

数据结构与算法设计

林永钢

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.

引言

旅行商问题Traveling Salesman Problem (TSP):
设有 **n** 个城市，已知任意两城市之间距离，现有一推销员想从某一城市出发巡回经过每一城市（且每城市只经过一次），最后又回到出发点，问如何找一条最短路径。

穷举法的时间复杂性为 **$O(n!)$**

$n = 21$ 时， $21! = 5.11 \times 10^{19}$ ，设每条路径需**CPU**时间为**1ns**，则总共要**1620**多年才能完成。

引言

- 算法的五个特征

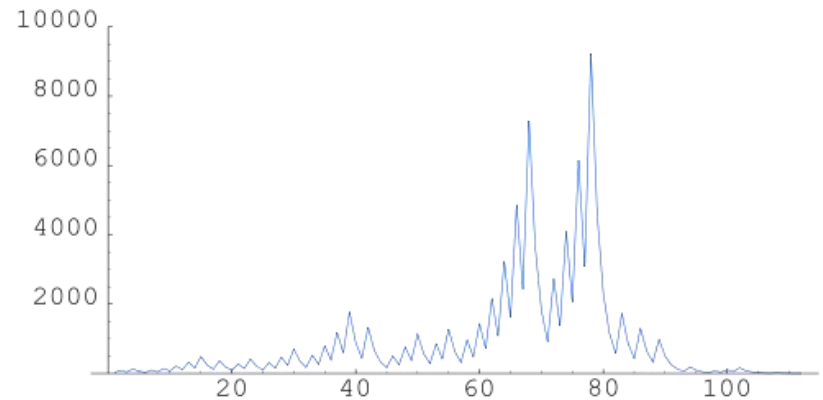
1) 有穷性 2) 确定性

3) 输入 4) 输出 5) 可行性

- 算法的定义: Informally, an *algorithm* is any well-defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output*. An algorithm is thus a sequence of computational steps that transform the input into the output.

3n+1问题目前不知道有没有算法

- 输入: 一个正整数 n ,
- 映射: $f(n) = n/2$, 若 n 是偶数;
 $f(n) = 3n+1$, 若 n 是奇数.
- 迭代: $5 \rightarrow 16 \rightarrow 8 \rightarrow \dots$, 到1则停止
- 输出: n 可在 f 迭代下是否能到1停止
- 直接模拟是正确的算法吗?
- 27需迭代111步(见右图)
- $1 \sim 5 \times 10^{18}$ 都能到1.([wiki])



不可判定问题(没有算法)举例

Hilbert第十问题:“多项式是否有整数根”有没有算法?

1970's 被证明不可判定.

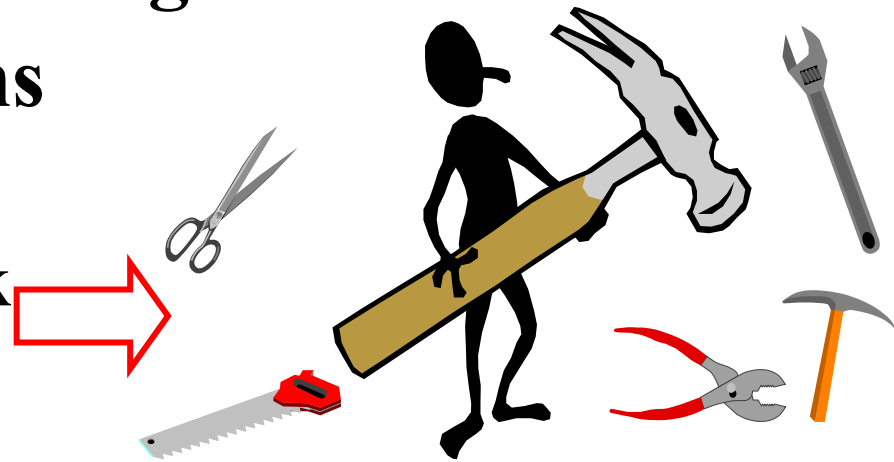
M = “对于输入 “**p**”, **p**是**k**元多项式,

1. 取**k**个整数的向量**x** (绝对值和从小到大)
2. 若 $p(x) = 0$, 则停机接受.
3. 否则转1.”

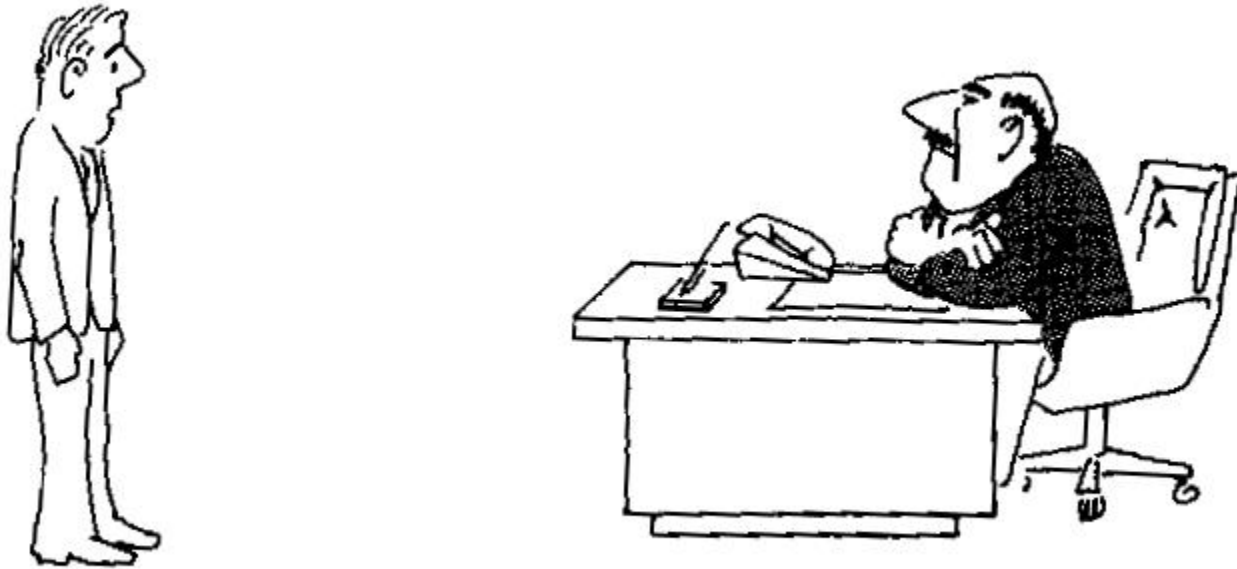
这个图灵机对输入 $p(x,y) = x^2+y^2-3$ 不停机

引言

- A survey of algorithmic design techniques.
- Abstract thinking.
- How to develop new algorithms for any problem that may arise.
- Be a great thinker and designer.
- ~~Not: A list of algorithms~~
 - ~~– - Learn their code~~
 - ~~– - Trace them until work~~
 - ~~– - Implement them~~
 - ~~– - be a mundane programmer~~



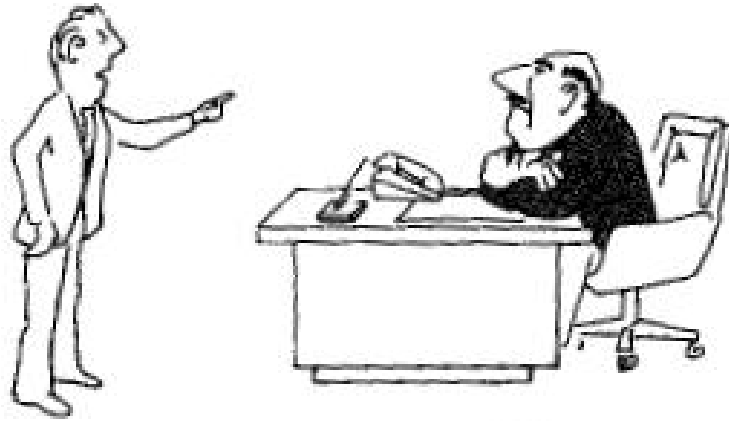
引言：学习算法的意义



**“I can’t find an efficient algorithm,
I guess I’m just too dumb.”**

**Serious damage to your
position within the company !!!**

引言：学习算法的意义



“I can’t find an efficient algorithm, because no such algorithm is possible!”

Unfortunately, proving intractability can be just as hard as finding efficient algorithms !!!

\therefore No hope !!!

$P \neq NP$

引言：学习算法的意义



"I can't find an efficient algorithm, but neither can all these famous people"

引言：学习算法的意义



**"Anyway, we have to solve this problem.
Can we satisfy with a good solution ? "**

一些有趣的问题

- (1) 背包问题1：有一人要从 n 种物品中选取不超过 b 千克重的行李随身携带，要求总价值最大。
例：设背包的容量为50千克。物品1重10千克，价值60元；物品2重20千克，价值100元；物品3重30千克，价值120元。求总价值最大。
- (2) 背包问题2：有一人要从 n 种货物中选取不超过 b 千克重的行李随身携带，要求总价值最大。
例：设背包的容量为50千克。物品1有60千克，每千克价值60元；物品2有20千克，每千克价值100元；物品3有40千克，每千克价值120元。求总价值最大。

The Drunk Jailer

(3) A certain prison contains a long hall of n cells, each right next to each other. Each cell has a prisoner in it, and each cell is locked. One night, the jailer gets bored and decides to play a game. For round 1 of the game, he takes a drink of whiskey, and then runs down the hall unlocking each cell. For round 2, he takes a drink of whiskey, and then runs down the hall locking every other cell (cells 2, 4, 6, ...). For round 3, he takes a drink of whiskey, and then runs down the hall. He visits every third cell (cells 3, 6, 9, ...). If the cell is locked, he unlocks it; if it is unlocked, he locks it.

The Drunk Jailer

He repeats this for n rounds, takes a final drink, and **passes out.**

Input: The first line of input contains a single positive integer. **This is the number of lines that follow. Each of the following lines contains a single integer between 5 and 100, inclusive, which is the number of cells n .**

Output: For each line, print out the number.

样例输入 样例输出

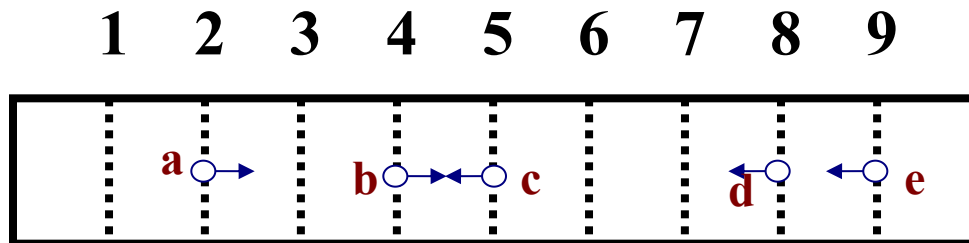
2 2

5 10

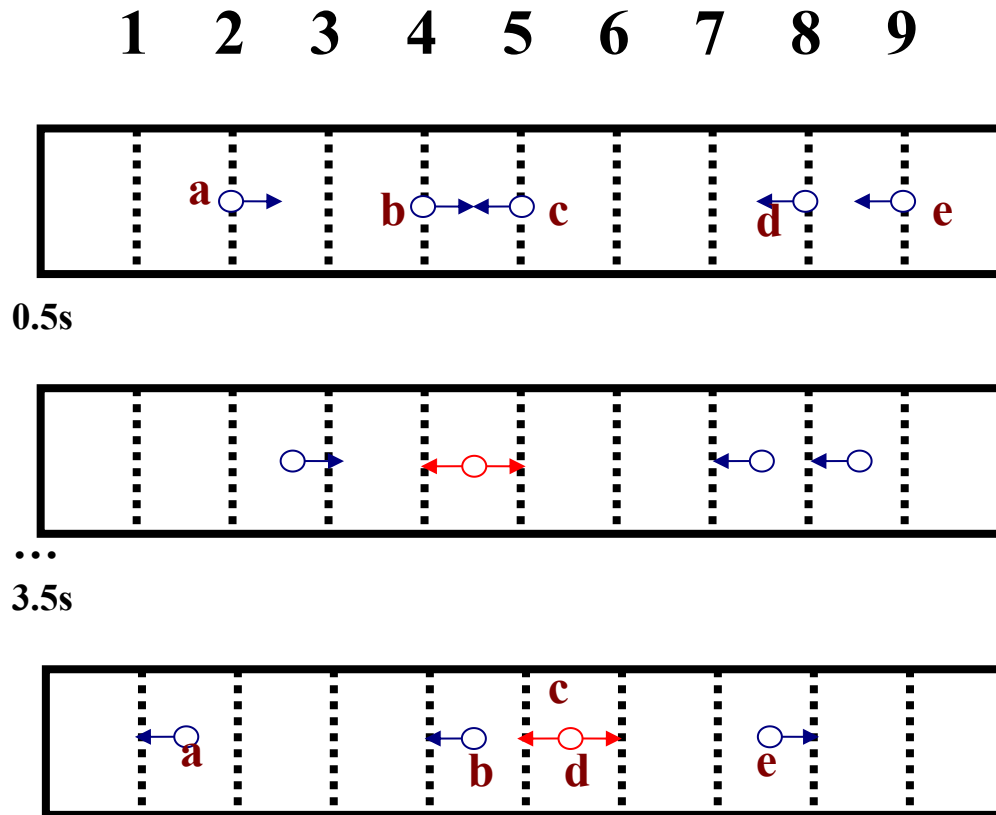
100

一些有趣的问题：蚂蚁问题

(4) 有一根10厘米的细木杆，在第2厘米、第4厘米、第5厘米、第8厘米、第9厘米这五个位置上各有一只蚂蚁。木杆很细，不能同时通过一只蚂蚁。开始时，蚂蚁的头朝左还是朝右是任意的，它们只能朝前走或调头，但不会后退。所有蚂蚁的速度都相同，均为 1cm/s 。当任意两只蚂蚁碰头时，两只蚂蚁会同时掉头并且仍然按 1cm/s 朝相反方向走。求蚂蚁掉下木杆的最小和最大时间。



一些有趣的问题： 蚂蚁问题



5秒，a掉； 6秒，e掉； 8秒，b和d掉； 9秒，c掉。

课程相关

- 课程内容

数学基础、分治与排序、动态规划、贪心法、自动机、图灵机、NP问题等.

