算法与计算理论

课程内容

数据结构

- 概述
- 线性表
- 栈与队列
- 数组与广义表
- 串
- 树
- 图
- 查找
- 内部排序
- 外部排序

算法与计算理论

- 概述
- 分治
- 动态规划
- 贪心
- 回溯
- ……
- ……
- ……
- ……
- 计算模型
- 可计算理论
- 计算复杂性

Contents

本章内容

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

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

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

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

递归函数-Ackermann函数

[C]
$B_n(1)$

$$B_k(j)=\begin{cases} j+1 & k=0 \\ B_{k-1}^{(j+1)}(j) & k>0 \end{cases}$$

上标表示复合次数。当上标为0时，定义 $A_k^{(0)}(j)=j$

[C]:

$B_0(1)=2$

$B_1(1)=B_0(B_0(1))=B_0(2)=3$

$B_2(1)=B_1(B_1(B_1(1)))=B_1(B_1(3))=B_1(B_0(B_0(3)))=B_1(B_0(4))=B_1(5)=B_0(B_0(5))=$

$B_3(1)= B_2\,B_2(B_2(B_2(1)))=\ldots= 2047$, $B_4(1) >> 2^{2048}$

递归函数-Ackermann函数

$$D_n = \begin{cases} 1 & n = 0 \\ 2^{D_{n-1}} & n > 0 \end{cases}$$

$D_1=2^1$, $D_2=2^2=4$, $D_3=2^4=16$, $D_4=2^{16}=65536$, $D_5=2^{65536}$

递归函数-Ackermann函数

[王]

$A(n,m)$

$$
\begin{cases}
A(1,0) = 2 & \\
A(0,m) = 1 & m \geq 0 \\
A(n,0) = n + 2 & n \geq 2 \\
A(n,m) = A(A(n-1,m),m-1) & n,m \geq 1
\end{cases}
$$

$A(1,1)$=A(A(0,1),0)=2

$A(1,2)$= A(A(0,2),1) =A(1,1) = 2

$A(2,2)$= A(A(1,2),1)=A(2,1)=A(A(1,2),0)=4

$A(3,3)$=A(A(2,3),2) = A(4,2) = A(A(3,2),1)=A(8,1) = A(A(7,1),0)…=16

$A(4,4)=D_{65536}$.

Ackerman函数的反函数: 增长很慢

经常用于复杂度分析中

$\alpha(n) = \min\{ k \mid A(k,k) \geq n \}$

或 log*n $= \min\{ k \mid D_k \geq n \}$

$A(1,1)=2$,

$A(2,2)=4$,

$A(3,3)=D_3=16$,

$A(4,4)=D_{65536}$,

$$\alpha(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ 2 & 3 \leq n \leq 4 \\ 3 & 5 \leq n \leq 16 \\ 4 & 17 \leq n \leq D_{65536} \end{cases}$$

Contents

本章内容

递归函数

分治原理, 主定理，二分法

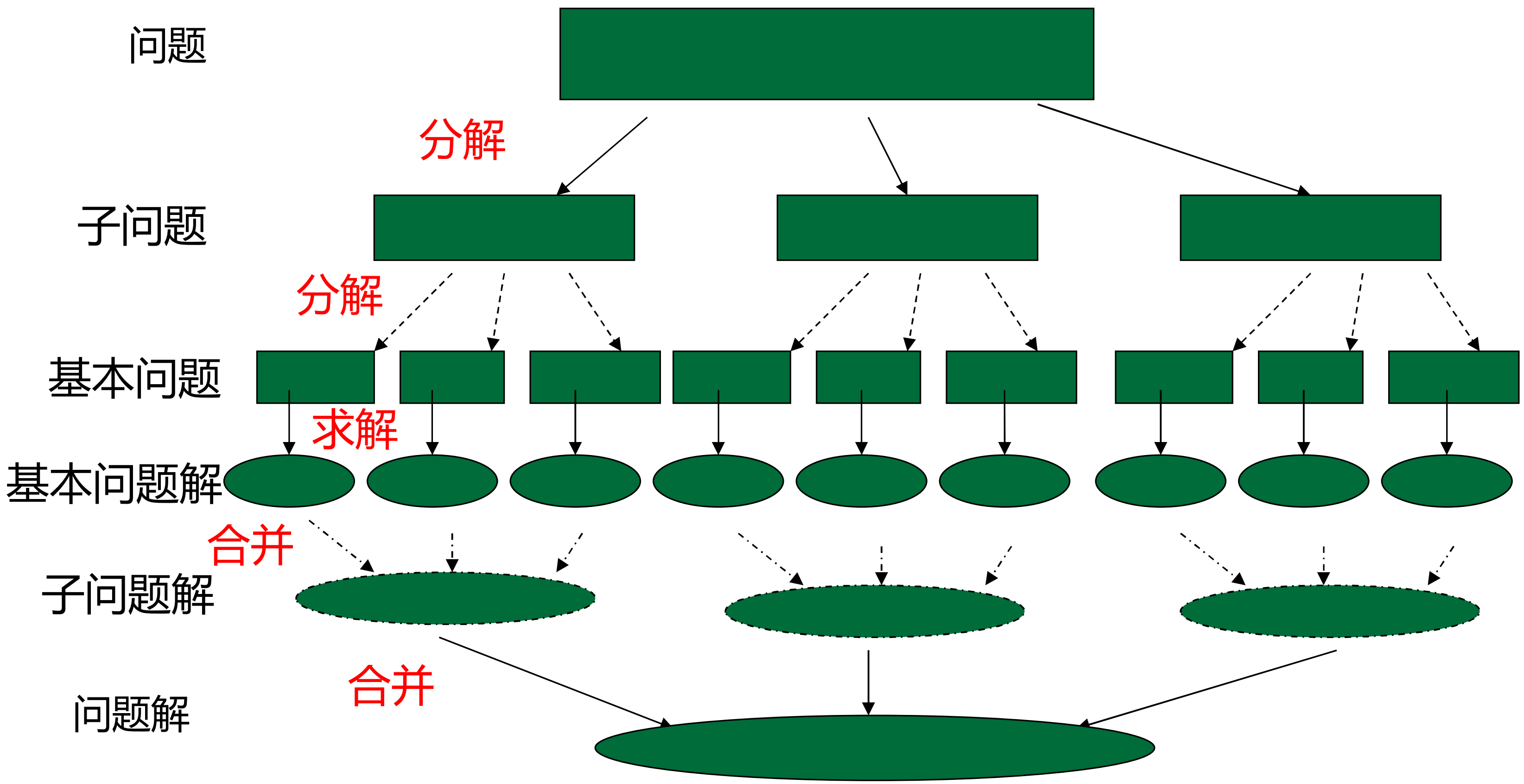
经典分治法求解方法

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 P1, P2,..., Pa;      //分解问题
        for (i=1,i<=a,i++)
          yi=divide-and-conquer(Pi); //递归的解各子问题
        return merge(y1,...,ya);       //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(1/*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, 时间f(n)

设分解出的子问题有*a* 个, 时间复杂度T(n):

