

数据结构与算法设计-分治

教材:

[1][殷] 数据结构(C语言版第2版),殷人昆编,清华大学出版社.

[2][王] 王晓东,计算机算法设计与分析(第5版),电子工业.

[3][S] 唐常杰等译, Sipser著, 计算理论导引, 机械工业.

参考资料:

[4][C] 潘金贵等译, Cormen等著, 算法导论, 机械工业.

[5][M] 黄林鹏等译, Manber著, 算法引论-一种创造性方法, 电子.

[6][L] Lewis等著, 计算理论基础, 清华大学.

符号

- 2 对数

$$\log n = \log_2 n, \quad \lg n = \log_{10} n \quad \lg 2 = 0.301$$

$$\log^k n = (\log n)^k$$

$$\log \log n = \log(\log n)$$

性质：

$$a^{\log_b n} = n^{\log_b a}$$

$$\log_k n = c \log_l n$$

符号

- 证明:

$$\begin{aligned} a^{\log_b n} &= b^{\log_b a^{\log_b n}} \\ &= b^{\log_b n \cdot \log_b a} \\ &= (b^{\log_b n})^{\log_b a} \\ &= n^{\log_b a} \end{aligned}$$

符号

- $\log_b N = \log_a N / \log_a b$
- $\log_e N = \ln N$
- $e = 2.71828$
- $\log_a b = 1 / \log_b a$

符号

- 二、阶乘

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log n! = \Theta(n \log n)$$

符号

- 三、求和

等比级数的求和公式

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$$

分治算法

主要内容:

1. 分治原理, 主定理, 二分法
2. 线性时间选择
3. 最大子段和
4. 最接近点对问题

快速排序

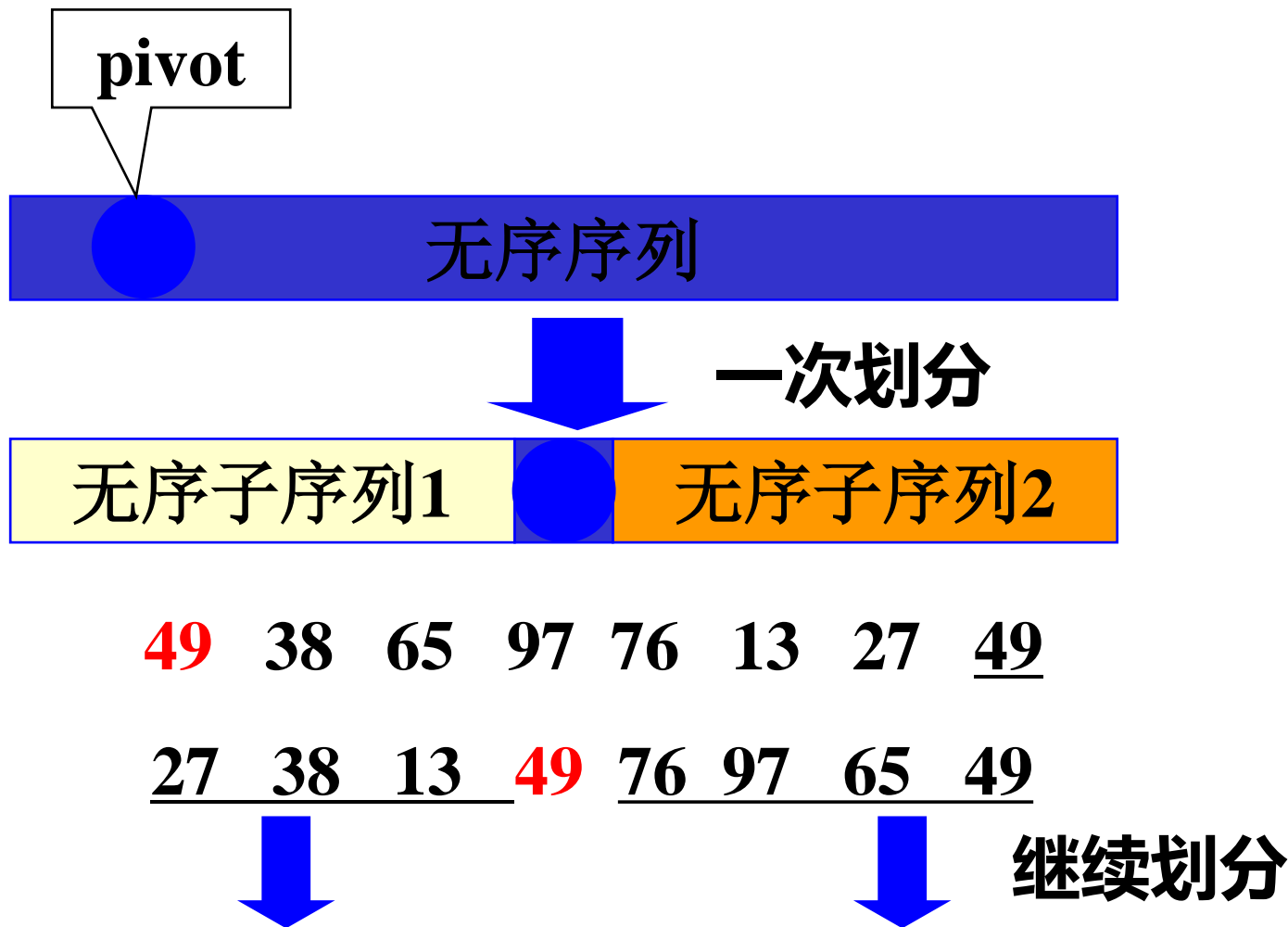
❖ 基本思想:

- 通过一趟排序将待排序记录分割成两个部分
- 一部分记录的关键字比另一部分的小。
- 选择一个关键字作为分割标准，称为pivot

❖ 基本操作:

- 选定一记录R (pivot)，将所有其他记录关键字 k' 与该记录关键字 k 比较
 - 若 $k' < k$ 则将记录换至R之前;
 - 若 $k' > k$ 则将记录换至R之后;
- 继续对R前后两部分记录进行快速排序，直至排序范围为1;

快速排序



时间复杂度分析

最坏情况分析: 每次分成大小为1和n-1的两段

$$T(n) = \begin{cases} O(1) & n \leq 1 \\ T(n-1) + O(n) & n > 1 \end{cases}$$

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

最好情况分析: 每次分成大小为n/2的两段

$$T(n) = \begin{cases} O(1) & n \leq 2 \\ 2T(n/2) + O(n) & n > 2 \end{cases}$$

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

快速排序

- ❖ 快速排序的基本思想是基于分治策略的。对于输入的子序列 $L[p..r]$ ，分三步处理：
- ❖ **1、分解(Divide)：**将待排序列 $L[p..r]$ 划分为两个非空子序列 $L[p..q]$ 和 $L[q+1..r]$ ，使前面任一元素的值不大于后面元素的值。
- ❖ 途径实现：在序列 $L[p..r]$ 中选择数据元素 $L[q]$ ，经比较和移动后， $L[q]$ 将处于 $L[p..r]$ 中间的适当位置，使得数据元素 $L[q]$ 的值小于 $L[q+1..r]$ 中任一元素的值。

快速排序

- ❖ **2、递归求解(Conquer)**: 通过递归调用快速排序算法, 分别对 $L[p..q]$ 和 $L[q+1..r]$ 进行排序。
- ❖ **3、合并(Merge)**: 由于对分解出的两个子序列的排序是就地进行的, 所以在 $L[p..q]$ 和 $L[q+1..r]$ 都排好序后不需要执行任何计算 $L[p..r]$ 就已排好序, 即自然合并。
- ❖ 这个解决流程是符合分治法的基本步骤的。因此, 快速排序法是分治法的经典应用实例之一。

归并排序

❖ 归并：

- 将两个或两个以上有序表组合成一个新的有序表。


❖ 2-路归并排序：

- 设初始序列含有 n 个记录，则可看成 n 个有序的子序列，每个子序列长度为1。
- 两两合并，得到 $\left\lfloor \frac{n}{2} \right\rfloor$ 个长度为 2 或 1 的有序子序列。
- 再两两合并，……如此重复，直至得到一个长度为 n 的有序序列为止。


归并排序

例


初始关键字: [49] [38] [65] [97] [76] [13] [27]



一趟归并后: [38 49] [65 97] [13 76] [27]



二趟归并后: [38 49 65 97] [13 27 76]



三趟归并后: [13 27 38 49 65 76 97]



归并排序

A: 1 3 8 9 11

B: 2 5 7 10 13

C: 1 2 3 5 7 8 9 10 11 13

归并排序

```
void Merge(T A[], int Alen, T B[], int Blen, T C[]){  
    int i=0,j=0,k=0;  
    while(i < Alen && j < Blen){  
        if(A[i] < B[j])  
            C[k++] = A[i++];  
        else  
            C[k++] = B[j++];  
    }  
    while(i < Alen)  
        C[k++] = A[i++];  
    while(j < Blen)  
        C[k++] = B[j++];  
}
```


归并排序

```
void MergeSort(int A[], int B[], int l, int h)
```

```
{
```

```
    if(l == h)
```

```
        return;
```

```
    int m = (l+h)/2;
```

```
    MergeSort(A, B, l, m);
```

```
    MergeSort(A, B, m+1, h);
```

```
    Merge(A, B, l, m, h);
```

```
}
```

$$T(n) = \begin{cases} 1 & n=2 \\ 2T(n/2) + cn & n>2 \end{cases}$$

归并排序

❖ 时间复杂度:

- 共进行 $\lceil \log_2 n \rceil$ 趟归并, 每趟对 n 个记录进行归并
- 所以时间复杂度是 $O(n \log n)$

❖ 空间复杂度:

- $O(n)$

❖ 稳定性:

- 稳定

分治基本过程

分: 将问题分解成若干个子问题

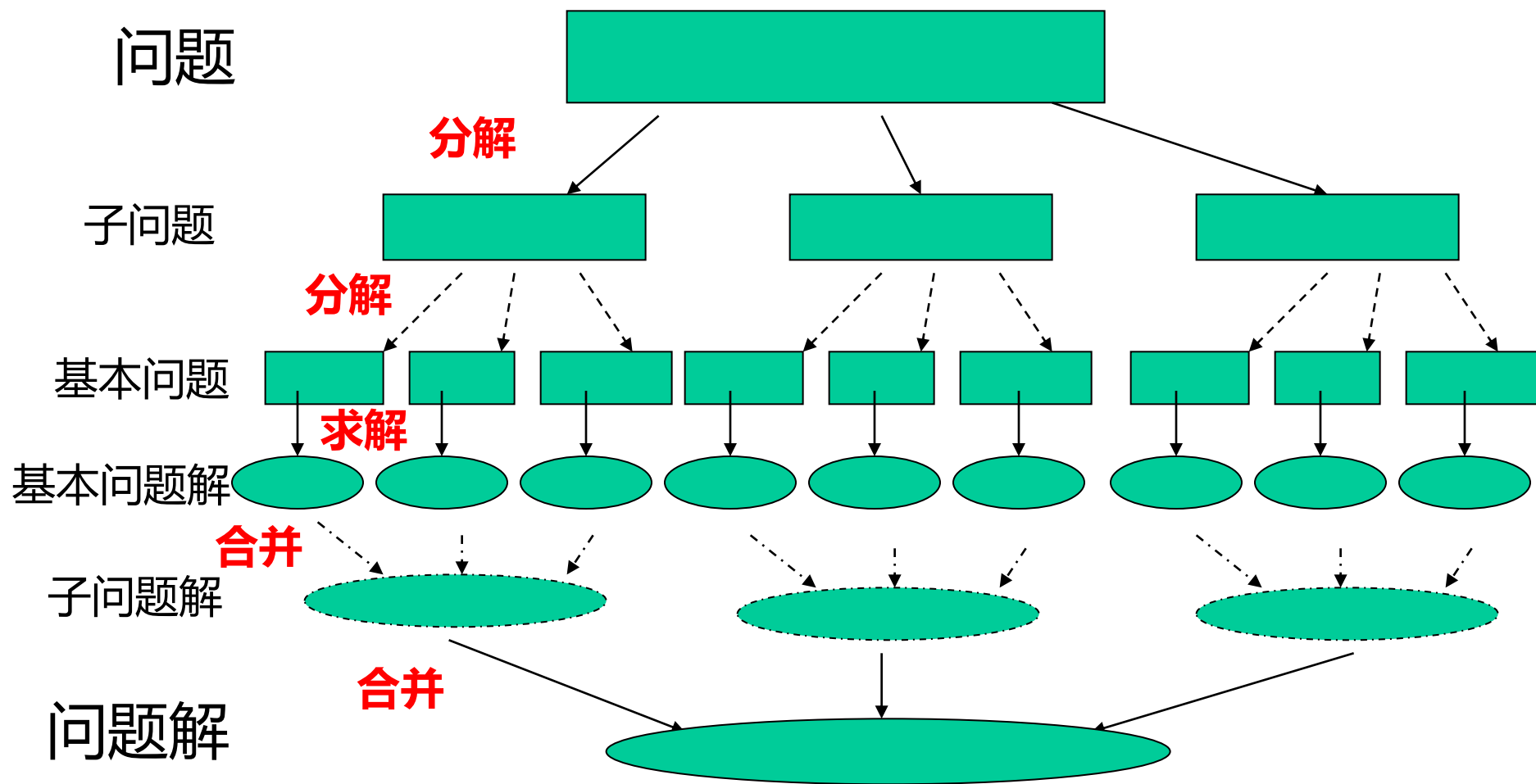
治: 递归求解子问题

合: 由子问题解合并得到原问题解

divide-and-conquer(P)

```
{ if ( | P | <= n0) adhoc(P);      //解决小规模的问题
  else
  { divide P into  $P_1, P_2, \dots, P_a$ ;    //分解问题
    for (i=1, i<=a, i++)
       $y_i = \text{divide-and-conquer}(P_i)$ ; //递归的解各子问题
    return merge( $y_1, \dots, y_a$ );      //合并出原问题的解
  }
}
```

分治过程图示



分治的原则

- 子问题相互独立(为什么), 无重复
若有大量重复子问题, 改用动态规划
- 子问题规模(n/b)大致相等(平衡思想)
- 子问题和原问题类似, 可递归求解
- 子问题解合并能得到原问题解

设分解出的子问题有 a 个, 时间复杂度 $T(n)$, 分解+合并时间 $f(n)$:

$$T(n) = \begin{cases} O(1) & n \leq n_0 \\ aT(n/b) + f(n) & n > n_0 \end{cases}$$

```
divide-and-conquer(P)
{ if ( | P | <= n0) adhoc(P);
  else
  { divide P into P1, P2,..., Pa;
    for (i=1,i<=a,i++)
      yi=divide-and-conquer(Pi);
    return merge(y1,...,ya);
  } }
```

分治中经常出现的递推关系

设 $a \geq 1$, $b \geq 2$, 分治中经常出现

$$T(n) = \begin{cases} O(1) & n \leq n_0 \\ aT(n/b) + f(n) & n > n_0 \end{cases}$$

教材中的公式 (Page 17)

$$T(n) = n^{\log_b a} + \sum_{j=0}^{\log_b n / n_0} a^j f(n / b^j)$$

这个公式有时使用不是很方便, 介绍分治主定理

分治主定理([M]Page37)

设 $a \geq 1, b \geq 2$

$$T(n) = \begin{cases} O(1) & n \leq n_0 \\ aT(n/b) + cn^k & n > n_0 \end{cases}$$

则

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & a > b^k \text{ or } k < \log_b a \\ \Theta(n^{\log_b a} \log n) & a = b^k \text{ or } k = \log_b a \\ \Theta(n^k) & a < b^k \text{ or } k > \log_b a \end{cases}$$

注:[M]中为大O记号, 无详细证明. 证明见附录.