- 课程安排

40学时授课

- 成绩

平时成绩30%， 期末闭卷成绩70%

时间复杂性

- 例：算法 A_1, A_2 的时间复杂性分别是 $n, 2^n$ ，设 $100\mu s$ 是一个单位时间，求 A_1, A_2 在 $1s$ 内能处理的问题规模。
- 已知 $\lg 2 = 0.301$

$$T(n) = n$$

$$T(n) * 10^{-4} = 1$$

$$\text{即 } n * 10^{-4} = 1$$

$$\text{所以 } n = 10^4$$

符号

一、符号说明

1. 取整函数

$\lfloor x \rfloor$: 小于等于 x 的最大整数

$\lceil x \rceil$: 大于等于 x 的最小整数

性质

$$x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$$

符号

$$\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n$$

$$\left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil$$

$$\left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$$

符号

$$\lceil a/b \rceil \leq [a+(b-1)]/b$$

$$\lfloor a/b \rfloor \geq [a-(b-1)]/b$$

符号

- 鸽巢原理（又名抽屉原理）

若 $n+1$ 只鸽子飞入 n 个鸽巢，那么至少有一个鸽巢至少有2只鸽子。

若 n 只鸽子飞入 m ($n > m$)个鸽巢，那么至少有一个鸽巢至少有 $\lfloor (n-1)/m \rfloor + 1$ 只鸽子。

Halloween treats

- Every year there is the same problem at Halloween: Each neighbor is only willing to give a certain total number of sweets on that day, no matter how many children call on him, so it may happen that a child will get nothing if it is too late. To avoid conflicts, the children have decided they will put all sweets together and then divide them evenly among themselves. From last year's experience of Halloween they know how many sweets they get from each neighbor. Since they care more about justice than about the number of sweets they get, they want to select a subset of the neighbors to visit, so that in sharing every child receives the same number of sweets. They will not be satisfied if they have any sweets left which cannot be divided.

Halloween treats

Sample Input

4 5 (the number of children and the number of neighbors, respectively.)

1 2 3 7 5

3 6

7 11 2 5 13 17

0 0

Sample Output

3 5

2 3 4

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

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

排序与分治算法

主要内容:

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

排序

什么是排序？ Sorting

排序是计算机内经常进行的一种操作，其目的是将一组“无序”的记录序列调整为“有序”的记录序列。

例如：

52, 49, 80, 36, 14, 58, 61, 23, 97, 75

14, 23, 36, 49, 52, 58, 61, 75, 80, 97

排序

排序：

假设含 n 个记录的序列为 $\{ R_1, R_2, \dots, R_n \}$

其相应的关键字序列为 $\{ K_1, K_2, \dots, K_n \}$

这些关键字相互之间可以进行比较，即在它们之间存在着这样一个关系： $K_{p1} \leq K_{p2} \leq \dots \leq K_{pn}$

按此固有关系将上式记录序列重新排列为

$\{ R_{p1}, R_{p2}, \dots, R_{pn} \}$

的操作称作**排序**。

排序

❖ 在待排记录序列中，任何两个关键字相同的记录，用某种排序方法排序后相对位置不变，则称这种排序方法是稳定的，否则称为不稳定的。

❖ 例如：

❖ 待排序列： **49**,38,65,97,76,13,27,49

❖ 排序后： 13,27,38,**49**,49,65,76,97 — 稳定

❖ 排序后： 13,27,38,49,**49**,65,76,97—不稳定

快速排序

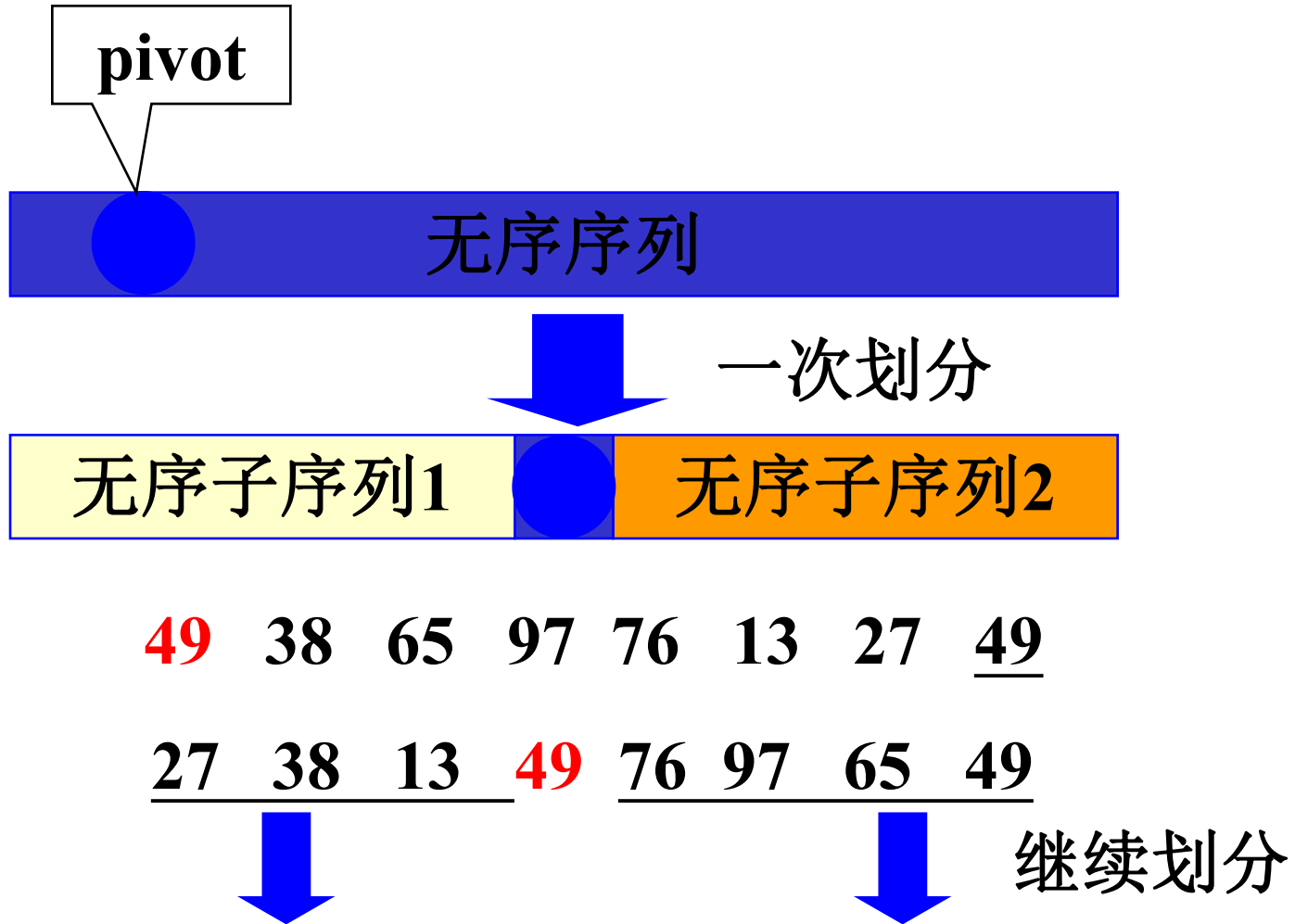
❖ 基本思想:

- 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



用第一个记录作pivot

49

49 38 65 97 76 13 27 49
l ↑ ↑ *h* ↑ ← *h*

27 38 65 97 76 13 49
l ↑ → ↑ *l* — *h* ↑ ↑

27 38 97 76 13 65 49
l ↑ ↑ *h* ↑ ← *h* ↑

27 38 13 97 76 65 49
l ↑ → ↑ *l* — *h* ↑ ↑

27 38 13 76 97 65 49
l ↑ *h* ↑ ← *h* ↑

27 38 13 49 76 97 65 49
l ↑ ↑ *h*

一趟快速排序

快速排序

[27 38 13] 49 [76 97 65] 49 一趟快速排序

[13] 27 [38] 49 [49 65] 76 [97] 两趟快速排序

13 27 38 49 49 65 76 97 三趟快速排序

快速排序

快速排序

❖ 设初始时

- low指针指向第一个记录;
- high指针指向最后一个记录;

❖ 一趟快速排序的算法过程:

1. 将第一个记录设置为pivot
2. 从表的两端交替地向中间扫描, 直到两个指针相遇
 - 先从高端扫描
 - 找到第一个比pivotkey小的记录
 - 将该记录移动到low指针指向的地方;
 - 再从低端扫描
3. 将pivot移动到low指针位置, 并返回该位置

```
int Partition(SqList &L, int low, int high)
```

```
{ /*对顺序表L中子表r[low..high]的记录  
   作一趟快速排序，并返回pivot记录所在位置。*/
```

```
    L.r[0]=L.r[low]; //用第一个记录作pivot记录  
    pivotkey=L.r[low].key; // pivotkey是pivot关键字
```

```
    while(low<high)
```

```
    { //从表的两端交替地向中间扫描
```

```
        while(low<high && L.r[high]. Key>=pivotkey)  
            --high;
```

```
        L.r[low]=L.r[high];
```

```
        while (low<high && L.r[low]. Key<=pivotkey)  
            ++low;
```

```
        L.r[high]=L.r[low];    } // 交替扫描结束
```

```
    L.r[low]=L.r[0]; //pivot位置
```

```
    return low;    //返回pivot位置
```

```
}//Partition
```

```
void Qsort(SqList &L, int low, int high)
```

```
{//对顺序表L中的子序列L.r[low.. high]作快速排序
```

```
  if (low<high)
```

```
  {   pivotloc=Partition(L, low, high);
```

```
      QSort(L, low, pivotloc-1);
```

```
      Qsort(L, pivotloc+1, high);
```

```
  }
```

```
}
```

递归结束
条件

```
void QuickSort(SqList &L )
```

```
{//对顺序表L快速排序
```

```
  QSort(L, 1, L.length);
```

```
}
```


快速排序特点

❖ 存储结构：顺序

❖ 时间复杂度

- 最坏情况：每次划分选择pivot是最小或最大元素
- 最坏情况： $O(n^2)$
- 最好情况（每次划分折半）： $O(n\log_2 n)$
- 平均时间复杂度为 $O(n\log_2 n)$

❖ 空间复杂度

- 最坏情况： $O(n)$
- 最好情况（每次划分折半）： $O(\log_2 n)$
- 平均空间复杂度 $O(\log_2 n)$

❖ 稳定性

- 不稳定

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

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

递归函数

- 递归算法: 直接或间接地调用自身的算法
- 递归函数: 用函数自身给出定义的函数
- 边界条件与递归方程是递归函数的两要素
保障在有限次计算后得出结果

$$n! = \begin{cases} 1 & n = 0 \text{ 边界条件} \\ n(n-1)! & n > 0 \text{ 递归方程} \end{cases}$$

$$fib(n) = \begin{cases} 1 & n \leq 1 \\ fib(n-1) + fib(n-2) & n > 1 \end{cases}$$

```
int fib(int n)
{ if (n <= 1) return 1;
  return fib(n-1)+fib(n-2); }
```