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

```
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≥1, b≥2, 分治中经常出现

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

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≥1, b≥2

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

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≥1, b≥2

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

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.

分治主定理-推广

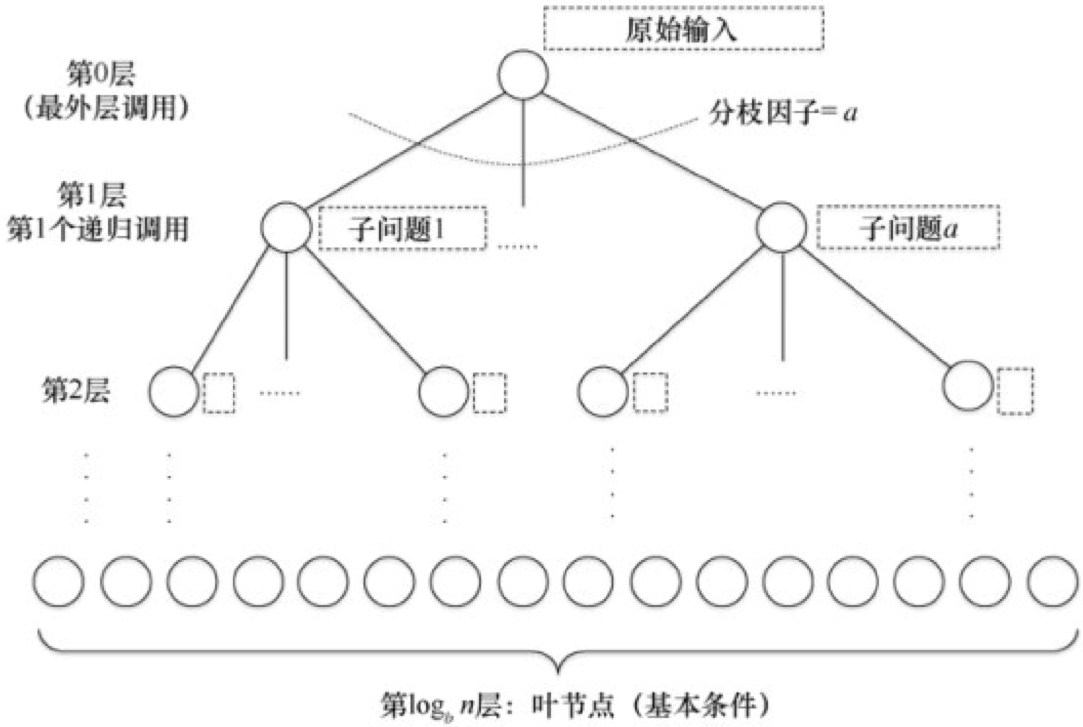
$$T(n)=\begin{cases} b & n=1 \\ T(\lfloor c_1 n \rfloor)+T(\lfloor c_2 n \rfloor)+bn & n>1 \end{cases}$$

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

特别地，当$c_1+c_2<1$时，有

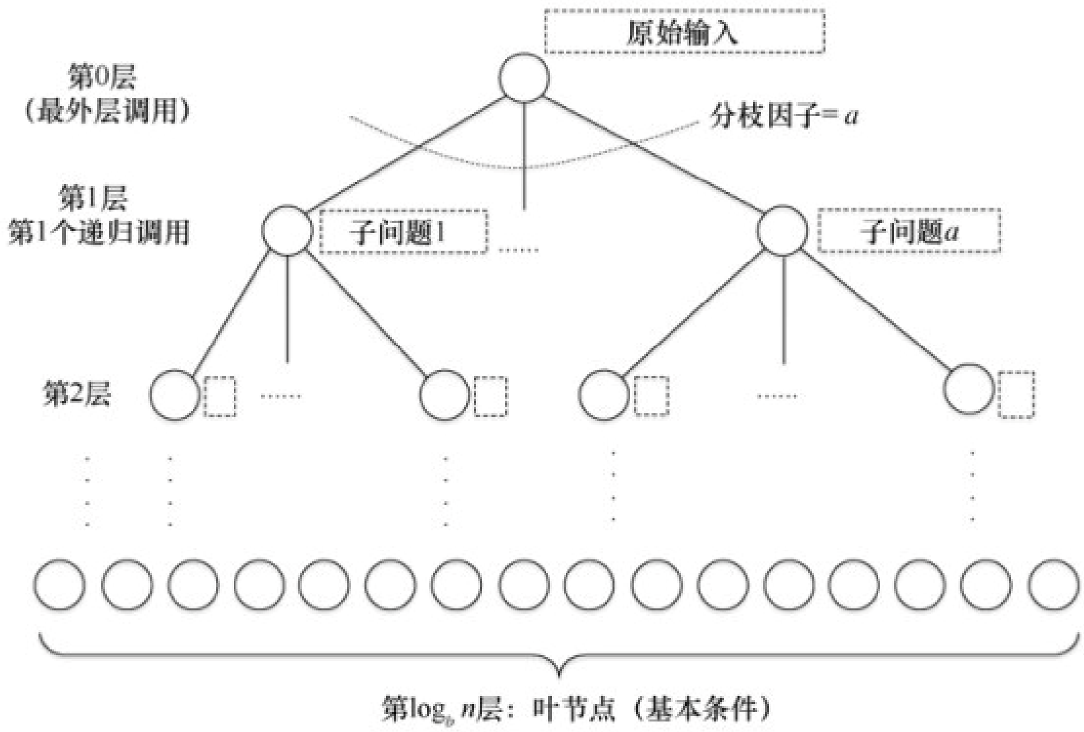
$$T(n)\le bn/(1-c_1-c_2)=O(n)$$

分治法的分析



递归树对应于一个标准的递归过程。节点对应于递归调用。第 0 层对应于最外层调用，
第 1 层对应于它制造的递归调用，接下来以此类推

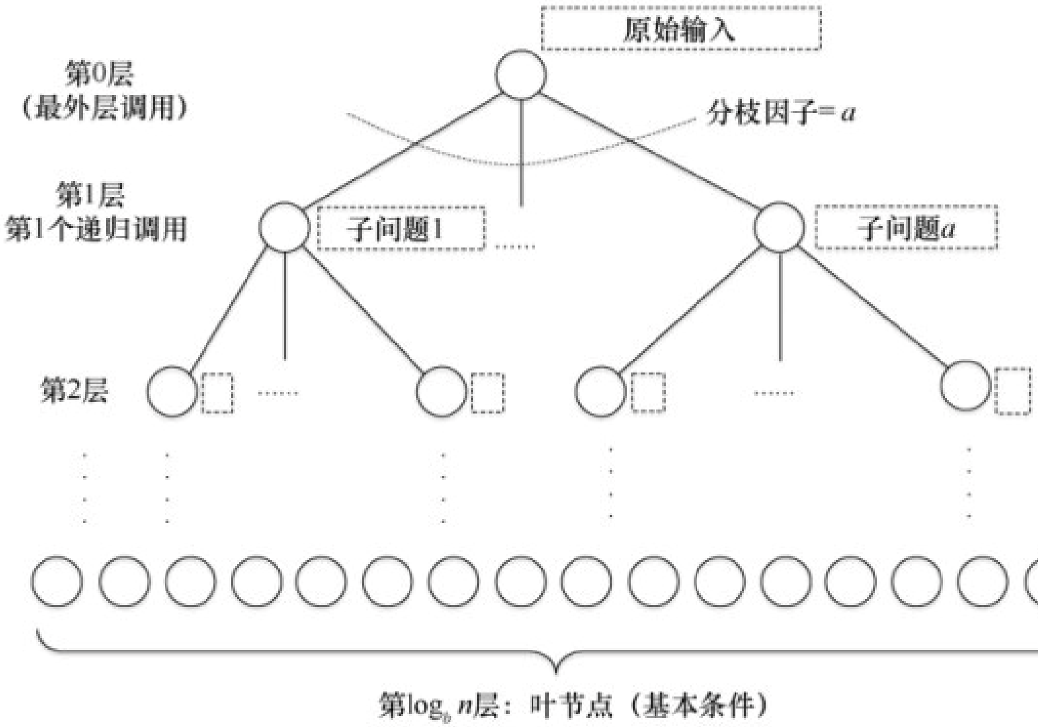
分治法的分析



递归树对应于一个标准的递归过程。节点对应于递归调用。第 0 层对应于最外层调用，第 1 层对应于它制造的递归调用，接下来以此类推

第 j 层一共有 a^j个不同的子问题， 每个子问题的输入长度为 n / b^j。

分治法的分析

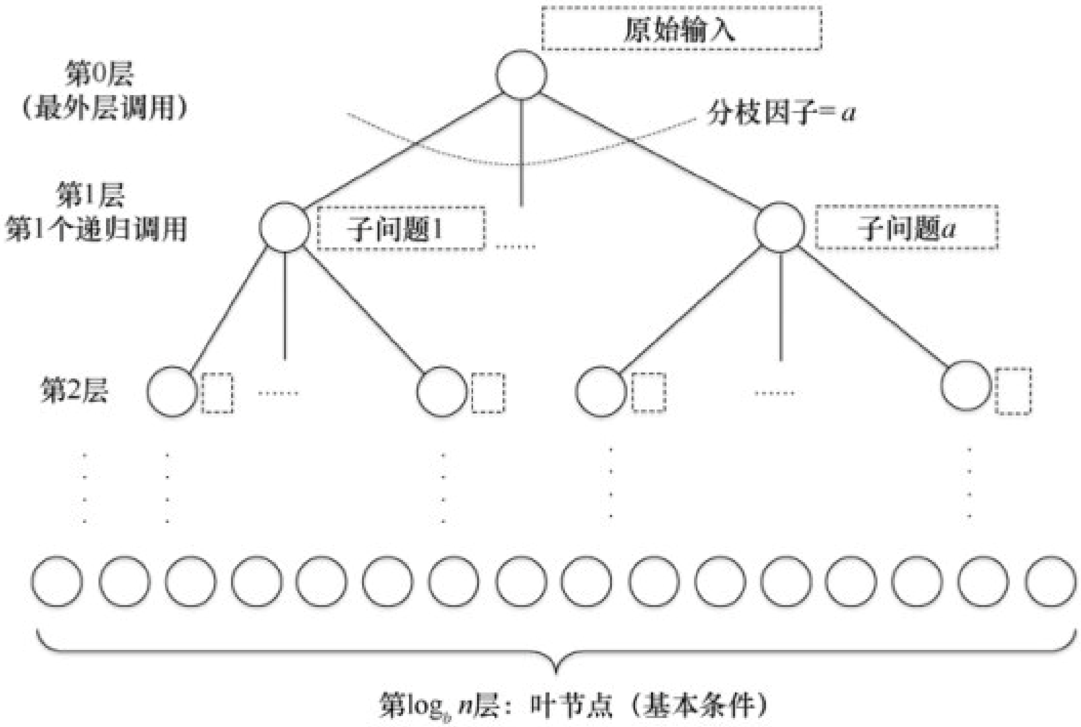


递归树对应于一个标准的递归过程。节点对应于递归调用。第 0 层对应于最外层调用，第 1 层对应于它制造的递归调用，接下来以此类推

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

$$T(n) = aT(n/b)+O(n^d)$$

分治法的分析



递归树对应于一个标准的递归过程。节点对应于递归调用。第 0 层对应于最外层调用，第 1 层对应于它制造的递归调用，接下来以此类推

第1层$aO(n^d)$

第2层$a/b^dO(n^d)$

第3层$(a/b^d)^2O(n^d)$

第i层$(a/b^d)^{i-1}O(n^d)$

分治法的分析

第1层$aO(n^d)$

最后1层的工作量

$$\left(\frac{a}{b^d}\right)^{x-1} O(n^d) \approx \left(\frac{a}{b^d}\right)^{x} O(n^d) = \left(\frac{a}{b^d}\right)^{\log_b n} O(n^d)$$

$$= \frac{a^{\log_b n}}{(b^{\log_b n})^d} O(n^d) = \frac{a^{\log_b n}}{n^d} O(n^d)$$

$$= O(a^{\log_b n})$$

$$= O(n^{\log_b a})$$

分治法的分析

关键因素的是a/b^d 这个关键比率。

a：也就是每个 递归层的子问题增殖率(RSP)，它是子问题数量爆炸性增长的扩张因子。

b^d：也就是工作量的收缩率(RWS)，在递归的每一层，每个子问题的工作量都会根据收缩因子 b^d 进行缩减。

工作量是会随着递归的层数增加而增加？还是随着递归的层数的增加而减少？又或者是每一层完成的工作量是相同的？

现在我们只需要观察每层产生的成本的发展趋势，是递减的还是递增的，还是每层都一样？每层成本的公比为a/b^d

分治法的分析

关键是a/b^d 这个比率。

如果每层的成本是递减的（ a/b^d < 1）

$$\text{第1层}O(n^d)$$

如果每层的成本是递增的（ a/b^d > 1）

$$\text{最后1层}O(n^{\log_b a})$$

如果每层的成本是相等的（ a/b^d == 1

$$T(n)=\begin{cases} O(n^d) & \text{公比}\dfrac{a}{b^d}<1 \\ O(n^{\log_b a}) & \text{公比}\dfrac{a}{b^d}>1 \\ O(n^d\log_b n) & \text{公比}\dfrac{a}{b^d}=1 \end{cases}$$

$$O(n^{\log_b a})*\log_b n = O(n^{\log_b a} * \log_b n)$$

二分法

输入: 实数序列$a_1,\ldots,a_n$, 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($a_1<\ldots<a_n$)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)=lnx+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。

```
void maxmin(int A[],int &e_max,int &e_min,int low,int high) {
    int mid,x1,y1,x2,y2;
    if ((high-low <= 1)) {
        if (A[high]>A[low]) {
            e_max = A[high];
            e_min = A[low];
        }  else {
            e_max = A[low];
            e_min = A[high];
        }
    } else {
        mid = (low + high) / 2;
        maxmin(A,x1,y1,low,mid);
        maxmin(A,x2,y2,mid+1,high);
        e_max = max(x1,x2);
        e_min = min(y1,y2);
    }
}
```

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

迭代法

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

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

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

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

$n=2^m$

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

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

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

$$=3n/2-2$$

课堂练习

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

答案

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

$n=2^k$

$$=3T(2^{k-1})$$

$$=3^2T(2^{k-2})$$

$$= \dots$$

$$=3^kT(2^{k-k})$$

$$= 3^kT(1)$$

$$= 3^k$$

$k=\log_2 n$

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。

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

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Contents

本章内容

递归函数

分治原理, 主定理， 二分法

经典分治法求解方法

大整数乘法

- 输入: 两个n位二进制数X,Y
- 输出: X×Y
- 输入规模: n

方案一: 直接计算

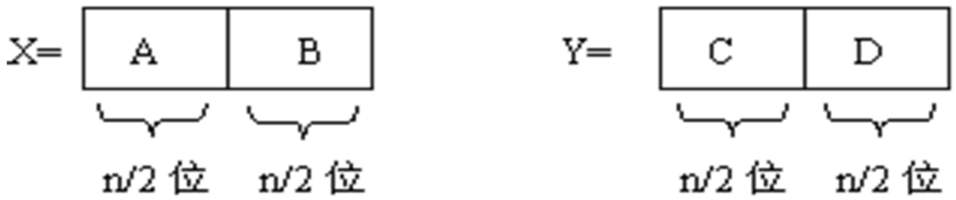
时间复杂度$O(n^2)$

方案二: 尝试分治法

$$
\begin{array}{cccccccc}
 & & & & 1 & 0 & 1 & 1 \\
 & & & \times & 1 & 1 & 0 & 1 \\
\cline{4-8}
 & & & & 1 & 0 & 1 & 1 \\
 & & 1 & 0 & 1 & 1 & & \\
+ & 1 & 0 & 1 & 1 & & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

大整数乘法: 分治

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



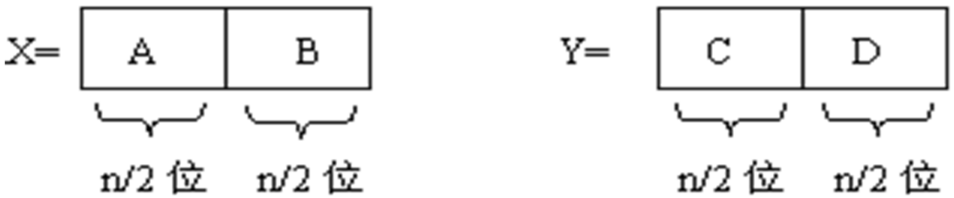
$$XY=(A2^{n/2}+B)(C2^{n/2}+D)=AC2^{n}+(AD+BC)\,2^{n/2}+BD$$

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 P1, P2,..., Pa;
       for (i=1,i<=a,i++)
          yi=divide-and-conquer(Pi);
       return merge(y1,...,ya);
} }
```