分治主定理([C]第4章)

设 $a \geq 1, b \geq 2$

$$T(n) = \begin{cases} O(1) & n \leq n_0 \\ aT(n/b) + f(n) & n > n_0 \end{cases}$$

则

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{若 } f(n) = O(n^{\log_b a - \varepsilon}), \varepsilon > 0 \\ \Theta(n^{\log_b a} \log n) & \text{若 } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{若 } f(n) = \Omega(n^{\log_b a + \varepsilon}), \varepsilon > 0 \end{cases}$$

注:[C]中有详细证明.

推广

$$T(n) = \begin{cases} b & n = 1 \\ T(\lfloor c_1 n \rfloor) + T(\lfloor c_2 n \rfloor) + bn & n > 1 \end{cases}$$

$$T(n) = \begin{cases} \Theta(n \log n) & c_1 + c_2 = 1 \\ \Theta(n) & c_1 + c_2 < 1 \end{cases}$$

特别地，当 $c_1 + c_2 < 1$ 时，有

$$T(n) \leq bn / (1 - c_1 - c_2) = O(n)$$

二分法

输入: 实数序列 a_1, \dots, a_n , 性质 P (关于序列单调)

输出: 满足性质 P 的临界点位置

例1: 输入序列($a_1 < \dots < a_n$)和 m , 判断 m 是否在序列中

枚举: 时间复杂度为 $O(n)$

二分法: 运算1次, 解范围缩小一半

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

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

条件: 性质 P 满足单调性

分治法求n元集最大最小元素

- 假设 $n=2^m$ 。要求每次平分成2个子集。

```
void maxmin(int A[],int &e_max,int &e_min,int low,int high)
2. {
3.     int mid,x1,y1,x2,y2;
4.     if ((high-low <= 1)) {
5.         if (A[high]>A[low]) {
6.             e_max = A[high];
7.             e_min = A[low];
8.         }
9.     } else {
10.         e_max = A[low];
11.         e_min = A[high];
12.     }
13. }
```

分治法求n元集最大最小元素

```
14.     else {  
15.         mid = (low + high) / 2;  
16.         maxmin(A,x1,y1,low,mid);  
17.         maxmin(A,x2,y2,mid+1,high);  
18.         e_max = max(x1,x2);  
19.         e_min = min(y1,y2);  
20.     }  
21. }
```

$$T(n) = \begin{cases} 1 & n=2 \\ 2T(n/2)+2 & n>2 \end{cases}$$

迭代法

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

$$= 2[2T(n/2^2) + 2] + 2$$

$$= 2^2 T(n/2^2) + 2(1 + 2)$$

$$= 2^3 T(n/2^3) + 2(1 + 2 + 2^2) = \dots$$

$$n = 2^m$$

$$= 2^{m-1} T(2) + 2(1 + 2 + \dots + 2^{m-2})$$

$$= 2^{m-1} + 2[1(1 - 2^{m-1}) / (1 - 2)]$$

$$= 2^{m-1} + 2^m - 2$$

$$= 3n/2 - 2$$

课堂练习

$$T(n)=3T(n/2) \quad T(1)=1 \quad n=2^m$$

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

答案

$$T(n)=3T(n/2) \quad T(1)=1$$

$$\begin{aligned} n=2^k \quad &= 3T(2^{k-1}) \\ &= 3^2T(2^{k-2}) \\ &= \dots \\ &= 3^kT(2^{k-k}) \\ &= 3^kT(1) \\ &= 3^k \end{aligned}$$

$$k=\log_2 n$$

课堂练习

用分治法求n个元素集合S中的最大、最小元素。写出算法，并分析时间复杂性（比较次数）。

假设 $n=3^m$ 。要求每次平分成3个子集。

$$T(n) = \begin{cases} 3 & n=3 \\ 3T(n/3)+4 & n>3 \end{cases}$$

$$T(n) = 5n/3 - 2$$

平分成2个子集 $T(n) = 3n/2 - 2$

求n元集最大最小元素

思考题

不用（递归的）分治法求n个元素集合S中的最大、最小元素，使得

$T(n)$ 仍为 $3n/2-2$ 。

这里假设 $n=2^m$ 。

求最小元素

算法

```
int FindMin( Array[], int Len)
{
    int MinIndex = 1;
    for(int i = 2; i <= Len; i++){
        if(Array[MinIndex] > Array[i]) MaxIndex = i;
    }
    return MinIndex;
}
```

求最小元素

● 最小问题

问题下界：假设集合中元素是互不相同的。则 $n-1$ 个元素不是最小元素。

对某一个元素，只有它在某一次比较中失败了，才能确定它不是最小元素。因此，有 $n-1$ 个元素在某次失败

每一次比较只能确定一个失败者，确定 $n-1$ 个在某次比较中的失败者需要 $n-1$ 次比较

确定最小元素至少需要 $n-1$ 次比较， $n-1$ 次比较是最小问题的下界

- 前面算法的比较次数是 $n-1$ 次，达到问题的下界，因此它是最优算法

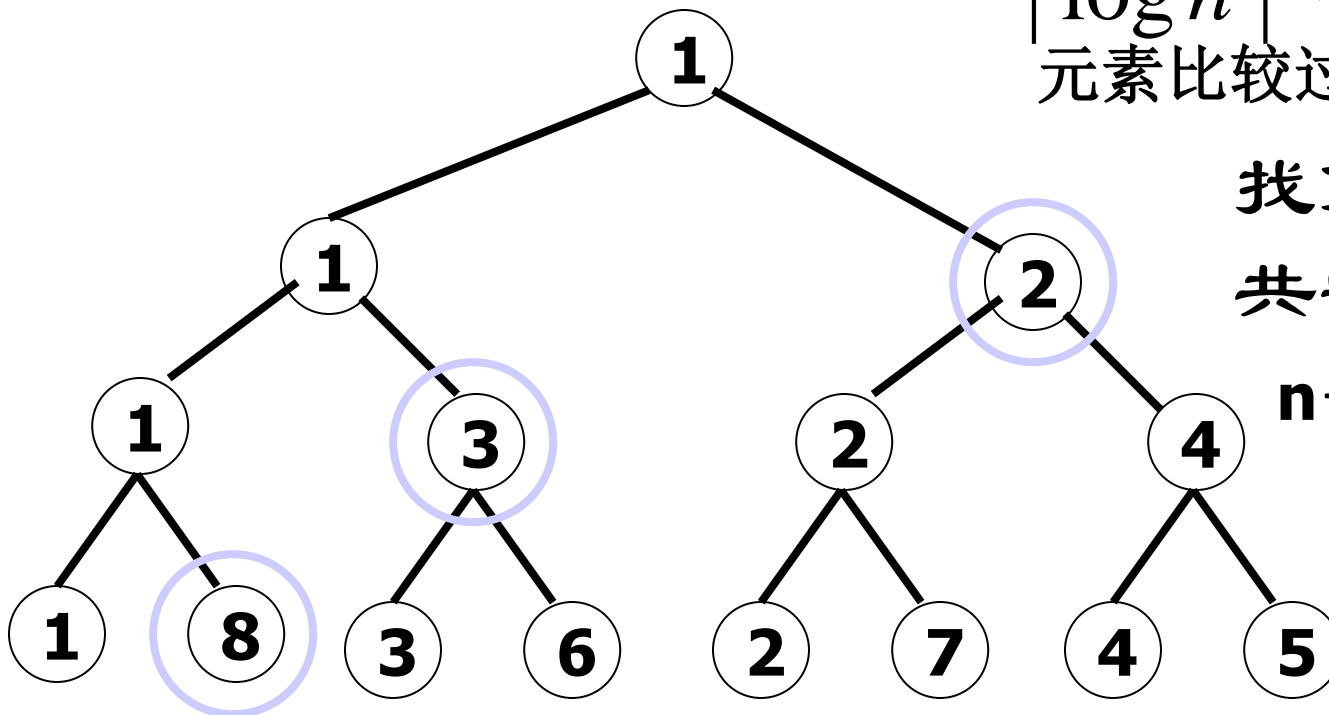
求第2小的元素

- 一般情况下 $2n-3$ 次比较
- 第2小元素一定存在于同最小元素比较过的元素之中

$\lceil \log n \rceil$ 个元素同最小元素比较过

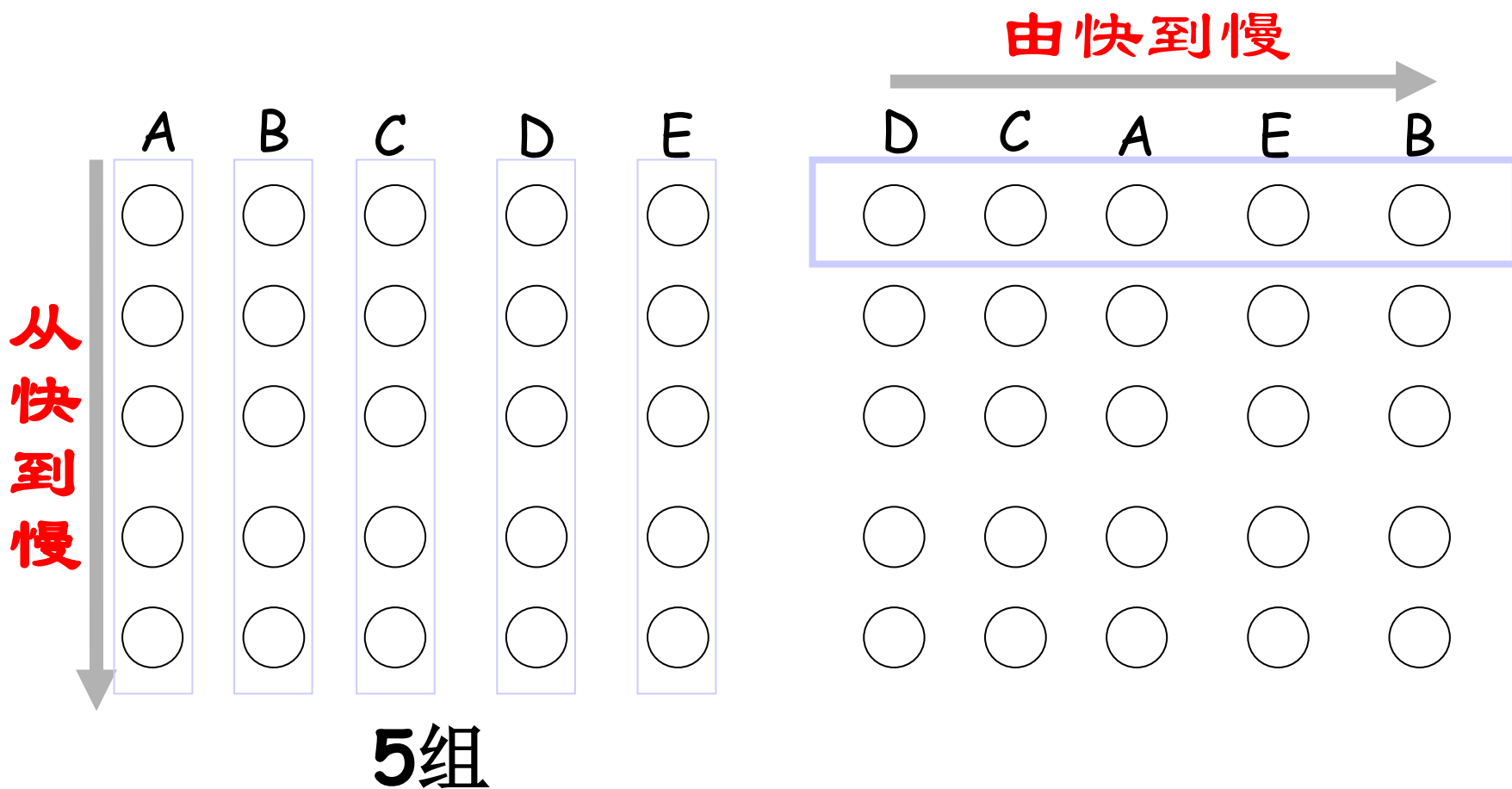
找第2小的元素
共需比较次数

$$n-1 + \lceil \log n \rceil - 1$$



引例

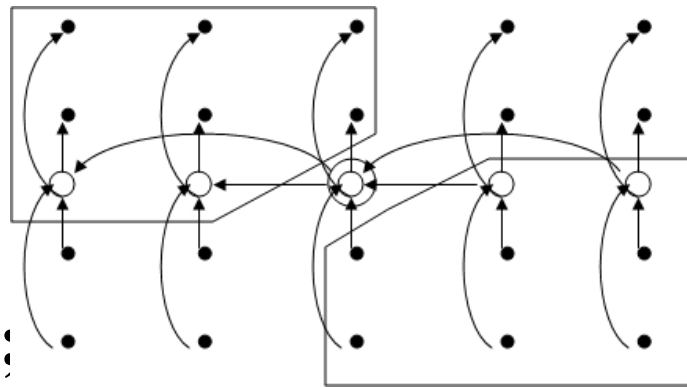
- 在中国的古代，25匹马通过赛跑来决出前3名，每5匹马一组，问最少需要几组？



线性时间选择算法

线性时间选择算法Select:

1. 将 n 个数划分成 $\lceil n/5 \rceil$ 组, 取出每组中位数(共 $\lceil n/5 \rceil$ 个),
2. 使用Select找这 $\lceil n/5 \rceil$ 个数的中位数
3. 以这个数为基准划分
4. 选一个部分继续执行Select



比如 **if** ($|S_1| \geq k$) **return**(Select(k, S_1));

else if ($|S_1| + |S_2| \geq k$) **return**(x);

else **return**(Select($k - |S_1| - |S_2|, S_3$))

线性时间选择

按递增顺序，找出下面29个元素的第18小元素：

8,31,60,33,17,4,51,57,49,35,11,43,37,3,13,
52,6,19,25,32,54,16,5,41,7,23, 22,46,29。

线性时间选择的一种实现方式

29个元素第18小：8,31,60,33,17,4,51,57,49,35,11,43,37,3,13,52,6,19,25,32,54,16,5,41,7,23,22,46,29。

前面25个元素划分为5组：(8,31,60,33,17),
(4,51,57,49,35), (11,43,37,3,13), (52,6,19,25,32),
(54,16,5,41,7)，其余4个元素暂不处理；

提取每一组的中值构成集合：(31,49,13,25,16)

递归求得 $x=25$ ；

{8, 17, 4, 11, 3, 13, 6, 19, 16, 5, 7, 23, 22} , 13
{25} , 1

{31, 60, 33, 51, 57, 49, 35, 43, 37, 52, 32, 54, 41, 46, 29} ; 15

线性时间选择程序

```
1 template <class Type>
2 Type Select(Type a[], int p, int r, int k)
3 {   if( r - p < 75 ) { 直接对数组a[p:r]排序; return a[p+k-1];}
4     for( int i = 0; i <= (r - p - 4) / 5 ; i++ ) //分 $\lceil n/5 \rceil$ 组, 取各组中位数
5         将a[p+5*i]至a[p+5*i+4]的第3小元素与a[p+i]交换位置;
6     Type x = Select(a,p,p+(r-p-4)/5, (r-p-4)/10); //取中位数的中位数,  $T(n/5)$ 
7     int i = Partition(a,p,r,x), j = i - p + 1;
8     if ( k == j ) return a[i];
9     elseif ( k < j ) return Select(a,p,i-1,k); //选择左片递归, 最多 $T(3n/4)$ 
10    else return Select(a,i+1,r,k-j); //选择右片递归, 最多 $T(3n/4)$ 
11 }
```

$$T(n) = \begin{cases} O(1) & n < 75 \\ T(n/5) + T(3n/4) + O(n) & n \geq 75 \end{cases} = O(n)$$

线性选择实现方式

29个元素第18小： 8,31,60,33,17,4,51,57,49,35,11,43,37,3,13,52,6,19,25,32,54,16,5,41,7,23, 22,46,29。

```
for( int i = 0; i <= (r - p - 4) / 5 ; i++ ) //分 $\lceil n/5 \rceil$ 组, 取各组中位数
{
    将a[p+5*i]至a[p+5*i+4]的各组分别排序;
    将a[p+5*i]至a[p+5*i+4]的第3小元素与a[p+i]交换位置;}
```

8	8	31	31	4	31	4	31	4
31	17	17	17	51	17	35	49	35
60	31	8	8	57	8	49	8	17
33	33	33	33	49	33	51	33	51
17	60	60	60	35	60	57	60	57

线性选择实现方式

29个元素第18小:

```
for( int i = 0; i <= (r - p - 4) / 5 ; i++ ) //分 $\lceil n/5 \rceil$ 组, 取各组中位数
{
    将a[p+5*i]至a[p+5*i+4]的各组分别排序;
    将a[p+5*i]至a[p+5*i+4]的第3小元素与a[p+i]交换位置;}
```

31 4 3 6 5

49 35 11 19 7

13 17 8 33 60

25 51 37 32 41

16 57 43 52 54

线性选择实现方式

29个元素第18小： 8,31,..., ,41,7, **23, 22,46,29**。

$x = \text{Select}(a, p, p + (r - p - 4) / 5, (r - p - 4) / 10);$ //取中位数的中位数

31 4 3 6 5

49 35 11 19 7

13 17 8 33 60

25 51 37 32 41

16 57 43 52 54

$x = 25$

$a[29] = \{31, 49, 13, 25, 16, 4, 35, 17, 51, 57, 3, 11, 8, 37, 43, 6, 19, 33, 32, 52, 5, 7, 60, 41, 54, \mathbf{23, 22, 46, 29}\}$

线性选择实现方式

int i = Partition(a,p,r,x), j = i - p + 1;

x = 25

**a[29]={31,49,13,25,16,4,35,17,51,57,3,11,8,37,
43,6,19,33,32,52,5,7,60,41,54,23,22,46,29}**

**a[29]={22,23,13,7,16,4,5,17,19,6,3,11,8,
25**a[13]**,43,37,57,33,32,52,51,35,
60,41,54,49,31,46,29}** **i = 13**

线性选择实现方式

29个元素第18小:

$a[29] = \{22, 23, 13, 7, 16, 4, 5, 17, 19, 6, 3, 11, 8,$
 $25, a[13], 43, 37, 57, 33, 32, 52, 51, 35, 60, 41, 54, 49, 3$
 $1, 46, 29\} \quad x = 25 \quad i = 13$

$j = i - p + 1;$

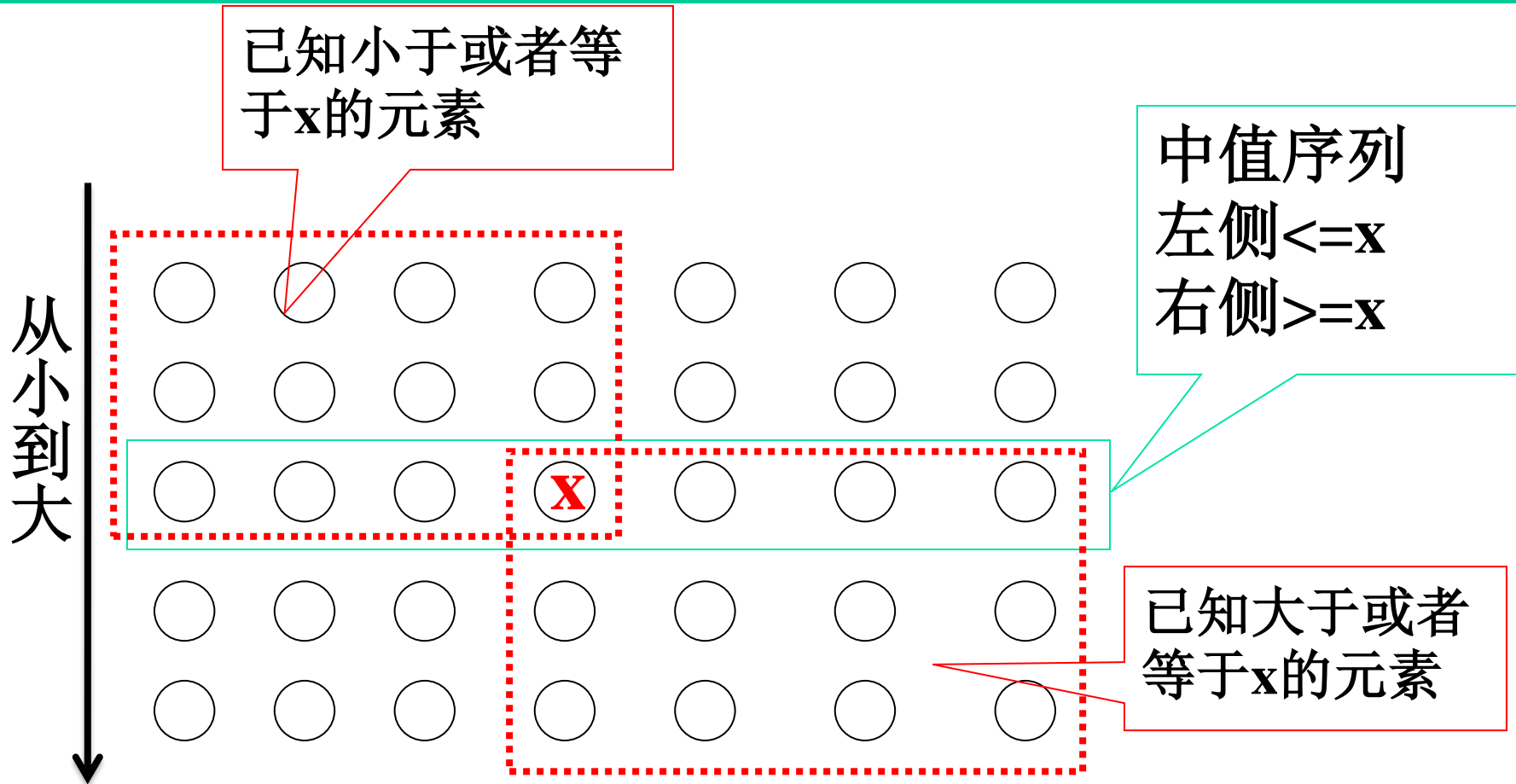
$\text{if } (k == j) \text{ return } a[i];$

$\text{else if } (k < j) \text{ return Select}(a, p, i-1, k);$

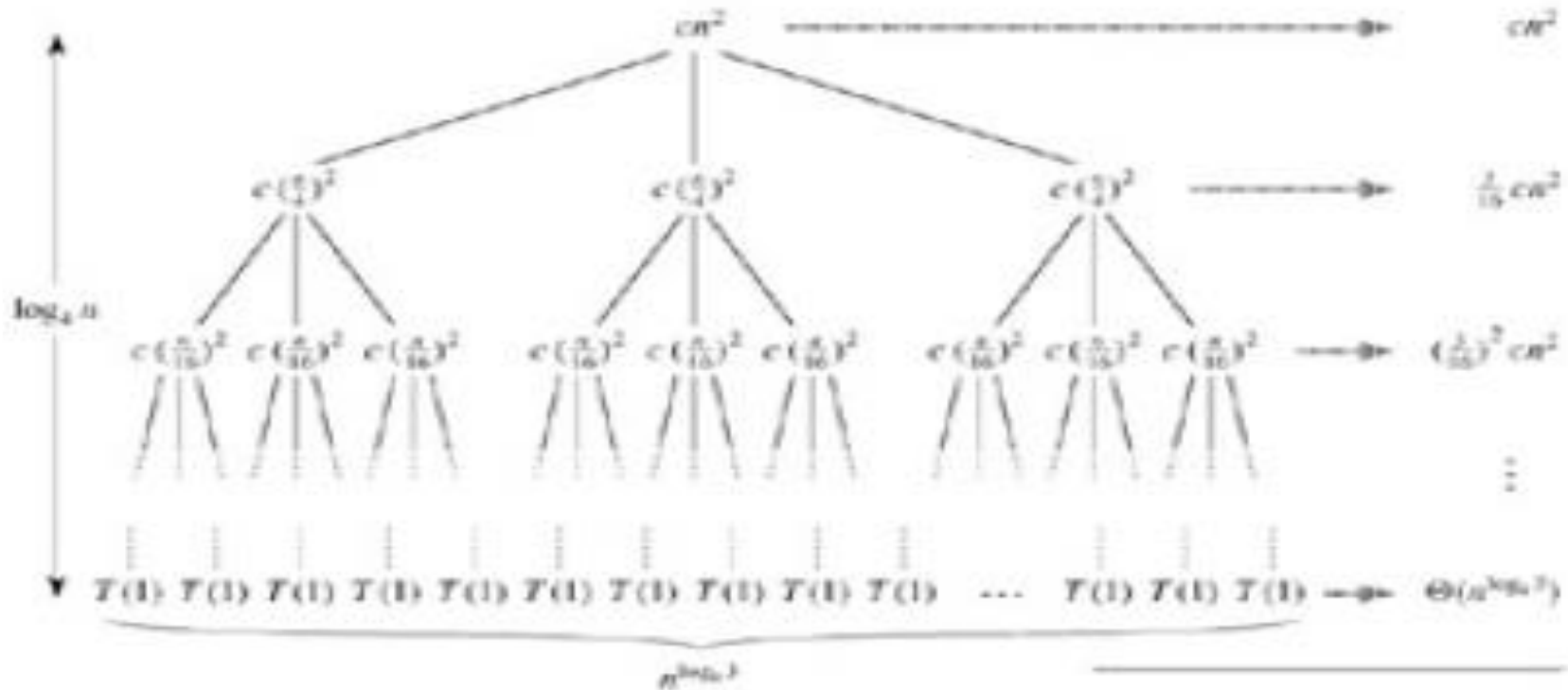
$\text{else return Select}(a, i+1, r, k-j);$

$\{43, 37, 57, 33, 32, 52, 51, 35, 60, 41, 54, 49, 31, 46, 29\}$

线性时间选择程序



$$T(n) = \begin{cases} O(1) & n < 75 \\ T(n/5) + T(3n/4) + O(n) & n \geq 75 \end{cases} = O(n)$$