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

分治基本过程

分: 将问题分解成若干个子问题

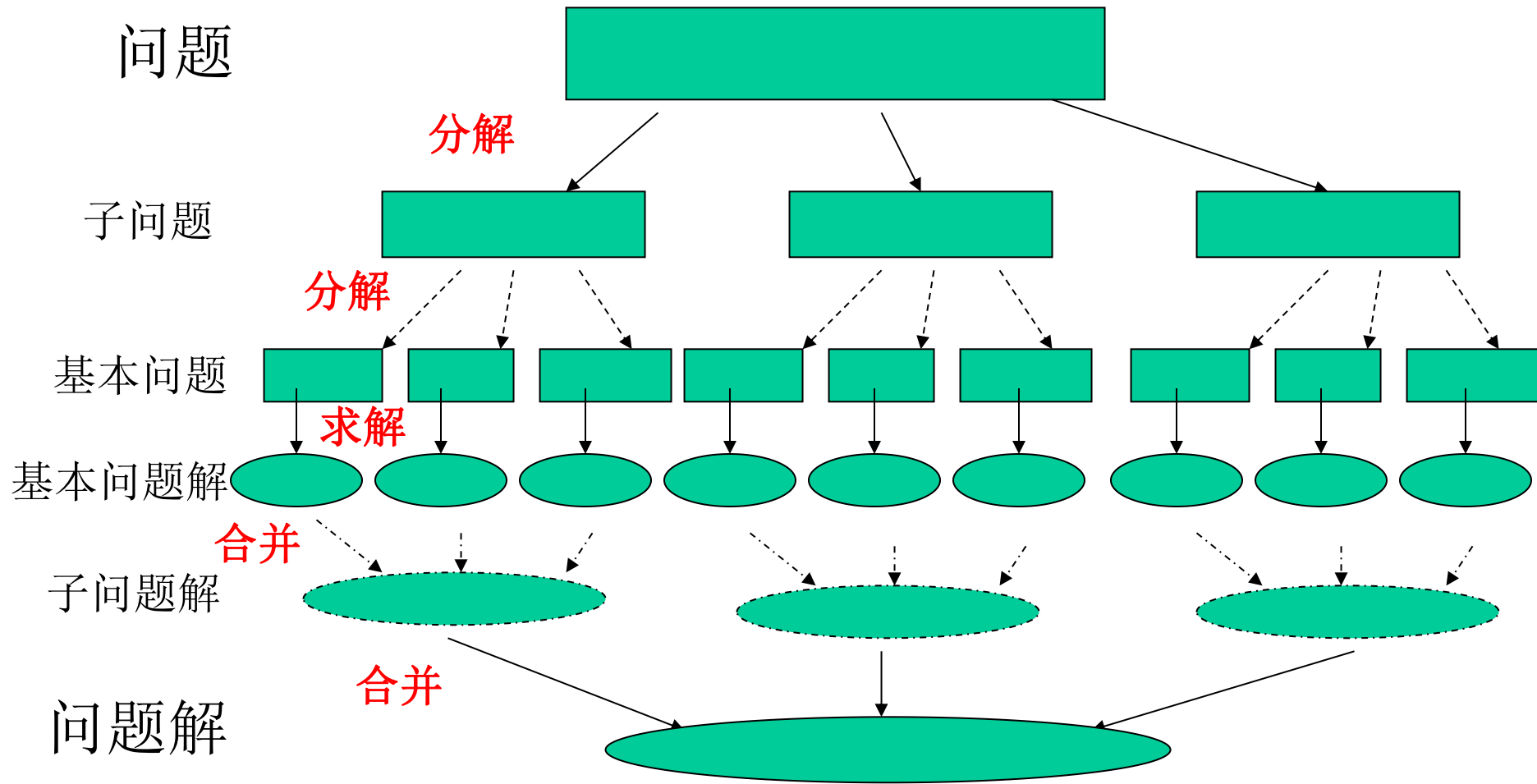
治: 递归求解子问题

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

例2: 求 $f(x) = \ln x + 2x - 6$ 在 $(2, 3)$ 中的近似零点.

二分法

例3: 现给出4根电缆，长度分别为8.02、7.43、4.57、5.39，要你把它们分割成11根等长的电缆，每根电缆的最大长度是多少？

使用二分法的条件

- 解具有递增（或递减）的特性
- 对于某个值不是问题的解，那么比这个值大（或小）的值均不是问题的解

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。

```
1. 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

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

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

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

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

$$n = 2^m$$

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

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

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

$$= 3n/2 - 2$$

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

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

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

$$k=\log_2 n$$

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 $n=2^m$ 。

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

大整数乘法

- 输入：两个 n 位二进制数 X, Y
- 输出： $X \times Y$
- 输入规模： n

方案一：直接计算

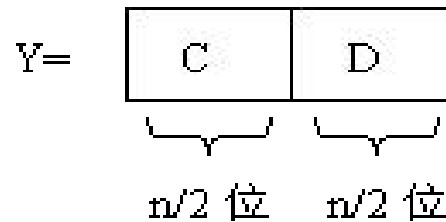
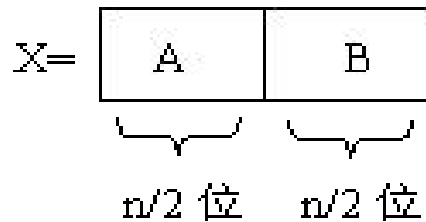
时间复杂度 $O(n^2)$

方案二：尝试分治法

$$\begin{array}{r} 1\ 0\ 1\ 1 \\ \times 1\ 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 1 \\ 1\ 0\ 1\ 1 \\ + 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \end{array}$$

大整数乘法: 分治

将X和Y都分两段, 即 $X=A2^{n/2}+B$, $Y=C2^{n/2}+D$



$$XY=(A2^{n/2}+B)(C2^{n/2}+D)=AC2^n+(AD+BC) 2^{n/2}+BD$$

Mt(X,Y,n)

1. if $n=1$, return($X*Y$)
2. $X=[A,B]$, $Y=[C,D]$, $k=\lceil n/2 \rceil$
3. $a=Mt(A,C,k)$, $b=Mt(A,D,k)$,
4. $c=Mt(B,C,k)$, $d=Mt(B,D,k)$
5. return($a2^n+(b+c)2^k+d$)

divide-and-conquer(P)

