9 分治策略

北京理工大学

高春晓

教材

- 教材
  - [1] [殷]殷人昆. 数据结构（C语言描述）(第二版)[M].清华大学出版社, 2017.04.
  - [2] [严]严蔚敏，吴伟民编. 数据结构（C语言版）[M].清华大学出版社, 2012.
  - [3] [王]王晓东编著，计算机算法设计与分析[M]. 电子工业出版社，2012.02
  - [4] [S] Sipser著，唐常杰等译. 计算理论导引[M]. 机械工业出版社，2017.11
- 参考书
  - [1] [C]Cormen等著，殷建平等译. 算法导论[M], 机械工业出版社，2013.01
  - [2] [M]Manber著，黄林鹏等译. 算法引论-一种创造性方法[M].电子工业出版社，2010.01
  - [3] [L]Lewis等著, 张利昂等译. 计算理论基础[M].清华大学出版社，2006.07
  - [4] [W]Mark Allen Weiss. Data Structures and Algorithm Analysis in C (Second Edition)[M]. 机械工业出版社, 2010.

本章内容

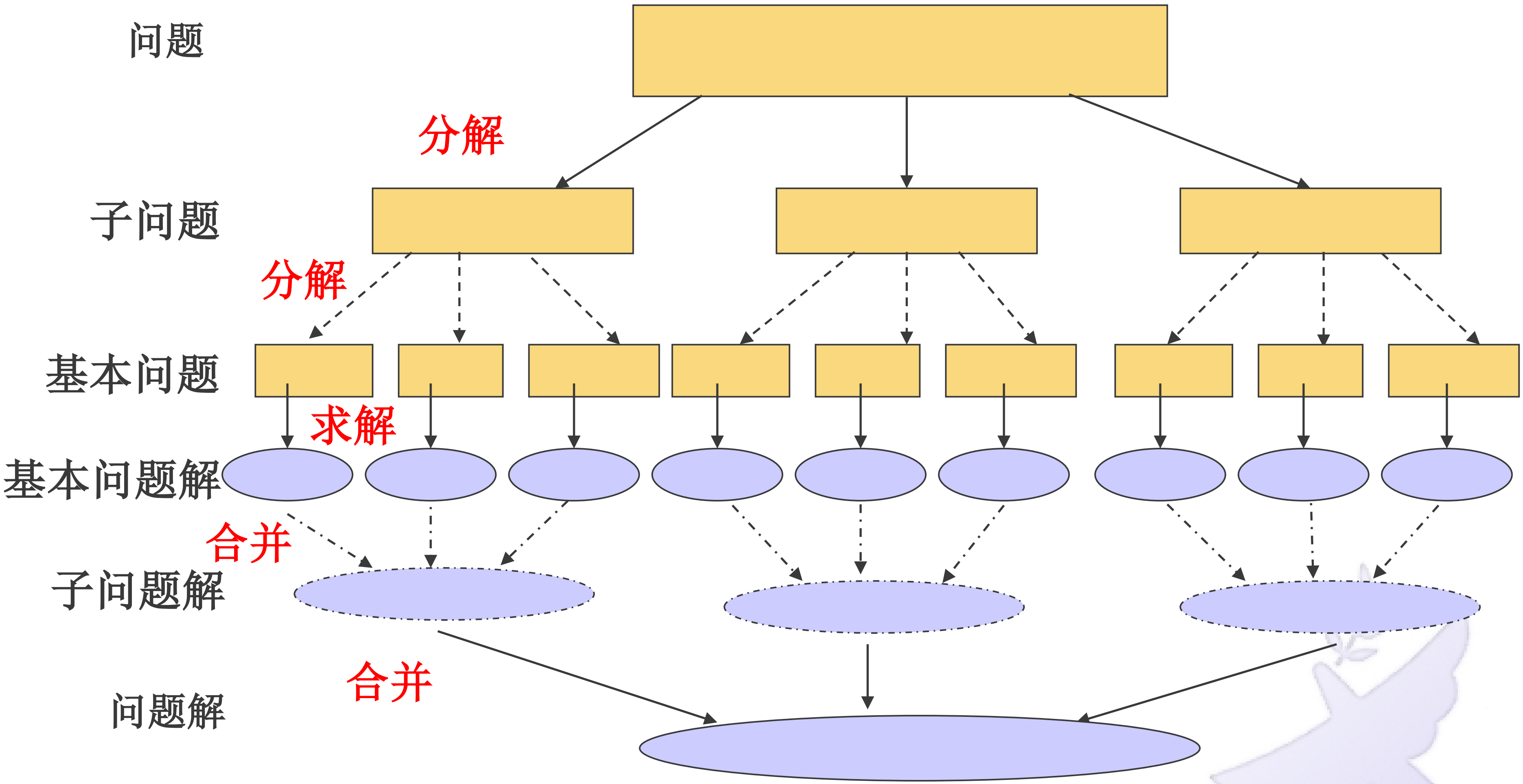
- 1. 分治原理, 主定理, 二分法
- 2. 大整数乘法
- 3. 线性时间选择
- 4. 最大子段和
- 5. 最接近点对问题
- 附录

1. 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

1. 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
2. 子问题规模(n/b)大致相等(平衡思想)
3. 子问题和原问题类似, 可递归求解
4. 子问题解合并能得到原问题解

```
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
   }
}
```

分治算法的时间

- 设时间复杂度为T(n)，每次分解出的子问题有a个,
- 分解+合并时间为f(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);        //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 ([王])

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

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

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

注:[C]中有详细证明.

