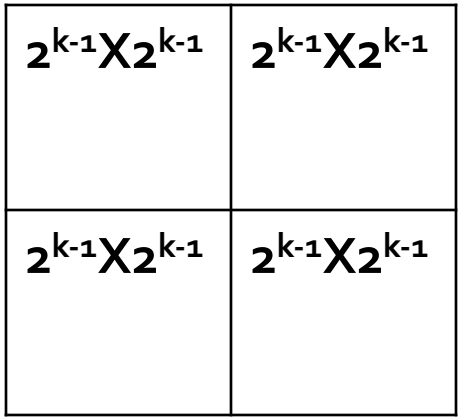
两种方法比较

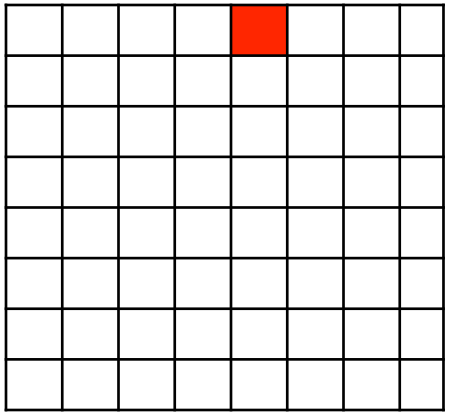
问题：棋盘覆盖

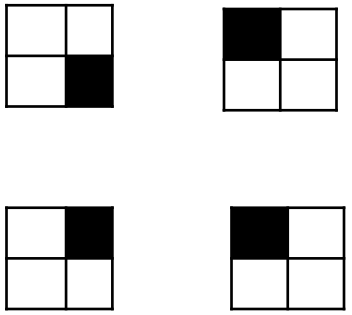
- 在一个$2^k \times 2^k$个方格组成的棋盘中，恰有一个方格与其它方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。

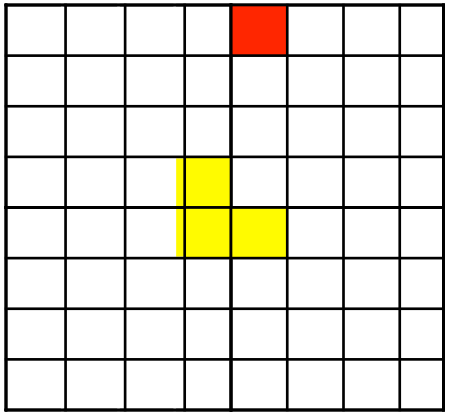





$2^{k-1}X2^{k-1}$
$2^{k-1}X2^{k-1}$
$2^{k-1}X2^{k-1}$
$2^{k-1}X2^{k-1}$







再来回顾一下排序

冒泡排序：O(n²)

插入排序：O(n²)

快速排序：O(nlgn)

归并排序：O(nlgn)

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

http://en.wikipedia.org/wiki/Algorithms

时间复杂度的意义?

π = 3.14159265358979323846264338327950288419716939937510...

e = 2.7 ...5236028747 ...4709369995...

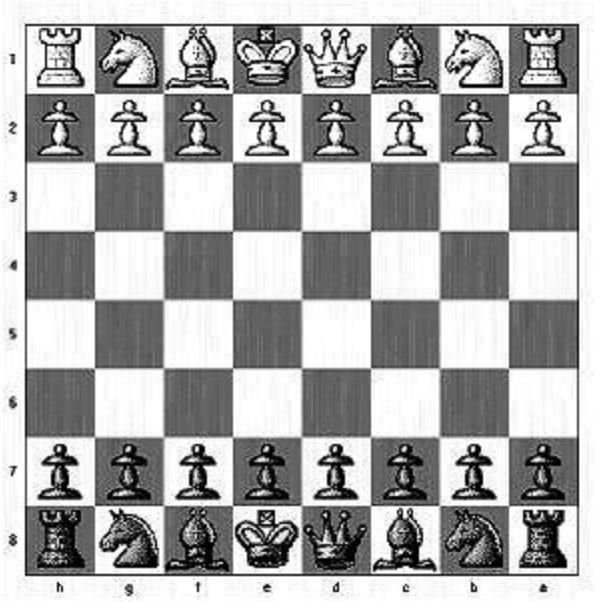
2^n vs n^2

e^π vs π^e

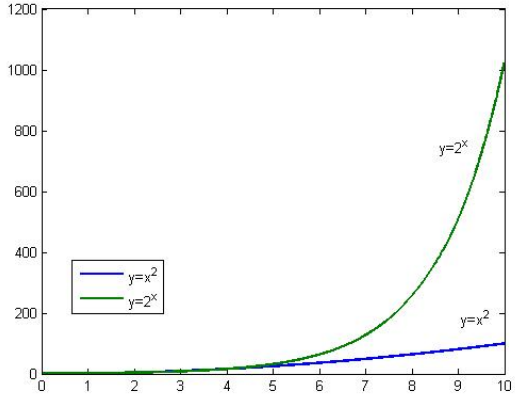
18446744073709551616

$1.844674407 \times 10^{19}$

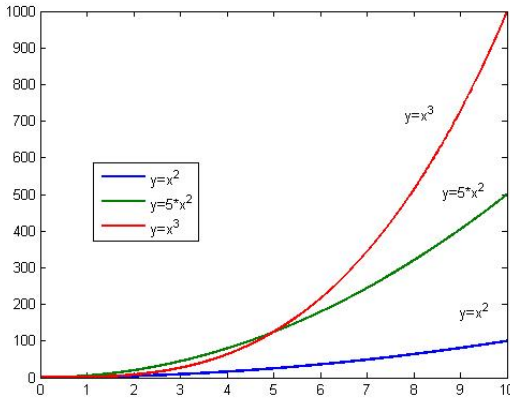
2^{64}♪

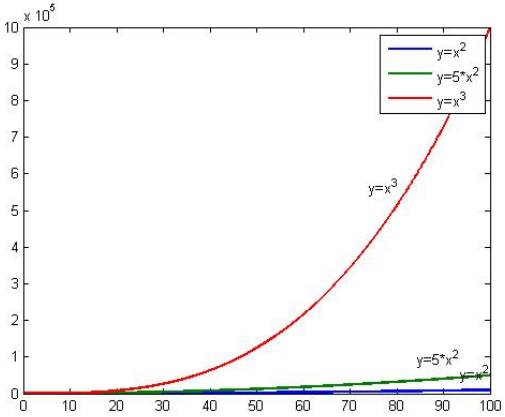


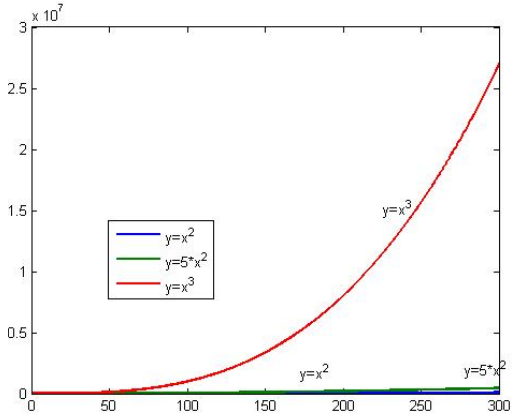
	2	3	4	5	6	7	8	9	10
x^2	4	9	16	25	36	49	64	81	100
2^x	4	8	16	32	64	128	256	512	1024



	2	5	10
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$f(N)<=Cg(N)$

g(N)是f(N)的上界

记做
$f(N)=O(g(N))$

当$N>N_0$时，
有$f(N)>=Cg(N)$

g(N)是f(N)的下界

记做
$f(N)=\Omega(g(N))$

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

+

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

→

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

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

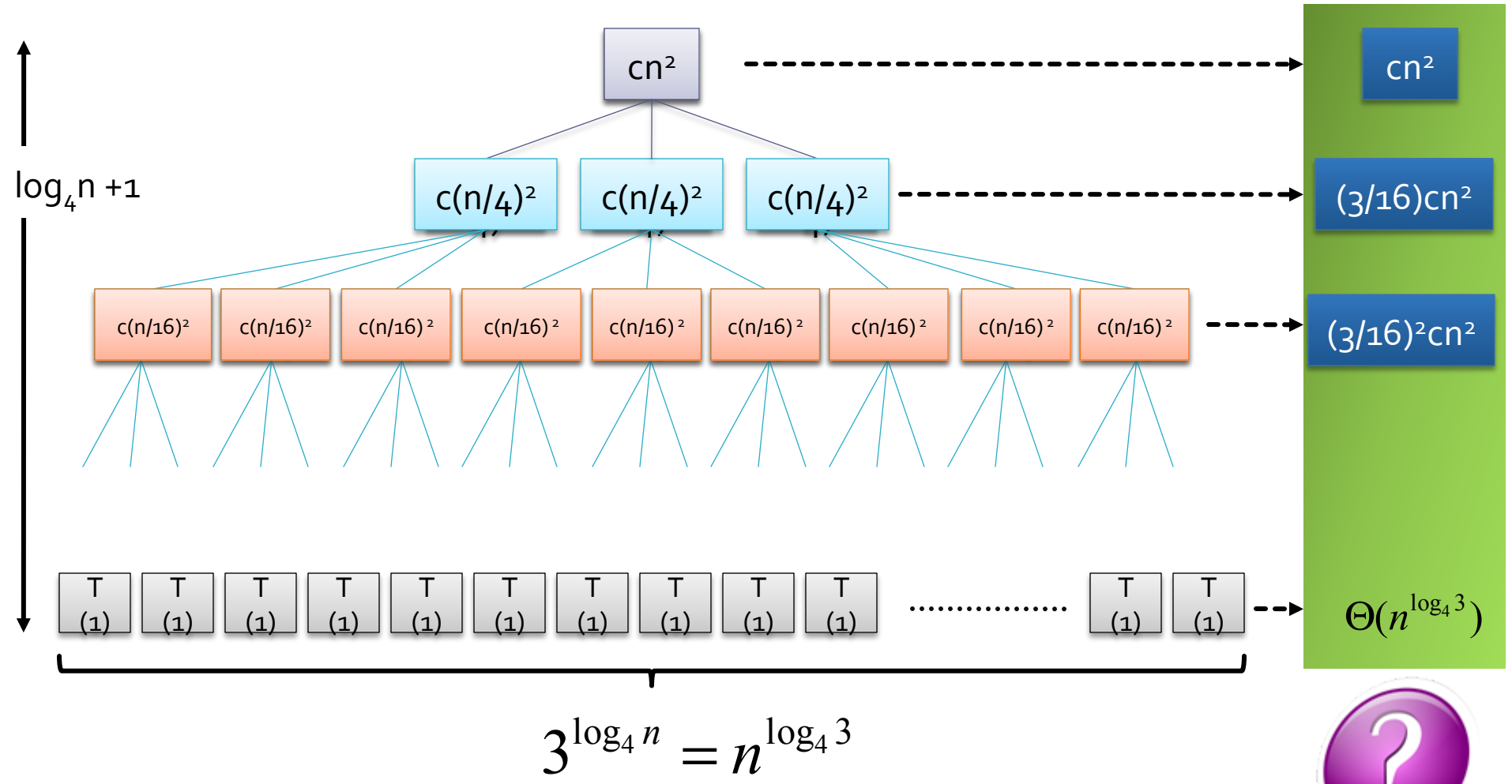
$$T(n) = aT(n/b) + f(n)$$

如何计算 T(n)

归并排序： $T(n) = 2T(n/2) + cn$

递归树法

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



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

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

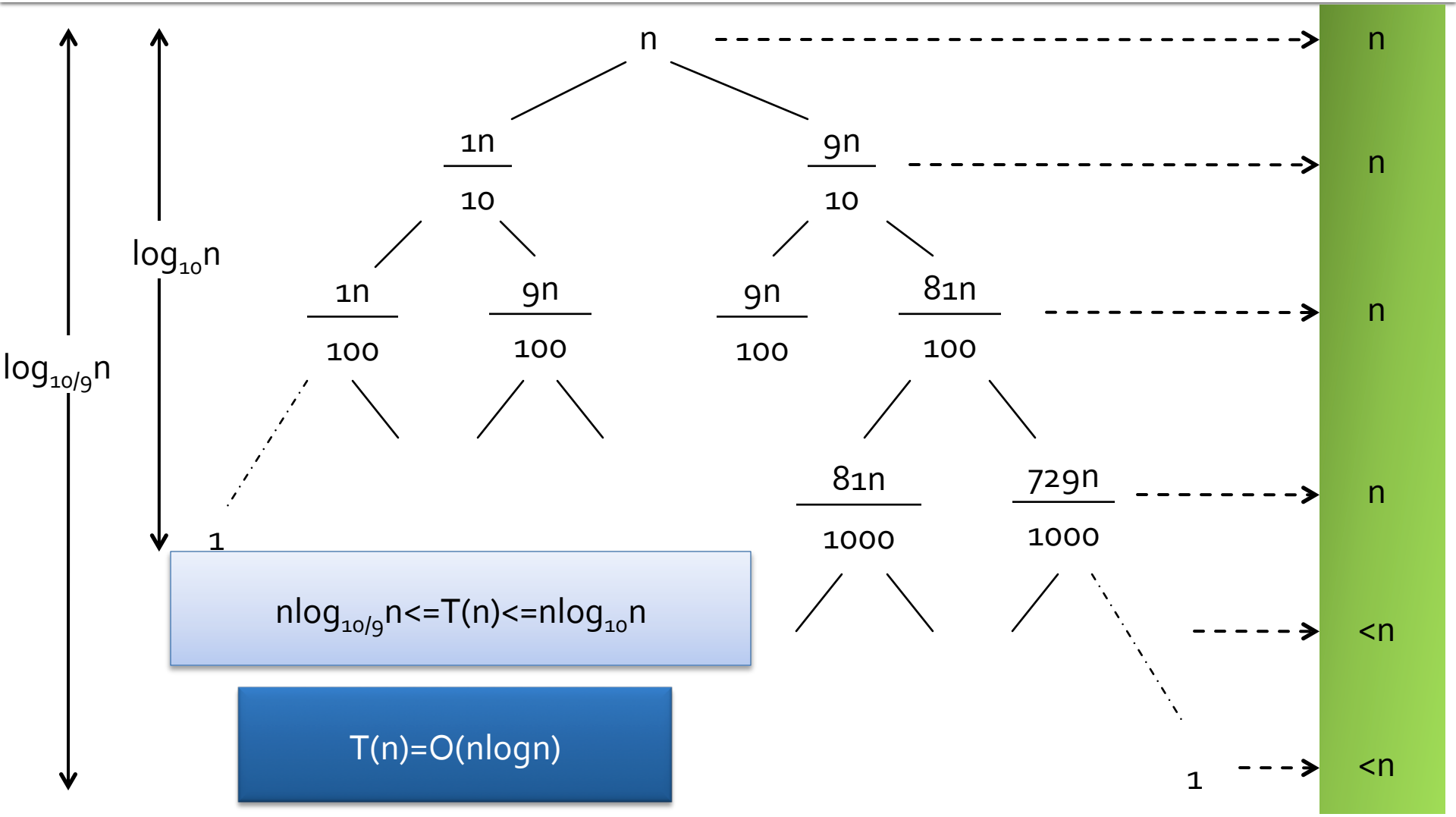
$$= \frac{1}{1 - \frac{3}{16}} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13}cn^2 + \Theta(n^{\log_4 3})$$

$$= \mathrm{O}(n^2)$$

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


n
1n/10
9n/10
1n/100
9n/100
9n/100
81n/100
81n/1000
729n/1000
1
1
n
n
n
n
<n
<n
log10n
log10/9n
nlog10/9n<=T(n)<=nlog10n
T(n)=O(nlogn)