```
{ if ( | P | <= n0) adhoc(P);  
  else  
  { divide P into  $P_1, P_2, \dots, P_a$ ;  
    for ( $i=1, i \leq a, i++$ )  
       $y_i = \text{divide-and-conquer}(P_i)$ ;  
    return merge( $y_1, \dots, y_a$ );  
  } }
```


时间复杂度T(n)分析

将X和Y都分两段, 即 $X=A2^{n/2}+B$, $Y=C2^{n/2}+D$



$$XY=(A2^{n/2}+B)(C2^{n/2}+D)=AC2^n+(AD+BC) 2^{n/2}+BD$$

令T(n)为n位乘法所需时间, T(n)的构成:

4次n/2位乘法, 3次不超过n位加法, 2次移位

$$T(n)=\begin{cases} O(1) & n=1 \\ 4T(n/2)+O(n) & n>1 \end{cases}$$

由分治主定理 $T(n)=O(n^2)$ 改进: AC, AD, BC, BD不独立

大整数乘法: 改进的分治

将X和Y都分两段, 即 $X=A2^{n/2}+B$, $Y=C2^{n/2}+D$

$$XY=(A2^{n/2}+B)(C2^{n/2}+D)=AC2^n+(AD+BC)2^{n/2}+BD$$

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

$T(n)$ 构成: 3次 $n/2$ 位乘法, 6次不超过 n 位加法, 2次移位

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

根据分治主定理 $T(n) = \Theta(n^{\log_2 3})$

注: 分多段可改进(见本章习题);

Strassen矩阵乘法(8次乘法改为7次乘法)

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

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

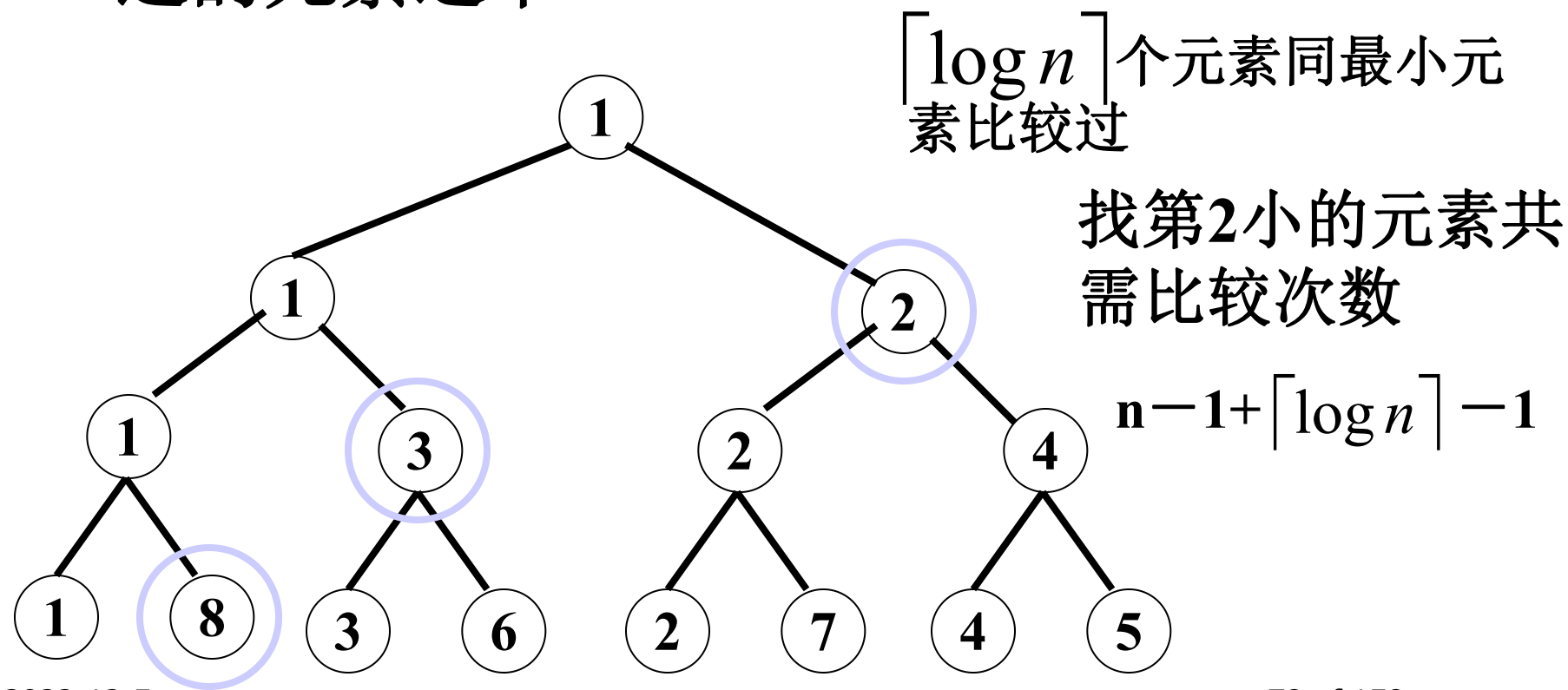
每一次比较只能确定一个失败者，确定 $n-1$ 个在某次比较中的失败者需要 $n-1$ 次比较

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

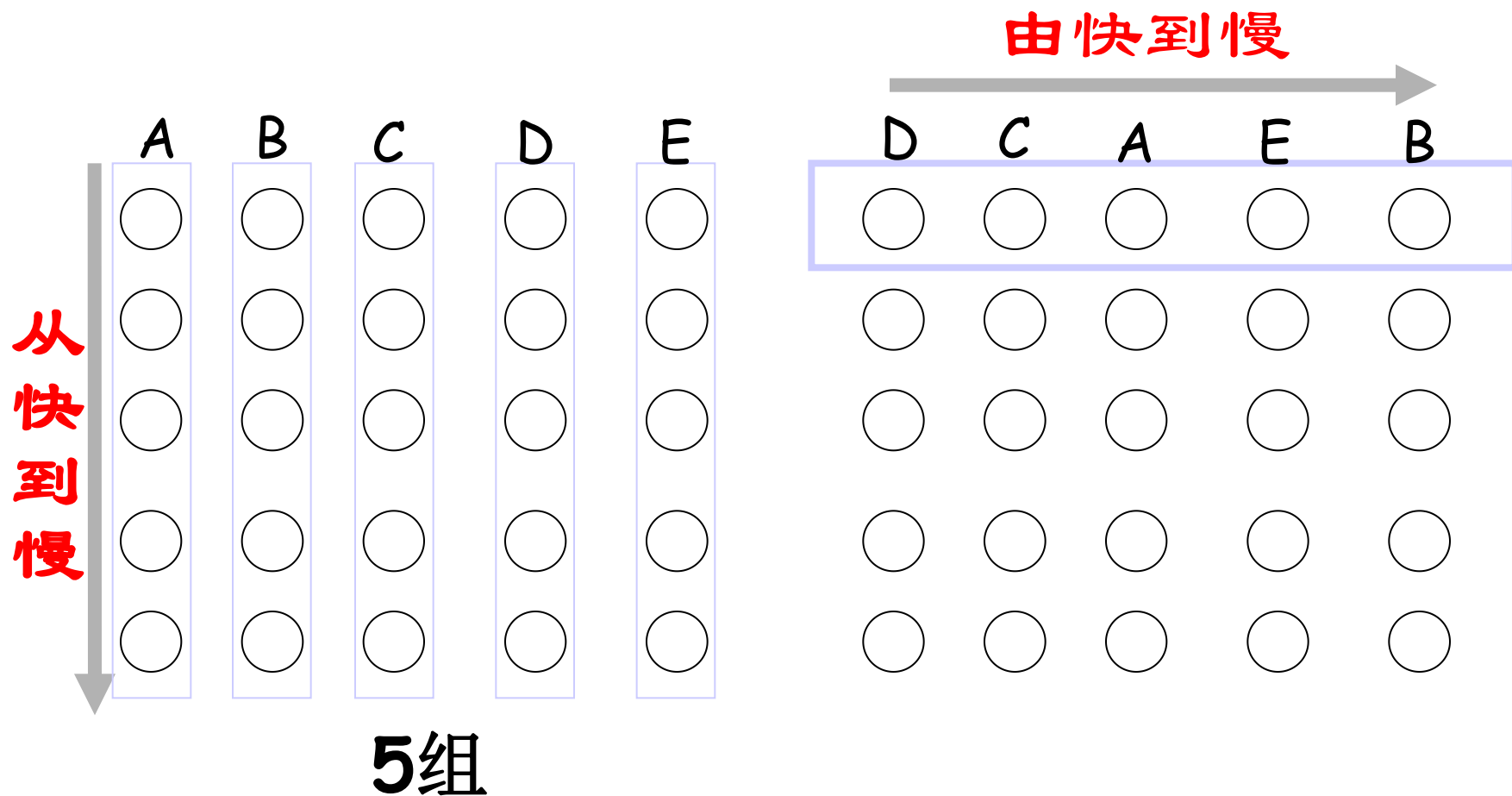
求第2小的元素

- 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



引例

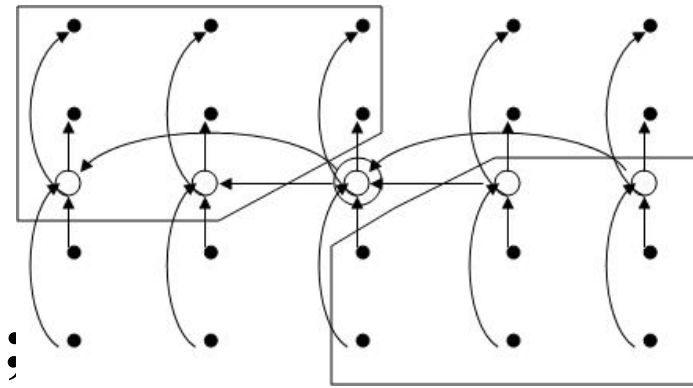
- 在中国的古代，25匹马通过赛跑来决出前3名，每5匹马一组，问最少需要几组？



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:

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

$j = i - p + 1;$

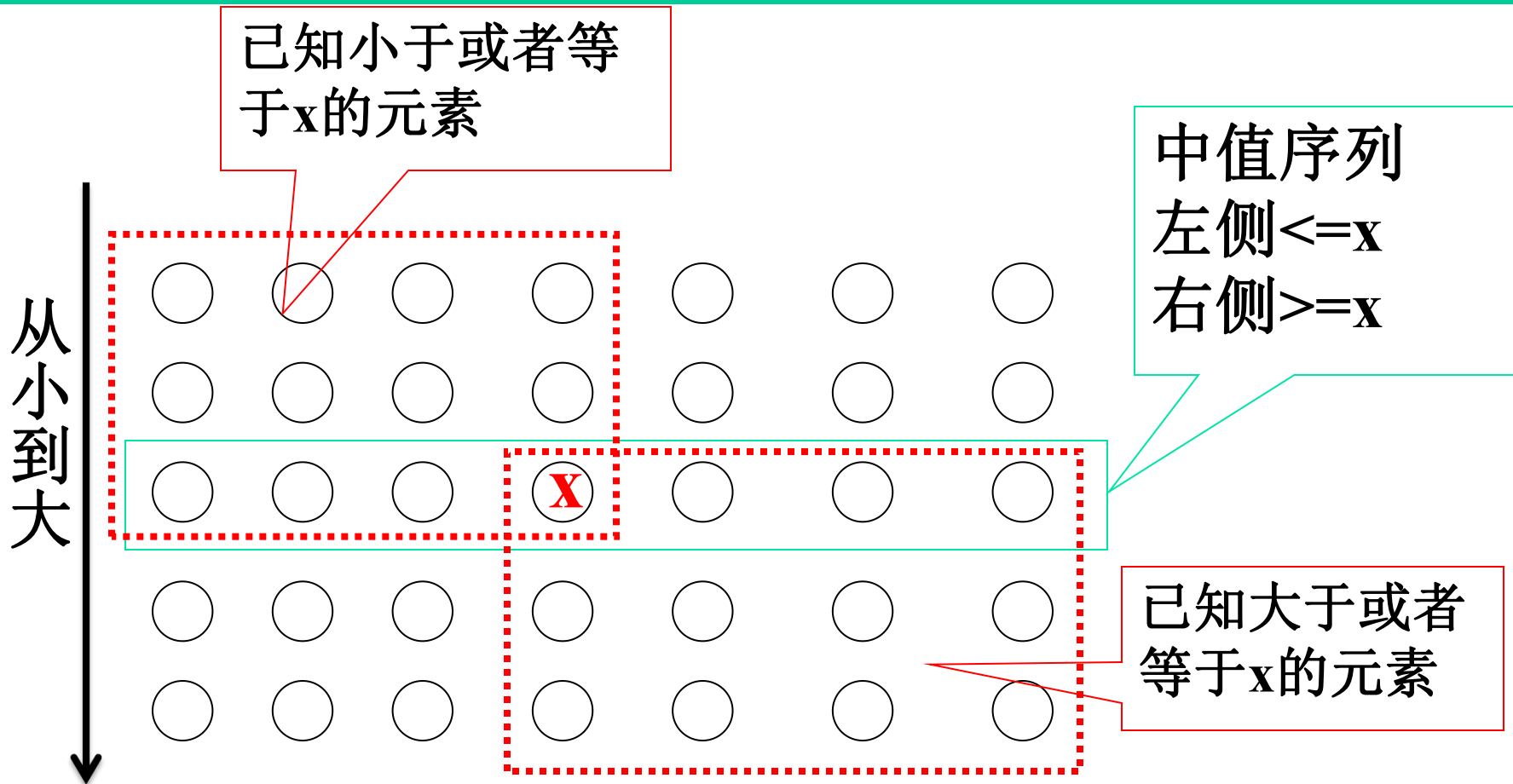
$\text{if } (k == j) \text{ return } a[i];$

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

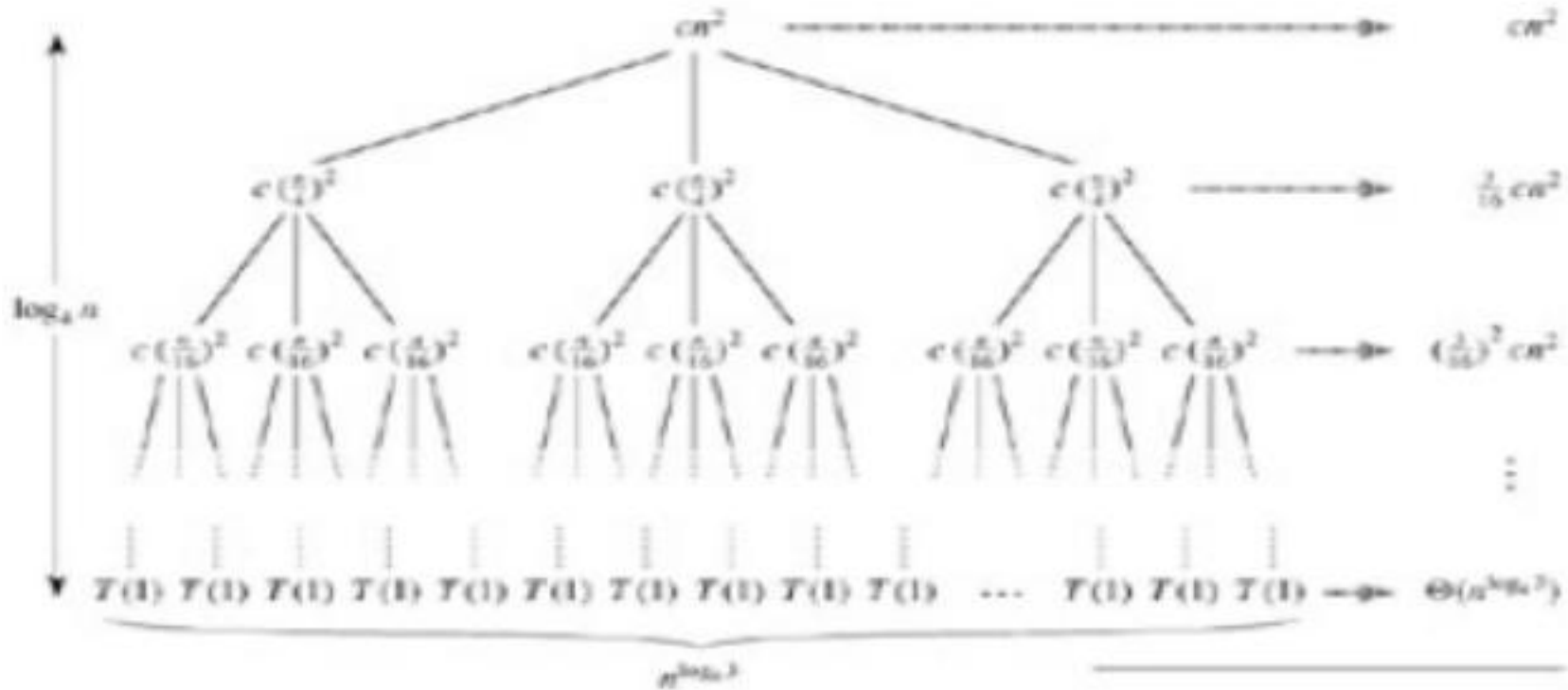
$\text{else return Select}(a, i+1, r, k-j);$

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

线性时间选择程序



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



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

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

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

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

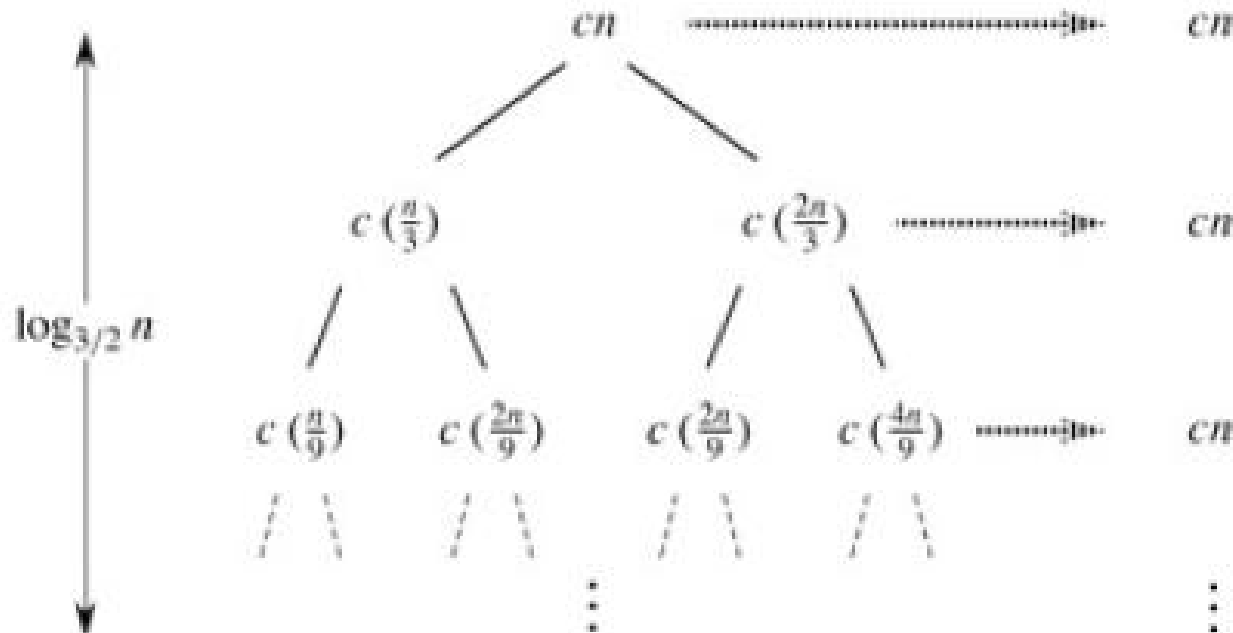
$$T(n) = T(n/3) + T(2n/3) + cn$$

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

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

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

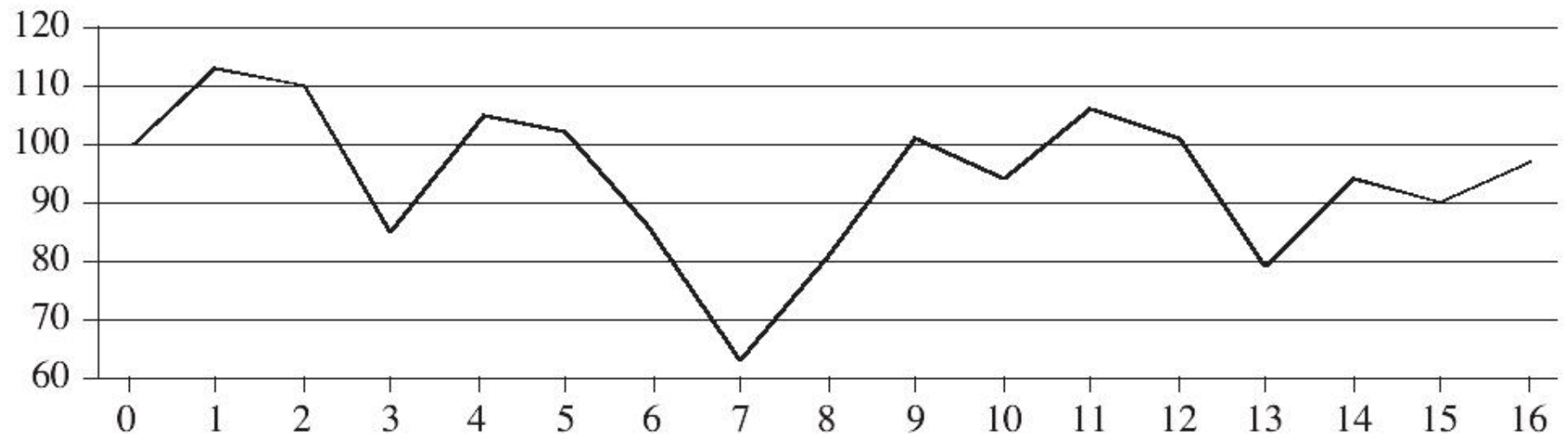
$$+ [c(n/3) + c(2n/3)] + cn$$



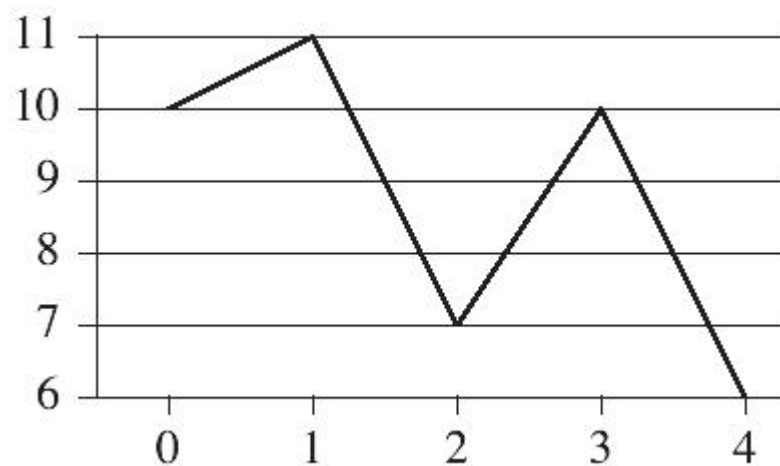
Total: $O(n \lg n)$

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

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