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

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

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

$$T(n) = 3^3T(n/4^3) + 3^2c(n/16)^2 + 3c(n/4)^2 + cn^2$$

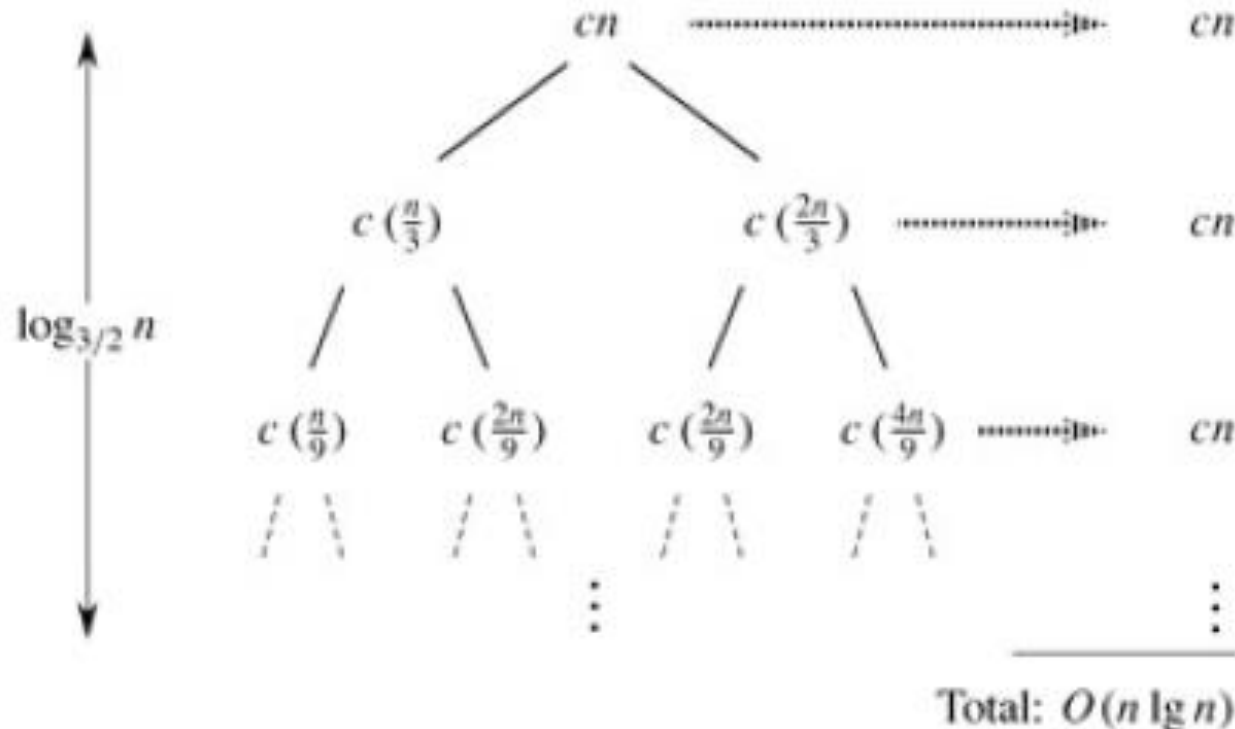
$$T(n) = T(n/3) + T(2n/3) + cn$$

$$T(n) = [T(n/3/3) + T(2n/3/3) + c(n/3)] +$$

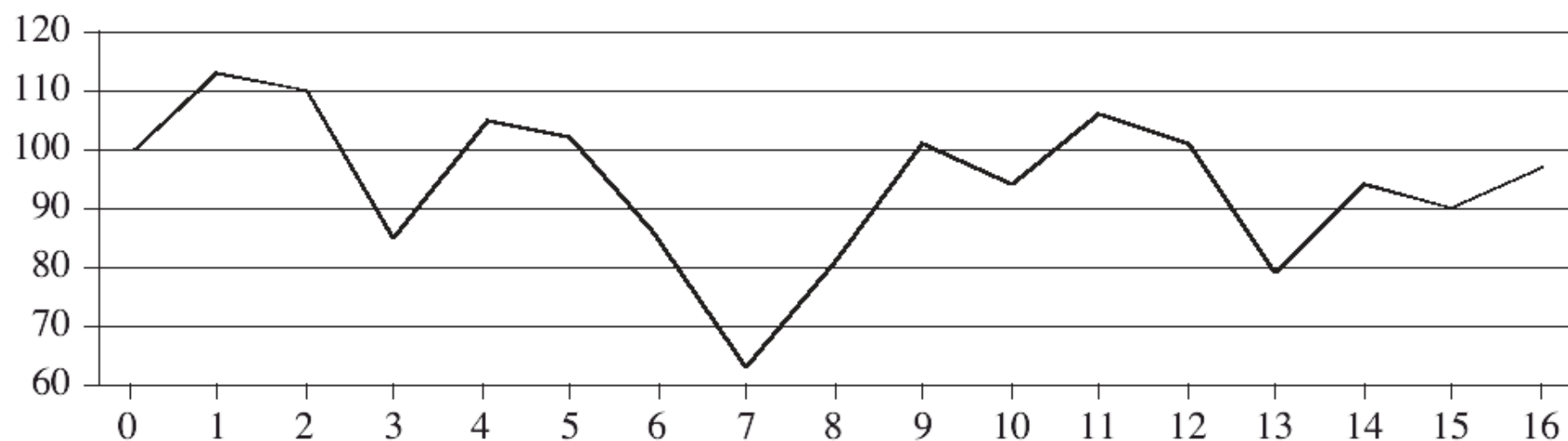
$$[T(2n/3/3) + T(4n/3/3) + c(2n/3)] + cn$$

$$T(n) = [T(n/9) + T(2n/9) + T(2n/9) + T(4n/9)]$$

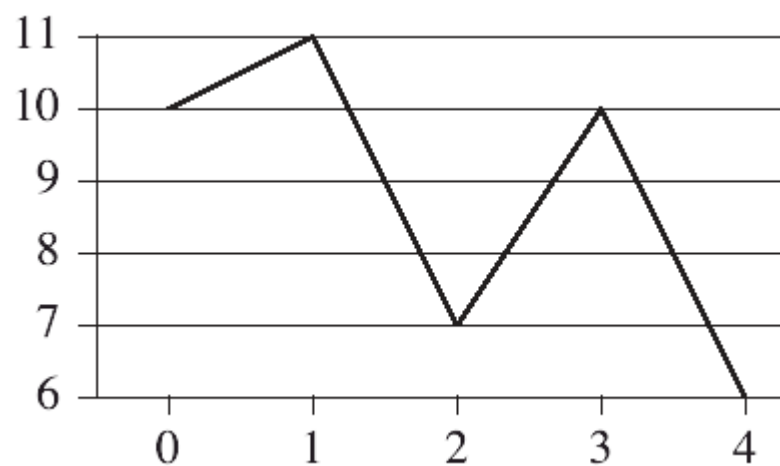
$$+ [c(n/3) + c(2n/3)] + cn$$



引例



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	-4	3	-4

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

maximum subarray

最大子段和

- 给定整数序列 a_1, a_2, \dots, a_n ，求形如 $\sum_{k=i}^j a_k$ 的子段和的最大值。规定子段和为负整数时，定义其最大子段和为0，即

$$\max \left\{ 0, \max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a_k \right\}$$

- 例如， $(a_1, a_2, a_3, a_4, a_5, a_6) = (-2, 11, -4, 13, -5, -2)$
最大子段和为

$$\sum_{k=2}^4 a_k = 20$$

1 可以把所有的子段和计算出来，找到最小的

2 找到所有子段算法：

每个子段有一个起点 i 和一个终点 j

把起点位置 i 从左到右进行扫描

确定起点后，把终点位置 j ，左到右进行扫描，确定起点终点后，把这个子段中所元素相加 $(i, i+1, \dots, j)$,

```

int MaxSubSum1(int n, int a[], int &besti, int &bestj)
{ //数组a[]存储ai, 返回最大子段和, 保存起止位置到
  Besti,Bbestj中
    int sum=0;
    for(int i=1; i<=n; i++)
      for(int j=i; j<=n; j++) {
        int thissum=0;
        for(int k=i; k<=j; k++)
          thissum += a[k];
        if(thissum>sum) {
          sum=thissum;
          besti=i; bestj=j;
        }
      }
    return sum;
  }
}

```

算法: $T(n)=O(n^3)$;

```

int MaxSubSum2(int n, int a[], int &besti, int &bestj)
{ //数组a[]存储ai, 返回最大子段和, 保存起止位置到
  Besti,Bbestj中
    int sum=0;
    for(int i=1; i<=n; i++){
        int thissum=0;
        for(int j=i; j<=n; j++) {
            thissum += a[j];
            if(thissum>sum) {
                sum=thissum;
                besti=i; bestj=j;
            }
        }
    }
    return sum;
}

```

改进算法: $T(n)=O(n^2)$;

最大子段和: 分治算法

● 基本思想

将 $A[1..n]$ 分为 $a[1..n/2]$ 和 $a[n/2+1..n]$, 分别对两区段求最大子段和, 这时有三种情形:

Case 1: $a[1..n]$ 的最大子段和的子段落在 $a[1..n/2]$;

Case 2: $a[1..n]$ 的最大子段和的子段落在 $a[n/2..n]$;

Case 3: $a[1..n]$ 的最大子段和的子段跨在 $a[1..n/2]$ 和 $a[n/2..n]$ 之间;

- 对 Case 1 和 Case 2 可递归求解；

对 Case 3，可知 $a[n/2]$ 和 $a[n/2+1]$ 一定在最大和的子段中，因此

在 $a[1..n/2]$ 中计算：

$$S_1 = \max_{1 \leq i \leq n/2} \sum_{k=i}^{n/2} a_k$$

在 $a[n/2..n]$ 中计算：

$$S_2 = \max_{n/2+1 \leq i \leq n} \sum_{k=n/2+1}^i a_k$$

易知： $S_1 + S_2$ 是 Case 3 的最大值

```
int MaxSubSum3(int a[], int left, int right)
{ //返回最大子段和
    int sum=0;
    if(left==right)
        sum=a[left]>0?a[left]:0;
    else {
        int center=(left+right)/2;
        int leftsum=
            MaxSubSum3(a, left,center);
        int rightsum=
            MaxSubSum3(a, center+1, right);
        int s1=0; int leftmidsum=0;
        for(int i=center; i>=left; i--) {
            leftmidsum += a[i];
            if(leftmidsum>s1) s1=leftmidsum;
        }
    }
}
```

```

int s2=0; int rightmidsum=0;
for(int i=center+1; i<=right; i++) {
    rightminsum += a[i];
    if(rightmidsum>s2)
        s2=rightmidsum;
}
int sum=s1+s2;
if(sum<leftsum) sum=leftsum;
if(sum<rightsum) sum=rightsum;
} //end if
return sum;
} //end

```

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

$$\Rightarrow T(n) = O(n \log n)$$

最接近点对问题

- 输入: 平面上点集 $P = \{p_1, p_2, \dots, p_n\}$
- 输出: (s, t) 使得
$$d(p_s, p_t) = \min \{ d(u, v) \mid u \neq v \in P \}$$
其中设 $u = (x_1, y_1)$, $v = (x_2, y_2)$,

$$d(u, v) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

算法设计分析过程:

直接法--一维--排序--分治--二维--改进--改进

最近点对-逐对求距离

- 输入: 平面上点集 $P = \{p_1, p_2, \dots, p_n\}$
- 输出: (s, t) 使得 $d(p_s, p_t)$ 是最小点间距

O(1) 1. 初始化: $\min = d(p_1, p_2); s=1; t=2;$

2. 对 $i = 1 : n-1$

3. 对 $j = i+1 : n$

O(n²)

O(1)

4. 若 $\min > d(p_i, p_j),$

5. 则 $s = i; t = j; \min = d(p_i, p_j)$

6. 输出 (s, t)

总时间 $O(C(n, 2)) = O(n^2)$

最近点对--一维方法一

排序再逐个计算距离:

$O(n \log n)$ 1. 排序: $p_{i_1} \leq p_{i_2} \leq \dots \leq p_{i_n}$.

2. 初始化: $\min = d(p_{i_1}, p_{i_2}); s = i_1; t = i_2;$

3. 对 $k = 2 : n-1$

$O(n)$

$\left\{ \begin{array}{l} O(1) \end{array} \right\}$

4. 若 $\min > d(p_{i_k}, p_{i_{k+1}})$

5. 则 $s = i_k; t = i_{k+1}; \min = d(p_{i_k}, p_{i_{k+1}})$

● 总时间 $O(n \log n)$

● 不能推广到二维

最近点对--一维分治

问题1: 设点集合为 S , 如何分成两个部分 S_L 和 S_R ?

- 取中点 $m = (\min S + \max S)/2$ 划分, 可能不平衡
- 取中位数划分(解决了平衡问题)

问题2: 如何合并?

- 最小距离 = $\min \{ d_L, d_R, \min S_R - \max S_L \}$

$O(n)$ 1. 分: 取 S 中位数, 划分为 $S_L < S_R$.

$2T(n/2)$ 2. 治: 递归求 $S_L(S_R)$ 的最近点对距离 $d_L(d_R)$

$O(n)$ 3. 合: 取 S_L 最大点 p , S_R 最小点 q

$O(1)$ 4. $\delta = \min \{ d_L, d_R, \mathbf{q-p} \}$

$$T(n) = \begin{cases} O(1) & n \leq 3 \\ 2T(n/2) + O(n) & n > 3 \end{cases} = O(n \log n)$$

最近点对—二维分治尝试

设点集合为 S ,

1. 分: 取 S 横坐标中位数 mid , 划分为 $S_L <_x S_R$.

2. 治: 递归求 $S_L(S_R)$ 的最近点对距离 $d_L(d_R)$

3. 合: $d = \min \{ d_L, d_R \}$

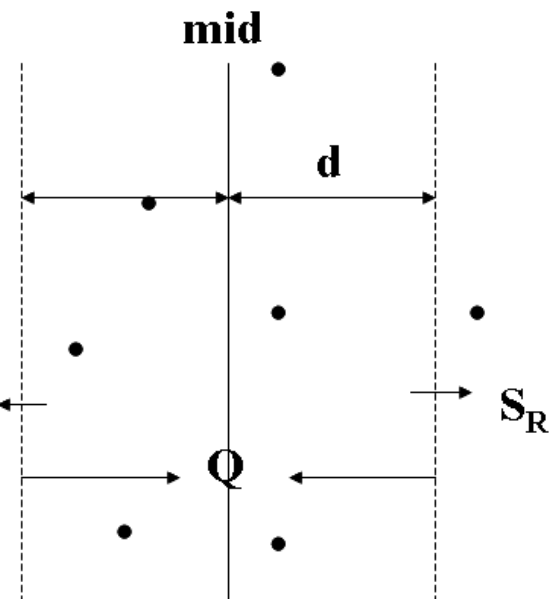
4. 取 $Q = \{ p \in S \mid |x(p) - \text{mid}| < d \}$

5. 逐对求 Q 中最近点对的距离 d .