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(nlgn)

假设T(m)<=cmlgm m<n
要证T(n)<=cnlgn

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

我们对结果有一个猜想T(n)=O(nlg_2n)

假设T(m)<=$cmlg_2m$ m<n
要证T(n)<=$cnlg_2n$

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

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)=Θ(nlgn)
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)=Θ(lgn)

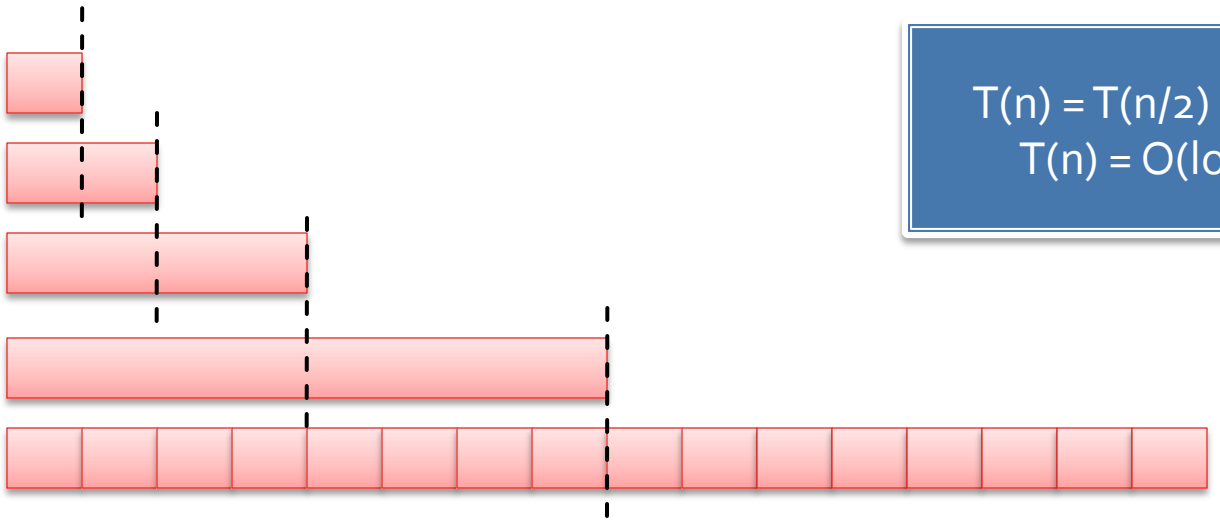
幂次计算

计算 a^n

a*a*a*a*a*.....*a

时间复杂度O(n)

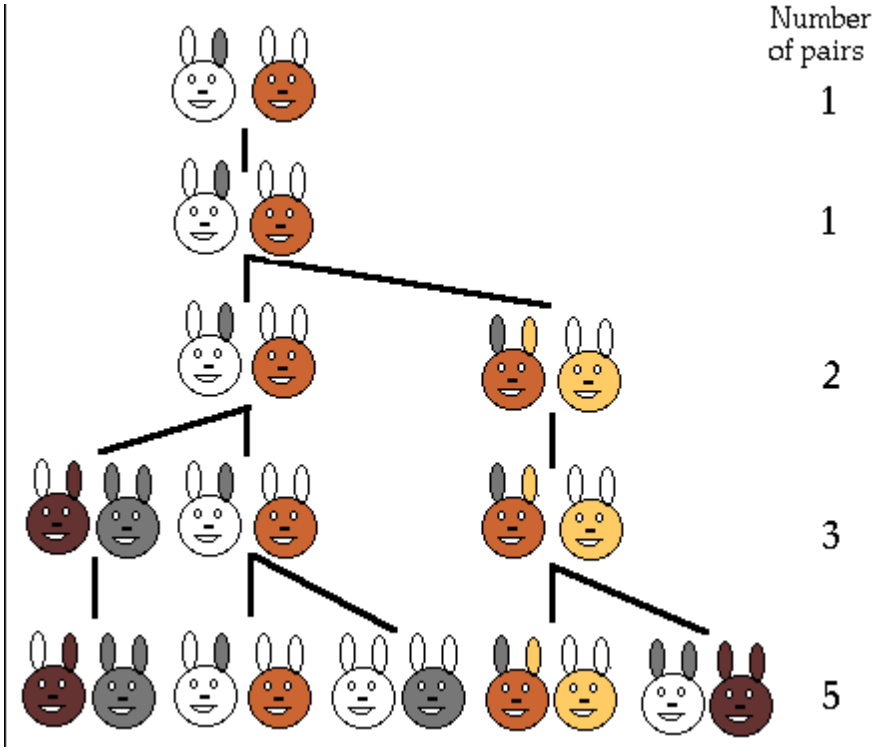
$$a^n = \begin{cases} a^{n/2}a^{n/2} & n\text{为偶数} \\ a^{(n-1)/2}a^{(n-1)/2}a & \text{n为奇数} \end{cases}$$



T(n) = T(n/2) + O(1)
T(n) = O(logn)

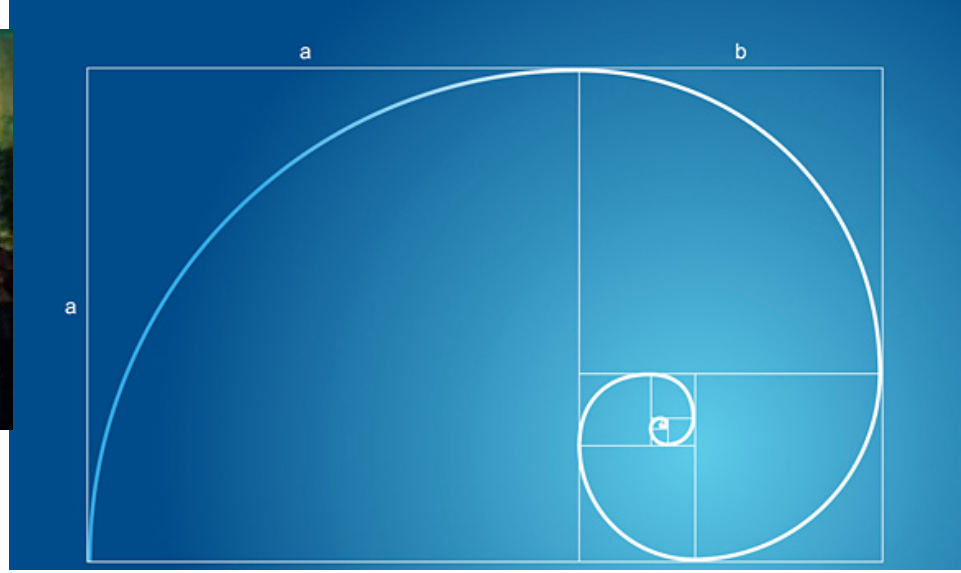
斐波那契（Fibonacci）数列

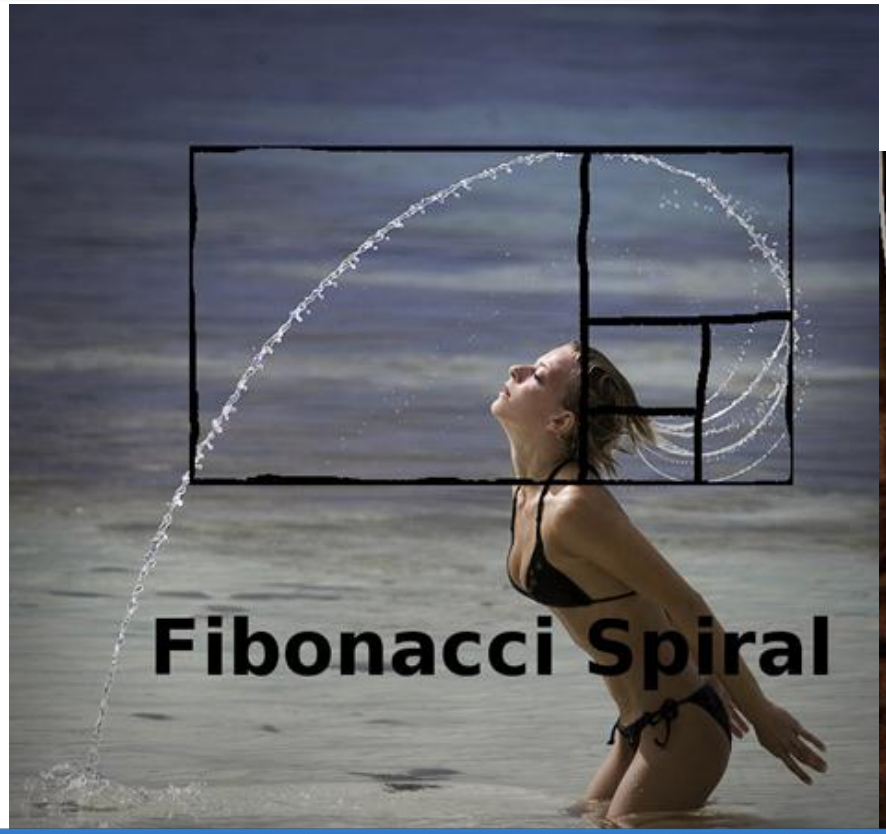
《珠算原理》：某人把一对兔子放入一个四面被高墙围住的地方。假设每对兔子每月能生下一对小兔，而每对新生小兔从第二个月开始又具备生育能力，请问：一年后应有多少对兔子

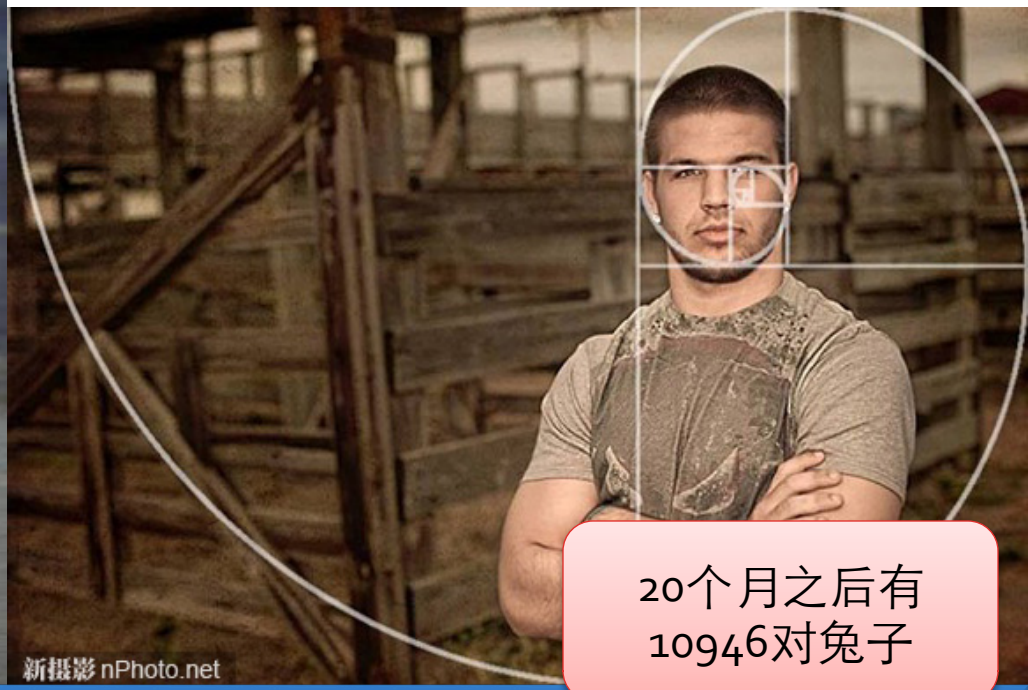


列昂纳多·斐波那契
意大利数学家

$$F_0 = 1$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

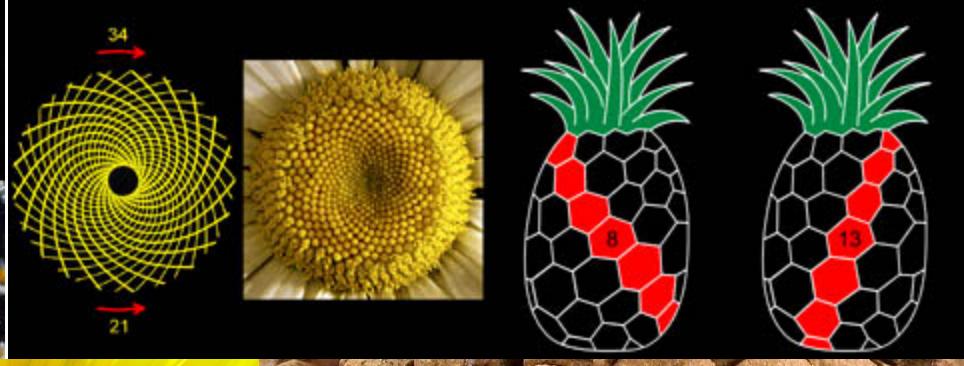


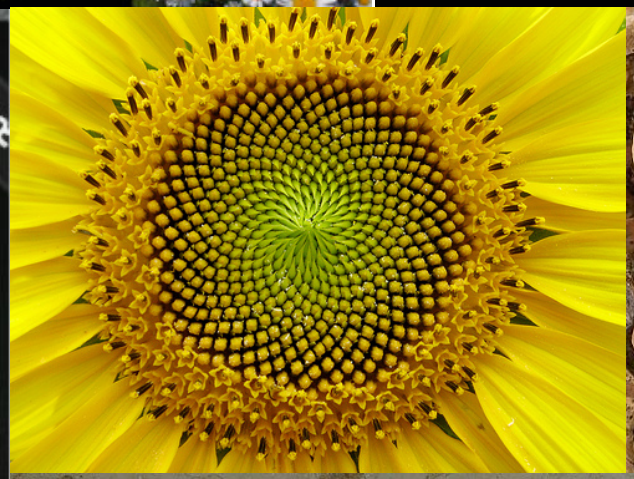


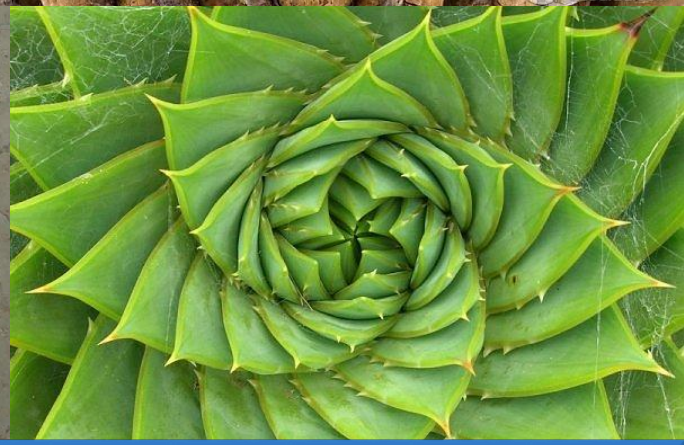


20个月之后有
10946对兔子

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,.....