大整数乘法: 分治

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



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

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)

大整数乘法的研究历史

http://en.wikipedia.org/wiki/Karatsuba_algorithm,
http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations,

- 1952年, A. Kolmogorov 猜$\Theta(n^2)$
- 1960年, Kolmogorov在自己组织的讨论班上提到这个猜测
- 几天后, 23岁的学生Karatsuba给出$O(n^{\log_2 3})$算法
- 1971, Schönhage和Strassen, 快速傅里叶变换([C],分治),
  $O(n \log n \log\log n)$, 猜测$O(n \log n)$
- 2007, Fürer, $O(n \log n \; 2^{O(\log^* n)})$

矩阵乘法

- 1969, Strassen算法, $O(n^3)$改进为$O(n^{\log_2 7})$
- 2010, CW算法, $O(n^{2.376})$
- 2014, 优化的CW-like算法, $O(n^{2.373})$.

排序问题(sorting)

输入: 一个实数序列 $a_1,\ldots,a_n$.

输出: 序列 $a_1,\ldots,a_n$的一个升序排列.

输入样例: 3,2,5,7,2,9,1,8

输出样例: 1,2,2,3,5,7,8,9

- 冒泡排序, 插入排序: $\Theta(n^2)$
- 堆排序 $\Theta(n\log n)$
- 合并(归并)排序: $\Theta(n\log n)$ 分少, 合多; 额外空间和复制.
- 快速排序: 最坏 $\Theta(n^2)$ 平均 $\Theta(n\log n)$, 分多合少
- 通过比较进行排序的算法为 $\Omega(n\log n)$

堆排序Williams1964, 建堆Floyd1964, 快排Hoare1962

快速排序

QuickSort(a, p, r) //当p<r时将a[p:r]段排序, 无需合并

1. 分解: Partition(a,p,r), 即以a[p](=b)为基准

通过交换将a[p:r]分成3段

a[p:middle-1] ≤ a[middle](=b) ≤ a[middle+1:r]

2. 递归求解: QuickSort(a, p, middle-1);

QuickSort(a, middle+1, r)

执行一次Partition(O(n)时间)举例:

6 2 8 5 10 9 12 1 15 7 3 13 4 11 16 14

6 2 4 5 3 1 12 9 15 7 10 13 8 11 16 14

1 2 4 5 3 6 12 9 15 7 10 13 8 11 16 14

程序主部


```
QuickSort(a, p, r)                          //排序a[p:r]
{     middle=Partition(a,p,r);
      QuickSort(a, p, middle-1);   //排序a[p:middle-1]
      QuickSort(a, middle+1, r);  //排序a[middle+1,r]
}
```

特点: 分解和分治时间多, 合并不需要时间

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

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

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

最好情况分析: 每次分成大小为n/2的两段

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

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

最小值问题

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;
}
```

最小值问题

最小值问题

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

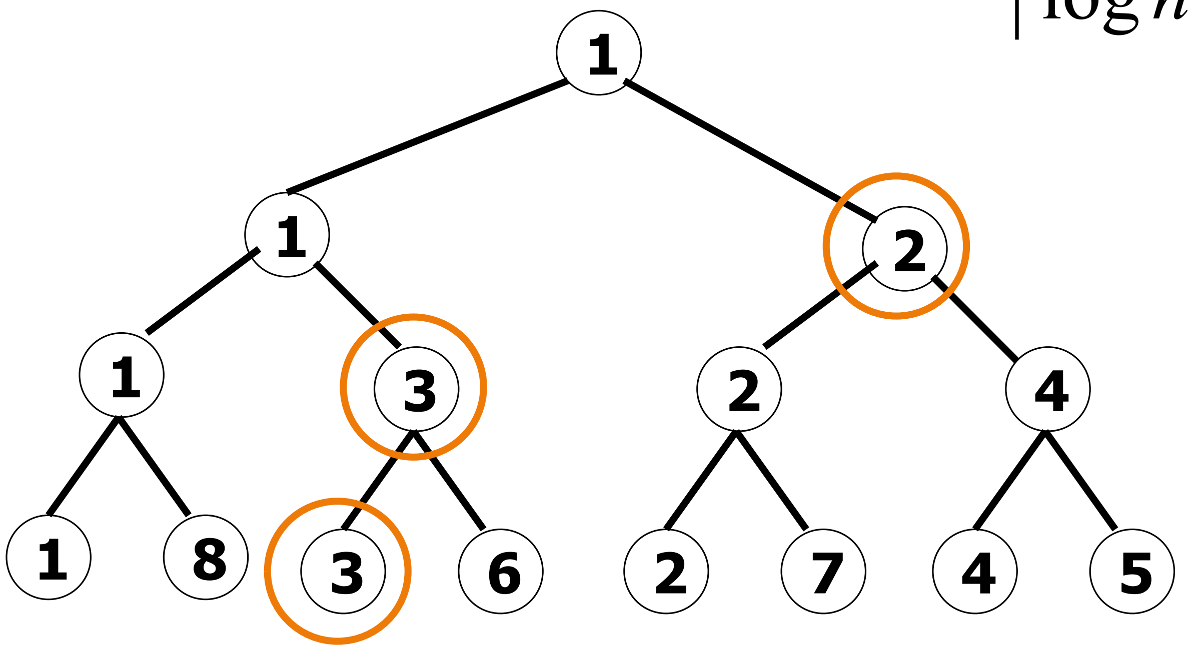
确定最小元素至少需要n-1次比较，n-1次比较是最小问题的下界

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

线性时间选择

找第k小数问题

- 输入: 一个实数序列 $a_1,\dots,a_n$, 和一个整数k.
- 输出: 序列中第k小的数.

方法一: 先排序, 再找第k小的数. O($n$log$n$)时间.

分析: 若k=1, 则直接找最小, O(n)时间.

若k<n/logn, 先建最小堆, O(n)时间([M]p100).

再弹出k个元素, O(k log$n$)

方法二: 使用快速排序方法, 最多对一段继续分解

最坏时间O(n^2), 平均时间O(n) ([C])

方法三: 改进, 最坏O(n)时间算法([王,C])

由快速排序改成的随机选择算法

```
QuickSort(a, p, r)                        //排序a[p:r]
{     mid=RamdomizePartition(a,p,r);
      QuickSort(a, p, mid-1);        //排序a[p:mid-1]
      QuickSort(a, mid+1, r);        //排序a[mid+1,r]
}
```

执行一次Partition举例:

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14

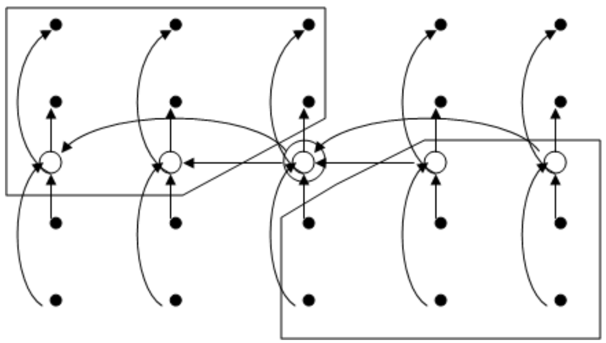
```
RSelect(a,p,r,k)      //选择a[p:r]中第k小数
{     mid=RamdomizePartition(a,p,r);
      if( mid >= k)return(RSelect(a, p, mid,k));
      else return(RSelect(a, mid+1, r, k-mid);
}   //初略时间分析: T(n) = T(9n/10) + O(n) = O(n)
```

改进选择算法

随机选择: 随机选基准, 划分, 继续随机选择

通过修改基准, 设计新的选择算法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$\lceil n/5 \rceil$个中位数的中位数
3. 以这个数为基准划分
4. 选一个部分继续执行Select

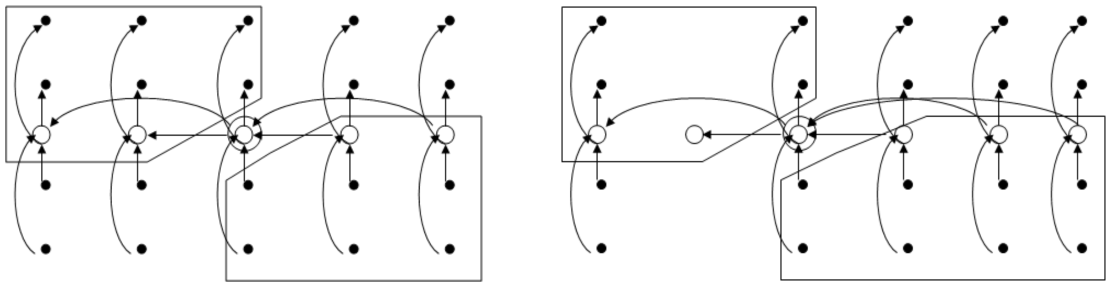


- 假设所有数互不相同
- 当n充分大, 至少有1/4的数<新基准, 1/4的数>新基准?

$$T(n)=\begin{cases} O(1) & n<n_0 \\ T(n/5)+T(3n/4)+O(n) & n\ge n_0 \end{cases} = O(n)$$

分治起始点

- 划分成⌈**n/5**⌉组, 取各组中位数的中位数做基准.



[王] $3\lfloor (n-5)/10 \rfloor \ge n/4$　　　　需要 $n \ge 75$.

明确比基准小的有$1/2*\lfloor n/5 - 1 \rfloor$组，因为在每一组中有2个元素小于本组的中位数，所以有$2*1/2*\lfloor n/5 - 1 \rfloor$个元素小于基准；

中位数那一行中又有$1/2*\lfloor n/5 - 1 \rfloor$个小于基准；

因此至少有$3\lfloor (n-5)/10 \rfloor$个元素小于基准。而当$n \ge 75$时$3\lfloor (n-5)/10 \rfloor \ge n/4$，所以按此基准划分所得的2个子数组的长度都至少缩短1/4。

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