分 $O(n)$, 治 $2T(n/2)$, 合 $O(n^2)$

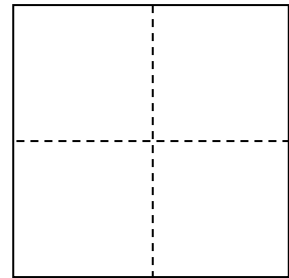
根据分治主定理 $T(n) = O(n^2)$

$$T(n) = \begin{cases} O(1) & n \leq 3 \\ 2T(n/2) + O(n^2) & n > 3 \end{cases}$$

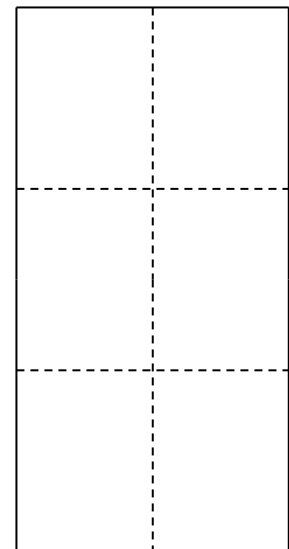


鸽巢(抽屉)原理的简单应用

任取一个 $d \times d$ 正方形内的点集 A ,
若 A 中任意两点距离都 $\geq d$, 则 A 中点数 ≤ 4 .



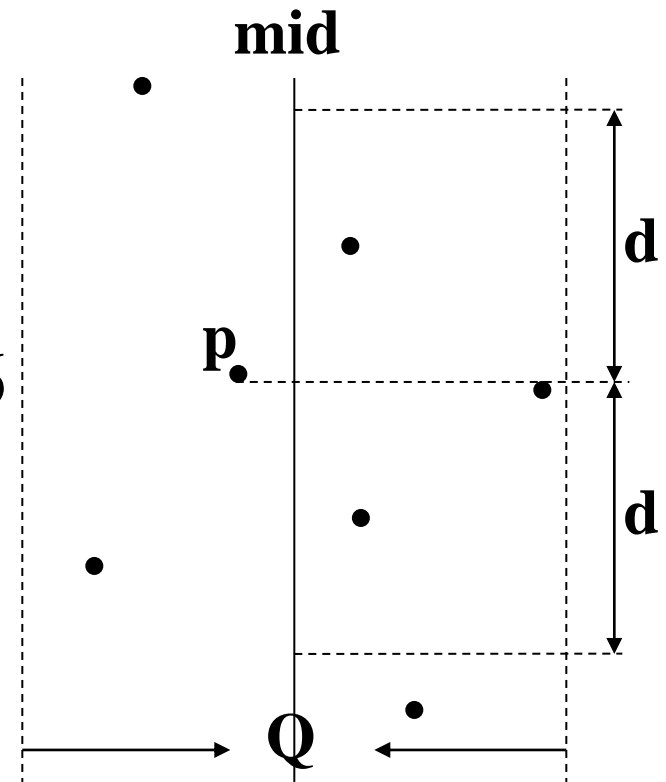
任取一个 $d \times 2d$ 矩形内点集 A ,
若 A 中任意两点距离都 $\geq d$, 则 A 中点数 ≤ 6 .



$$\sqrt{\left(\frac{d}{2}\right)^2 + \left(\frac{2d}{3}\right)^2} = \frac{5d}{6}$$

方案一：Q左右分开

Q右侧中与p距离 $< d$ 的点数 ≤ 6

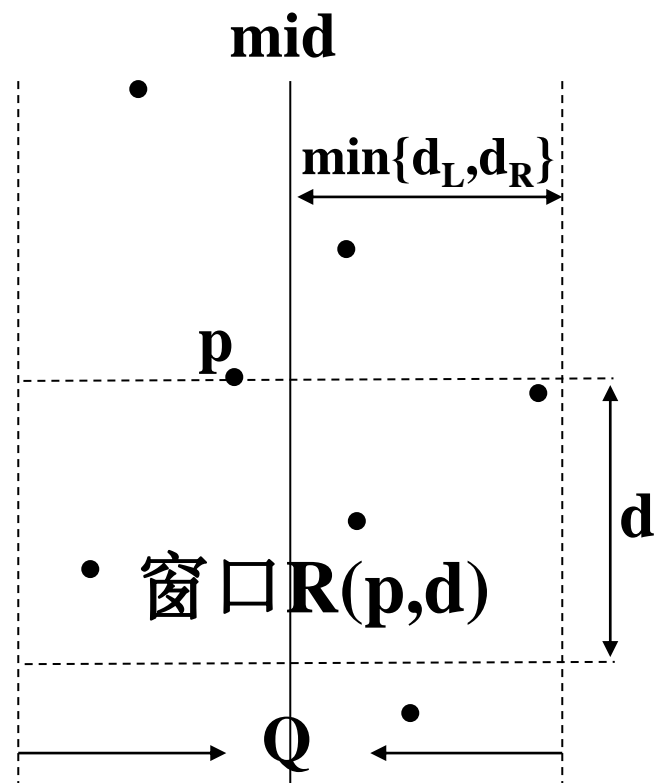


方案二: 检查p下方的点

- 定义窗口

$$R(p,d) = \{(x,y) : |x-\text{mid}| < \min\{d_L, d_R\}, 0 \leq y(p) - y \leq d\}$$

- Q中p下方与p距离 $\leq d$ 的点一定在 $R(p,d)$ 中
而且点数 $\leq 7 = 4 + 3$



最近点对--合并时间改进一

1. 分: 取S横坐标中位数mid, 划分为 $S_L \leq_x S_R$.

2. 治: 递归求 $S_L(S_R)$ 的最近点对距离 $d_L(d_R)$

3. $d = \min \{ d_L, d_R \}$

4. $Q = \{ p \in S \mid |x(p) - \text{mid}| < d \}$ 按纵坐标升序

5. 对 $i = 1$ 到 $|Q|-1$,

6. $j = i + 1$,

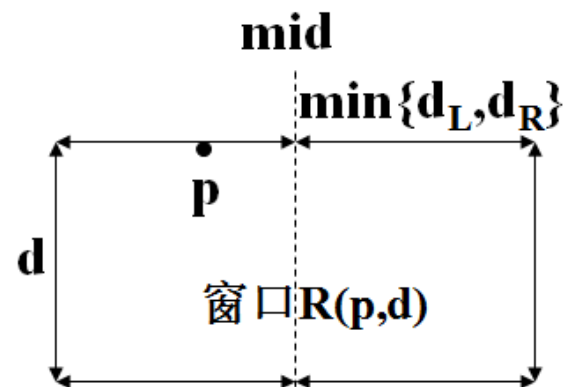
7. while $(y(j) - y(i) < d)$

8. {若 $d(p_i, p_j) < d$, 更新 d ; $j = j + 1$ }

步4: $O(n \log n)$, 步78循环至多7次, 步5-8循环至多n次

$T(n) = O(n \log^2 n)$, 进一步改进?

$$T(n) = \begin{cases} O(1) & n \leq 3 \\ 2T(n/2) + O(n \log n) & n > 3 \end{cases}$$



$$\begin{aligned} T(2^k) &= 2T(2^{k-1}) + k2^k, \\ &= 2^2T(2^{k-1}) + \\ &\quad (k-1)2^k + k2^k. \\ &= O(k^2 2^k) \\ &= O(n \log^2 n) \end{aligned}$$

称5--8过程为:

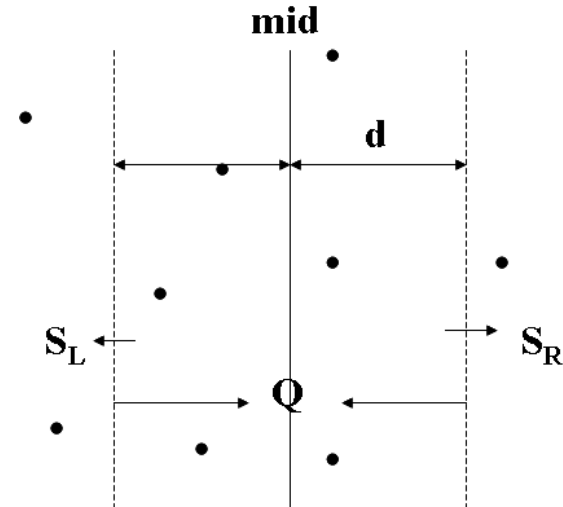
对Q中每个点p,
检查窗口 $R(p, d)$,
更新最短距离d

最近点对--合并时间改进二

排序放到分治前:

设有平面点集 S , 按 y 坐标升序(预处理)

1. 分: 取 S 横坐标中位数 mid , 划分为 $S_L <_x S_R$.
2. 治: 递归求 $S_L(S_R)$ 的最近点对距离 $d_L(d_R)$
3. 合: $d = \min \{ d_L, d_R \}$
4. 从 S_L, S_R 中归并取 $Q = \{ p \in S \mid |x(p) - mid| < d \}$
5. 对 Q 中每个点 p , 检查窗口 $R(p, d)$, 更新最短距离 d



$$T(n) = \begin{cases} O(1) & n \leq 3 \\ 2T(n/2) + O(n) & n > 3 \end{cases} \quad O(\textcolor{red}{n} \log n)$$

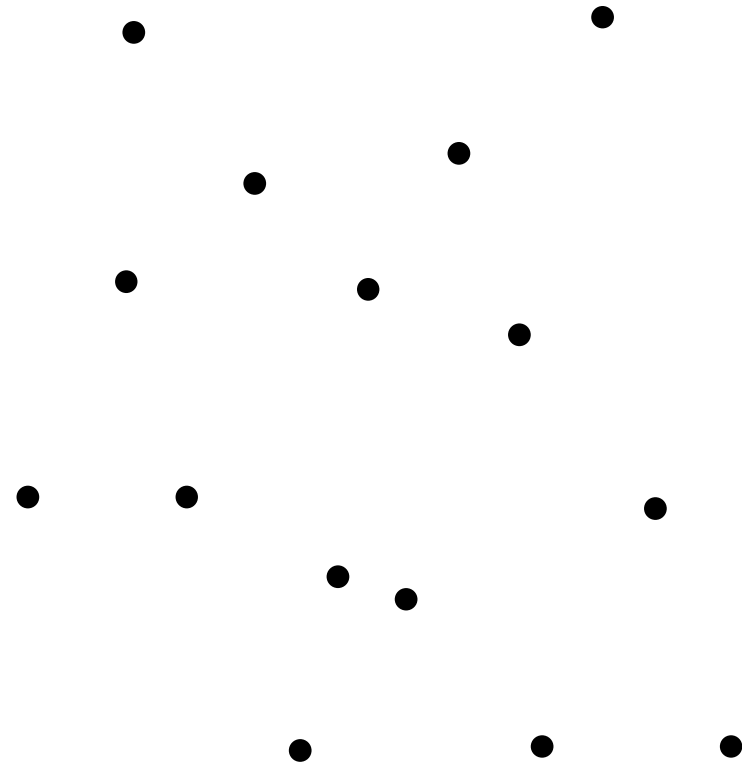
算法图示--初始

S

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .
2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$
3. 合: $d = \min \{ d_L, d_R \}$
4. 由 S_L, S_R 按纵坐标大小归并得 Q
5. 对Q中每个点p,
6. 检查窗口 $R(p, d)$
7. 更新最短距离



算法图示--预处理

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .

2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$

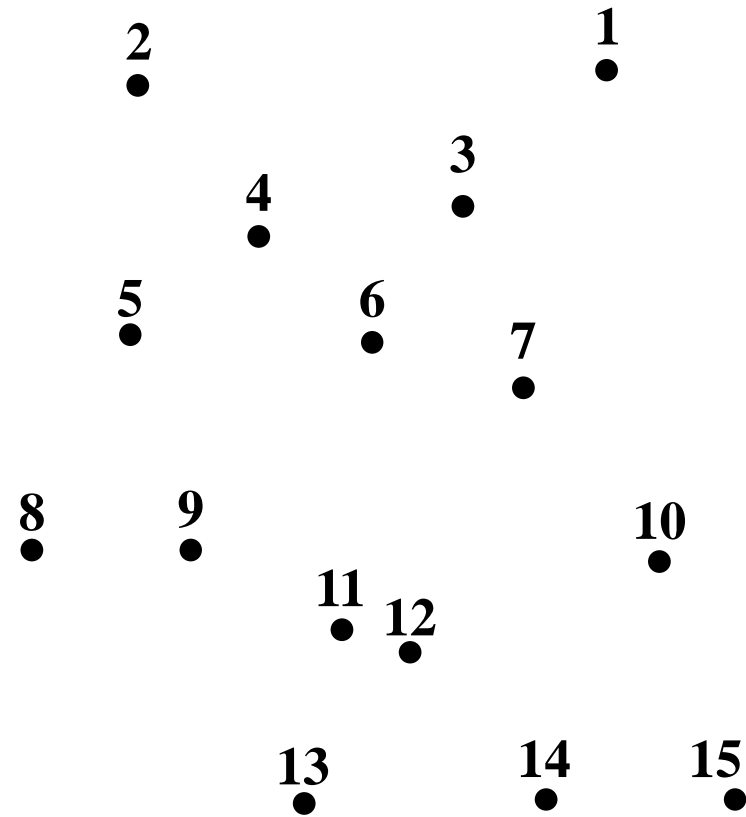
3. 合: $d = \min \{ d_L, d_R \}$

4. 由 S_L, S_R 按纵坐标大小归并得 Q

5. 对Q中每个点p,

6. 检查窗口 $R(p, d)$

7. 更新最短距离

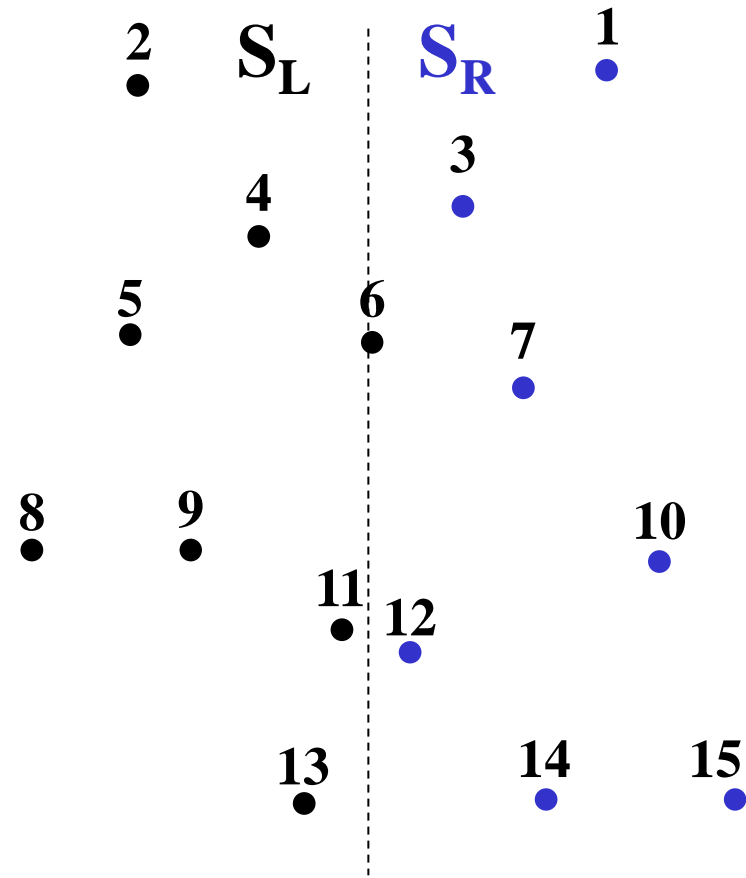


算法图示--分

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .
2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$
3. 合: $d = \min \{ d_L, d_R \}$
4. 由 S_L, S_R 按纵坐标大小归并得 Q
5. 对Q中每个点p,
6. 检查窗口 $R(p, d)$
7. 更新最短距离