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

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
F0,F1,F2,F3,F4,F5.....,F1000000

时间复杂度O(n)

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

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

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(lgn)

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?

C语言中，int的范围是$-2^{31} \sim 2^{31}-1$

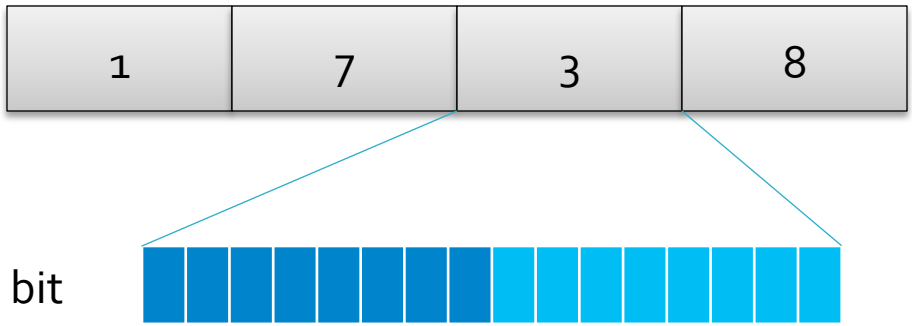
可以表示的整数位数不超过10位

如何进行任意位整数的加减乘除运算呢?

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]



元素之间的运算容易实现。

如何将整数输入到这个数组中。

sacnf

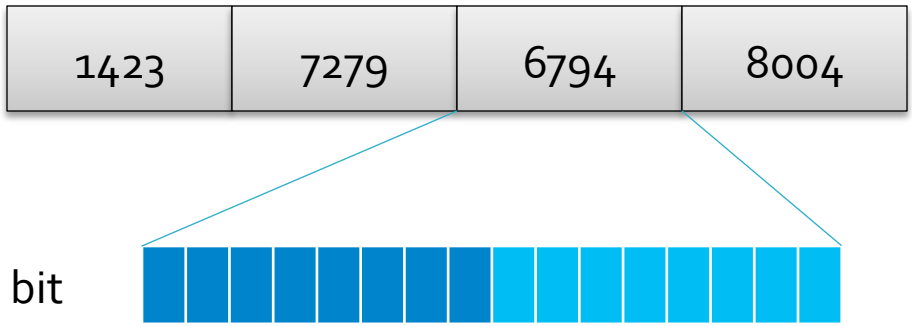
short int的范围是-32768～32767，有很大的浪费

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]



元素之间的运算容易实现。

如何将整数输入到这个数组中。

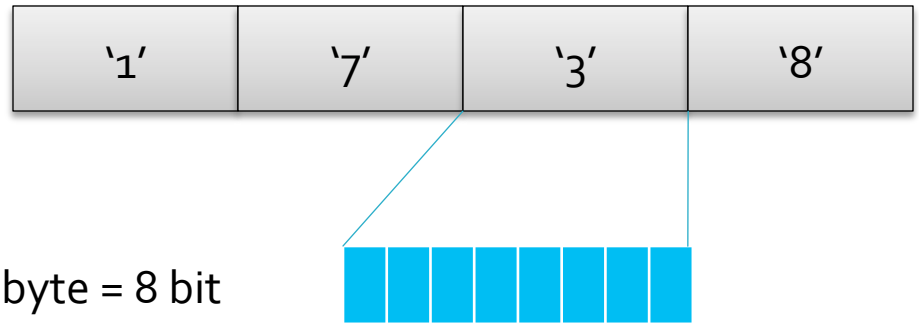
sacnf

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	212	117	61

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

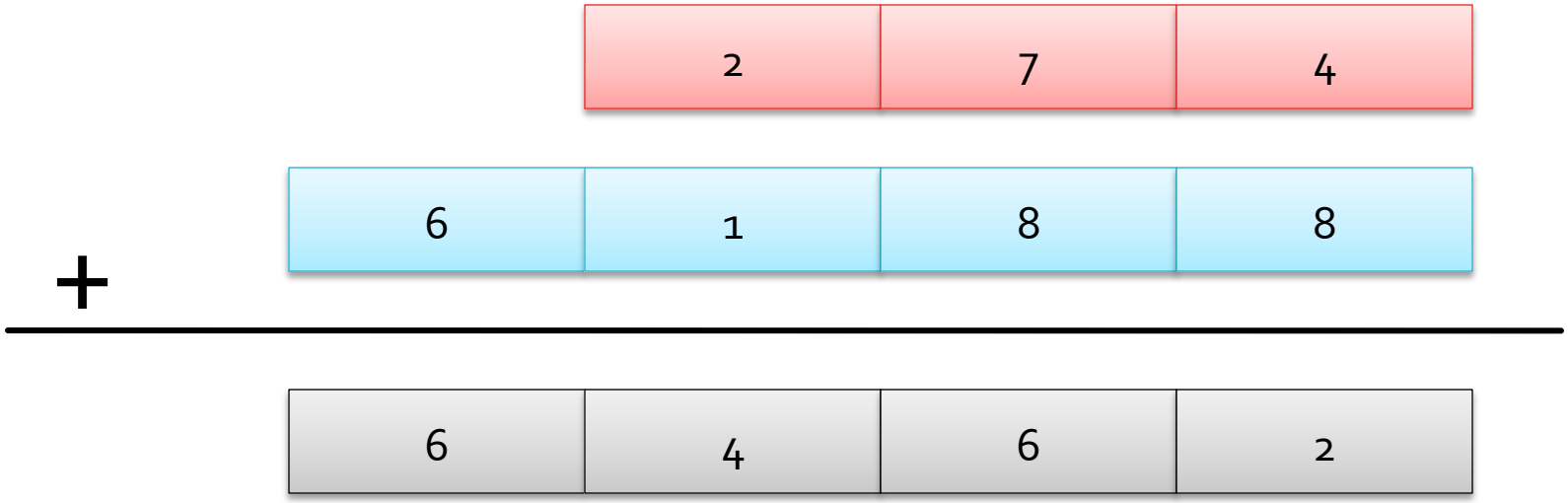
元素之间的运算需要另外的转化

字符串输入需要转化

空间最省

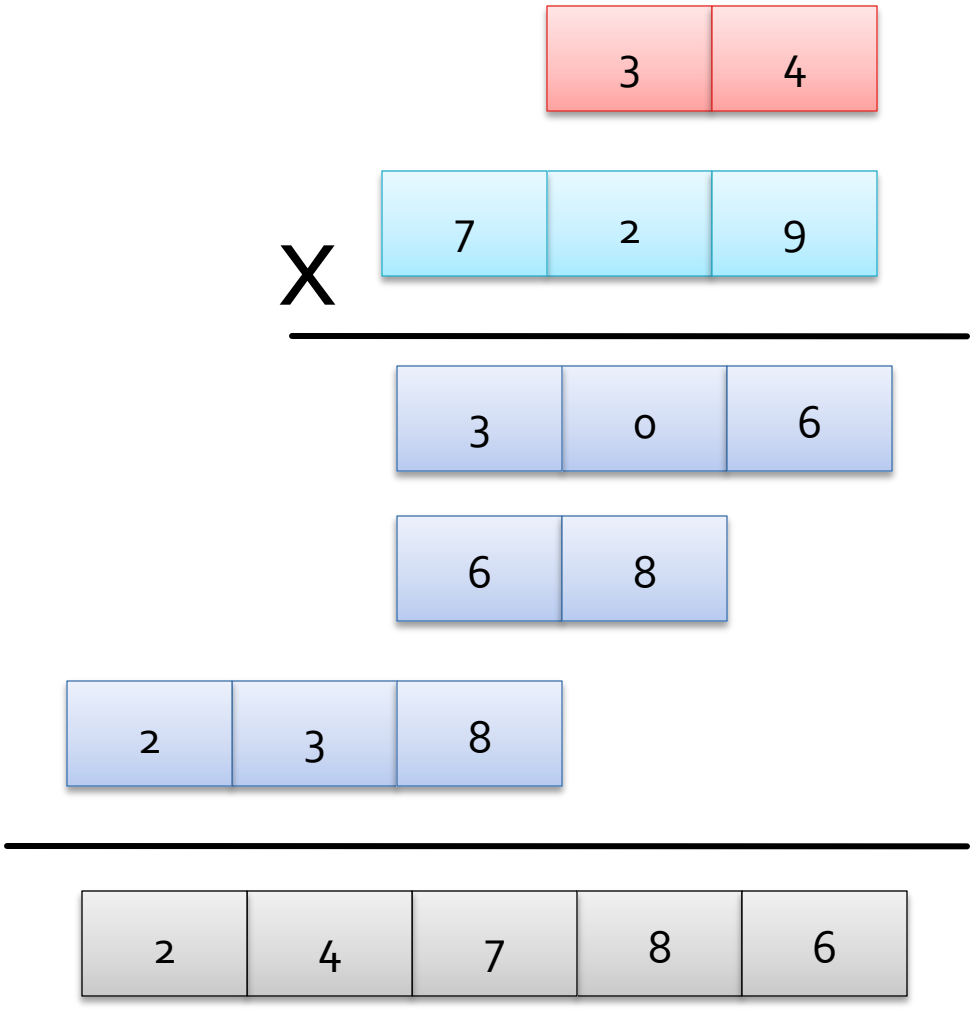
大整数加法

数组对应的位置分别作加法



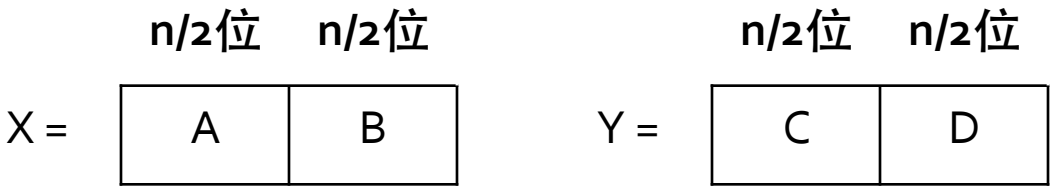
时间复杂度O(n)

大整数乘法



时间复杂度$O(n^2)$

尝试分治思想



$X = A10^{n/2} + B$
$Y = C10^{n/2} + D$

$XY = (A10^{n/2} + B)(C10^{n/2} + D)$
$=AC10^{n} +(AD+BC)\ 10^{n/2} +BD$

运算量与整数的位数有关，用位数n来代表问题的规模

$T(n) = 4T(n/2) + O(n)$

$T(n) = O(n^2)$

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

Carl Friedrich Gauss (1777-1855)

$(a+bi)(c+di)=ac-bd+(bc+ad)i$

4次乗法

$bc+ad$
$= (a+b)(c+d) -ac-bd$

ac　　bd　　(a+b)(c+d)

3次乗法

算法改进

$$XY = (A10^{n/2} + B)(C10^{n/2} + D)$$
$$= AC10^{n} + (AD+BC)\,10^{n/2} + BD$$

$$AD+BC = (A+B)(D+C) - AC - BD$$

$$XY = AC2^{n} + ((A+B)(D+C) - AC - BD)\,2^{n/2} + BD$$

$$T(n) = 3T(n/2) + O(n)$$

$$T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

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

$$\Rightarrow T(n) = \Theta(n^{\log_b a}) .$$

如何运行大整数除法?

线性时间选择

寻找10亿个浮点数中最大的10个数

如何保存10亿个浮点数?

float[1000 000 000];

一个float变量占用4byte内存

一共需要4G的内存

堆和栈的作用有什么不同，他们都存放了哪些变量?