```
template <class Type>
Type Select(Type a[], int p, int r, int k)
{   if( r - p < 75 ) { 直接对数组a[p:r]排序; return a[p+k-1];;}
    for( int i = 0; i <= (r - p - 4) / 5 ; i++ ) //分⌈n/5⌉组, 取各组中位数
        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;
    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)
    int i = Partition(a,p,r,x), j = i - p +1;
    if ( k == j ) return a[i];
    elseif ( k < j ) return Select(a,p,i-1,k);     //选择左片递归, 最多T(3n/4)
    else return Select(a,i+1,r,k-j);               //选择右片递归, 最多T(3n/4)
}
```

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

线性时间选择实现方式

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⌈n/5⌉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	17	17
60	31	8
33	33	33
17	60	60

31	4
17	51
8	57
33	49
60	35

31	4
17	35
8	49
33	51
60	57

31	4
49	35
8	17
33	51
60	57

线性时间选择实现方式

29个元素第18小：

for(int i = 0; i <= (r - p - 4) / 5 ; i++) //分⌈n/5⌉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

线性时间选择实现方式

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

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

线性时间选择实现方式

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

以x = 25为轴，快排分区

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}

i = 13

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

线性时间选择实现方式

29个元素第18小：

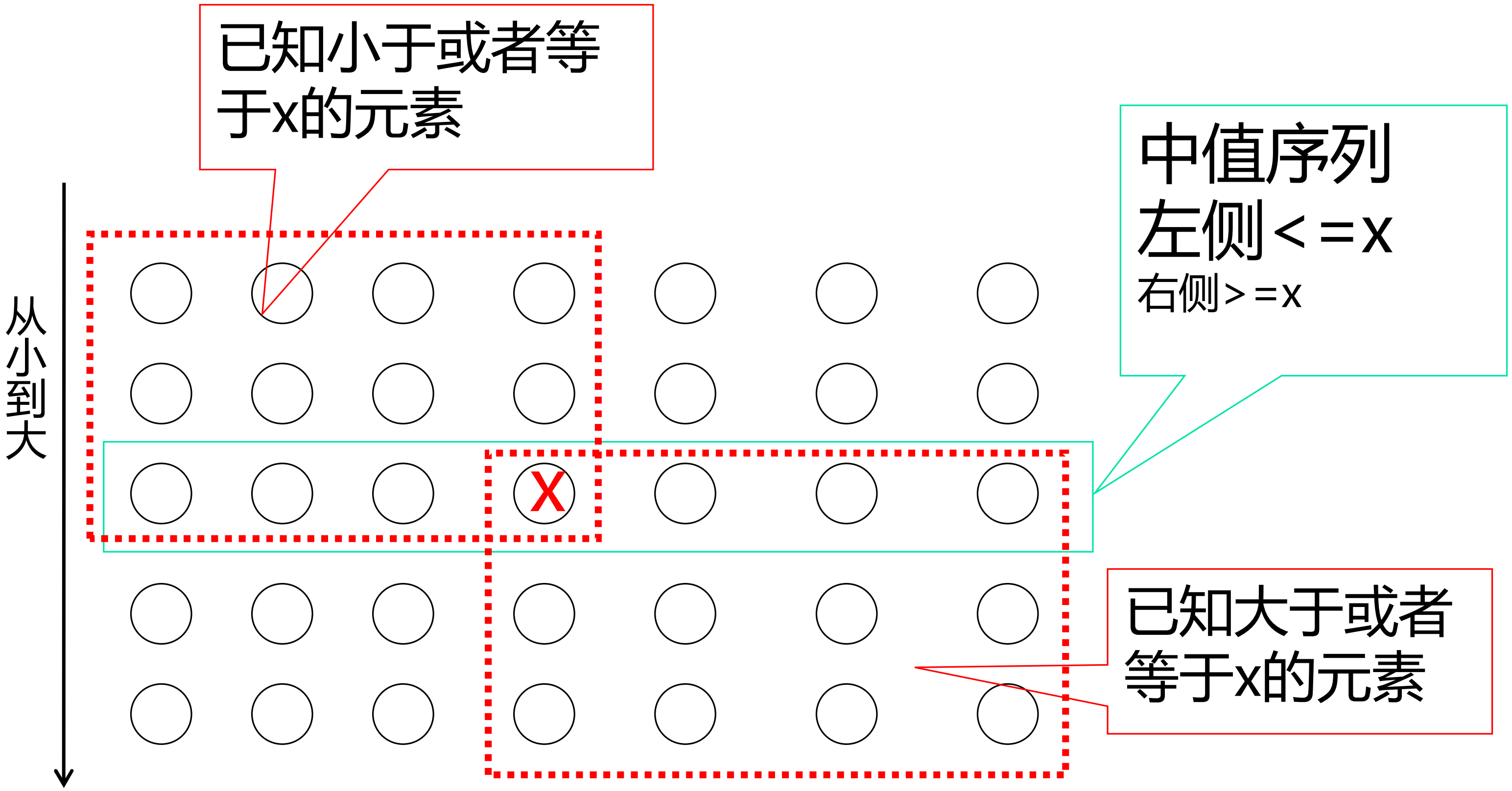
x = 25　i = 13

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

```
j = i - p +1;
 if ( k == j ) return a[i];
 else if ( k < j ) return Select(a,p,i-1,k);
 else return Select(a,i+1,r,k-j);
```

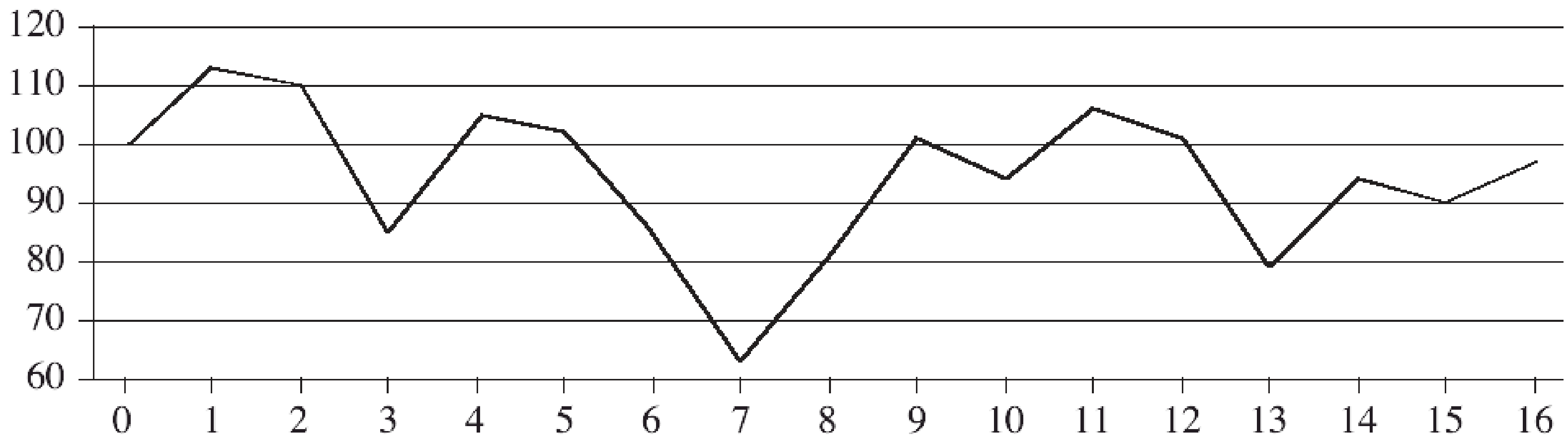
{43,37,57,33,32,52,51,35,60,41,54,49,31,46,29}

线性时间选择程序



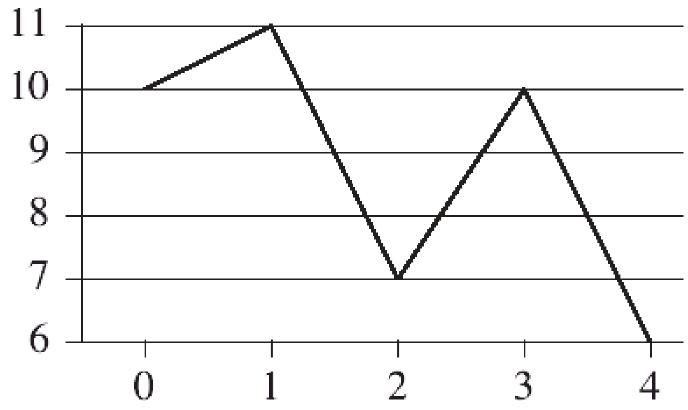
$$T(n)=\begin{cases} O(1) & n<75 \\ T(n/5)+T(3n/4)+O(n) & n\geq 75 \end{cases} = O(n)$$

最大子段和



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

最大子段和



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$a_1,a_2,\ldots,a_n$，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

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

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,...,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：

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

在a[n/2..n]中计算：

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

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
```

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

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

最接近点对问题

- 输入: 平面上点集 P = { $p_1, p_2, \ldots, p_n$ }
- 输出: (s, t) 使得

  d(p_s, p_t) = min { d(u,v) | $u \neq v \in$ 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 = { $p_1, p_2,\ldots, p_n$}
- 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;

O(n^2)

2. 对 i = 1 : n-1
3. 对 j = i+1 : n

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:

1. 排序: $p_{i_1} \le p_{i_2} \le \ldots \le p_{i_n}$. O(n logn)
2. 初始化: min = $d(p_{i_1}, p_{i_2})$; $s = i_1$; $t = i_2$;
3. 对 $k = 2 : n-1$
4. 若 min > $d(p_{i_k}, p_{i_{k+1}})$
5. 则 $s = i_k$; $t = i_{k+1}$; min = $d(p_{i_k}, p_{i_{k+1}})$

O(n)

O(1)

- 总时间 O(nlogn)
- 不能推广到二维

最近点对--一维分治

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

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. δ = min { d_L, d_R, q-p }

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

德以明理 学以精工

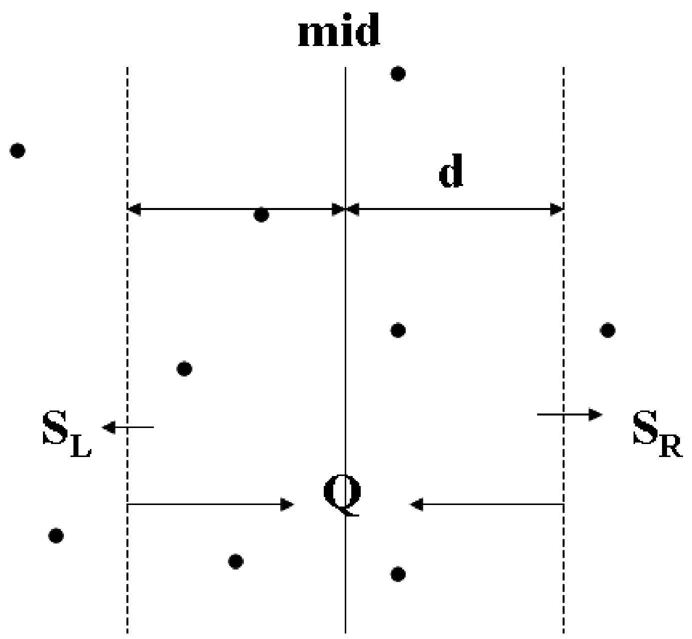
最近点对—二维分治尝试

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 $Q = \{ p \in S \mid |x(p) - mid| < d \}$
5. 逐对求Q中最近点对的距离d.