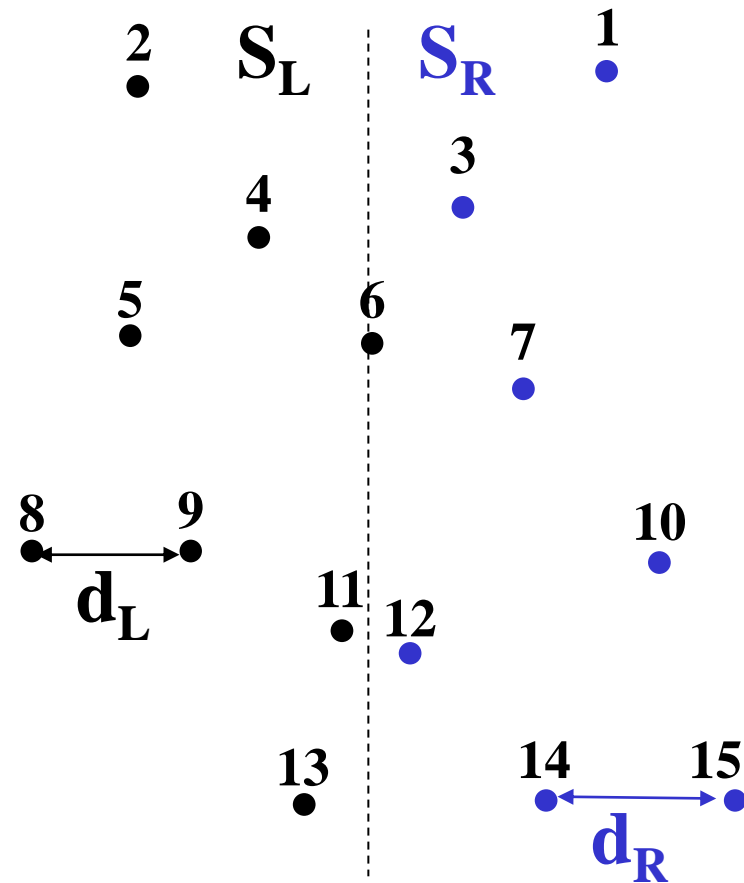


算法图示--治

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .
2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$
3. 合: $d = \min \{ d_L, d_R \}$
4. 由 S_L, S_R 按纵坐标大小归并得 Q
5. 对Q中每个点p,
6. 检查窗口 $R(p, d)$
7. 更新最短距离



算法图示--合3

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .

2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$

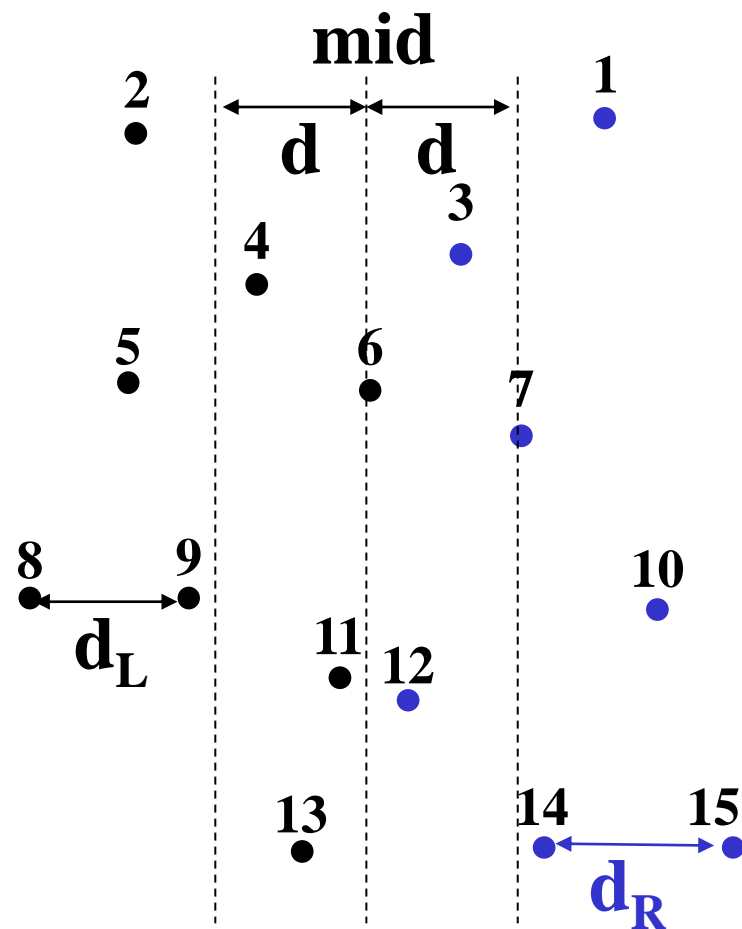
3. 合: $d = \min \{ d_L, d_R \}$

4. 由 S_L, S_R 按纵坐标大小归并得 Q

5. 对Q中每个点p,

6. 检查窗口 $R(p, d)$

7. 更新最短距离

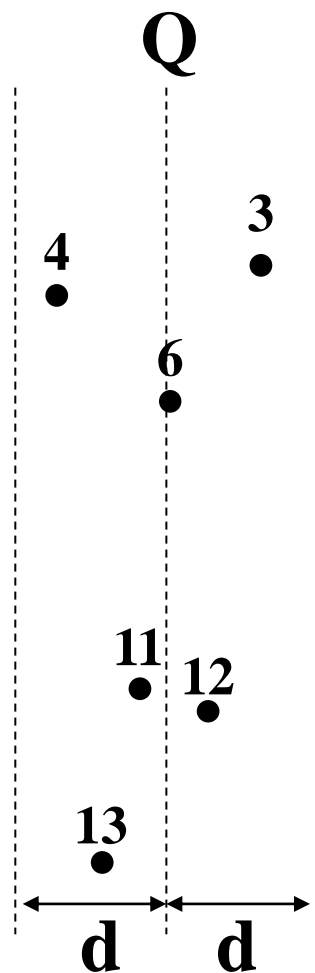


算法图示--合4

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .
2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$
3. 合: $d = \min \{ d_L, d_R \}$
4. 由 S_L, S_R 按纵坐标大小归并得 Q
5. 对Q中每个点p,
6. 检查窗口 $R(p, d)$
7. 更新最短距离



算法图示--合67:p₃

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .

2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$

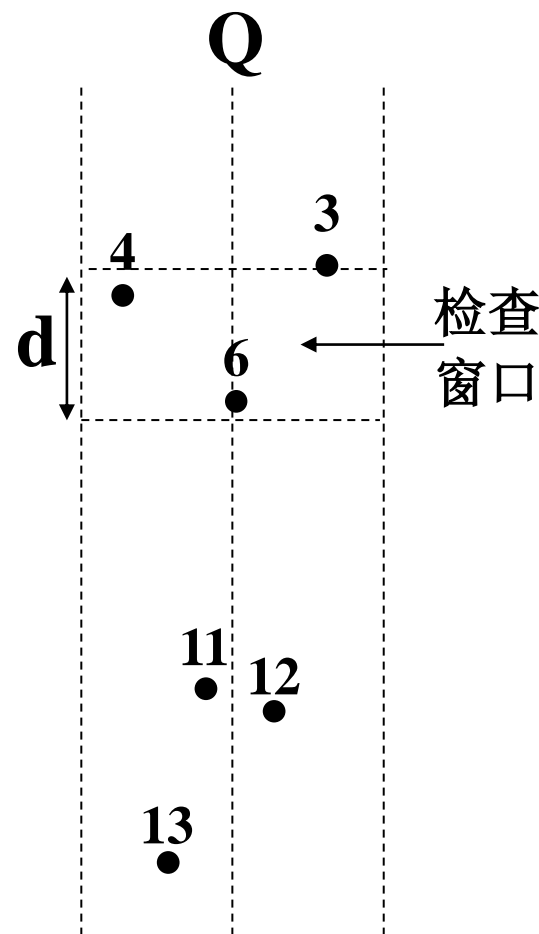
3. 合: $d = \min \{ d_L, d_R \}$

4. 由 S_L, S_R 按纵坐标大小归并得 Q

5. 对Q中每个点p,

6. 检查窗口 $R(p, d)$

7. 更新最短距离



算法图示--合67:p₄

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .

2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$

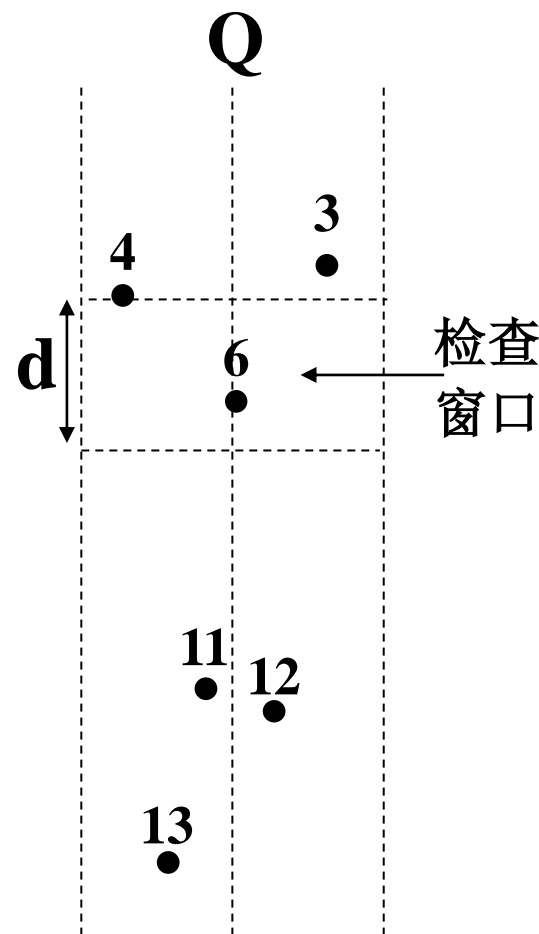
3. 合: $d = \min \{ d_L, d_R \}$

4. 由 S_L, S_R 按纵坐标大小归并得 Q

5. 对Q中每个点p,

6. 检查窗口 $R(p, d)$

7. 更新最短距离



算法图示--合67:p₆

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .

2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$

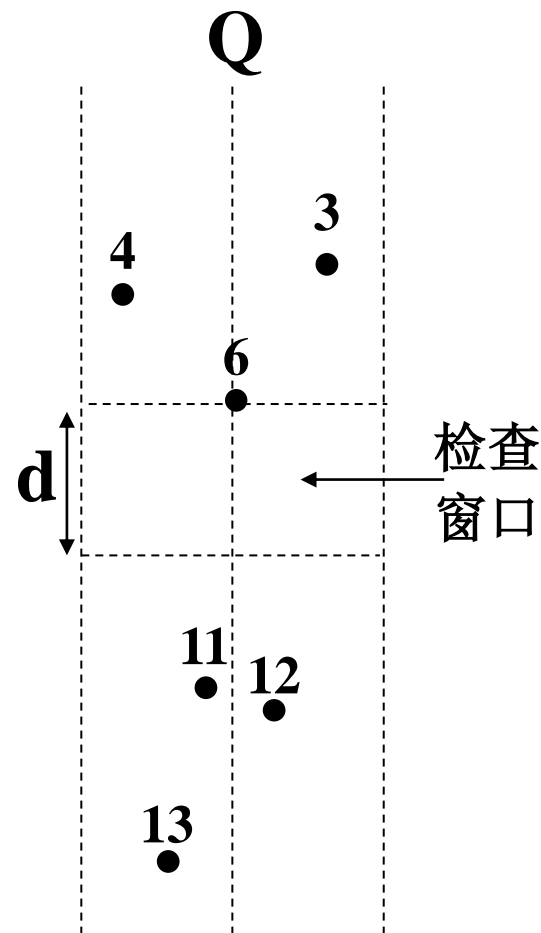
3. 合: $d = \min \{ d_L, d_R \}$

4. 由 S_L, S_R 按纵坐标大小归并得 Q

5. 对Q中每个点p,

6. 检查窗口 $R(p, d)$

7. 更新最短距离



算法图示--合6:p₁₁

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .

2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$

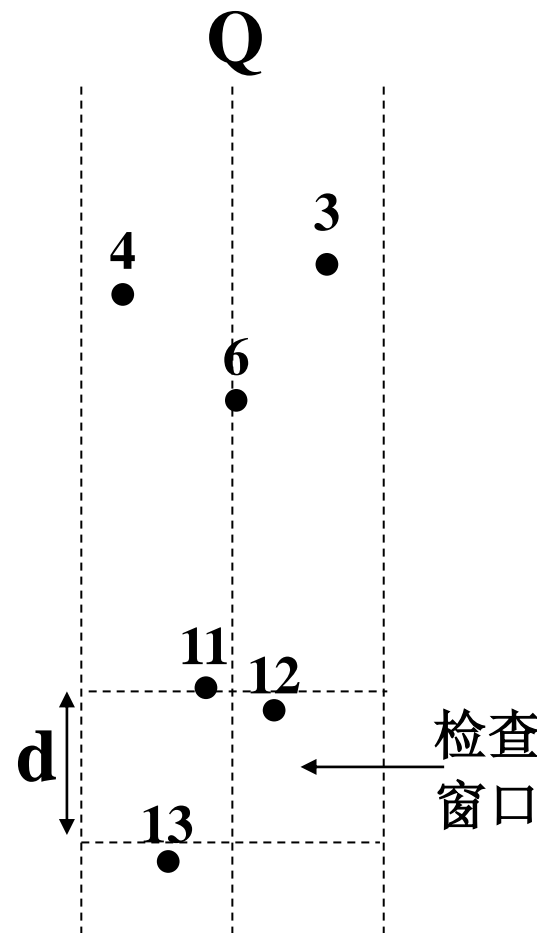
3. 合: $d = \min \{ d_L, d_R \}$

4. 由 S_L, S_R 按纵坐标大小归并得 Q

5. 对Q中每个点p,

6. **检查窗口 $R(p, d)$**

7. 更新最短距离



算法图示--合7:p₁₁

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .

2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$

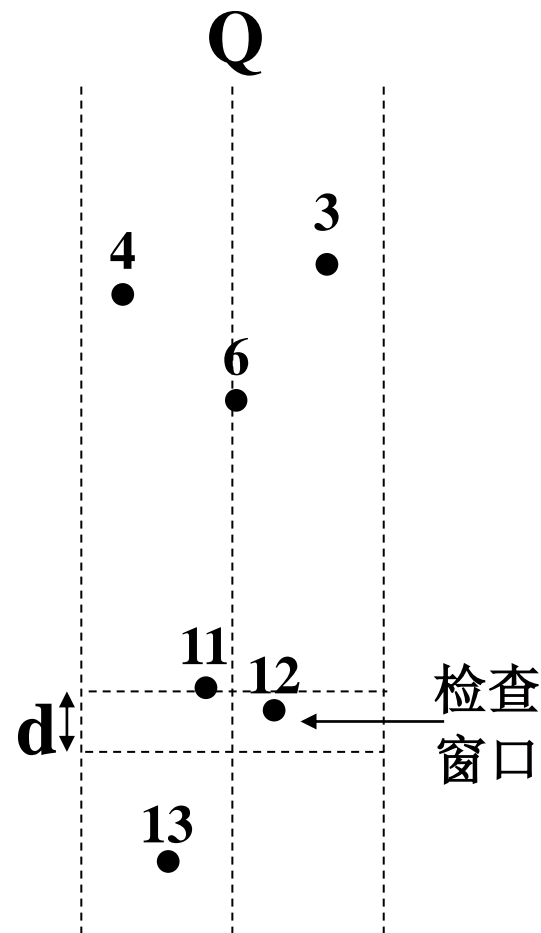
3. 合: $d = \min \{ d_L, d_R \}$

4. 由 S_L, S_R 按纵坐标大小归并得 Q

5. 对Q中每个点p,

6. 检查窗口 $R(p, d)$

7. 更新最短距离

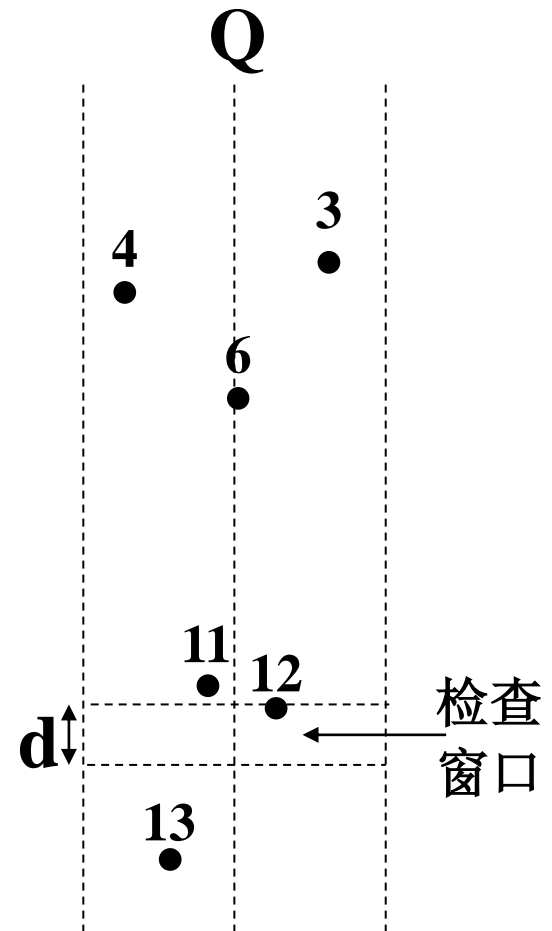


算法图示--合67:p₁₂

设有平面点集S

按y坐标递减(预处理)

1. 分: 取S横坐标中位数mid, 划分 S_L, S_R .
2. 治: 递归求 $S_L(S_R)$ 最近点对距离 $d_L(d_R)$
3. 合: $d = \min \{ d_L, d_R \}$
4. 由 S_L, S_R 按纵坐标大小归并得 Q
5. 对Q中每个点p,
6. 检查窗口 $R(p, d)$
7. 更新最短距离



最近点对程序-定义

```
class PointX
{ public:
    int operator<=(PointX a) const
    {return(x<=a.x);}
private:
    int ID; //点编号
    float x,y;//点坐标
};

class PointY
{ public:
    int operator<=(PointX a) const
    {return(y<=a.y);}
private:
    int p; //同一点在数组X中的编号
    float x,y;//点坐标
};
```

```
template<class Type>
inline float distance(const Type& u,
                      const Type& v)
{
    float dx = u.x-v.x;
    float dy = u.y-v.y;
    return sqrt(dx*dx+dy*dy);
}
```

最近点对程序-预排序

```
bool Cpair2(PointX X[], int n, PointX& a, PointX& b, float& d)
{
    if(n<2)return false;
    MergeSort(X,n);                // X按横坐标排序
    PointY *Y = new PointY [n];
    for(int i = 0; i < n; i++)      //将数组X中的点复制到数组Y中
    {
        Y[i].p = i;
        Y[i].x = X[i].x;
        Y[i].y = Y[i].y;
    }
    MergeSort(Y,n);                //Y按纵坐标排序
    PointY *Z = new PointY [n];
    closest(X,Y,Z,0,n-1,a,b,d);    //求最近点对
    delete [] Y;
    delete [] Z;
    return true;
}
```

最近点对程序-输入

```
int main()
{
    int n;
    scanf("%d",&n);
    PointX *X = new PointX [n];
    float xx,yy;
    for(int i = 0; i < n; i++) //输入数组X
    {
        scanf("%f %f",&xx,&yy);
        X[i].ID = i; X[i].x = xx; X[i].y = yy;
    }
    PointX& a; PointX& b; float& d;
    Cpair2(X, n, a, b, d);
    printf("%d, %d, %.2f\n",a,b,d); //输出a,b,d.
}
```

最近点对程序

```
void closest(PointX X[], PointY Y[], PointY Z[], int l,
            int r, PointX& a, PointX& b, float& d)
{
    if( r - l <= 2) {直接计算; return;}           //2点和3点的情形
    int m = ( l + r ) / 2; int f = l, g = m + 1; //多于3点的情形, 用分治法
    for( int i = l; i <= r; i++) if( Y[i].p > m ) Z[g++] = Y[i]; else Z[f++] = Y[i]; //分
    closest(X,Z,Y,l,m,a,b,d);                     //治: 左边
    float dr; PointX ar, br; closest(X,Z,Y,m+1,r,ar,br,dr); //治: 右边
    if( dr < d ) { a = ar; b = br; d = dr;}        //合: d
    Merge(Z,Y,l,m,r);                             //Z的两个有序段合并到数组Y
    int k = l; for( int i = l; i <= r; i++ ) //合: 从Y中取d矩形条内的点置于Z中
        if( fabs( X[m].x - Y[i].x ) < d ) Z[k++] = Y[i];
    for( int i = 1; i < k; i++)                     //合: 对d矩形条中的每点(Z[l:k-1])
    {
        for(int j = i+1; j < k && Z[j].y - Z[i].y < d; j++) //合: 检查R(p,d)中的点
        {
            float dp = distance( Z[i], Z[j]);
            if( dp < d){ d = dp; a = X[Z[i].p]; b = X[Z[j].p]; } //合: 更新最小距离
        }
    }
}
```

分治附录

附录：中位数原理

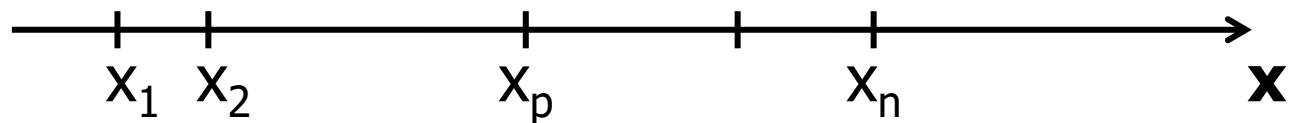
某公司有五个分公司依次设置在同一条铁路线的沿线A、B、C、D、E站。现在该公司希望在该铁路沿线设立一个仓库，要求该仓库离这五个站的火车行驶距离之和最小。如用数轴表示该铁路线，A、B、C、D、E各站的坐标依次为a、b、c、d、e（ $a < b < c < d < e$ ），则经过数学计算，该仓库大致应设置在坐标（1）处。

- (1) A. c B. $(a+b+c+d+e)/5$
C. $(a+2b+3c+2d+e)/9$
D. $(a+4b+6c+4d+e)/16$

附录：中位数原理

- 中位数原理

X轴上有**n**个点，由左至右依次排列为

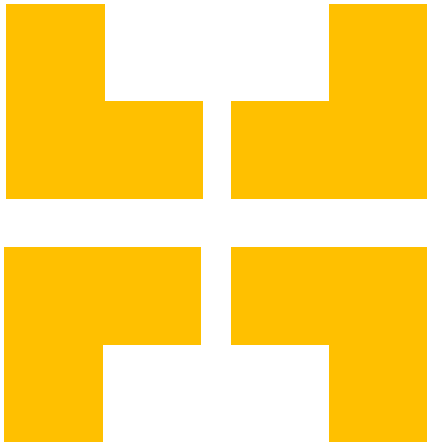


找一个点 x_p (不一定是**n**个点之一)，使 x_p 到各点距离和最小，解为：

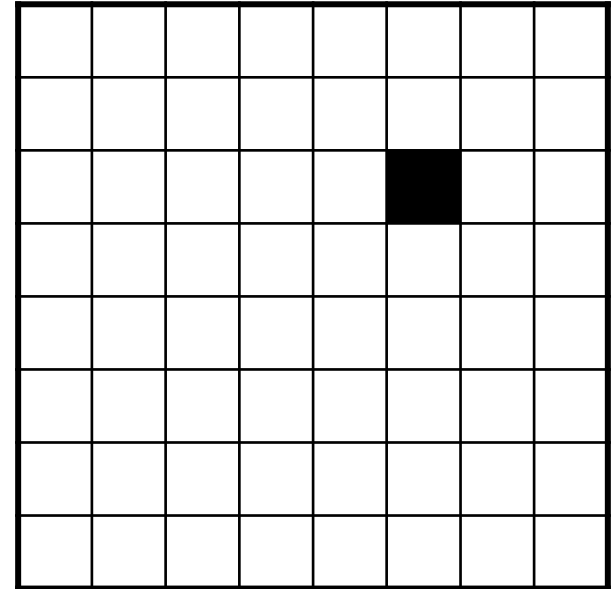
$$x_p = \begin{cases} x_{(n+1)/2} & \text{当 } n \text{ 为奇数时} \\ \text{中间两点的闭区间上} & \text{当 } n \text{ 为偶数时} \end{cases}$$