类型	说明	长度(字节)	表示范围	备注
char	字符型	1	-128～127	-2^7～(27-1)
unsigned char	无符号字符型	1	0～255	0～(2^8-1)
signed char	有符号字符型	1	-128～127	-2^7～(2^7_1)
int	整形	2	-32768～32767	-2^{15}～(2^{15}-1)
unsigned int	无符号整形	2	0～65536	0～(2^{16}-1)
short int	短整形	2	-32768～32767	-2^{15}～(2^{15}-1)
unsigned shord int	无符号短整形	2	0～65535	0～(2^{16}-1)
signed shord int	有符号短整形	2	-32768～32767	-2^{15}～(2^{15}-1)
long int	长整形	4	-2147483648～2147483647	-2^{31}～(2^{31}-1)
unsigned long int	无符号长整形	4	0～4294967296	0～(2^{35}-1)
signed long int	有符号长整形	4	-2147483648～2147483647	-2^{31}～(2^{31}-1)
float	浮点型	4	-3.4×10^{38}～-3.4×10^{38}	7位有效位
double	双精度型	8	-1.7×10^{308}～-1.7×10^{308}	15位有效位
long double	长双精度型	16	-3.4×10^{4392}～1.1×10^{4392}	19位有效位

寻找10亿个浮点数中最大的数

N-1次比较

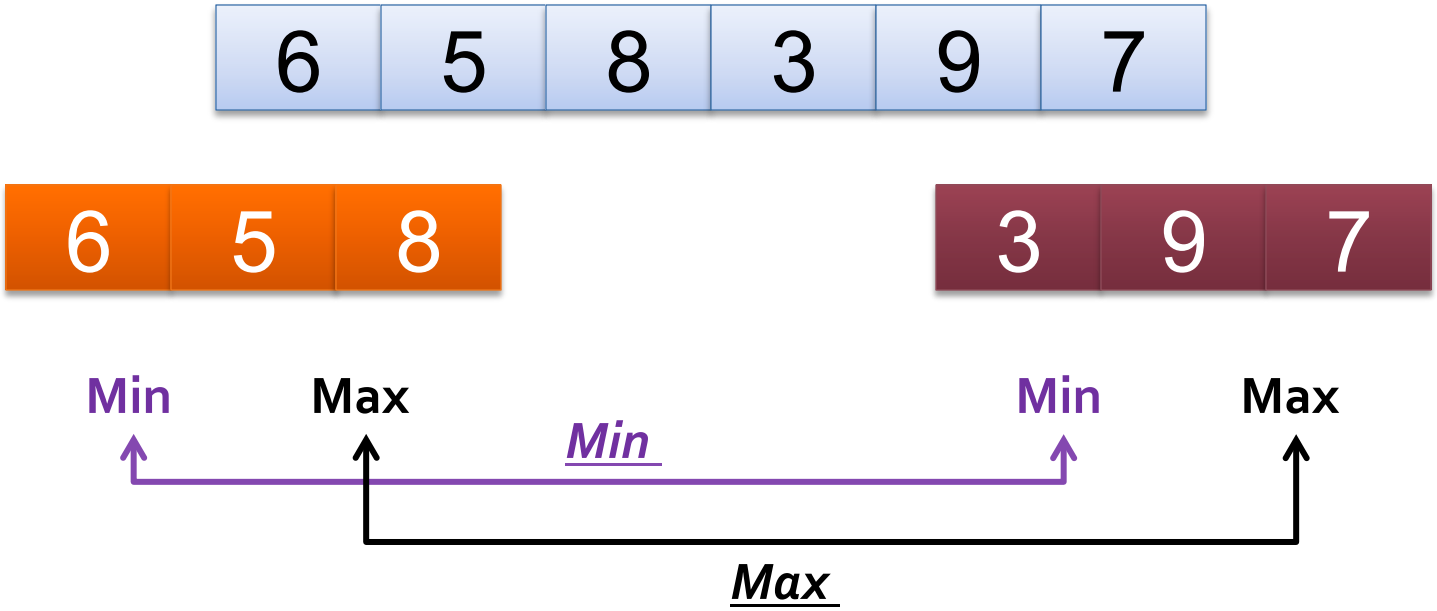
寻找10亿个浮点数中最小的数

N-1次比较

寻找10亿个浮点数中最大的数和最小的数

6	5	8	3	9	7

优化无处不在

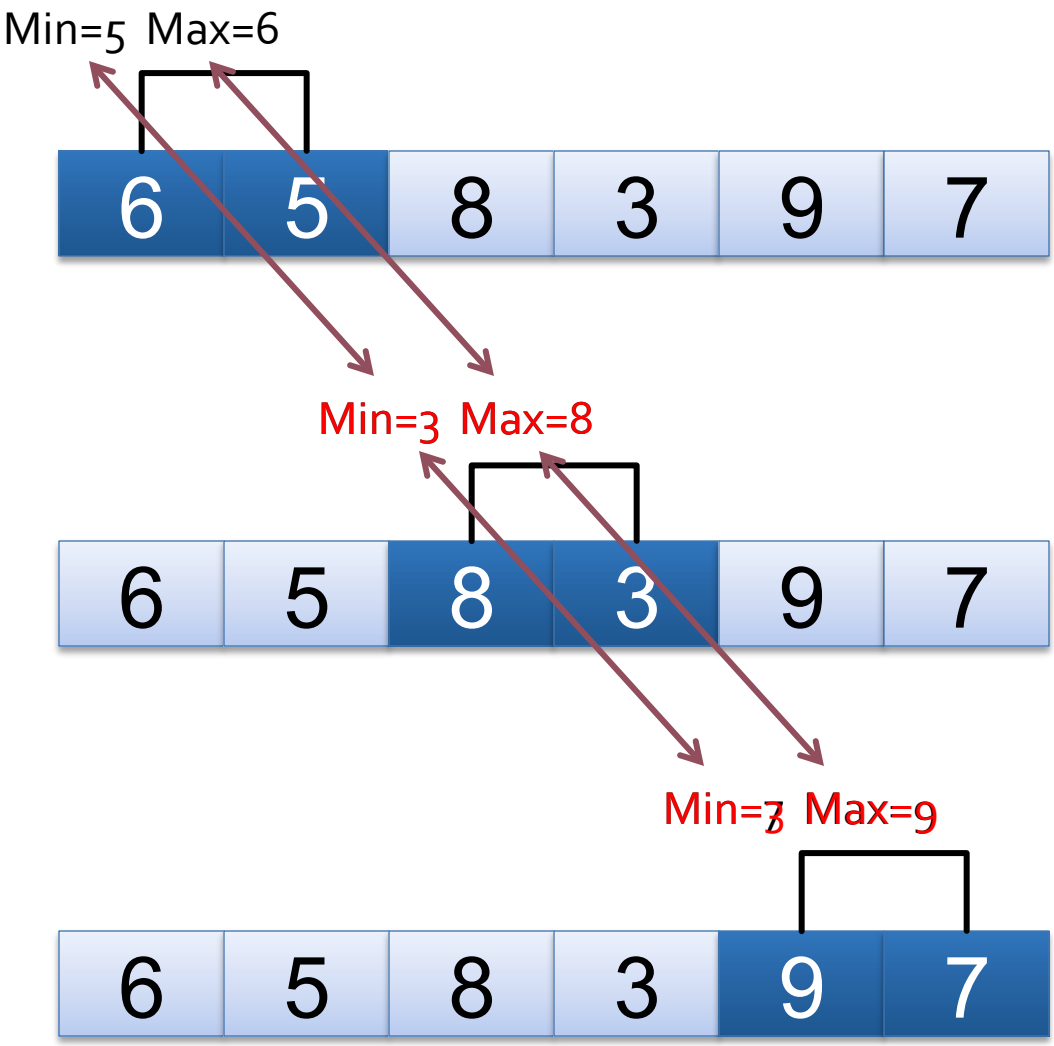


$$f(N) = 2 * f(N/2) + 2$$

$$f(2) = 1$$

$$
\begin{aligned}
f(N) &= 2 * f(N/2) + 2 \\
&= 2 * (2 * f(N/2^2) + 2) + 2 \\
&= 2^2 * f(N/2^2) + 2^2 + 2 \\
&\cdots\cdots \\
&= 2^{(\log_2 N)-1} * f(N/2^{(\log_2 N)-1}) + 2^{(\log_2 N)-1} + \cdots + 2^2 + 2 \\
&= \frac{N}{2} f(2) + 2^{(\log_2 N)-1} + \cdots + 2^2 + 2 \\
&= \frac{N}{2} + \frac{2 - 2^{(\log_2 N)}}{1-2} \\
&= \frac{N}{2} + \frac{2-N}{1-2} \\
&= 1.5N - 2
\end{aligned}
$$

无需递归的方法



1.5N-2

问题思考

寻找10亿个浮点数中最大的10个数

如果我要找最大的一个呢?

N次比较

先对所有的数进行排序

O(NlgN)

O(NlgN) vs. O(N*K)

是否需要对N个数进行排序?

只需对前K个数保持排序

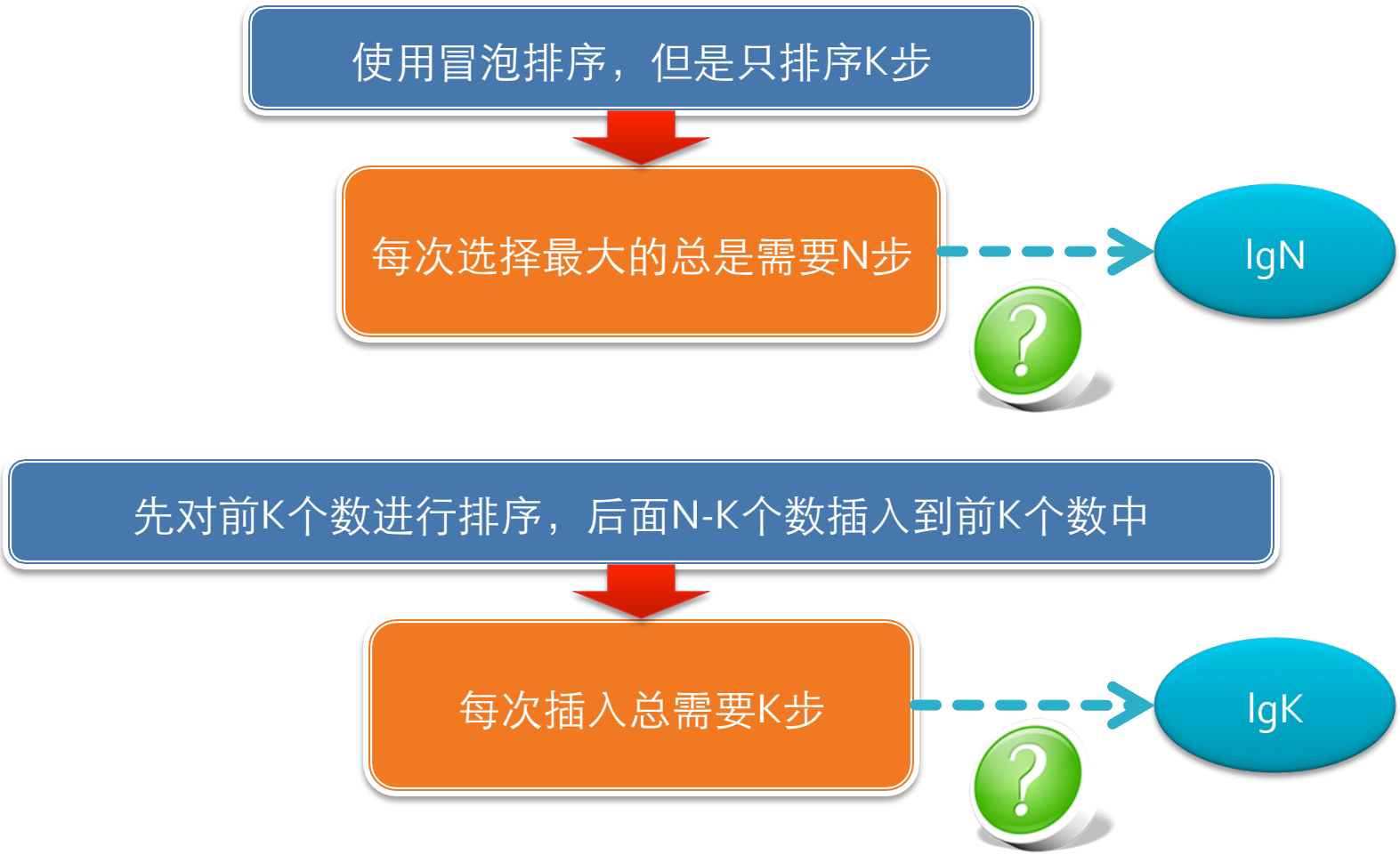
冒泡不一定是最差

使用冒泡排序，但是只排序K步

O(N*K)

先对前K个数进行排序，后面N-K个数插入到前K个数中

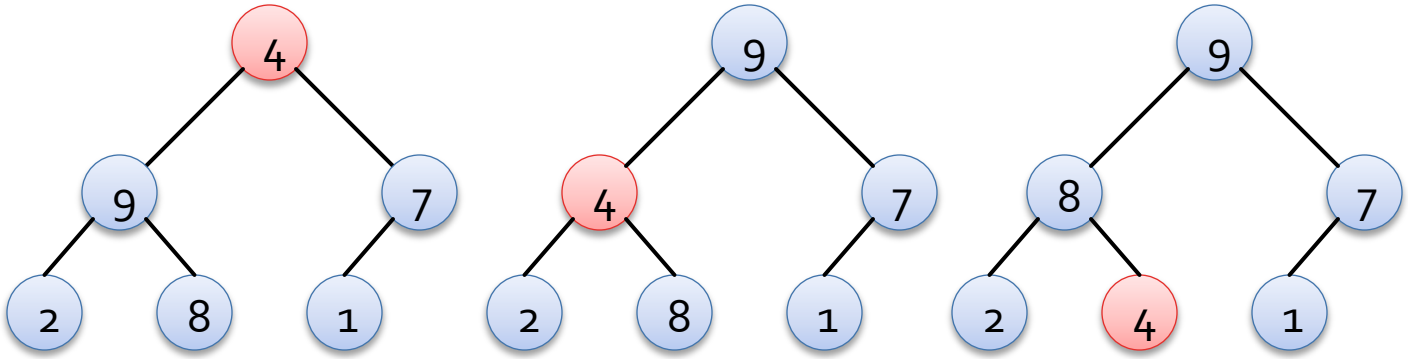
两种算法存在的问题


使用冒泡排序，但是只排序K步
每次选择最大的总是需要N步
lgN
先对前K个数进行排序，后面N-K个数插入到前K个数中
每次插入总需要K步
lgK

关于堆排序

保持堆性质

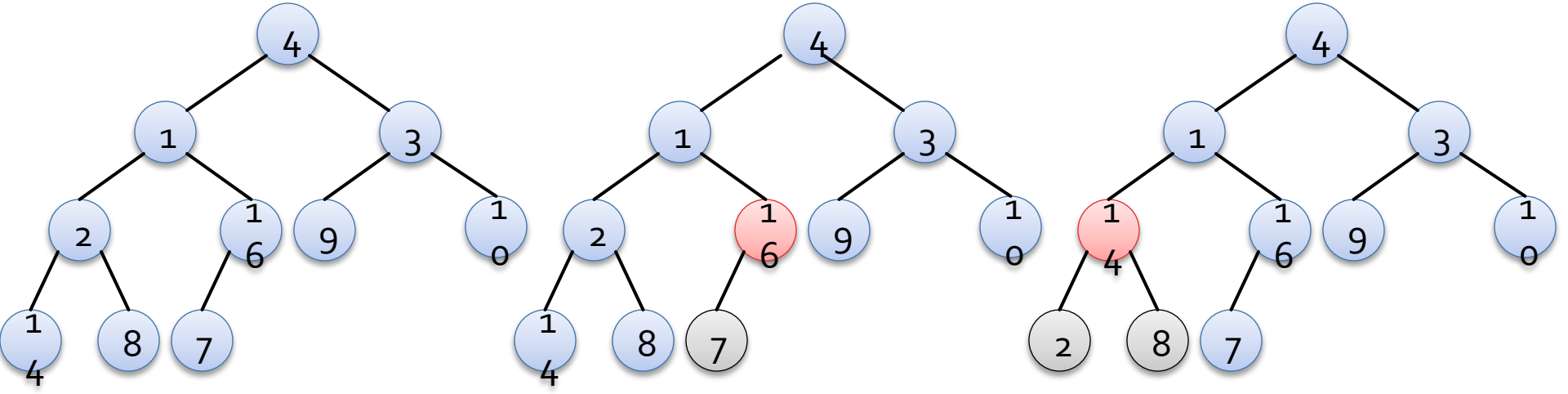
时间复杂度
O(lgn)