分O(n), 治2T(n/2), 合$O(n^2)$

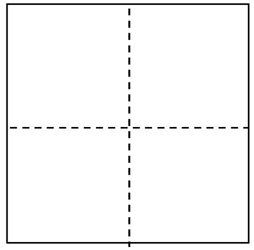
根据分治主定理 $T(n) = O(n^2)$



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

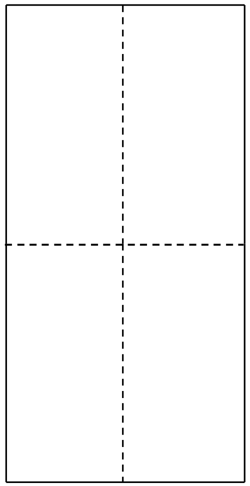
鸽巢(抽屉)原理的简单应用

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



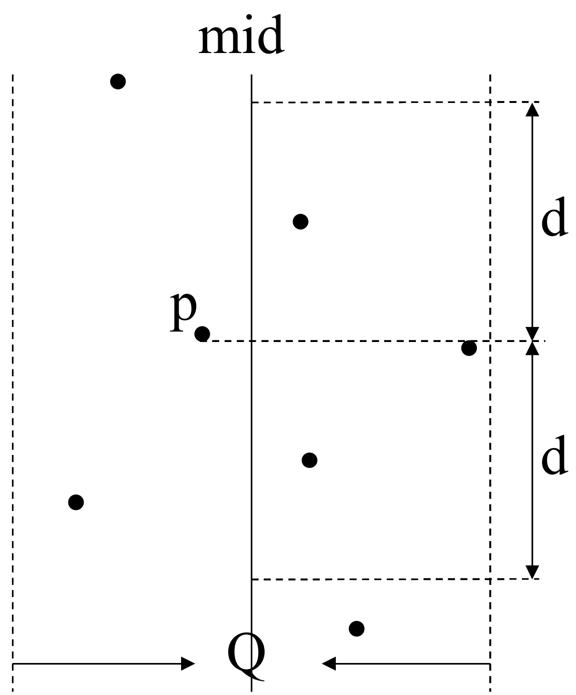
任取一个2d×3d矩形内点集A,
若A中任意两点距离都 ≥ d, 则A中点数 ≤ 6.

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



方案一: Q左右分开

Q右侧中与p距离 $< d$ 的点数 ≤ 6



德以明理　学以精工

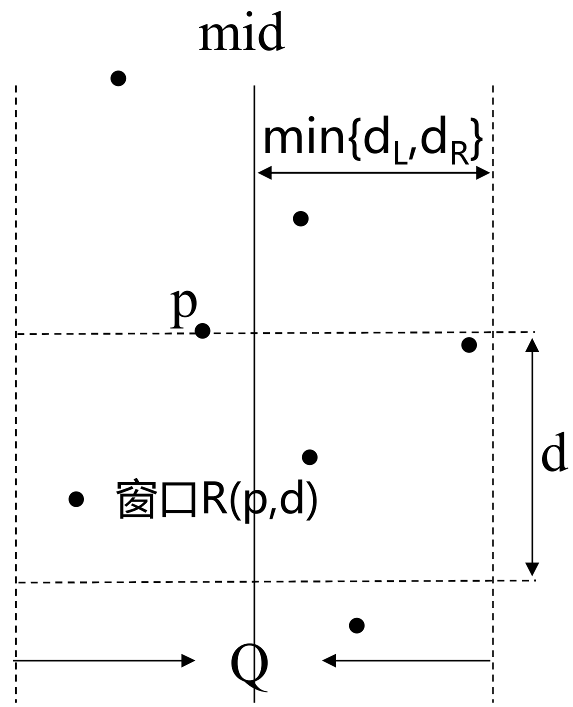
方案二: 检查p下方的点

- 定义窗口

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

- Q中p下方与p距离 $\le d$ 的点一定在R(p,d)中

  而且点数 $\le 7 = 4 + 3$



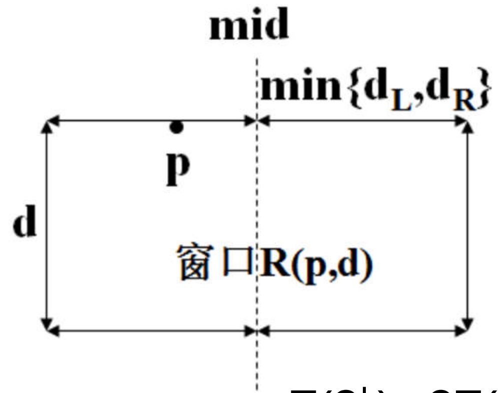
最近点对--合并时间改进一

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. $Q = \{ p \in S \mid |x(p) - mid| < d \}$ 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(pi,pj)<d, 更新d; j=j+1}

步4: O(nlogn), 78步循环至多7次, 步5-8循环至多n次

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

T(n) = O(n log²n), 进一步改进?



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

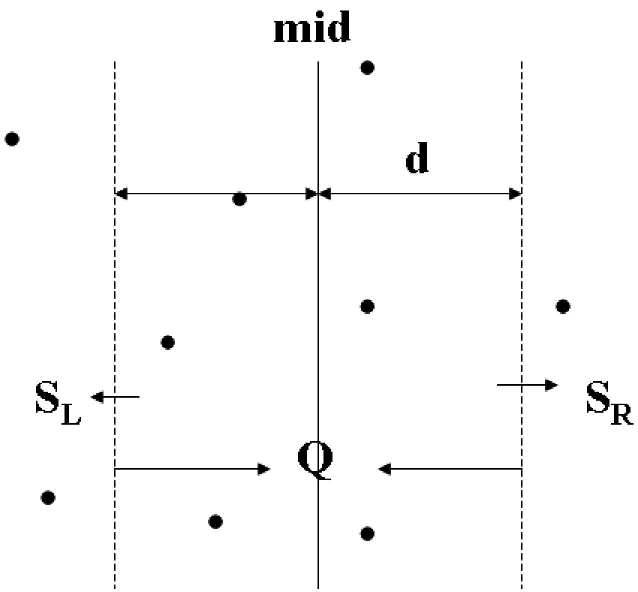
称5--8过程为:
对Q中每个点p,
检查窗口R(p,d),
更新最短距离d

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



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

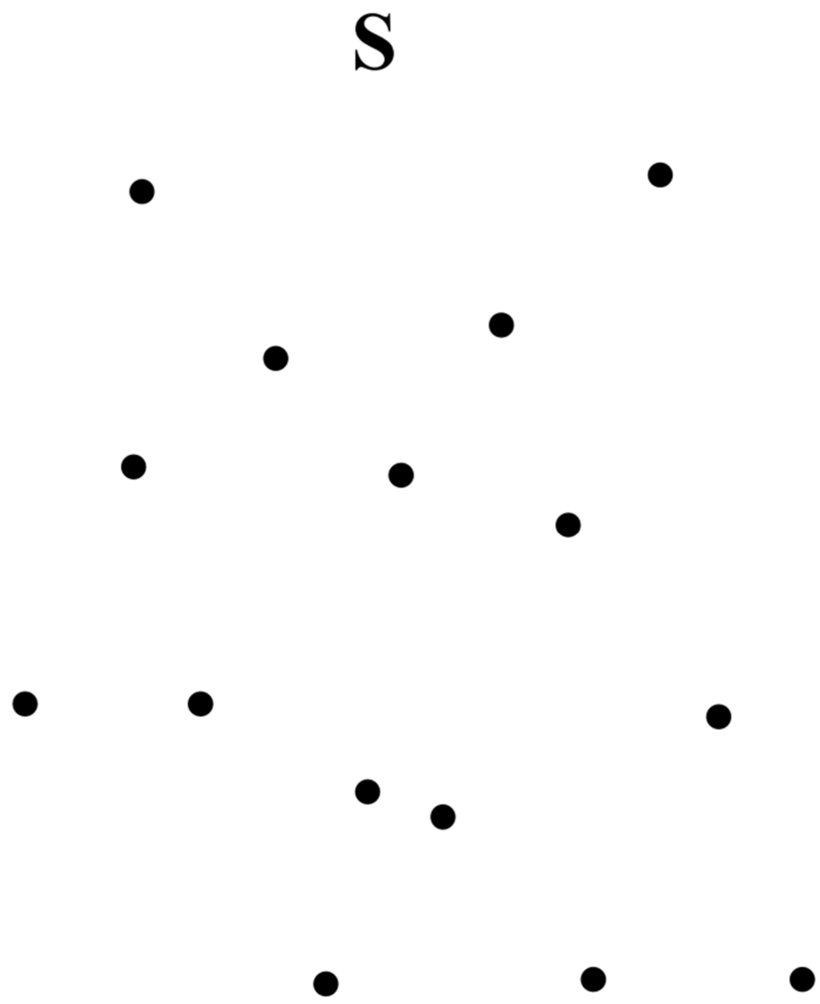
O(nlogn)