```

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

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

1. 排序算法
2. 递归函数
3. 分治原理, 主定理, 二分法
4. 大整数乘法
5. 线性时间选择
6. 最大子段和
7. 最接近点对问题

Have you ever played quoit in a playground? Quoit is a **game in which flat rings are pitched at some toys, with all the toys encircled awarded.**

In the field of Cyberground, the position of each toy is fixed, and the ring is carefully designed so it can only encircle one toy at a time. On the other hand, to make the game look more attractive, the ring is designed to have the largest radius. Given a configuration of the field, you are supposed to find the radius of such a ring. Assume that all the toys are points on a plane. A point is encircled by the ring if the distance between the point and the center of the ring is strictly less than the radius of the ring. If two toys are placed at the same point, the radius of the ring is considered to be 0.

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 $k = 2 : n-1$

$O(n)$ {
 $O(1)$ {

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: $d = \min \{ d_L, d_R \}$

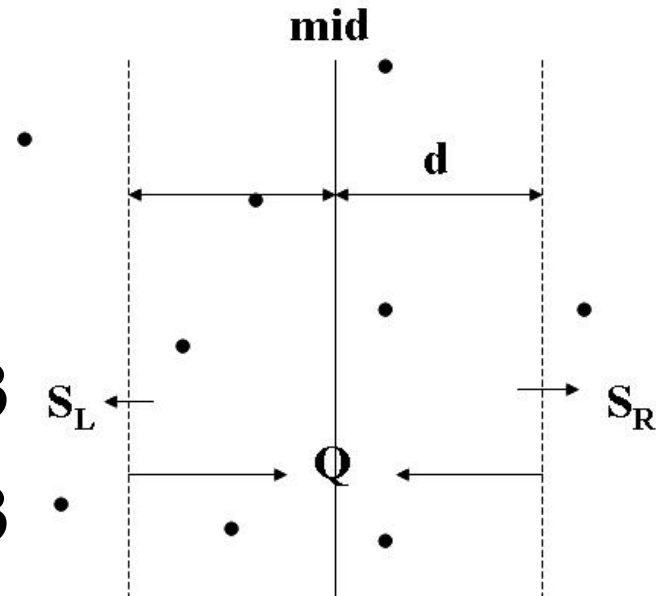
4. 取 $Q = \{ p \in S \mid |x(p) - \text{mid}| < d \}$

5. 逐对求 Q 中最近点对的距离 d .

分 $O(n)$, 治 $2T(n/2)$, 合 $O(n^2)$

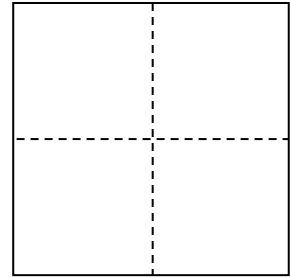
根据分治主定理 $T(n) = O(n^2)$

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

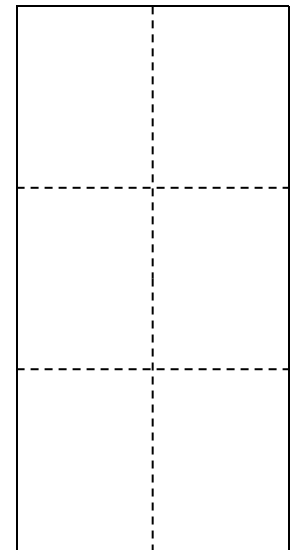


鸽巢(抽屉)原理的简单应用

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 .



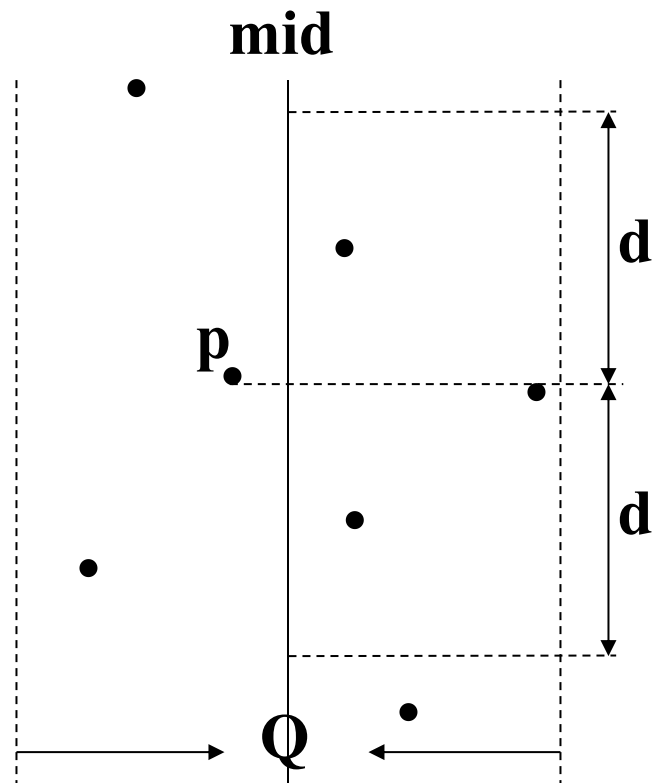
任取一个 $d \times 2d$ 矩形内点集 A ,
若 A 中任意两点距离都 $\geq d$, 则 A 中点数 ≤ 6 .



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

方案一：Q左右分开

Q右侧中与p距离 $< d$ 的点数 ≤ 6

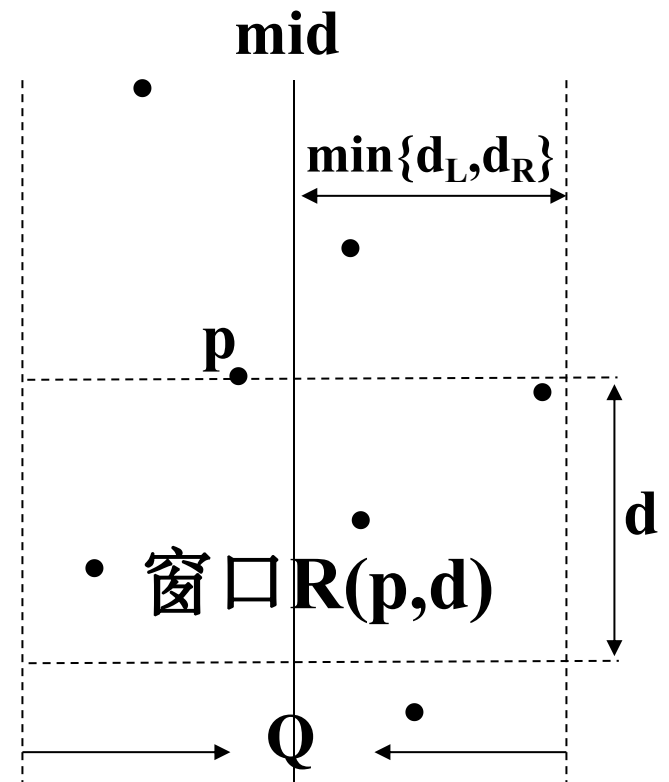


方案二: 检查p下方的点

- 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. $j = i + 1$,

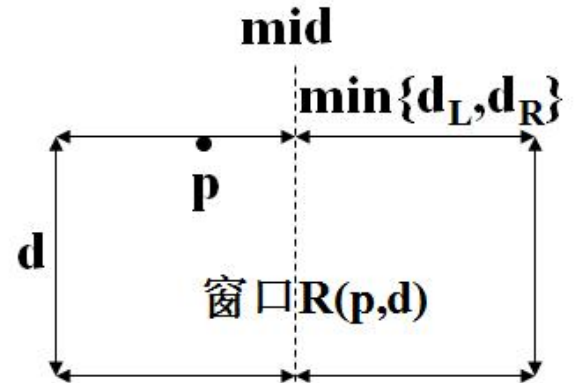
7. while($y(j) - y(i) < d$)

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?

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



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

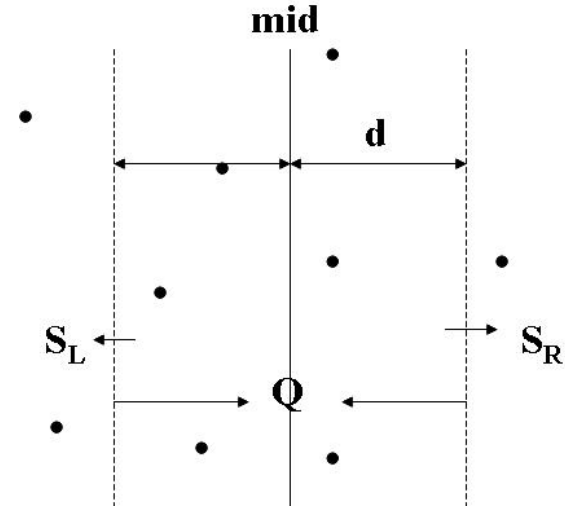
称5--8过程为:
对Q中每个点p,
检查窗口 $R(p, d)$,
更新最短距离d