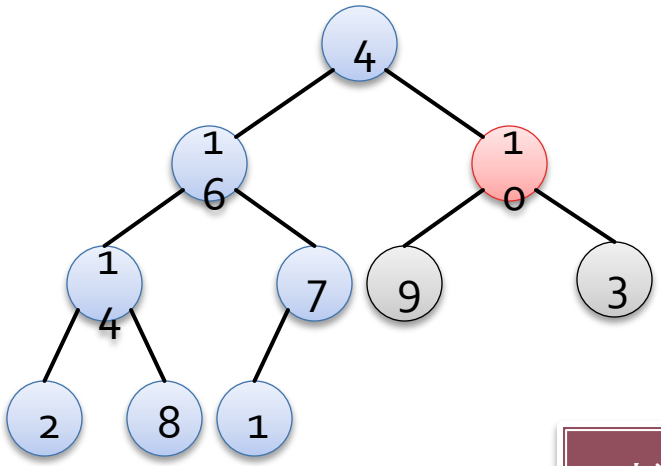


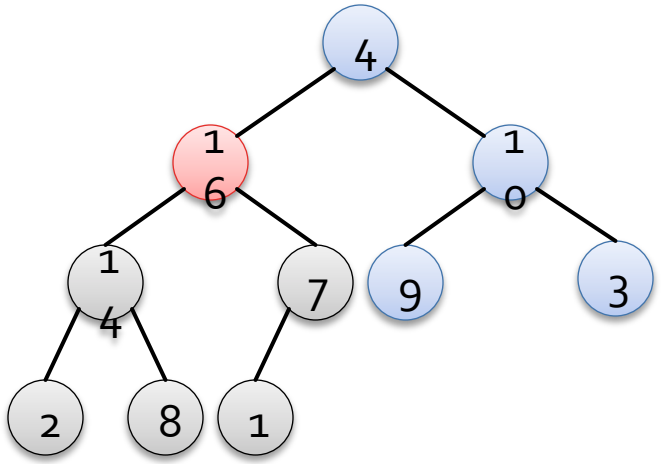
建堆

从叶子节点开始建堆

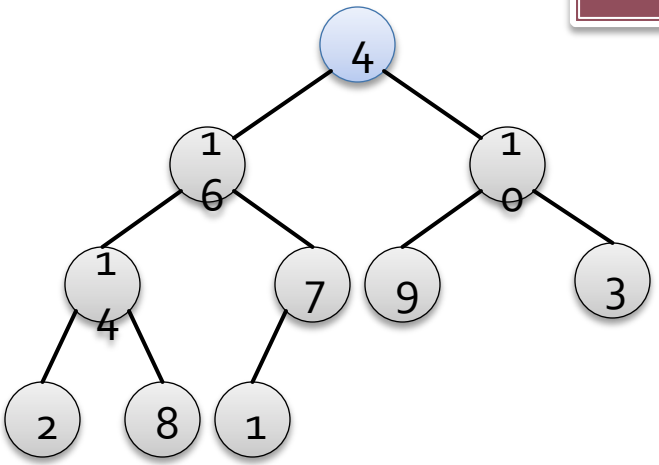
4	1	3	2	16	9	10	14	8	7

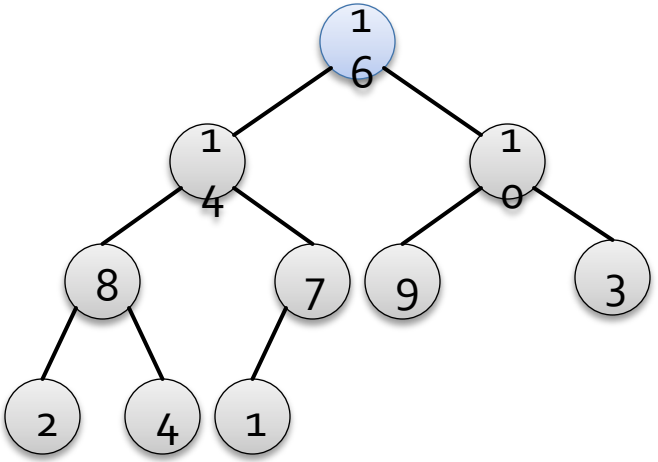



4
16
10
14
7
9
3
2
8
1


4
16
10
14
7
9
3
2
8
1

时间复杂度
O(n)

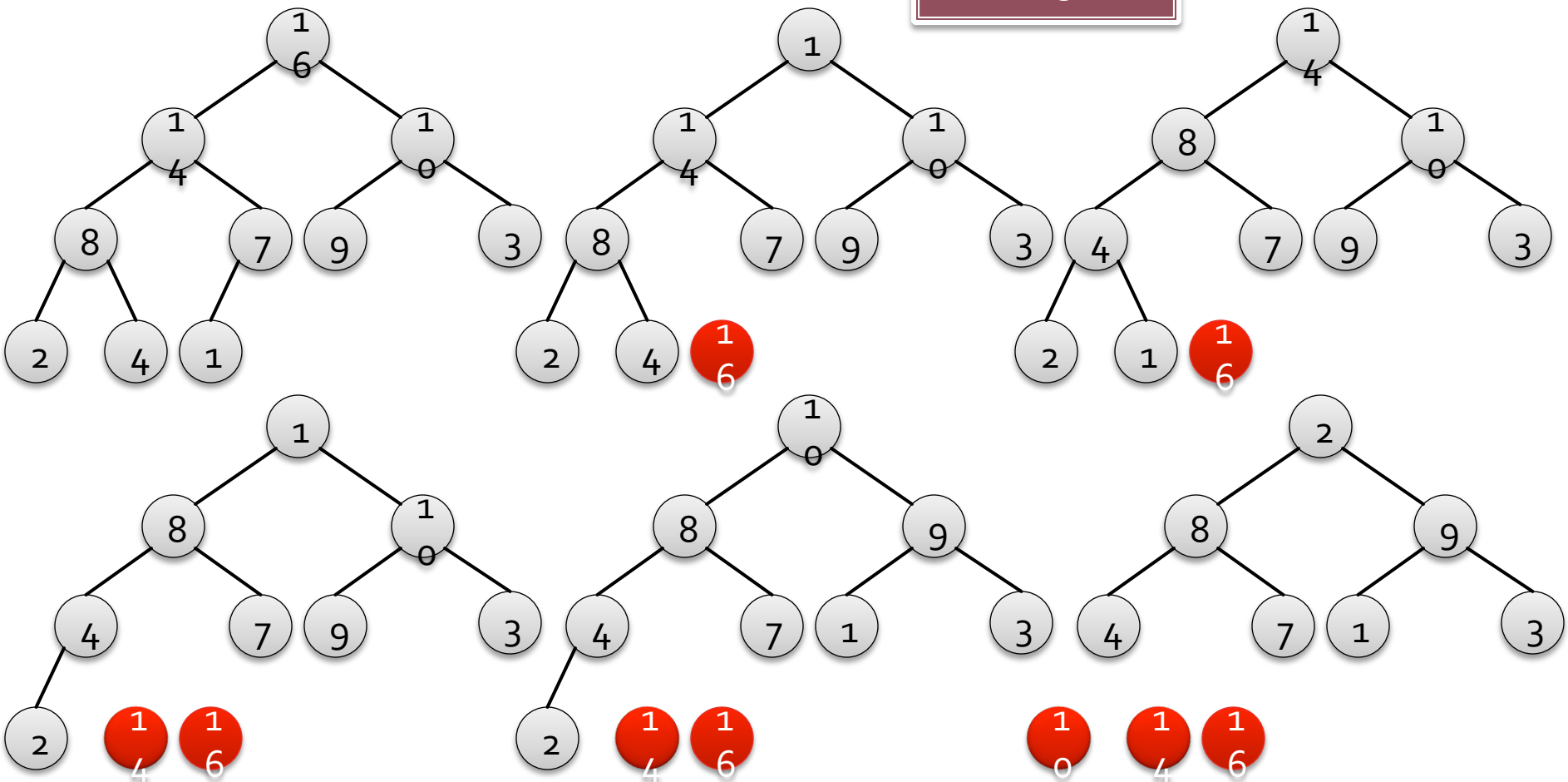

4
16
10
14
7
9
3
2
8
1


16
14
10
8
7
9
3
2
4
1

堆排序

建堆

时间复杂度
O(nlgn)



与堆相关的方法

建堆的时间复杂度 O(N)

更新堆的时间复杂度$O(\log_2 N)$

使用冒泡排序，但是只排序K步

每次选择最大的总是需要N步

用O(N)的时间为原数组建最大堆，然后弹出最大的元素K次，每次时间复杂度为$O(\log_2 N)$，总时间复杂度为$O(N+K*\log_2 N)$，当$K<N/\log_2 N$时，时间复杂度为O(N)

先对前K个数进行排序，后面N-K个数插入到前K个数中

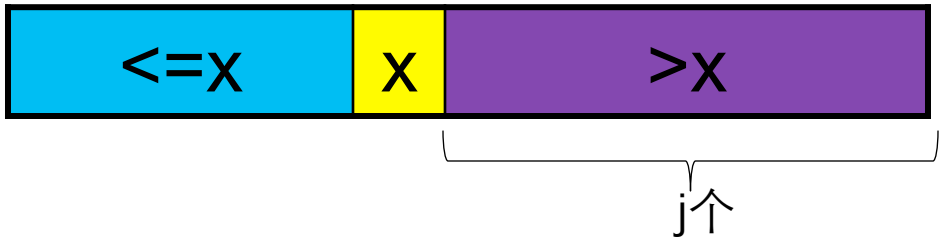
每次插入总需要K步

用O(K)的时间建立大小为K的最小堆，然后用数组中的每个元素与堆顶元素比较，若大于堆顶元素，则弹出原有堆顶，并将新的元素插入堆中，重建堆。时间复杂度为$O(N*\log_2 K)$

中位数怎么办?

有没有其他方法

快速排序 + 二分查找



Case 1　j<K，>X中的全部和<=X中最大的K-j个

Case 2　j>K，>X中的最大K个元素

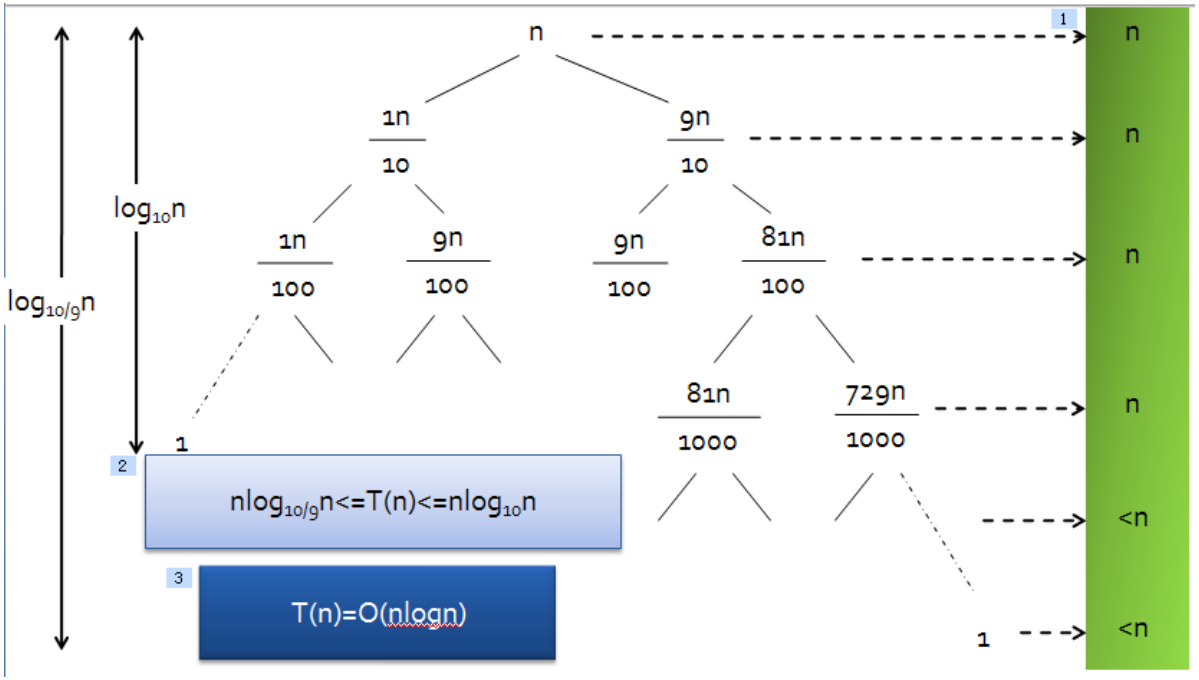
平均时间复杂度Θ(n)，最坏时间复杂度Θ(n²)

把问题化简为规模更小的子问题，如何计算时间复杂度呢？

参加《算法导论》第9章

如何寻找每次的划分点x?