算法图示--初始

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

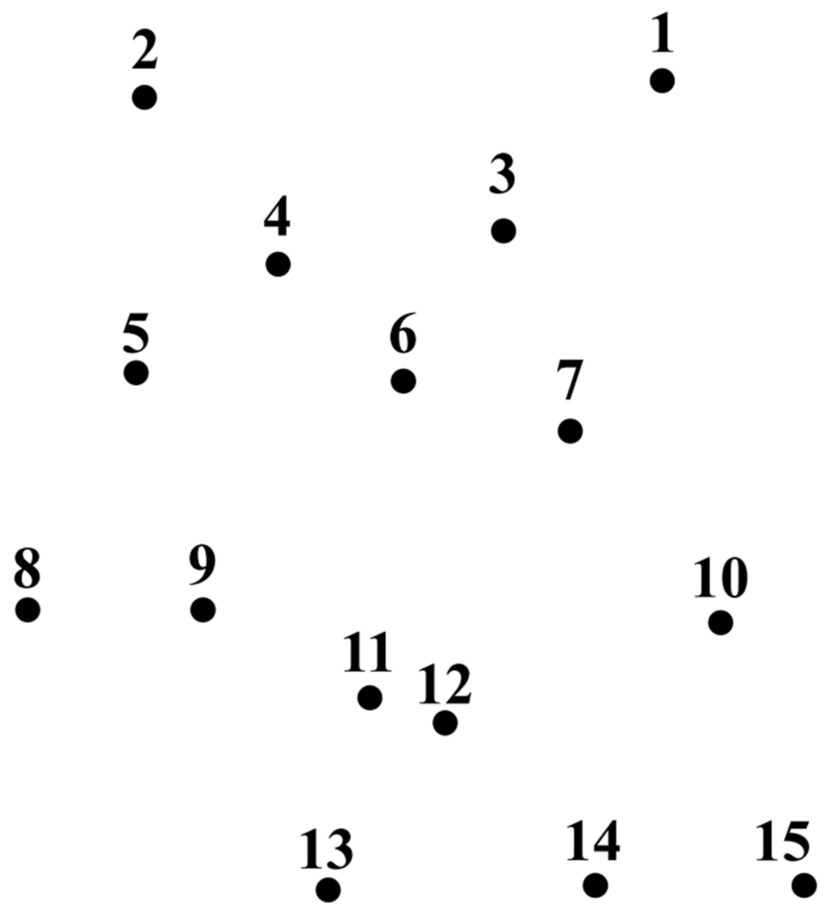


算法图示--预处理

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

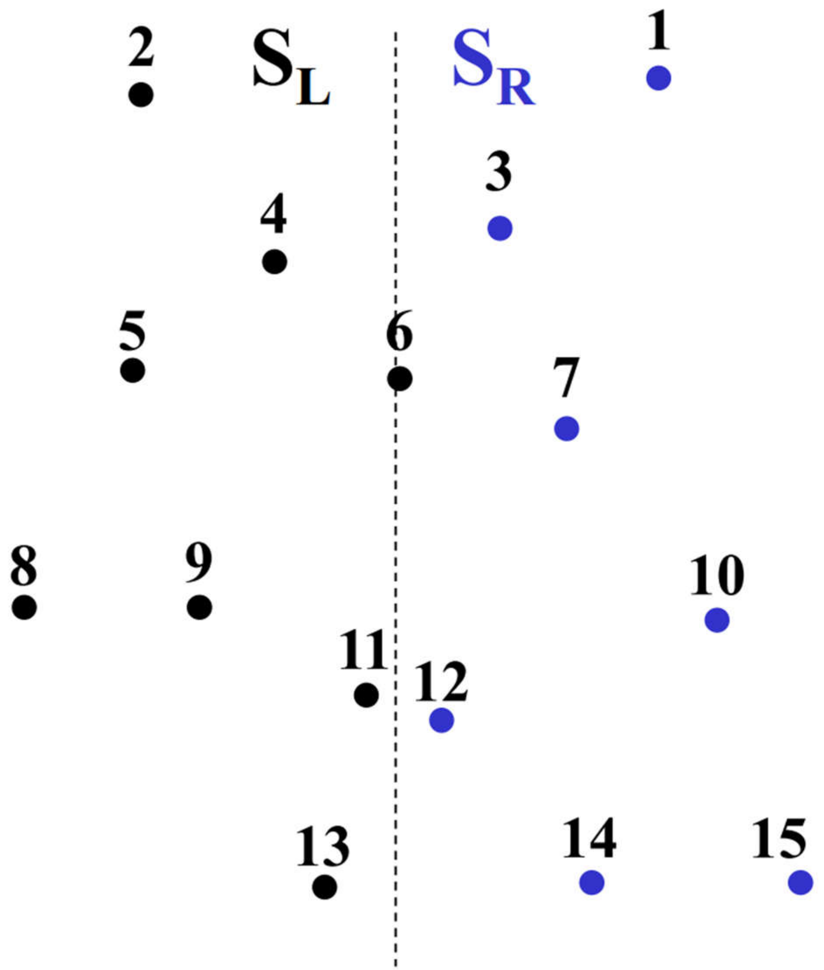


算法图示--分

设有平面点集S

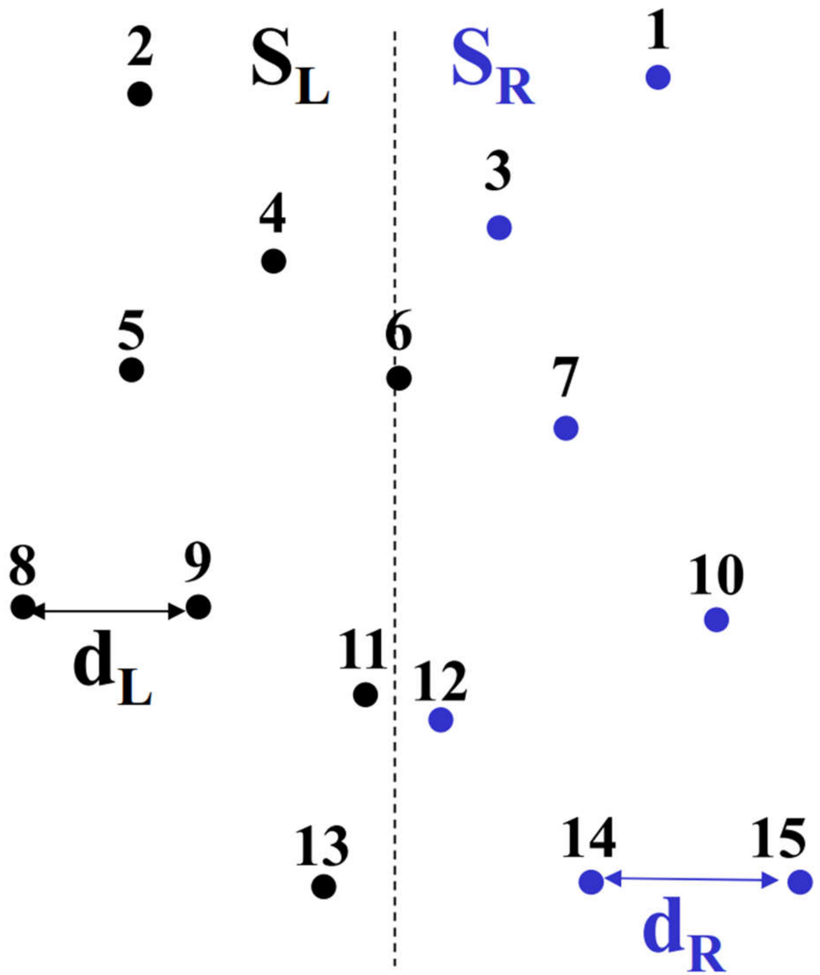
按y坐标递减(预处理)