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



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

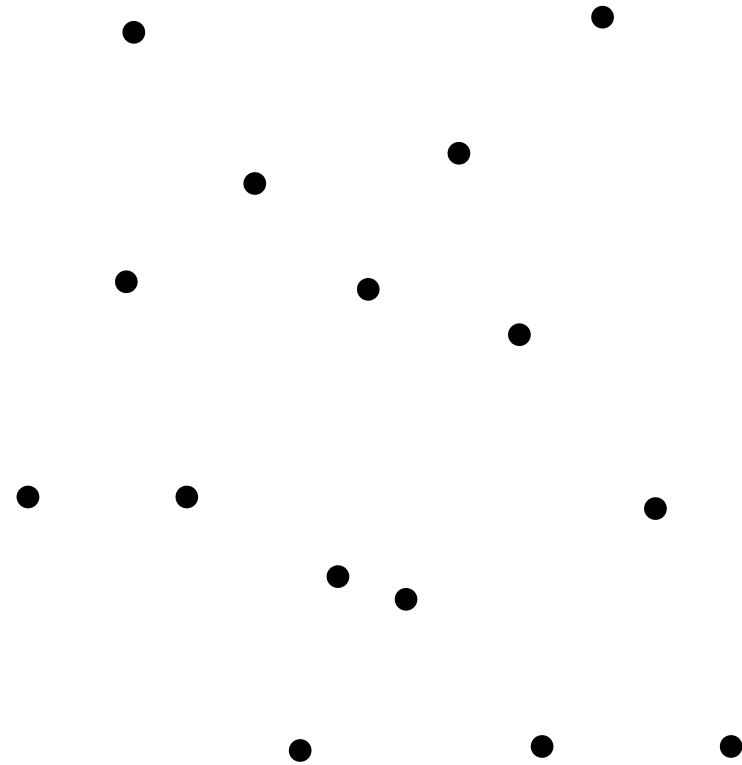
算法图示--初始

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 $d_L(d_R)$

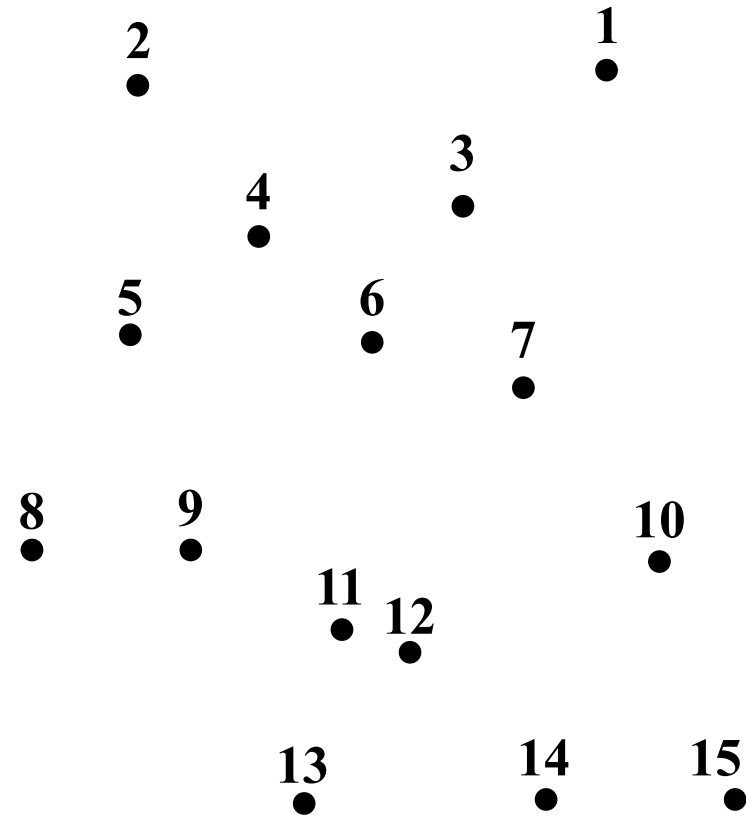
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

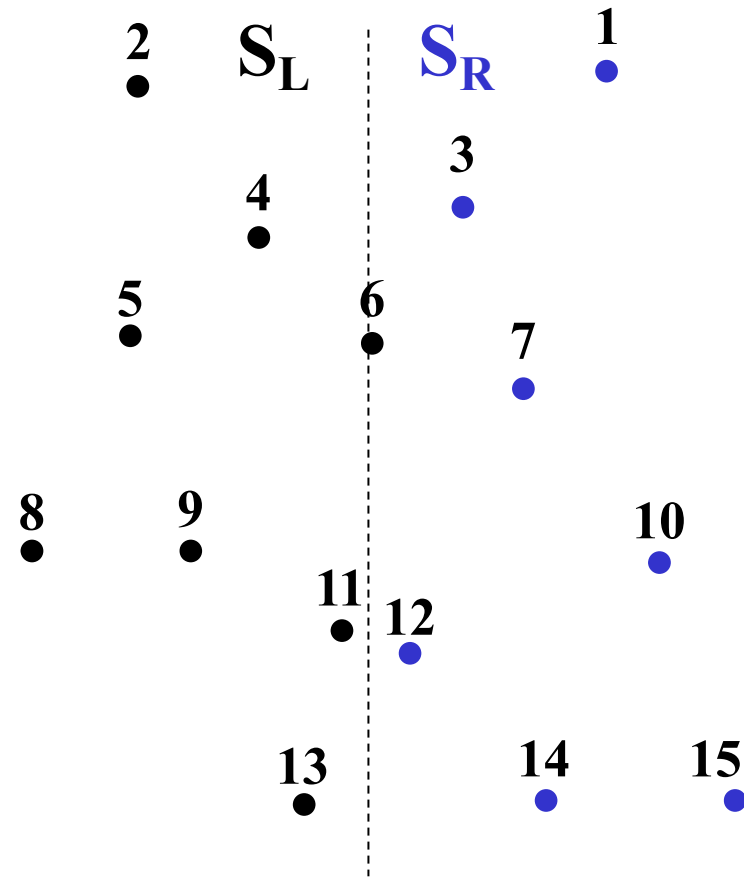
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

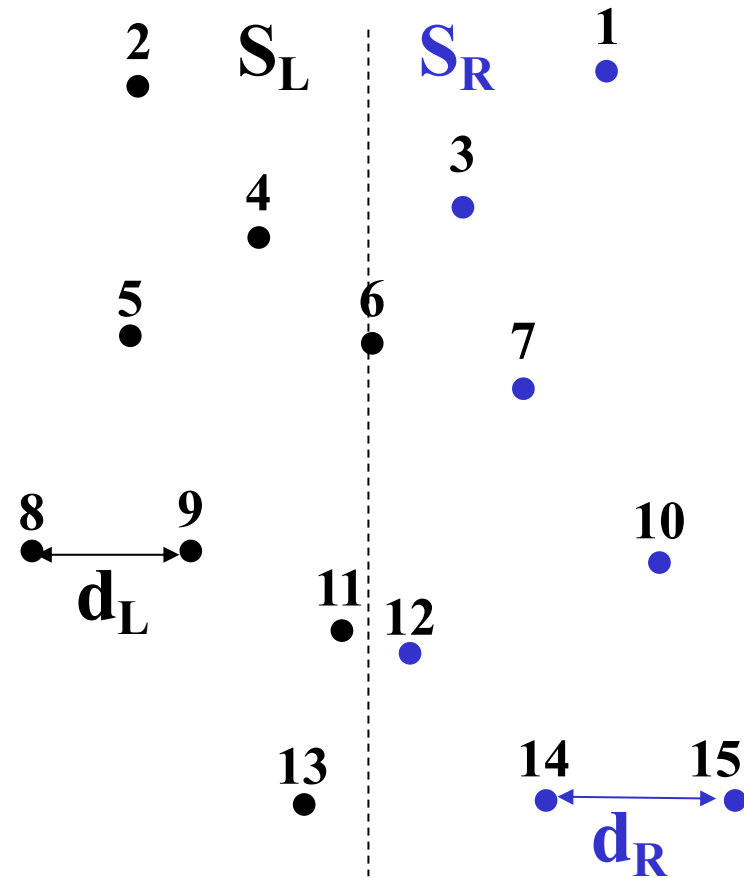
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

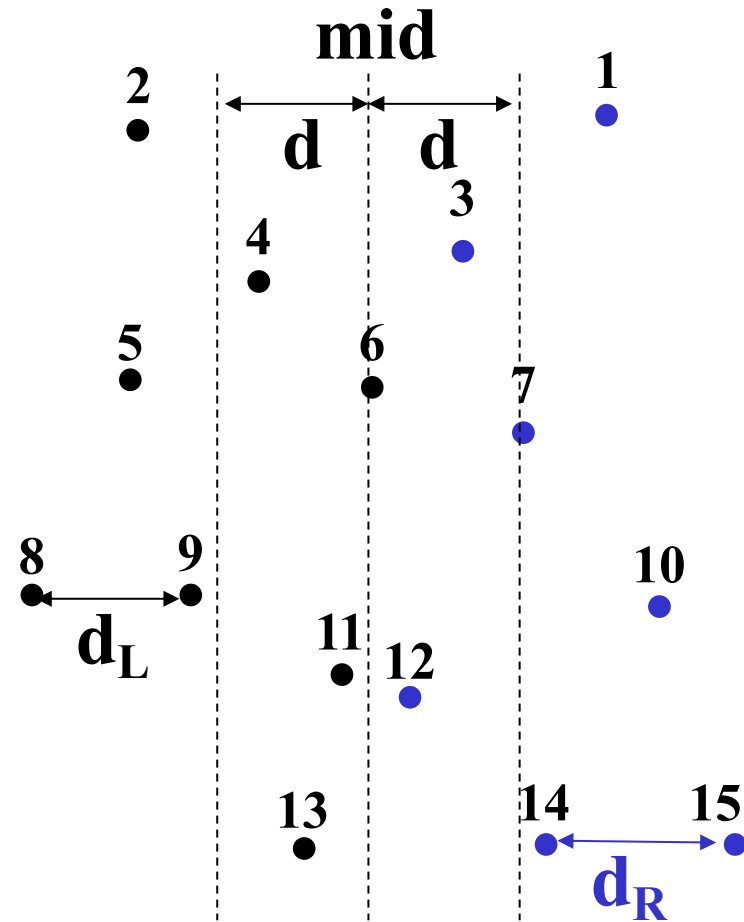
3. 合: $d = \min \{ d_L, d_R \}$

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

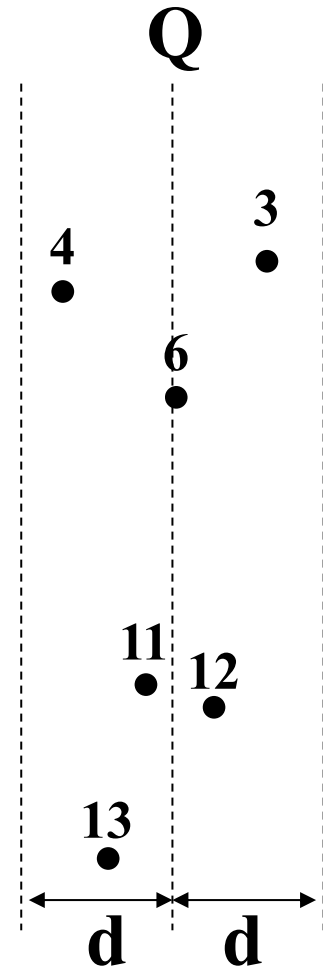


算法图示--合4

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 $d_L(d_R)$

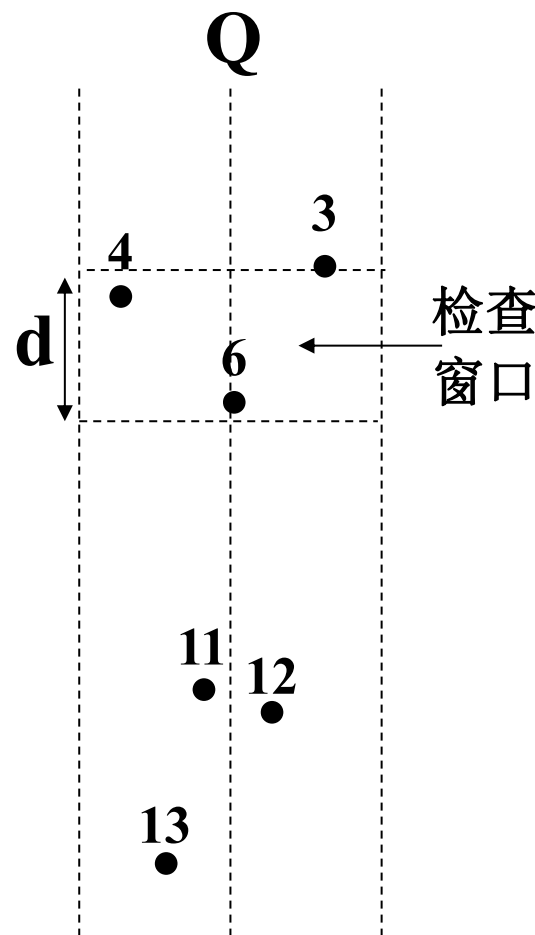
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