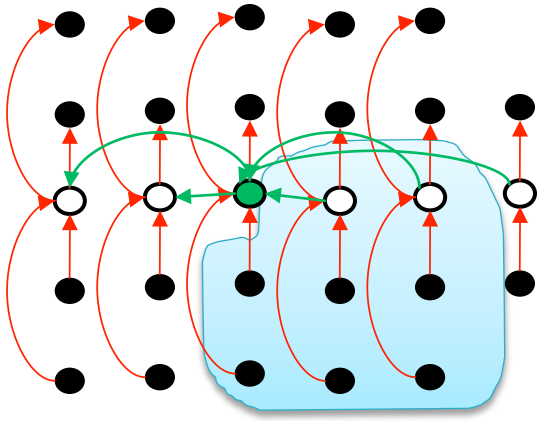
假设一个数列用快速排序，每次都能将其分解为9:1的两段，试用递归树法分析在此种情况下算法的时间复杂度



最坏情况线性时间选择

如果能在线性时间内，使得划分后的两个数组的大小都至少是原数组的一个固定倍数ε，那么最坏时间也可以在O(n)内找到

在n个数中寻找中位数的问题T(n)



将n个元素按5个一组分为n/5组，将每组元素排序，找到每组的中位数，共n/5个

O(n)

找出这n/5个元素的中位数

T(n/5)

每组中有两个小于中位数，而n/5个中位数中有(n-5)/10个小于我们选定的中位数X

用X进行划分，每次至少缩短1/4

O(3n/4)

X至少比3(n-5)/10个元素小，
也至少比3(n-5)/10个元素大，
n>75时， 3(n-5)/10 > n/4

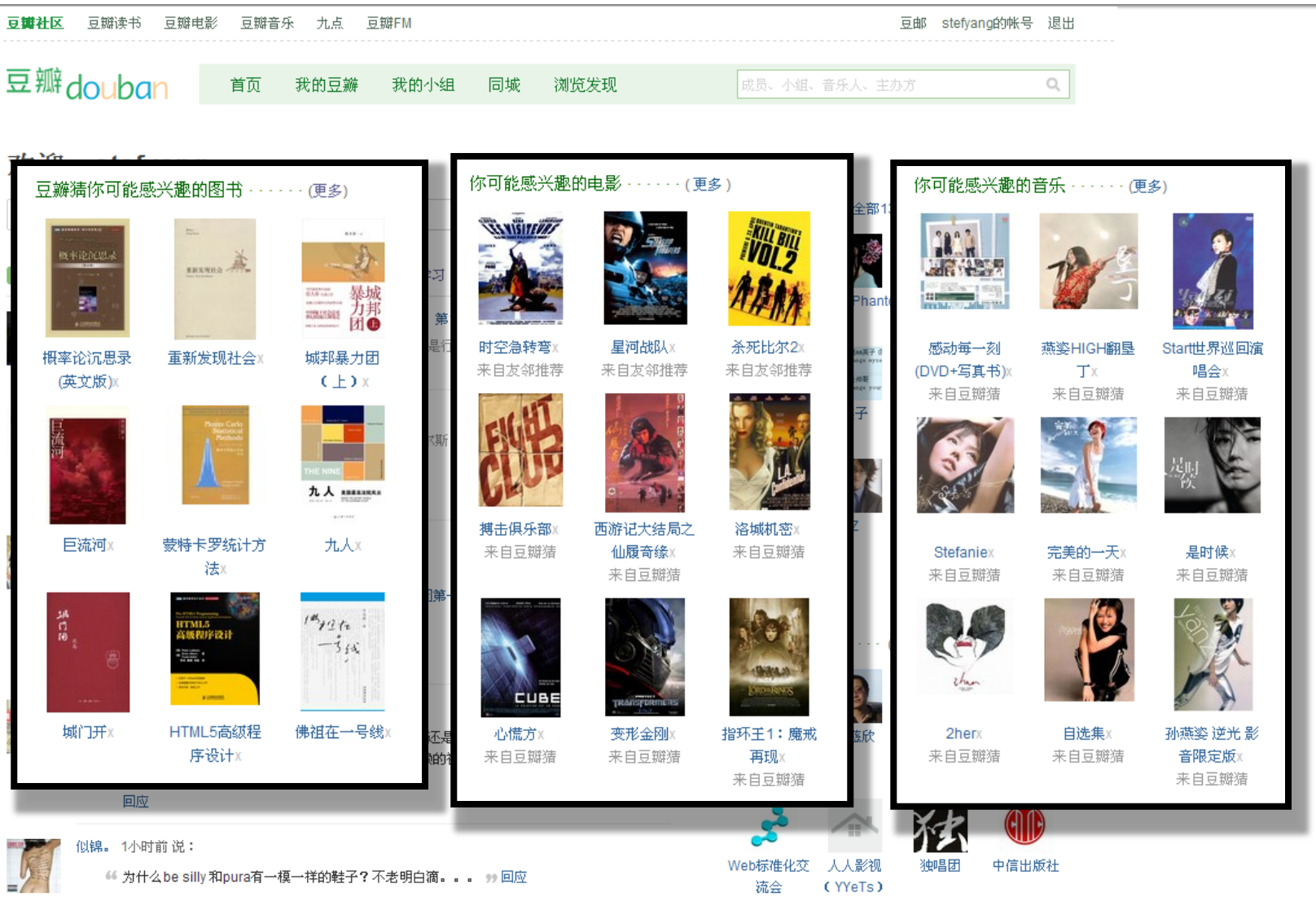
随堂测验

$$T(n) \leq T(n/5) + T(3n/4) + O(n)$$

最坏时间复杂度

$T(n)=O(n)$

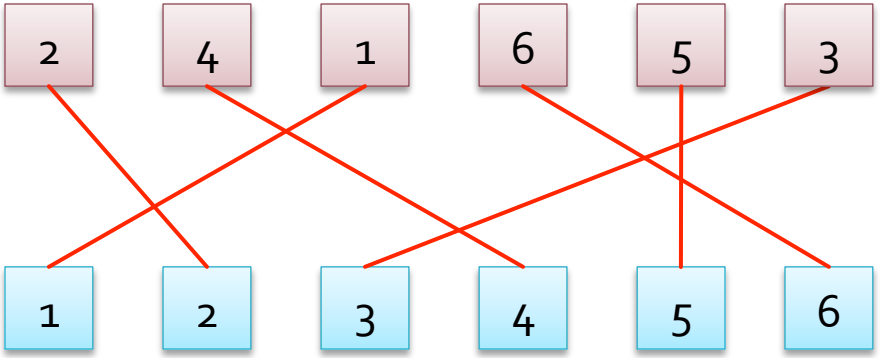
应用实例——排名分析


豆瓣社区 豆瓣读书 豆瓣电影 豆瓣音乐 九点 豆瓣FM
豆邮 stefyang的帐号 退出
豆瓣 douban
首页 我的豆瓣 我的小组 同城 浏览发现
成员、小组、音乐人、主办方
豆瓣猜你可能感兴趣的图书 (更多)
概率论沉思录(英文版)x
重新发现社会x
城邦暴力团(上)x
巨流河x
蒙特卡罗统计方法x
九人x
城门开x
HTML5高级程序设计x
佛祖在一号线x
你可能感兴趣的电影 (更多)
时空急转弯x 来自友邻推荐
星河战队x 来自友邻推荐
杀死比尔2x 来自友邻推荐
搏击俱乐部x 来自豆瓣猜
西游记大结局之仙履奇缘x 来自豆瓣猜
洛城机密x 来自豆瓣猜
心慌方x 来自豆瓣猜
变形金刚x 来自豆瓣猜
指环王1：魔戒再现x 来自豆瓣猜
你可能感兴趣的音乐 (更多)
感动每一刻(DVD+写真书)x 来自豆瓣猜
燕窝HIGH翻昱丁x 来自豆瓣猜
Start世界巡回演唱会x 来自豆瓣猜
Stefanie x 来自豆瓣猜
完美的一天x 来自豆瓣猜
是时候x 来自豆瓣猜
2her x 来自豆瓣猜
自选集x 来自豆瓣猜
孙燕姿 逆光 影音限定版x 来自豆瓣猜
回应
似锦。 1小时前 说：
为什么be silly 和pura有一模一样的鞋子？不老明白滴。。。 回应
Web标准化交流会
人人影视（YYeTs）
独唱团
中信出版社

它是怎么猜的

如果两个人同时对n个电影进行了喜爱程度的排名，如何比较两人对电影喜好是否一致呢

假设按照我对电影的喜爱程度给电影编号为1，2,.....,n，另外一个人对这n个电影的排序是a1，a2,.....,an，如果i<j，但是ai>aj，我们则称ij逆序，我们用两个序列的逆序个数来度量两个序列的相似度



计算逆序

穷举法：列举出所有可能，看一共出现了多少对逆序

暴力求解$O(n^2)$

$a_1,a_2,\ldots\ldots,a_n$

A: $a_1,a_2,\ldots\ldots,a_m$

B: $a_{m+1},a_{m+2},\ldots\ldots,a_n$

A中的逆序个数

B中的逆序个数

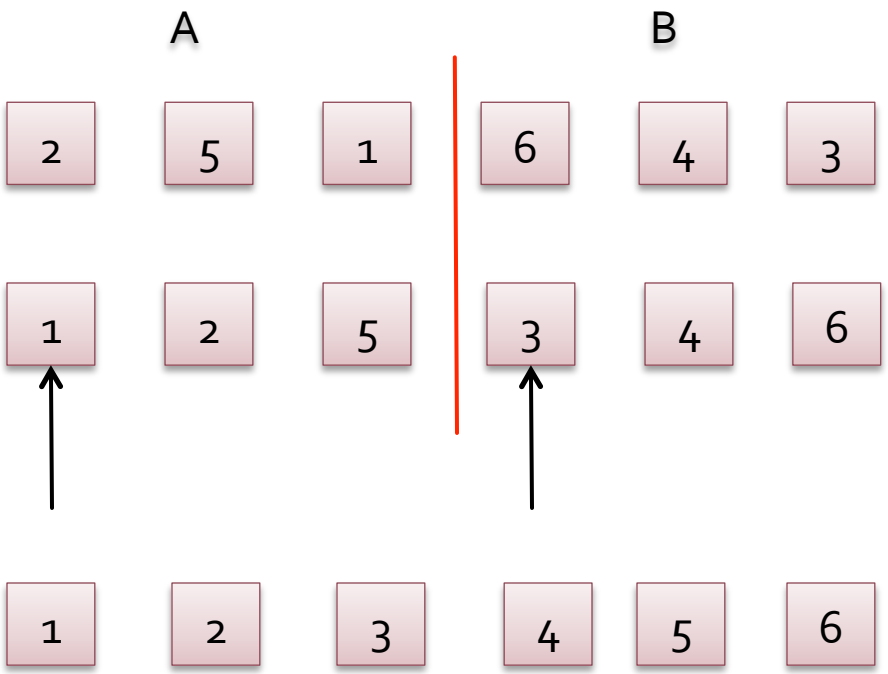
如何找到ai属于A，aj属于B这样的(ai,aj)逆序
我们称为集合A与集合B的逆序数

分解（Divide）

解决
（Conquer）

合并
（Combine

如何高效的计算第三类逆序数



相对于3来说，1 2都不构成逆序

相对于3来说，5构成逆序

相对于4来说，5构成逆序

将A和B中的元素进行归并排序时，如果选择B中元素的时候，A中还有元素，那么A中所剩的元素将和B中选取的这个元素构成逆序，所以当选取B中元素的时候，A中剩余几个元素，就会产生几个逆序，这样通过O(n)的时间就可以算出集合A与集合B的逆序数。

- 每次为了计算两个集合的逆序数都要将两个集合中的元素进行归并排序，计数作为排序的副产品。这样也可以为下一次的计算更大的集合的逆序数做好准备。

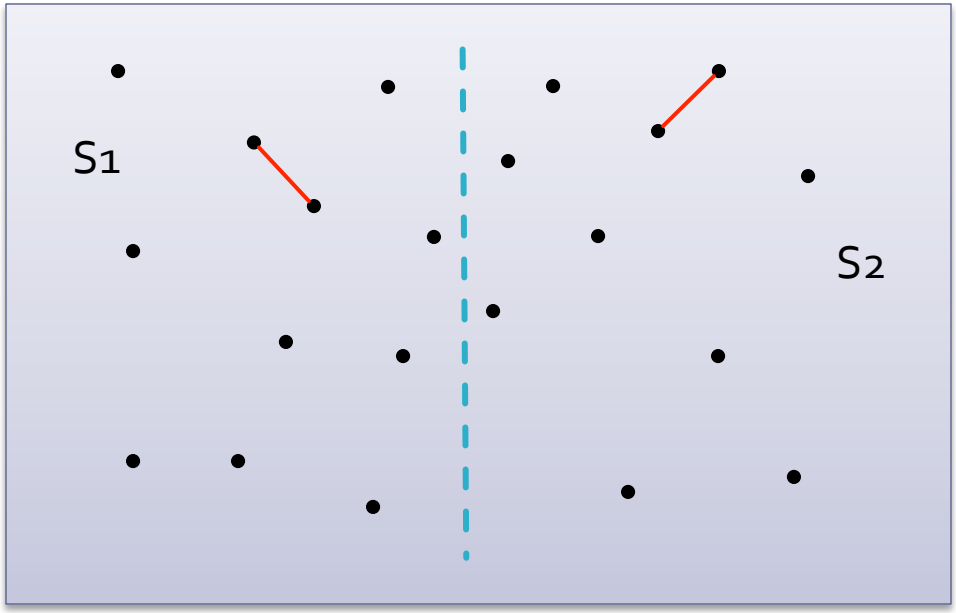
$T(n) = 2T(n/2) + O(n)$

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

最近点对

问题：给出平面上n个点，找出距离最近的一对点。

暴力求解$O(n^2)$



分解（Divide）

解决（Conquer）

合并（Comb

问题简化

问题：给出直线上n个点，找出距离最近的一对点。

时间复杂度
O(nlgn)

[m-d,m+d]

d = min{ |p3-p2| , |q3-q2| }

[m-d,m)中至多有S1中一个点
(m, m+d]中至多有S2中一个点

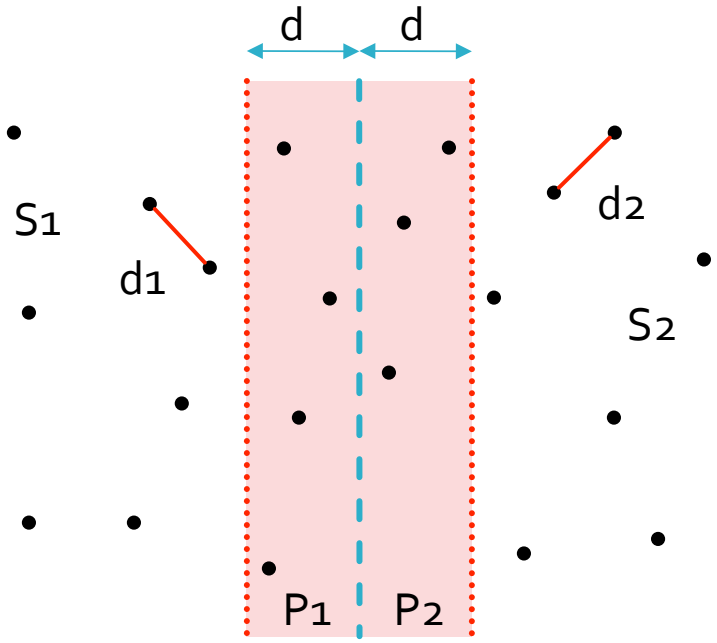
找到p4和q1点所需的时间是O(n)，
确定其中是否包含更近的点的时间是
O(1)

合并的时间是O(n)

T(n)=2T(n/2)+O(n)