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

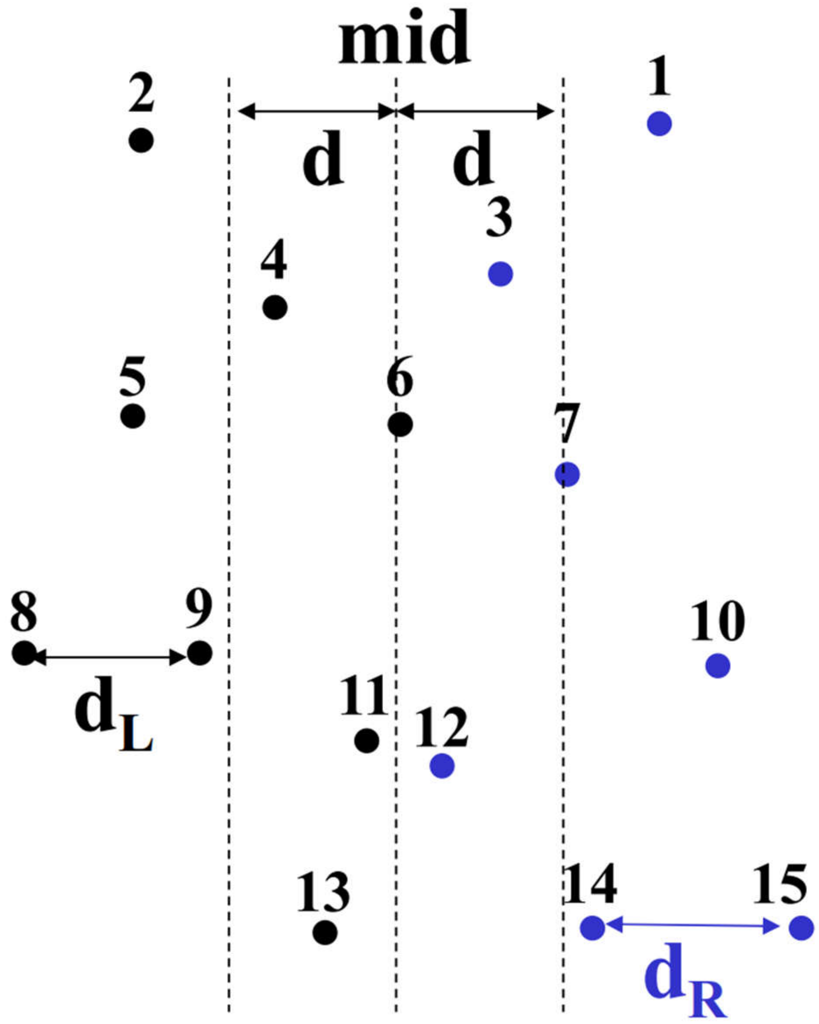


算法图示--合3

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

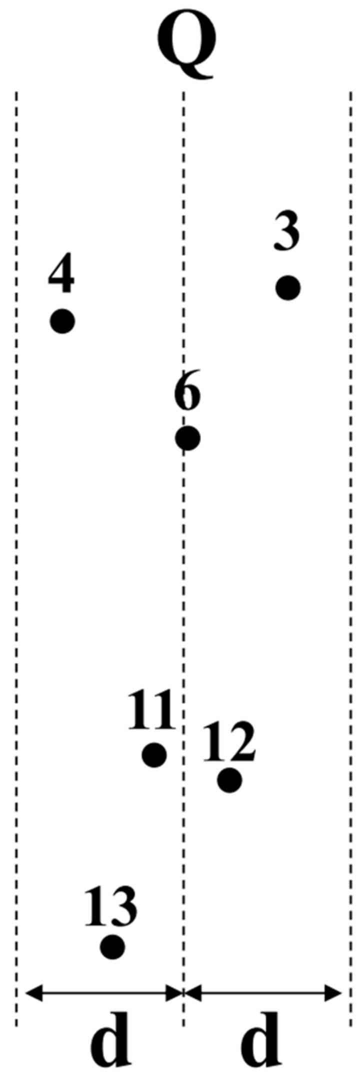


算法图示--合4

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

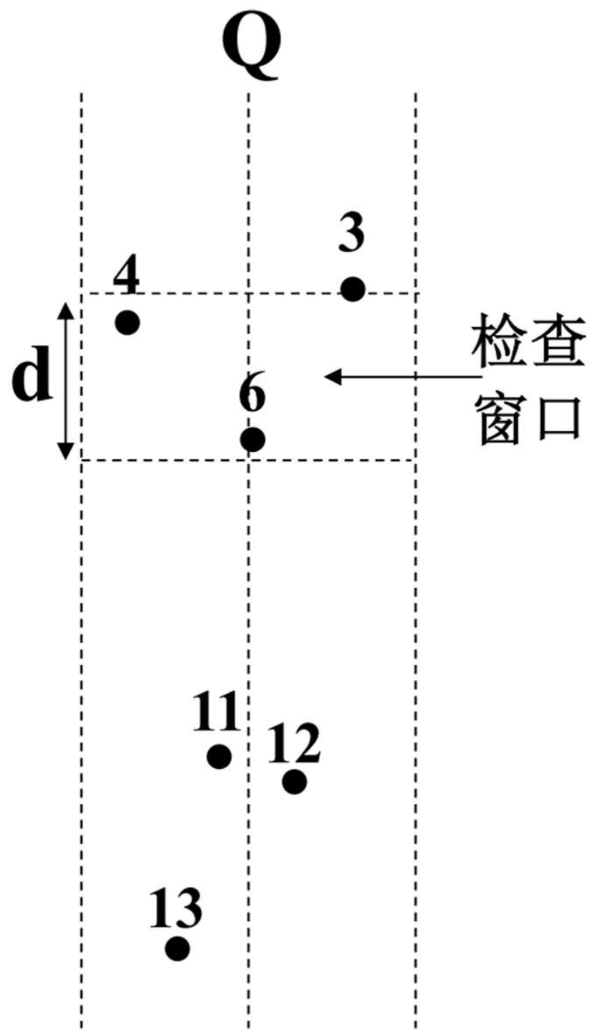


算法图示--合67:p_3

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

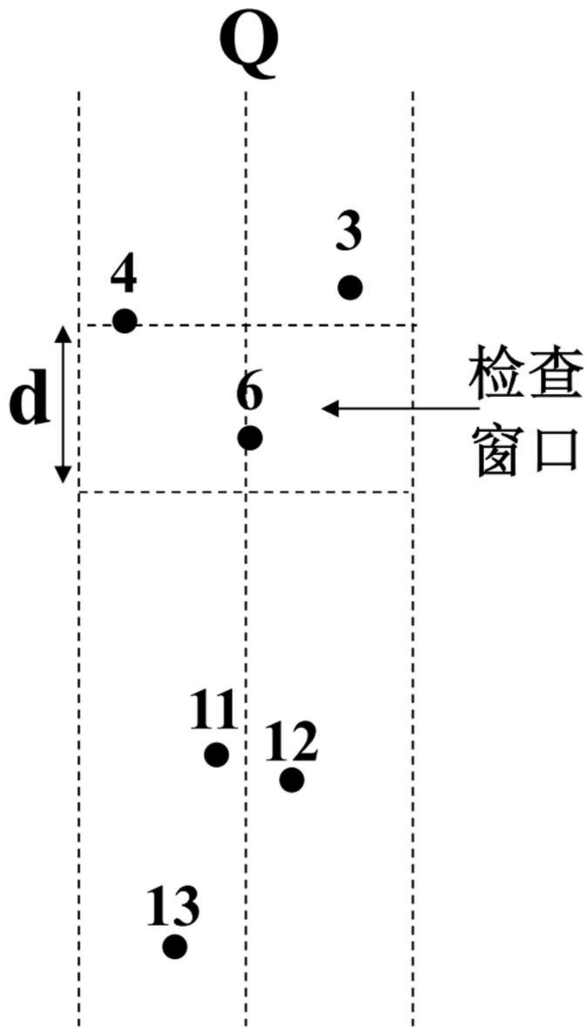


算法图示--合67:p_4

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

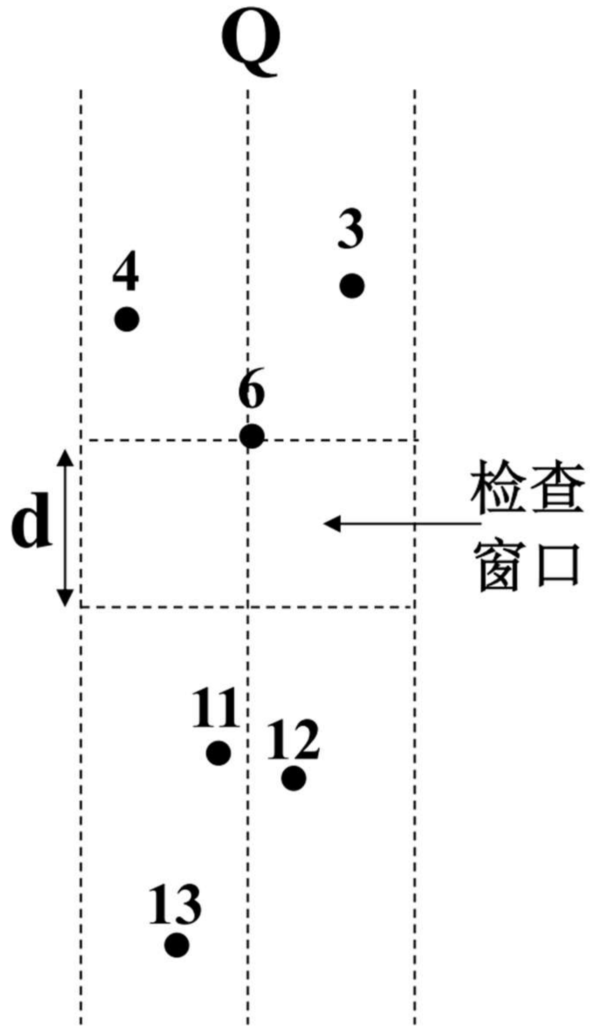


算法图示--合67:p_6

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

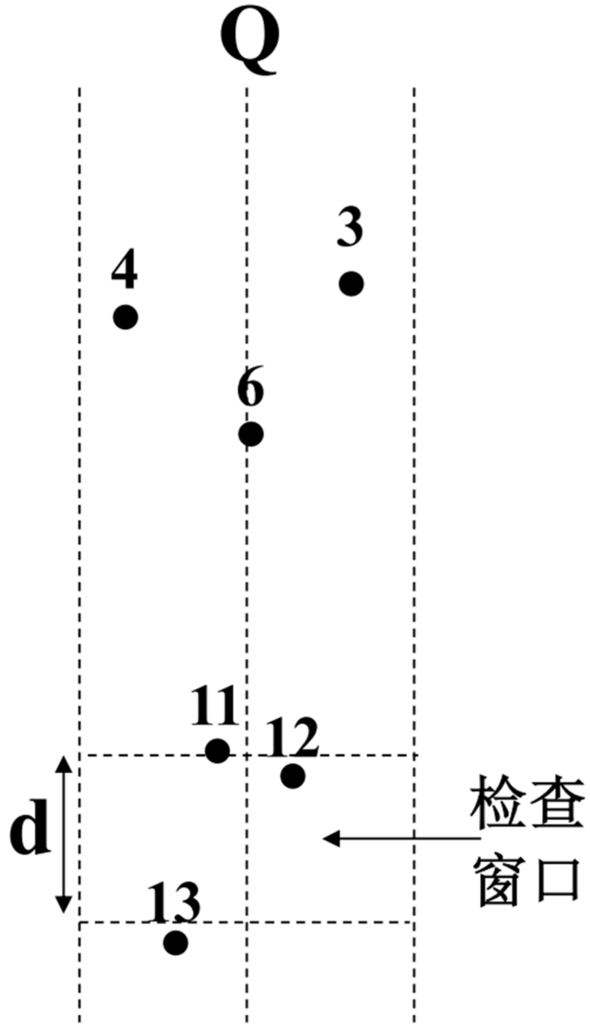


算法图示--合6:p_{11}

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

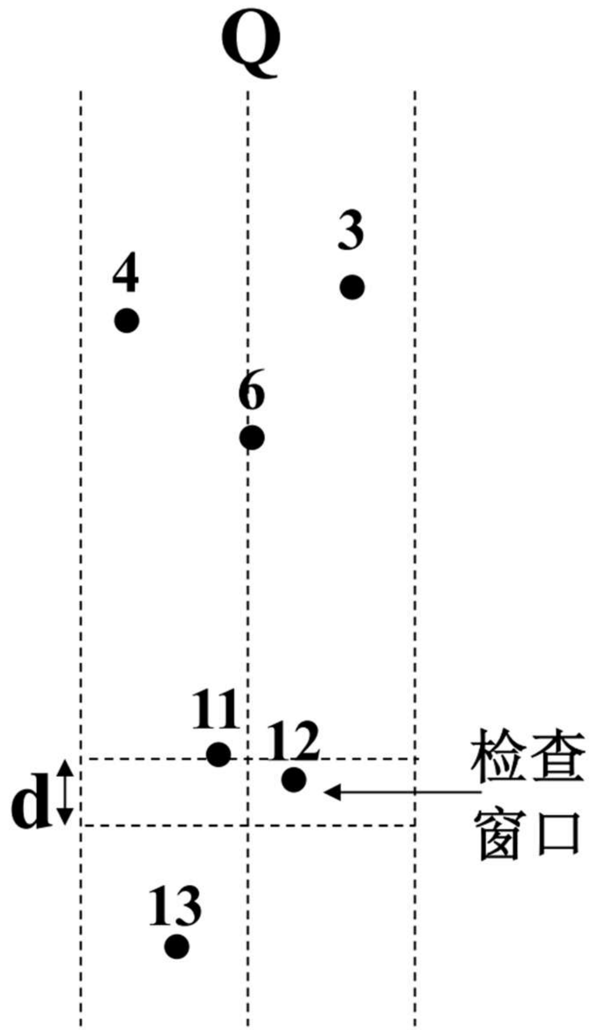


算法图示--合7:p_{11}

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

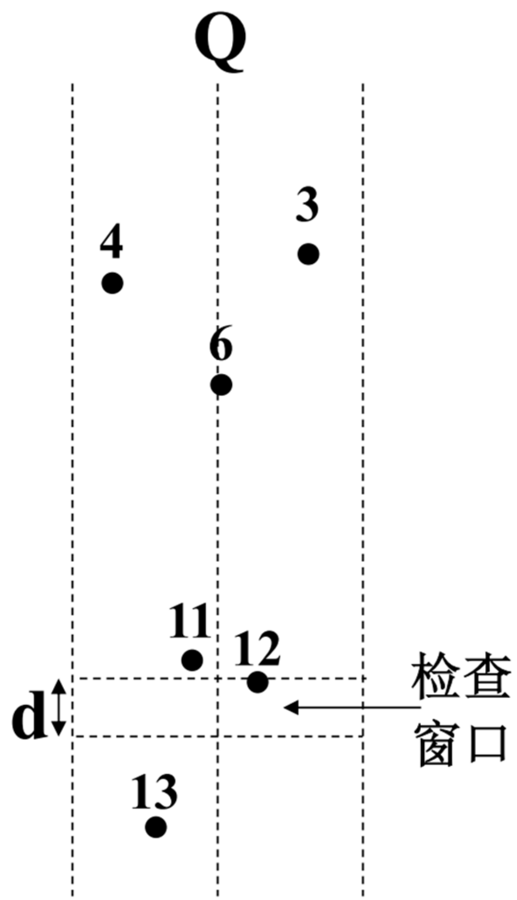


算法图示--合67:p_{12}

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

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

```