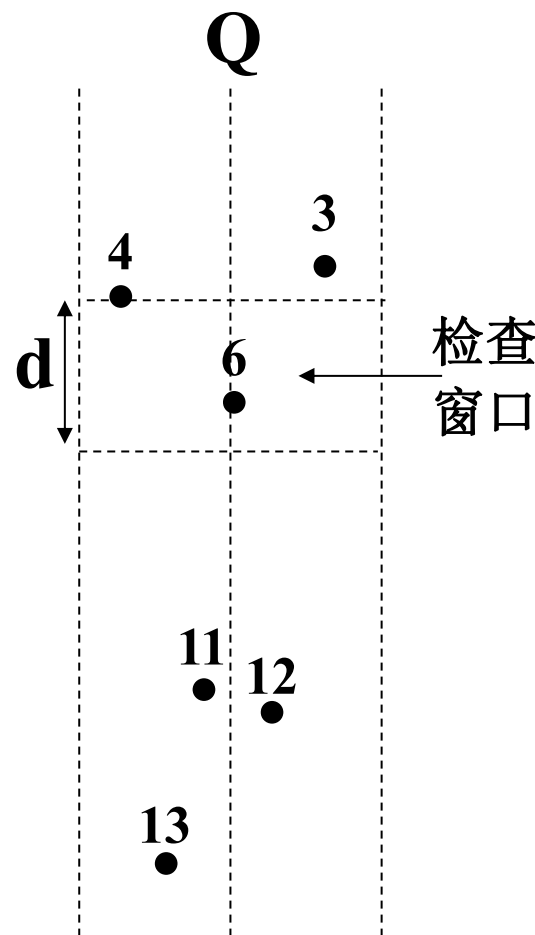
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

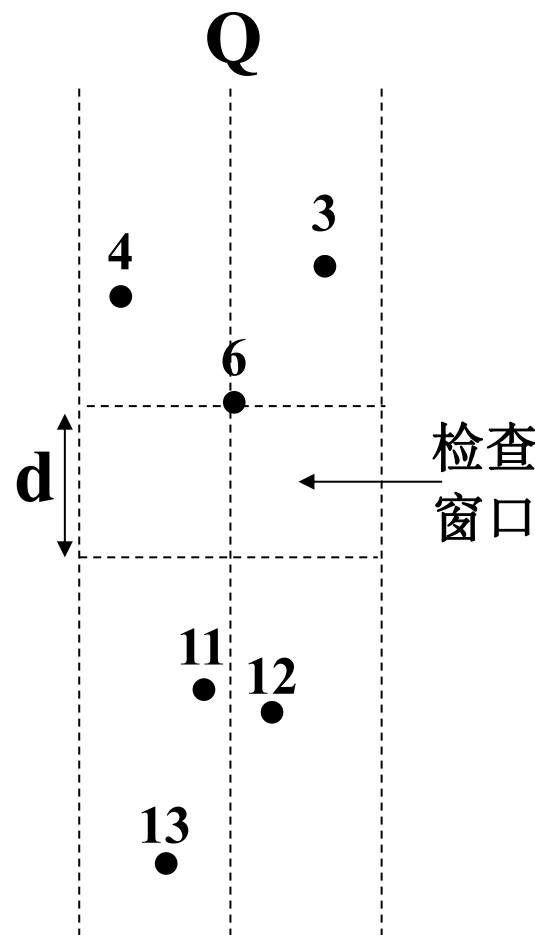
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

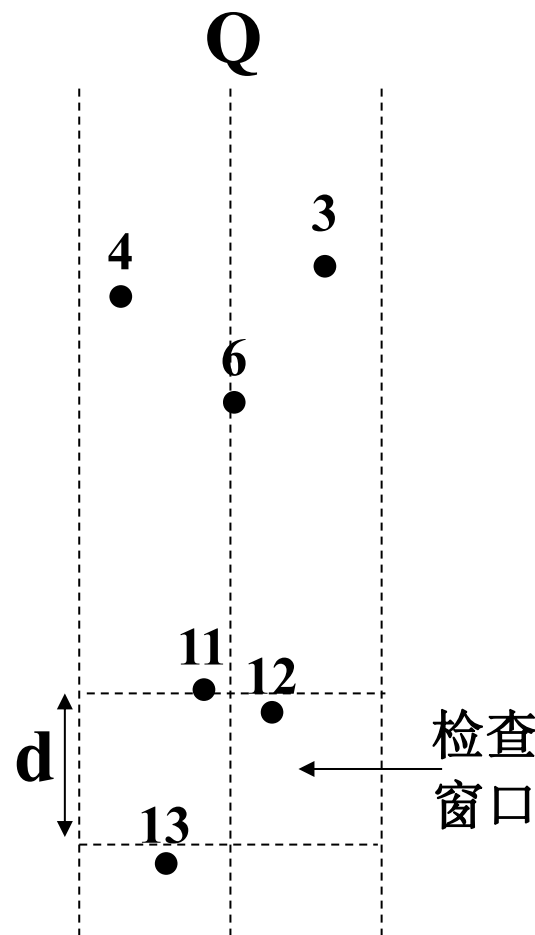
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

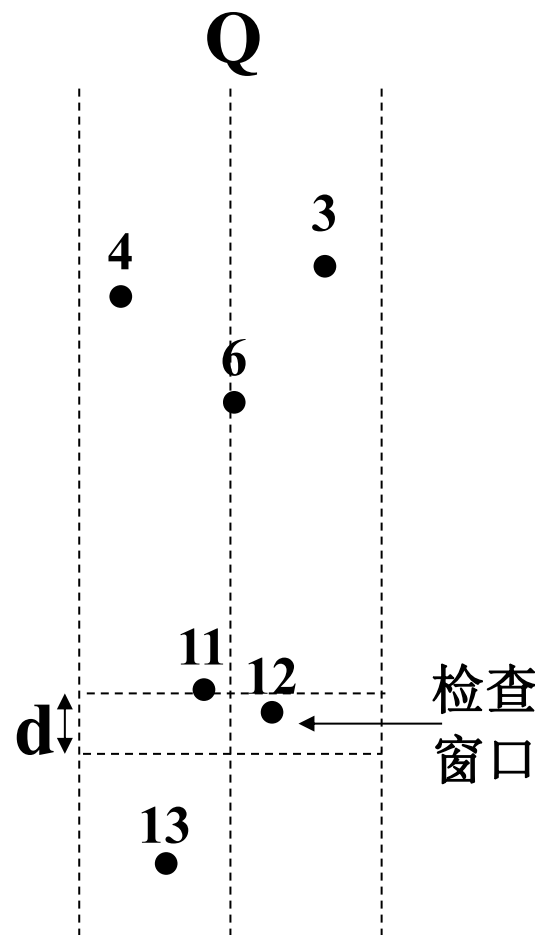
3. 合: $d = \min \{ d_L, d_R \}$

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 $d_L(d_R)$

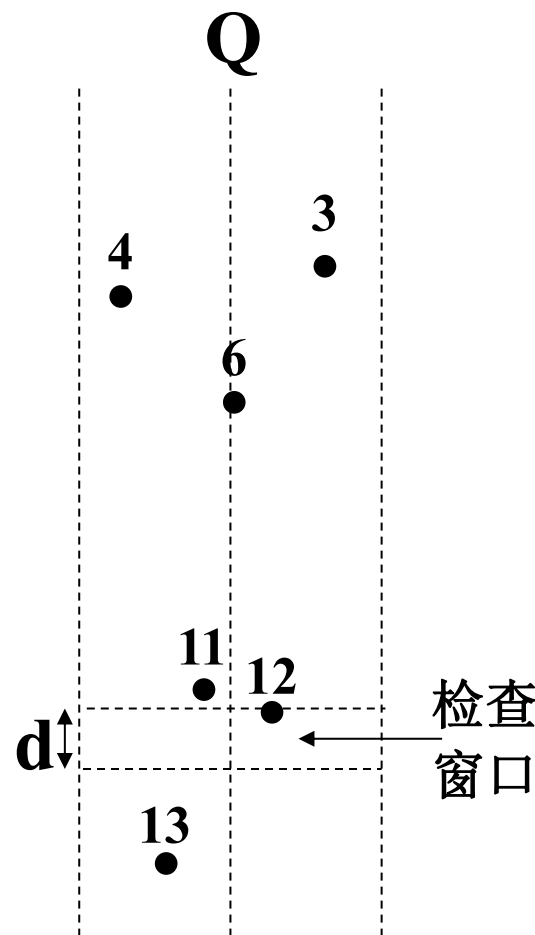
3. 合: $d = \min \{ d_L, d_R \}$

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

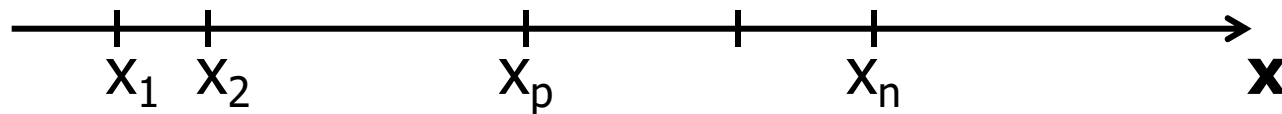
某公司有五个分公司依次设置在同一条铁路线的沿线A、B、C、D、E站。现在该公司希望在该铁路沿线设立一个仓库，要求该仓库离这五个站的火车行驶距离之和最小。如用数轴表示该铁路线，A、B、C、D、E各站的坐标依次为 a 、 b 、 c 、 d 、 e （ $a < b < c < d < e$ ），则经过数学计算，该仓库大致应设置在坐标 (1) 处。

- (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：

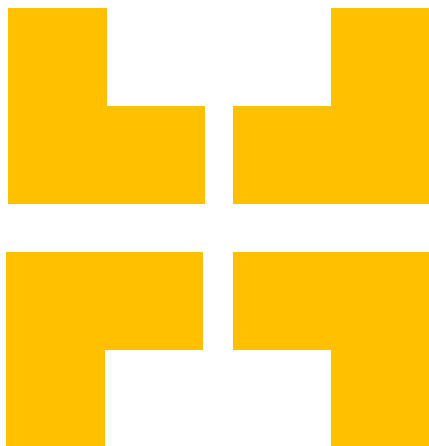
$$x_p = \begin{cases} x_{(n+1)/2} \\ \text{中间两点的闭区间上} \end{cases}$$