回到二维问题

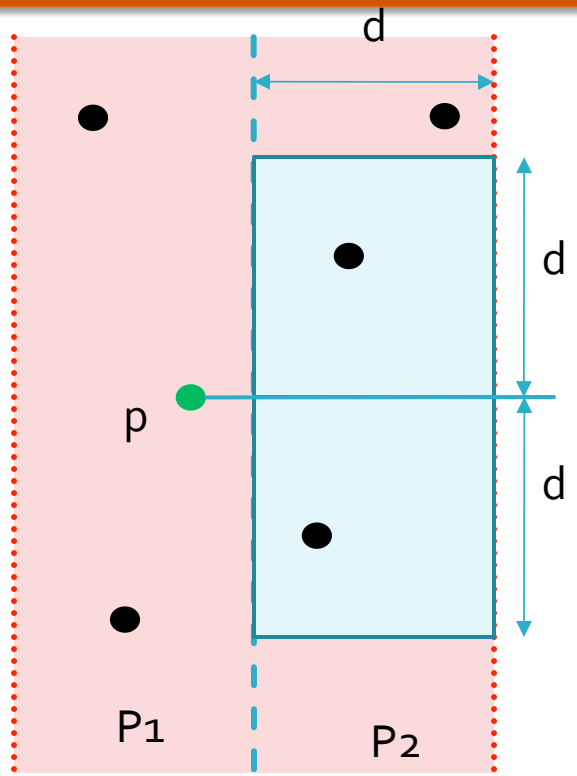
确定筛选区域中的点



d = min{d1 , d2 }

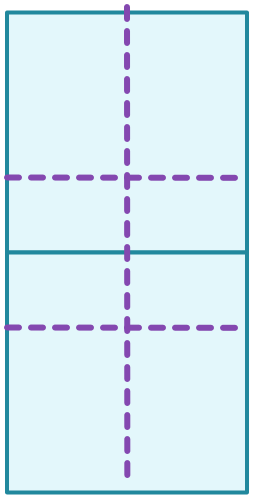
P1中与P2中的点对均构成最近点对的候选者，最多$n^2/4$

确定哪些点对需要检查



在P2中与p构成最近点对候选的点q应该满足dis(p,q)<d

此区域中最多有6个点



总共需要检查n/2*6=3n个点对

合并步骤

确定筛选区域中的点

时间复杂度O(n)

扫描所有的点，纵坐标在m-d到m+d范围内的点需要检查

检查点对的距离中最小的

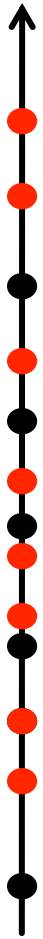
时间复杂度O(n)

对3n个点计算距离，并找出最小的

确定哪些点对需要检查

时间复杂度O(n)

如果在O(n)时间内为P1中的所有点找到需要检查的6个点，那么合并步骤的时间复杂度为O(n)



如果所有点已经按y排序

对于任何一个P1中的点，我们只需考虑在y轴上和他距离最近的6个P2中的点，这个只需要O(n)的时间

合并的时间复杂度O(n)

● P1中的点

● P2中的点

总的时间复杂度

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

每次分解O(n)—点已经按x排序
每次合并O(n)

$S(n) = T(n) + O(nlgn)$

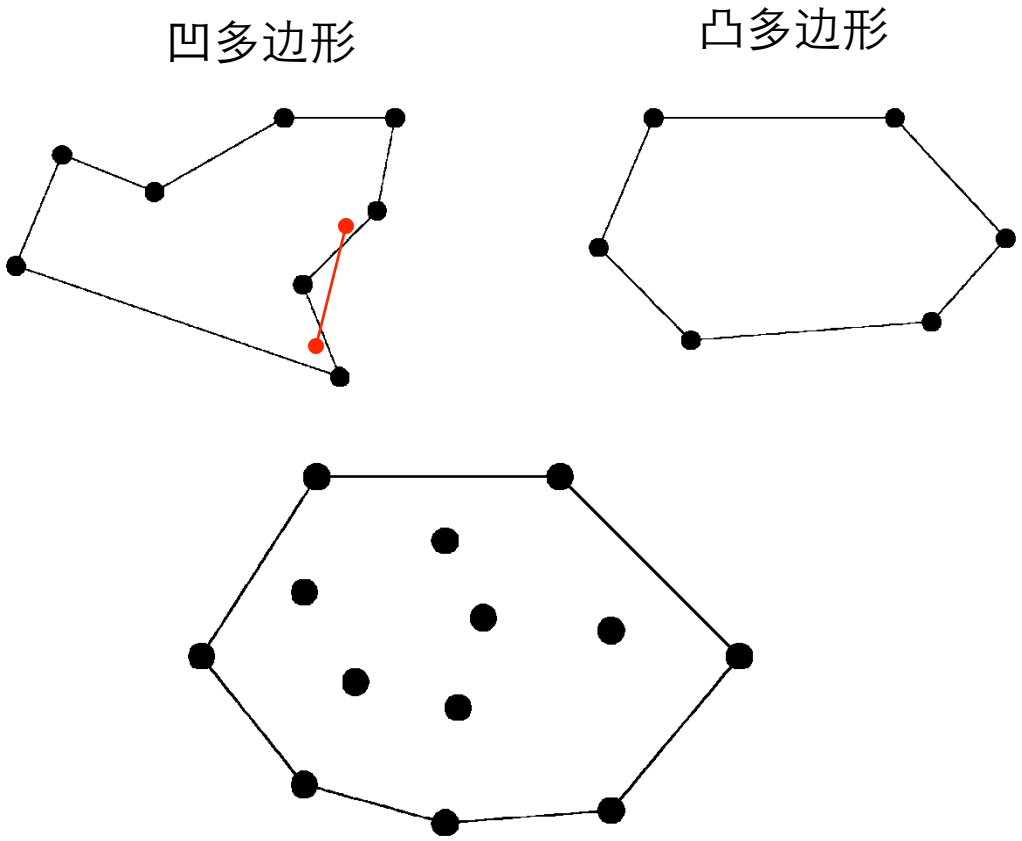
将所有点按y值排序记为listy，时间复杂度O(nlgn)
将所有点按x值排序记为listx，时间复杂度O(nlgn)

$S(n) = O(nlgn)$

凸包（CONVEX HULL）

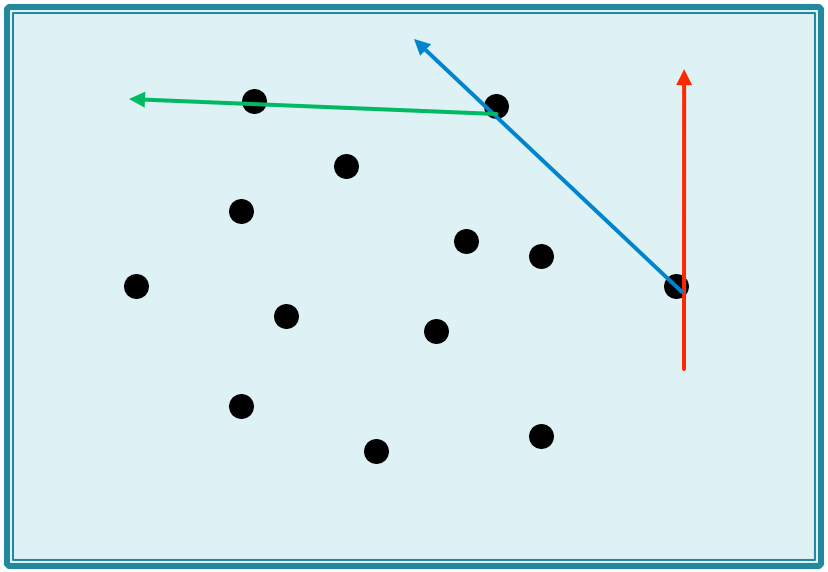
凸多边形：平面多边形内任意两点的连线上的所有点都在这个多边形内部，则称这个多边形是凸的，否则称其为凹的

平面上一个点集的凸包是指包含这些点的最小的凸多边形



问题描述

- 如何用最短的时间找到平面上一个点集的凸包。



时间复杂度
O(nh)

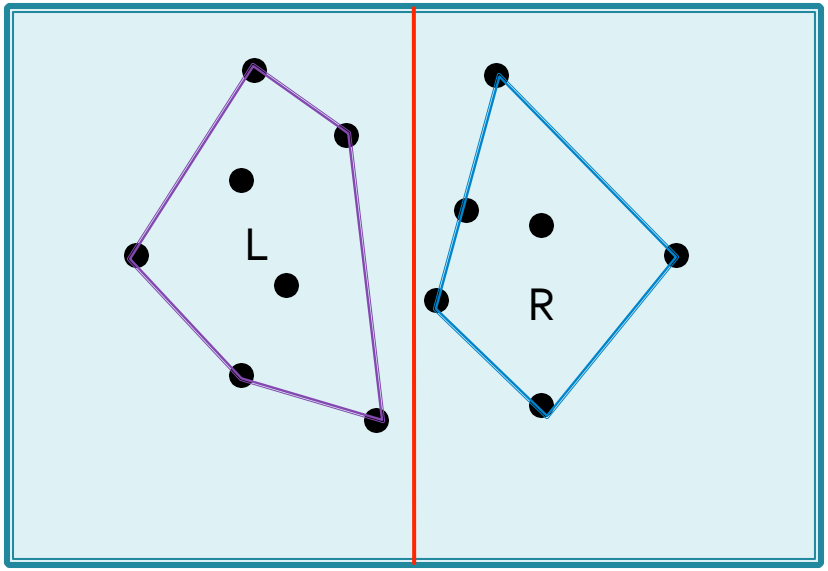
能否采用分治算法

分解
（Divide）

解决
（
Conquer）

合并
（
Combine）

取所有点横坐标的中位数，划为两部分



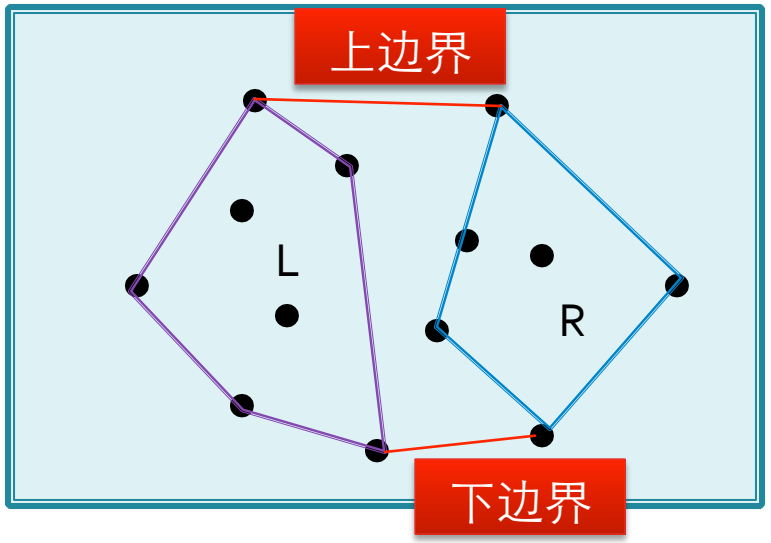
当有三个以下点时，已经是一个凸包了

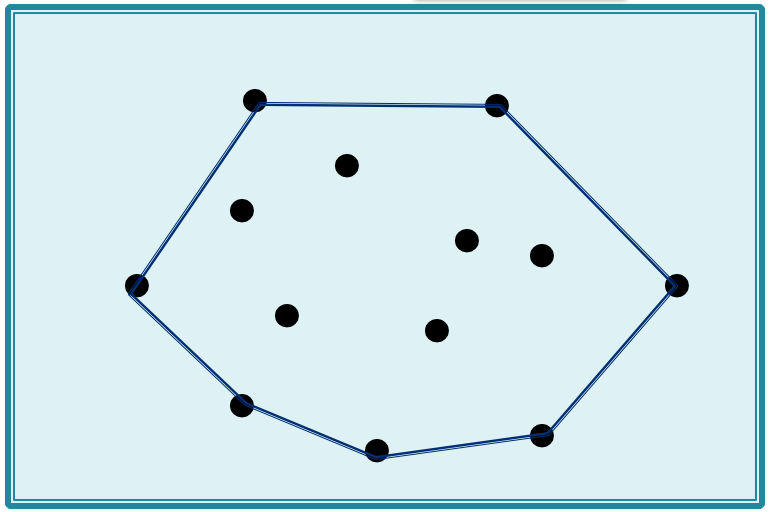
合并两个凸包

如何找到上边界和下边界

我们给L和R中的边界上的点分别按逆时针排序，并作标号

以下边界为例：
A = L中最右侧的点
B = R中最左侧的点
While(AB不是上边界){
    while(AB不是下边界)
        A--
    while(AB不是下边界)
        B++
}