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<u>（1）</u>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

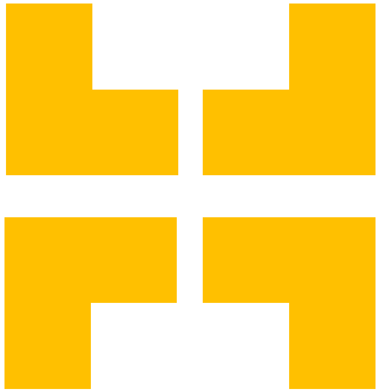
X轴上有n个点，由左至右依次排列为

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：

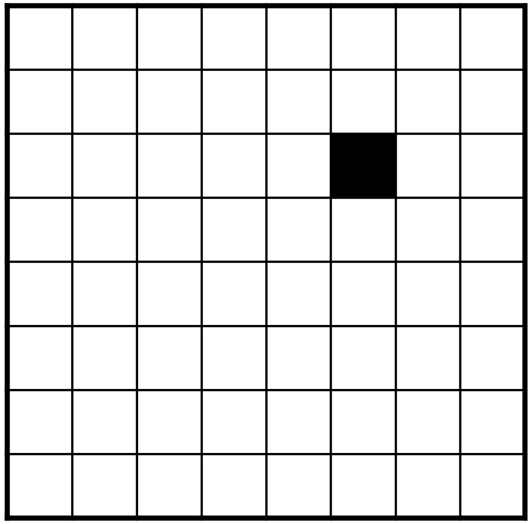
$$x_p=\begin{cases} x_{(n+1)/2} & \text{当}\,n\,\text{为奇数时} \\ \text{中间两点的闭区间上} & \text{当}\,n\,\text{为偶数时} \end{cases}$$

附录: 棋盘覆盖

L型骨牌



$2^k \times 2^k$棋盘

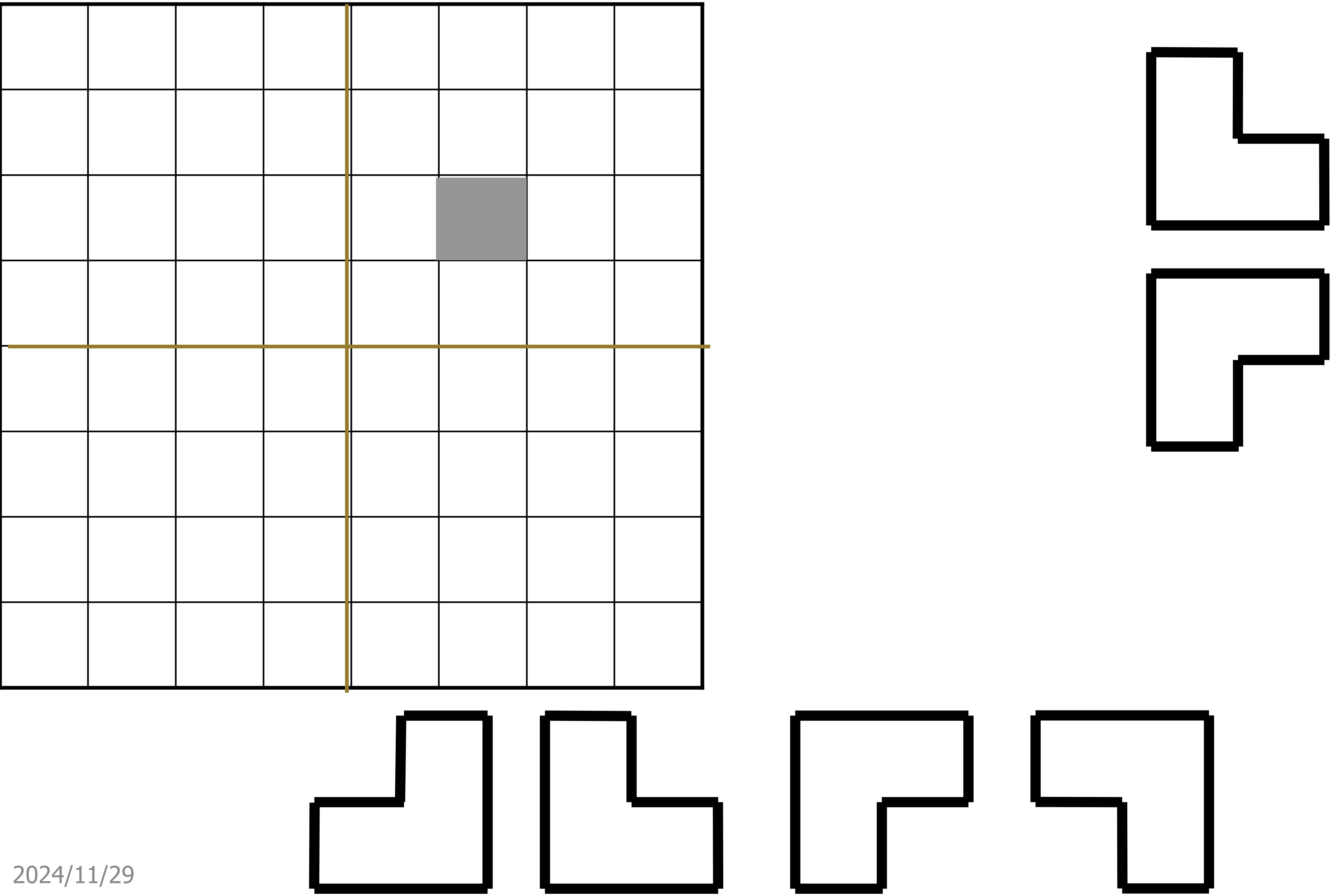


3	3	4	4	8	8	9	9
3	2	2	4	8	7	7	9
5	2	6	6	10		7	11
5	5	6	1	10	10	11	11
13	13	14	1	1	18	19	19
13	12	14	14	18	18	17	19
15	12	12	16	20	17	17	21
15	15	16	16	20	20	21	21

输入: k, 代表$2^k \times 2^k$棋盘

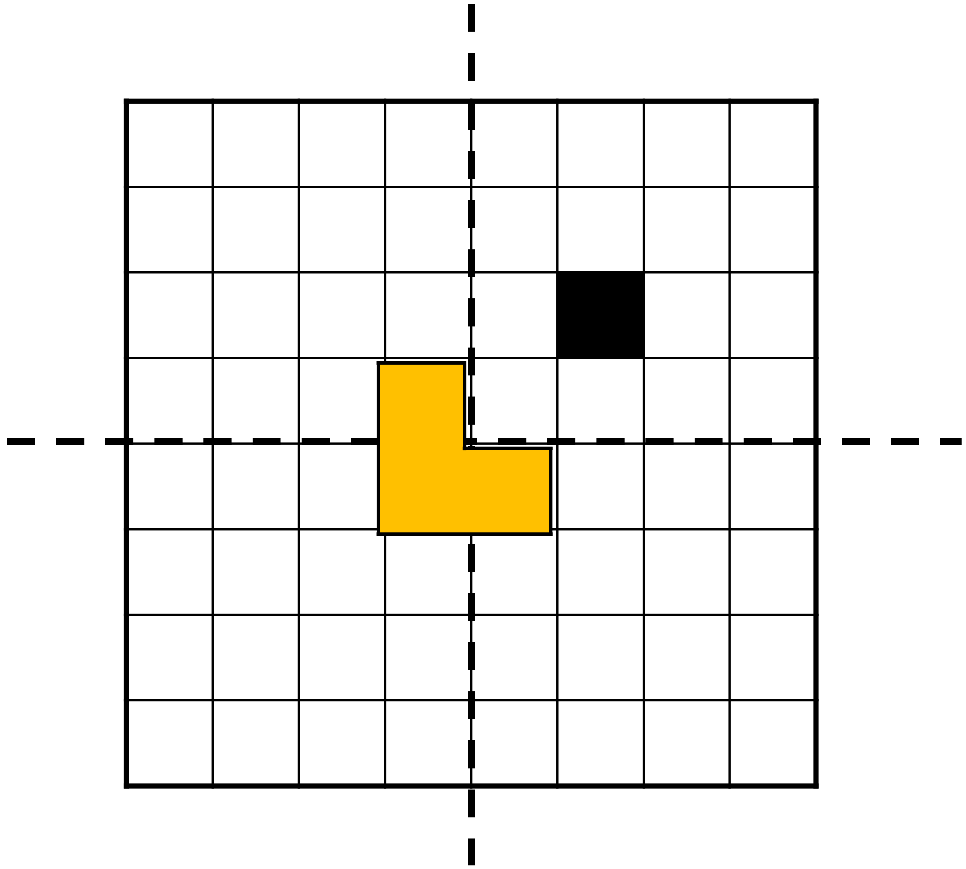
输出: 用L型骨牌覆盖棋盘的方案

说明: 有很多方案,
构造出一种方案即可

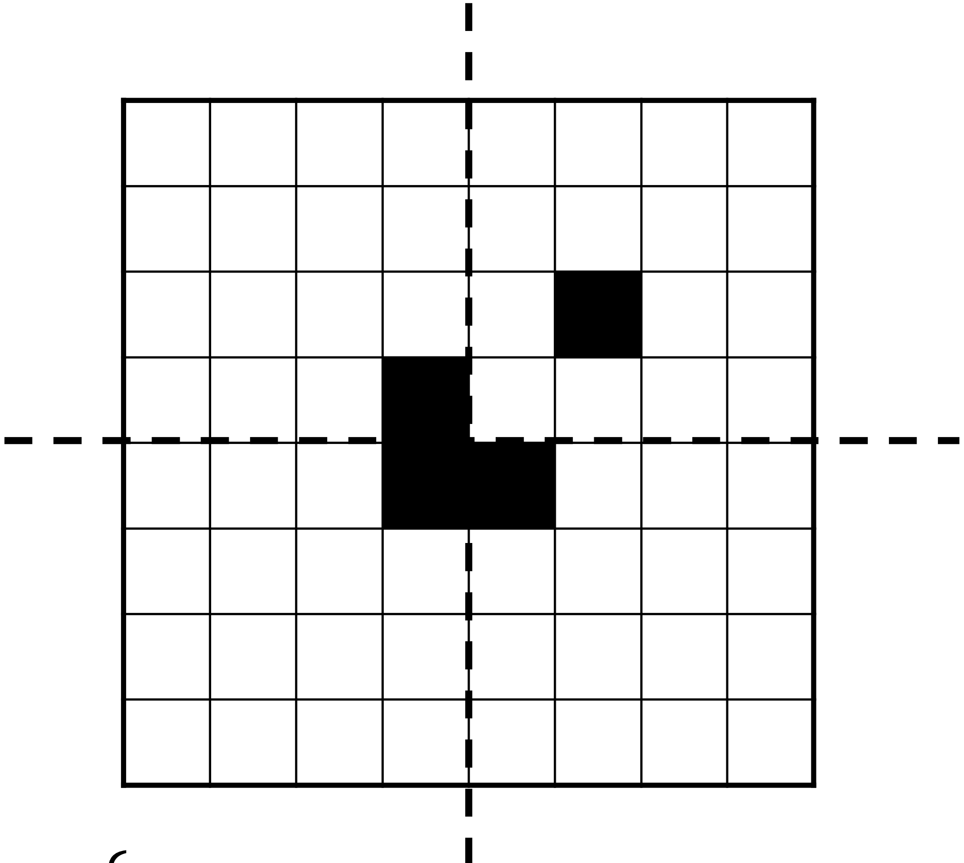


97 of 158

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

加2

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

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，$1\le i \le n$, 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 | 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?