当 n 为奇数时

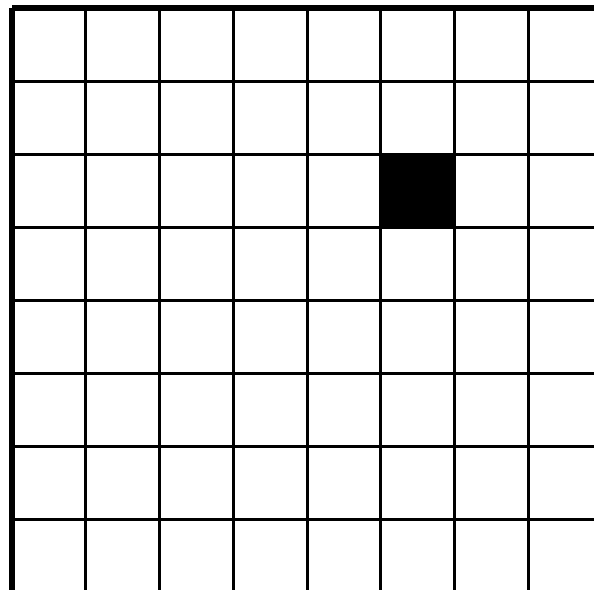
当 n 为偶数时

附录：棋盘覆盖

L型骨牌



$2^k \times 2^k$ 棋盘

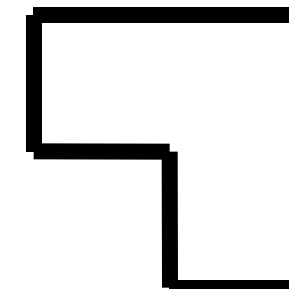
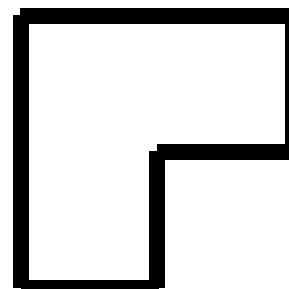
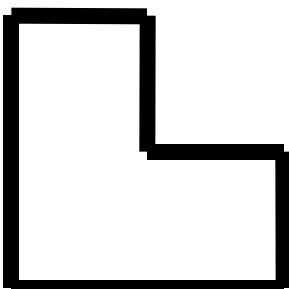
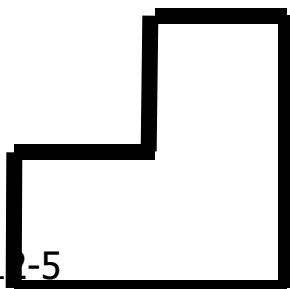
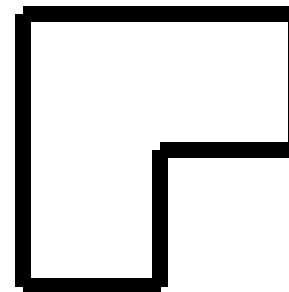
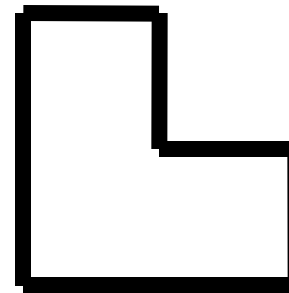
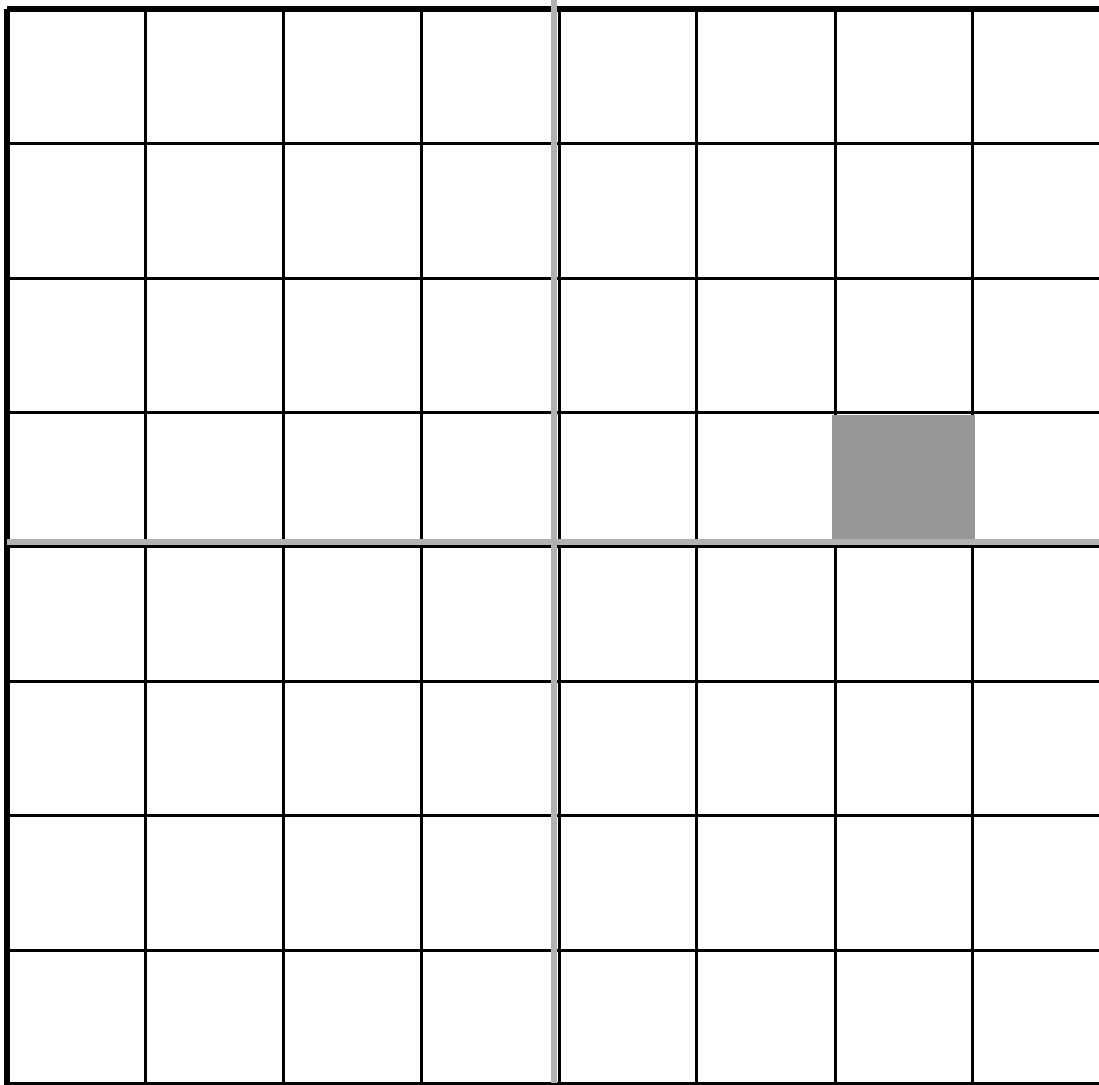


输入: k , 代表 $2^k \times 2^k$ 棋盘

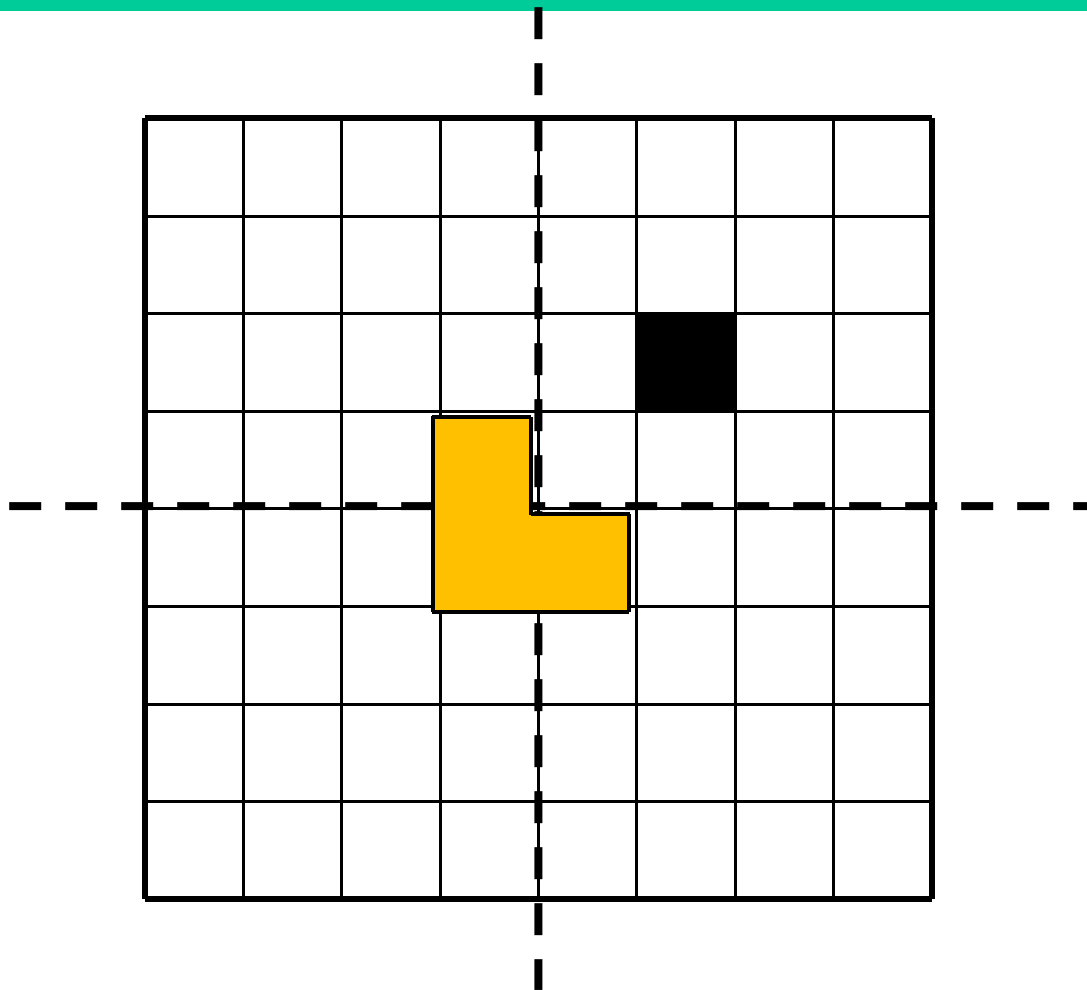
输出: 用 L 型骨牌覆盖棋盘的方案

说明: 有很多方案,
构造出一种方案即可

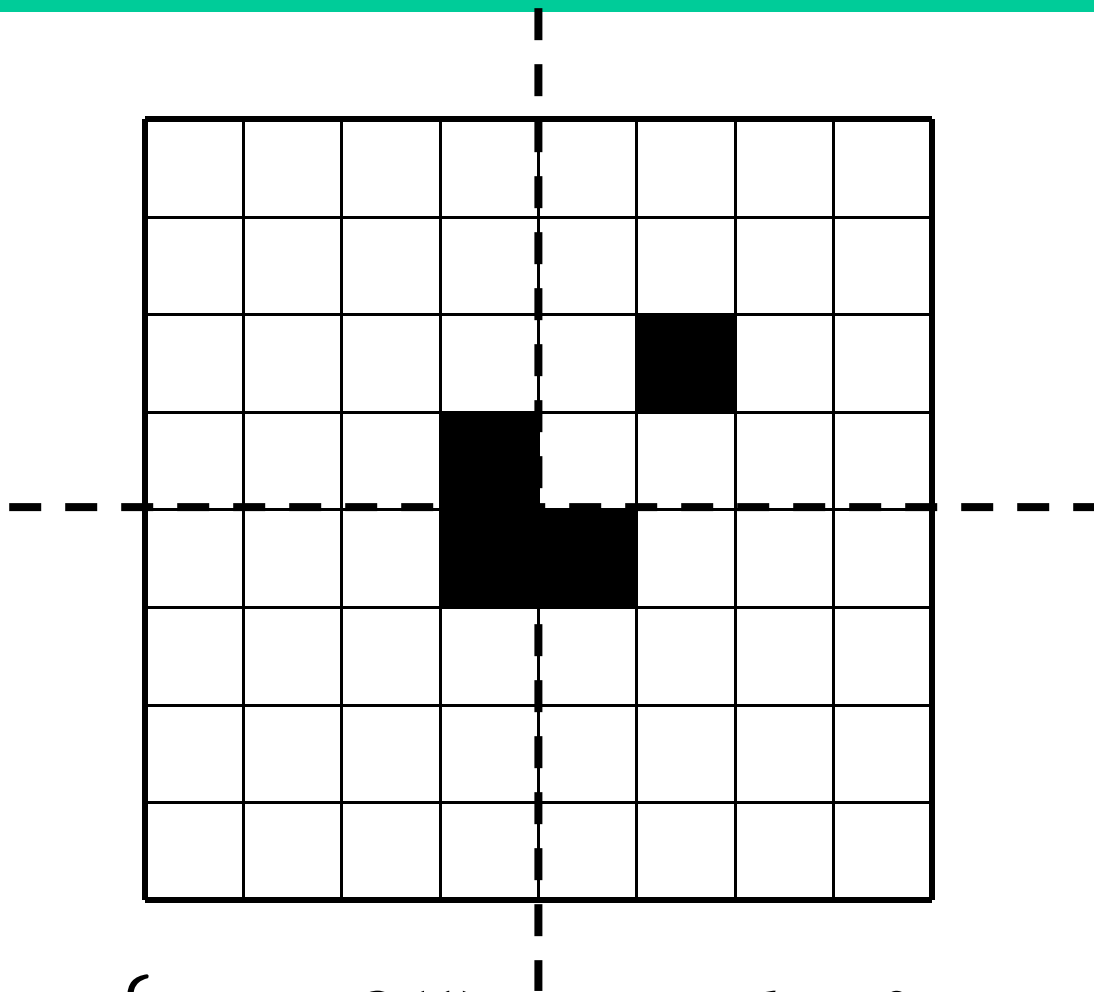
3	3	4	4	8	8	9	9
3	2	2	4	8	7	7	9
5	2	6	6	10		7	11
5	5	6	1	10	10	11	11
13	13	14	1	1	18	19	19
13	12	14	14	18	18	17	19
15	12	12	16	20	17	17	21
15	15	16	16	20	20	21	21



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

课堂练习-猜牌问题

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：甲只能回答“是”或“否”。

作业

1-1 求下列函数的渐近表达式:

$3n^2+10n$; $n^2/10+2^n$; $2^{1+1/n}$; $\log n^3$; $10\log 3^n$.

1-2 试论 $O(1)$ 与 $O(2)$ 的区别.

作业

- 1-4 (1) 假设某算法在输入规模为 n 时的计算时间为 $T(n)=3 \times 2^n$. 在某台计算机上实现并完成该算法的时间为 t 秒. 现有另一台计算机, 其运行速度是第一台的64倍, 那么在这台新机器上用同一算法在 t 秒内能解输入规模为多大的问题?
- (2) 若上述算法的计算时间改进为 $T(n)=n^2$, 其余条件不变, 则在新机器上用 t 秒时间能解输入规模为多大的问题?
- (3) 若上述算法的计算时间改进为 $T(n)=8$, 其余条件不变, 那么在新机器上用 t 秒时间能解输入规模为多大的问题?

作业

- 2-8 设 n 个不同的整数排好序后存于 $T[1:n]$ 中. 若存在一个下标 i , $1 \leq i \leq n$, 使得 $T[i]=i$. 设计一个有效算法找到这个下标. 要求算法在最坏情况下的计算时间 $O(\log n)$.
- 2.9 设 $T[0:n-1]$ 是 n 个元素的数组. 对任一元素 x , 设 $S(x)=\{ i \mid T[i]=x \}$. 当 $|S(x)| > n/2$ 时, 称 x 为主元素. 设计一个线性时间算法, 确定 $T[0:n-1]$ 是否有一个主元素.
- 2.25 在线性时间选择算法中, 输入元素被划分为5个一组, 如果将它们划分为7个一组, 该算法仍然是线性时间算法吗? 划分成3个一组又怎样?