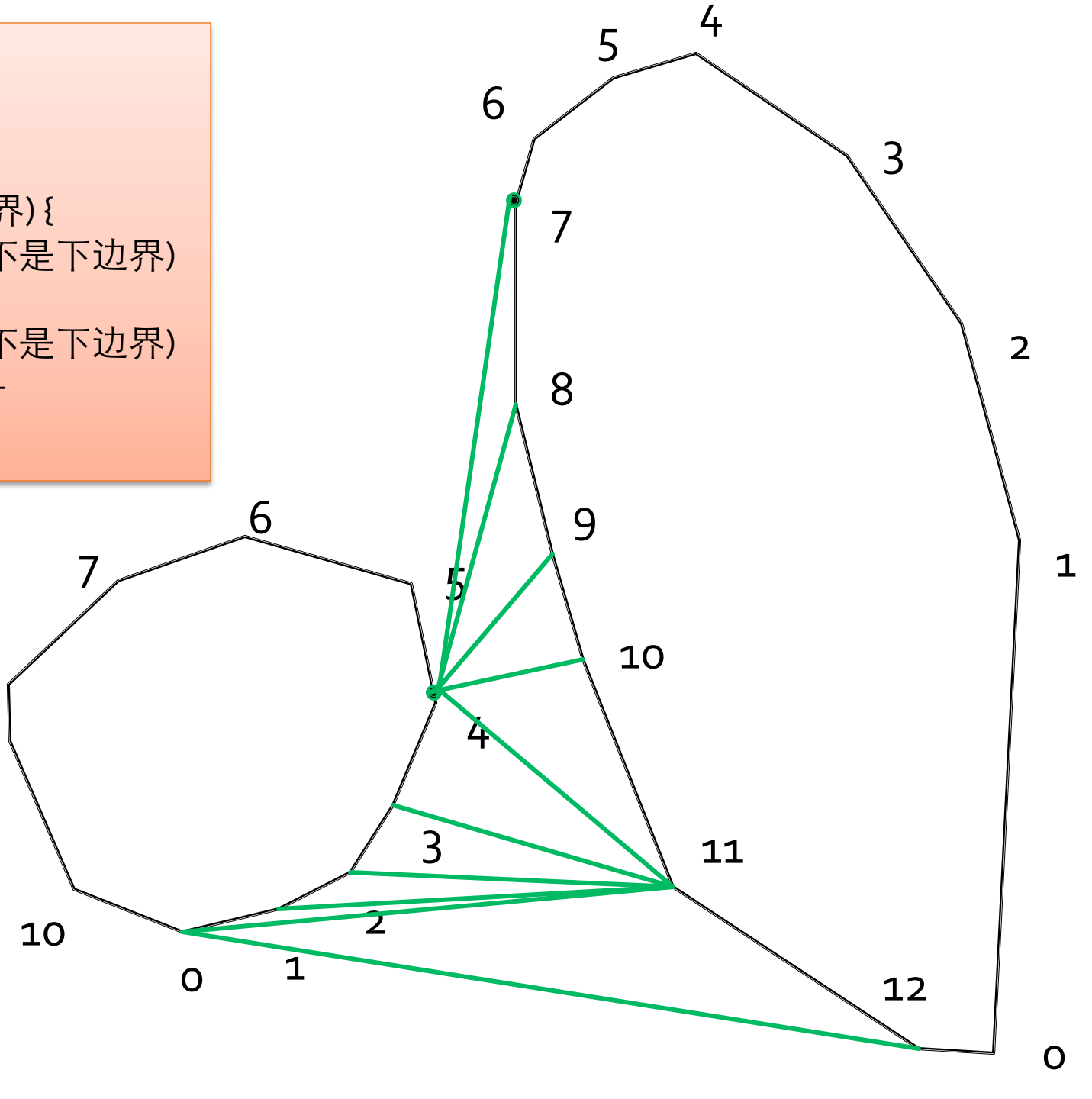




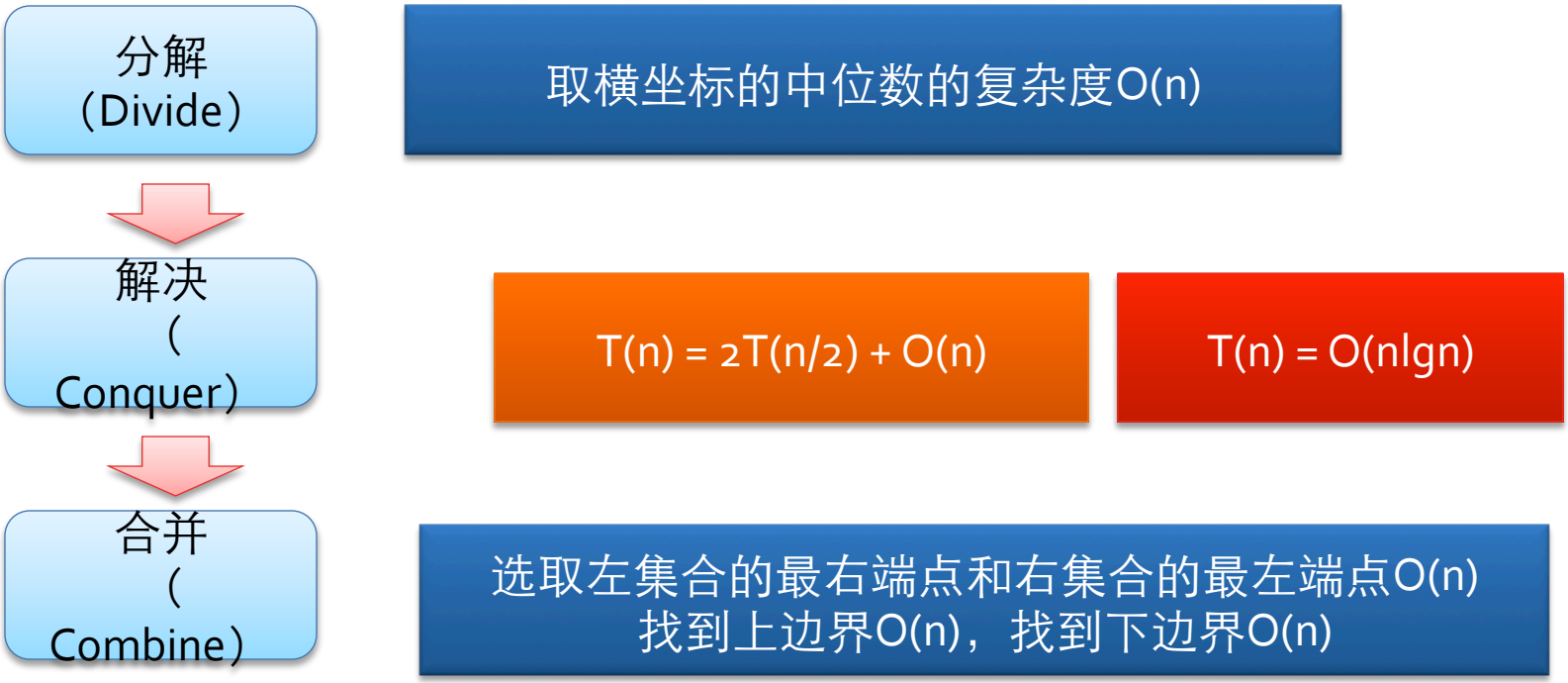
以下边界为例：
A = L中最右侧的点
B = R中最左侧的点
While(AB不是上边界){
while(AB不是下边界)
A--
while(AB不是下边界)
B++
}

B – A ↑

时间复杂度：O(n)

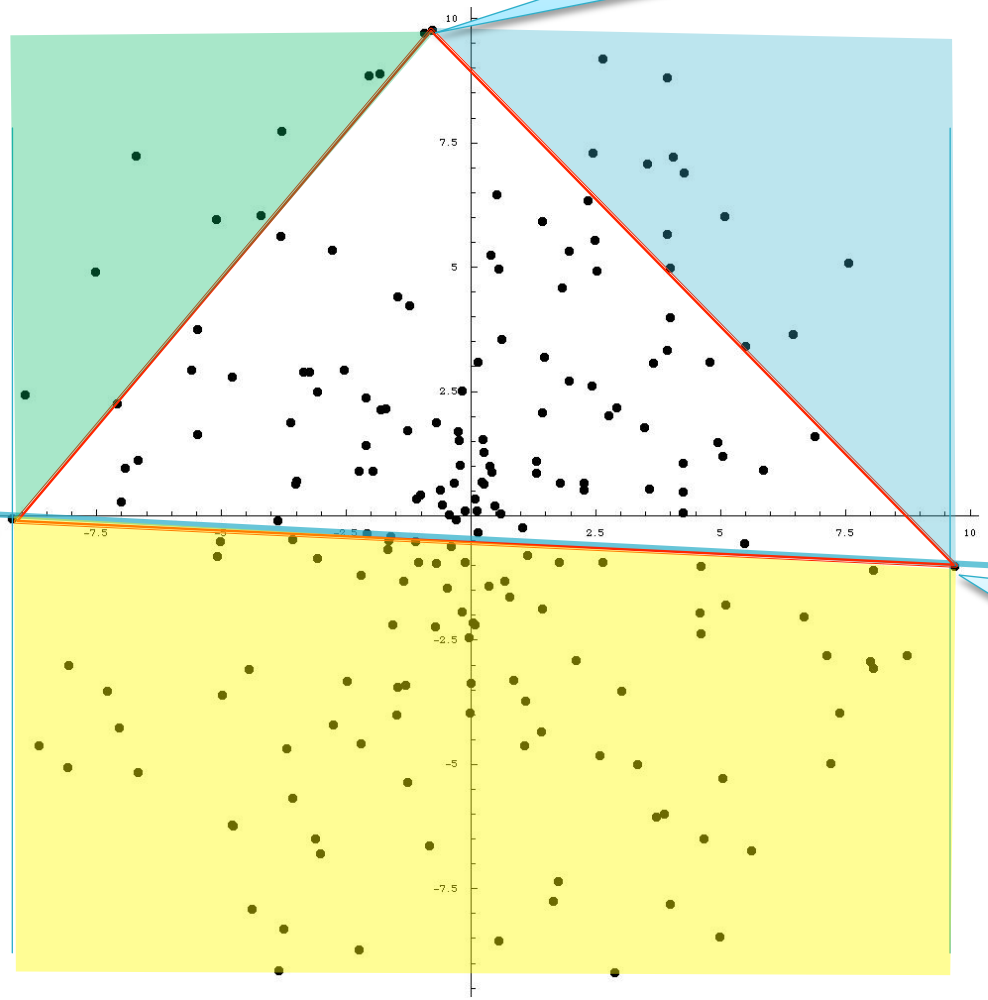


时间复杂度


分解
（Divide）
取横坐标的中位数的复杂度O(n)
解决
（
Conquer）
T(n) = 2T(n/2) + O(n)
T(n) = O(nlgn)
合并
（
Combine）
选取左集合的最右端点和右集合的最左端点O(n)
找到上边界O(n)，找到下边界O(n)

QUCKHULL

离分割线距离最远的点



通过简单的计算那些点一定是构成凸包的点?

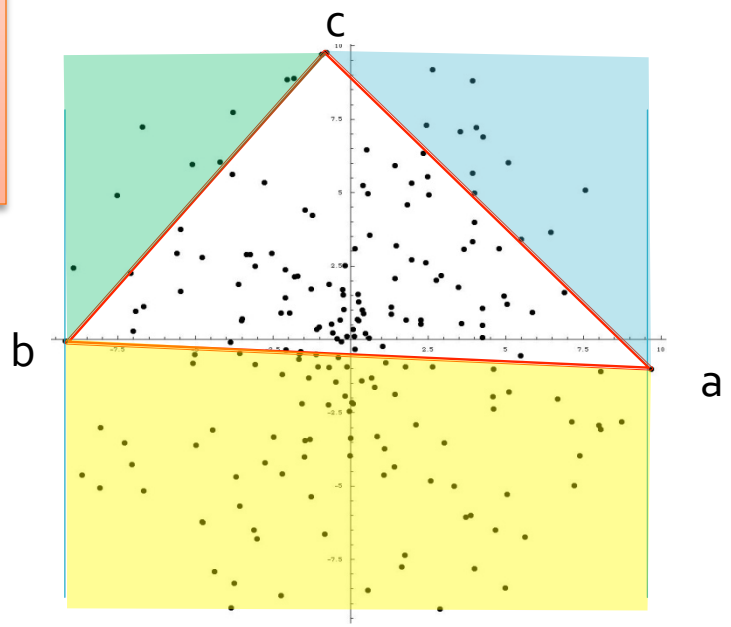
最左端点

最右端点

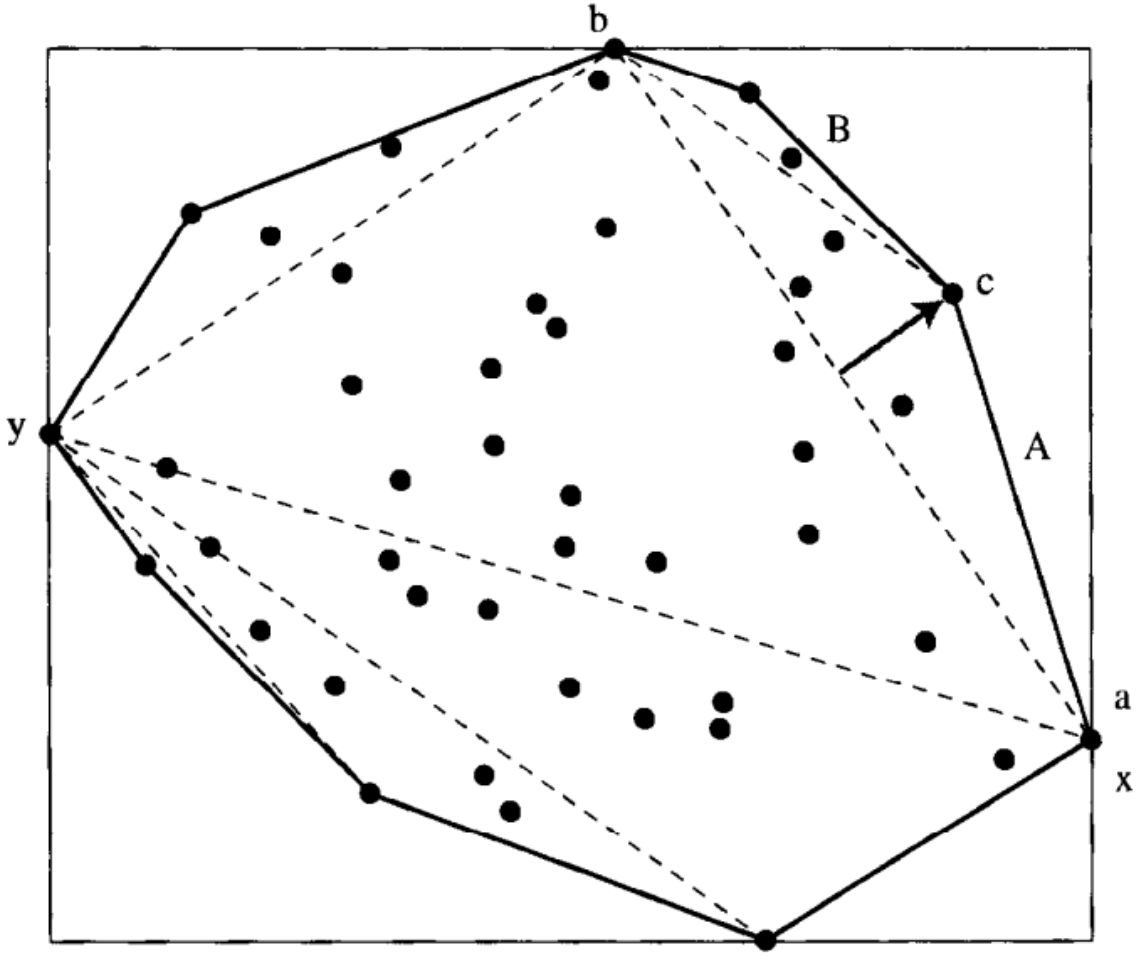
基本算法

Function **QuickHull**(a, b, S)
    if S为空直接返回
    else
        c = S中离ab最远的点
        A = 严格在ac右端的点的集合
        B= 严格在cb右侧的点的集合
        返回 **QuickHull**(a,c,A) + **QuickHull**(c,b,B)

类比**QuickSort**，选择一个标兵x，将数组分为<x的部分与>x的部分，对这两个部分再分别使用QuickSort



算法演示



时间复杂度