附录：棋盘覆盖

L型骨牌



$2^k \times 2^k$ 棋盘

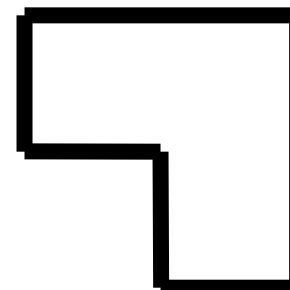
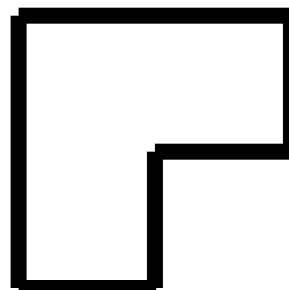
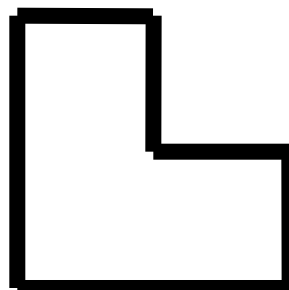
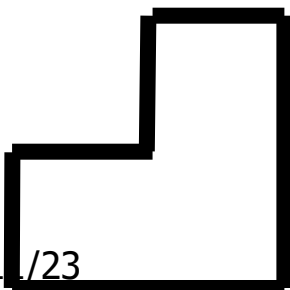
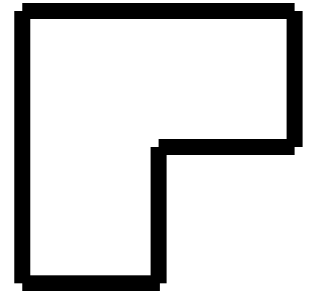
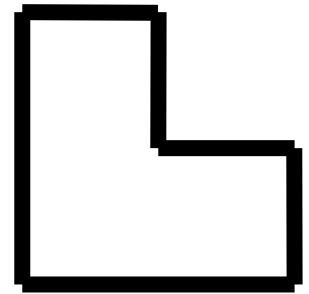
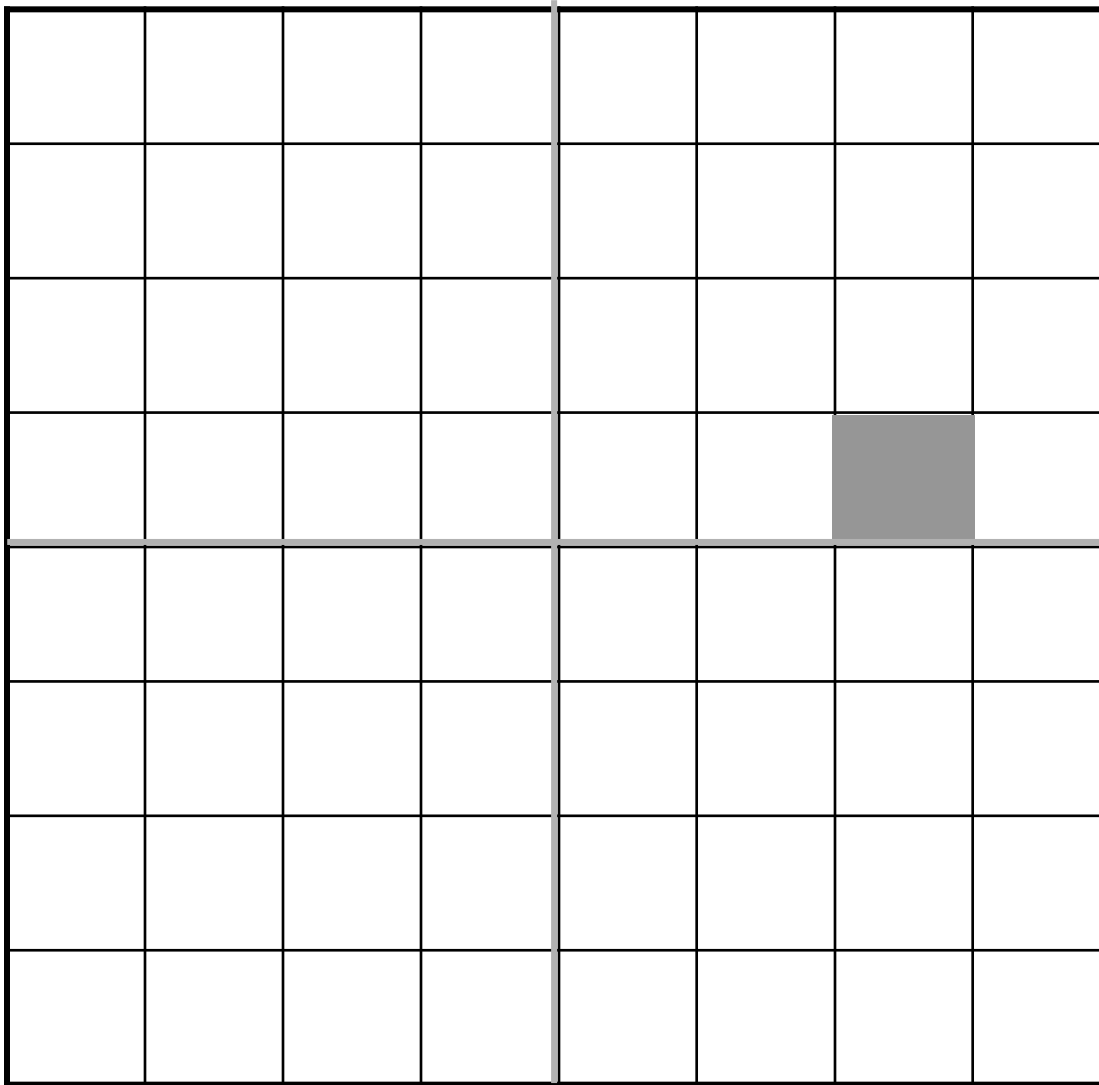


输入: k , 代表 $2^k \times 2^k$ 棋盘

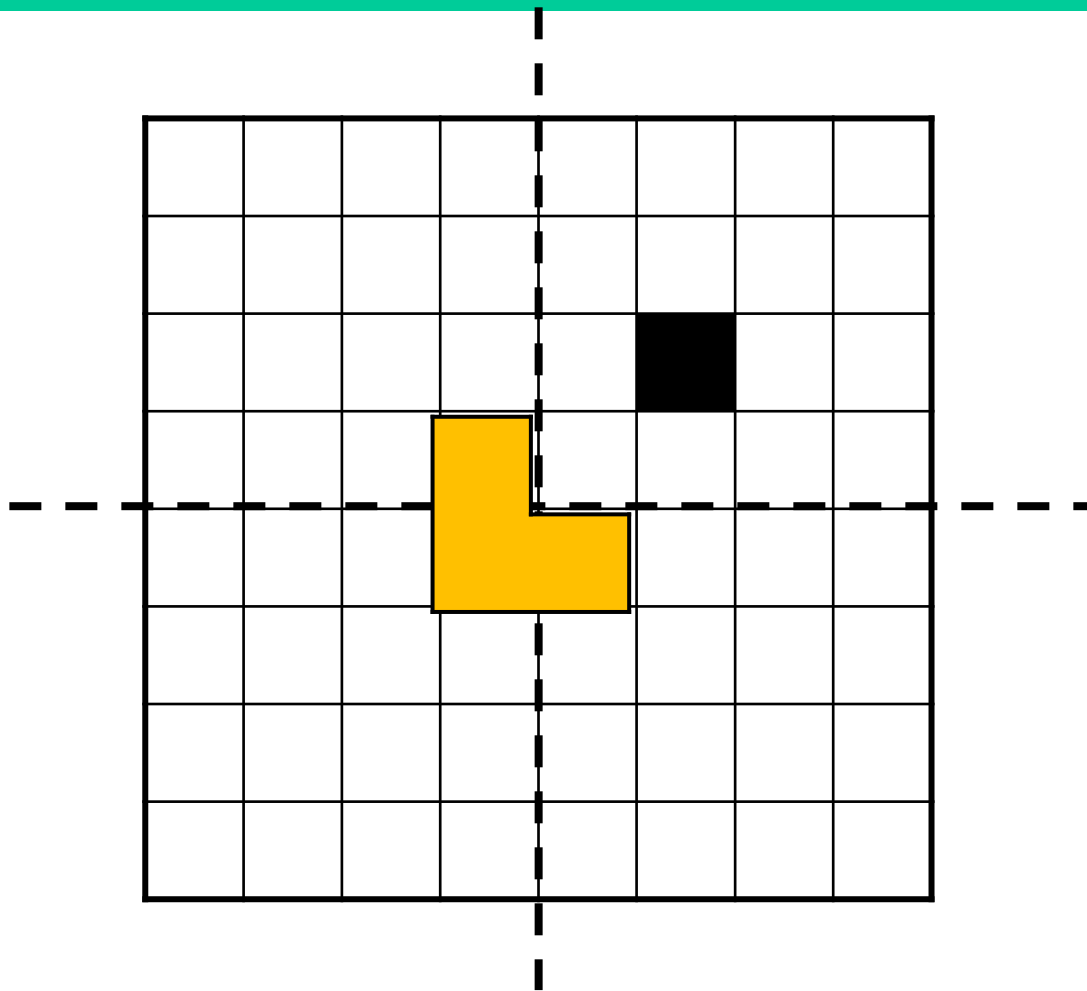
输出: 用 L 型骨牌覆盖棋盘的方案

说明: 有很多方案,
构造出一种方案即可

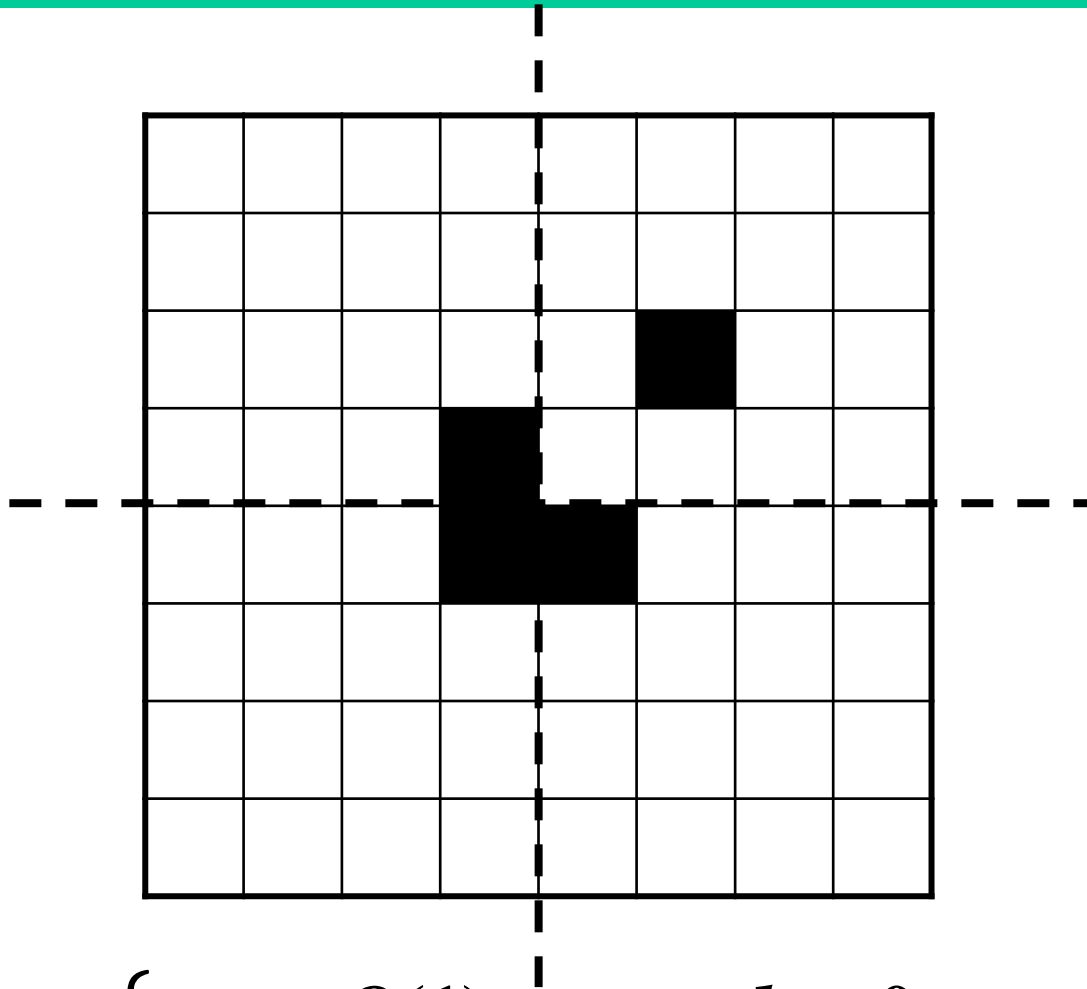
3	3	4	4	8	8	9	9
3	2	2	4	8	7	7	9
5	2	6	6	10		7	11
5	5	6	1	10	10	11	11
13	13	14	1	1	18	19	19
13	12	14	14	18	18	17	19
15	12	12	16	20	17	17	21
15	15	16	16	20	20	21	21



分治：递归构造



分治：递归构造



$$T(k) = \begin{cases} O(1) & k = 0 \\ 4T(k-1) + O(1) & k > 0 \end{cases} = O(4^k)$$

附录：循环赛日程表

$n=2^k$ 球员循环赛, 设计满足以下要求的比赛日程表:

- (1) 每个选手必须与其他 $n-1$ 个选手各赛一次
- (2) 每个选手一天只能赛一次
- (3) 循环赛一共进行 $n-1$ 天

球员	第1天
1	2
2	1

球员	第1天	第2天	第3天
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

循环赛日程表

1	2
2	1

(a) $2^k(k=1)$ 个选手比赛

1 2	3 4
2 1	4 3
3 4	1 2
4 3	2 1

(b) $2^k(k=2)$ 个选手比赛

1 2 3 4	5 6 7 8
2 1 4 3	6 5 8 7
3 4 1 2	7 8 5 6
4 3 2 1	8 7 6 5
5 6 7 8	1 2 3 4
6 5 8 7	2 1 4 3
7 8 5 6	3 4 1 2
8 7 6 5	4 3 2 1

(c) $2^k(k=3)$ 个选手比赛



循环赛日程表的推广

设计一个满足以下要求的比赛日程表：

(1)每个选手必须与其他 $n-1$ 个选手各赛一次；

(2)每个选手一天只能赛一次；

(3) n 为偶数时，循环赛一共进行 $n-1$ 天。

n 为奇数时，循环赛一共进行 n 天。

	1	2	3	4
第1天				
第2天				
第3天				

循环赛日程表的推广

	1	2	3	4
第1天				
第2天				
第3天				

	1	2	3
第1天			
第2天			
第3天			

	1	2	3
第1天	2	1	-
第2天	3	-	1
第3天	-	3	2

	1	2	3	4	5	6
第1天						
第2天						
第3天						
第4天						
第5天						

循环赛日程表的推广

	1	2	3	4	5
第1天	2	1	-	5	4
第2天	3	5	1	-	2
第3天	4	3	2	1	-
第4天	5	-	4	3	1
第5天	-	4	5	2	3

	1	2	3	4	5	6	7	8	9	10
第1天	2	1	8	5	4	7	6	3	10	9
第2天	3	5	1	9	2	8	10	6	4	7
第3天	4	3	2	1	10	9	8	7	6	5
第4天	5	7	4	3	1	10	2	9	8	6
第5天	6	4	5	2	3	1	9	10	7	8
第6天	7	8	9	10	6	5	1	2	3	4
第7天	8	9	10	6	7	4	5	1	2	3
第8天	9	10	6	7	8	3	4	5	1	2
第9天	10	6	7	8	9	2	3	4	5	1

练习

甲手中有**1张A**，**2张2**，**3张3**，**4张4**，**5张5**，**6张6**，**7张7**，**8张8**，**9张9**共**45张牌**，现甲从中任取一张牌，然后乙开始提问来猜出这张牌。请给出乙提问的平均最少次数。

注意：甲只能回答“是”或者“否”。