分治主定理([M]Page37)

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

设a≥1, b≥2

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

- 例1： a=1, b=2, k=0， $a=b^k$.
  T(n) = logn
- 例2： a=2, b=2, k=1， $a=b^k$.
  T(n) = nlogn
- 例3： a=4, b=2, k=1， $a>b^k$.
  $T(n) = n^2$
- 例4： a=2, b=2, k=2， $a<b^k$.
  $T(n) = n^2$.

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

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

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

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

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

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

条件: 性质P满足单调性

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

例2: 求f(x)=lnx+2x-6在(2,3)中的近似零点.

本章内容

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

2. 大整数乘法

- 问题描述
  - 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

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

大整数乘法: 分治

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

$X=\underbrace{A}_{n/2位}\underbrace{B}_{n/2位}$ $Y=\underbrace{C}_{n/2位}\underbrace{D}_{n/2位}$

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

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);
} }
```

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(n²)　　问题: AC, AD, BC, BD不独立

$$T(n)=\begin{cases} O(1) & n\le n_0 \\ aT(n/b)+cn^k & n>n_0 \end{cases} \quad T(n)=\begin{cases} \Theta(n^{\log_b a}) & a>b^k \text{ or } k<\log_b a \\ \Theta(n^{\log_b a}\log n) & a=b^k \text{ or } k=\log_b a \\ \Theta(n^k) & a<b^k \text{ or } k>\log_b a \end{cases}$$

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

- **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, $O(n^{2.373})$.**

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

本章内容

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

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;
}
```

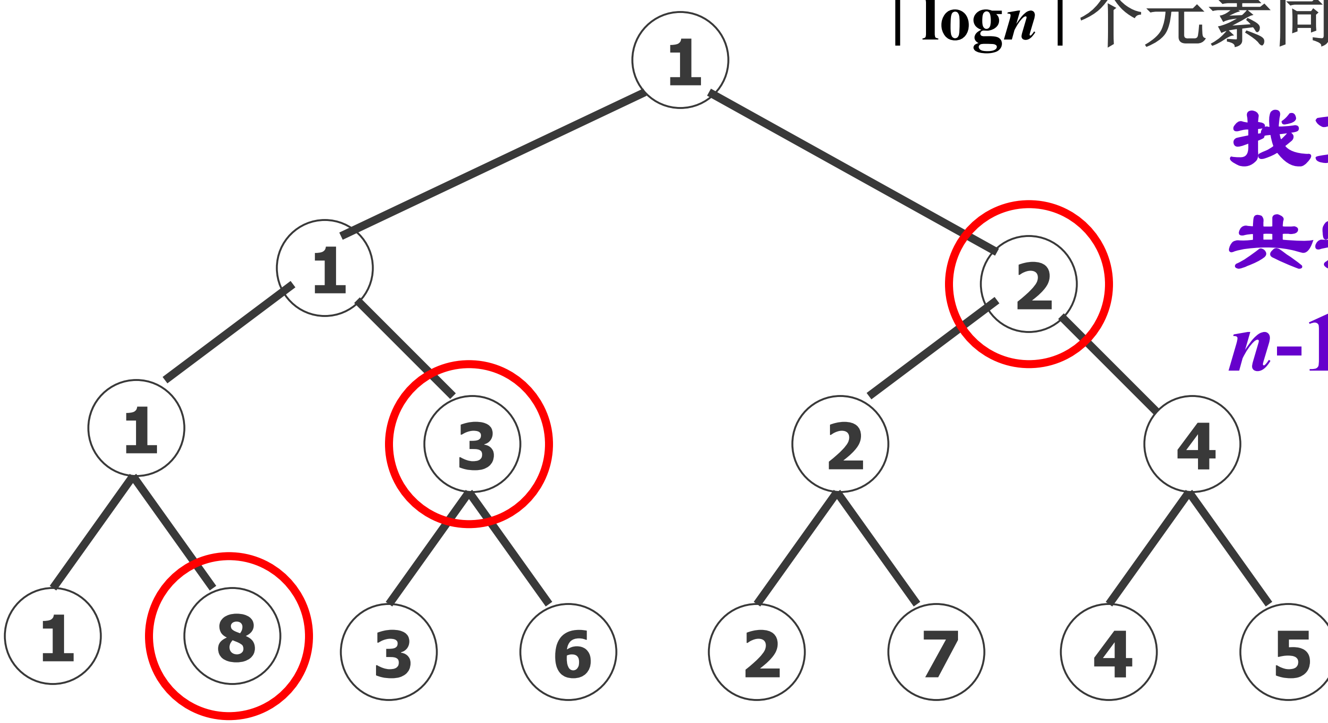
最小问题

- 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.

求次小的元素

- 一般情况下比较次数为：(n-1)+(n-2)=2n-3
- 次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 $a_1,\ldots,a_n$, 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.

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: 改进快排, 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

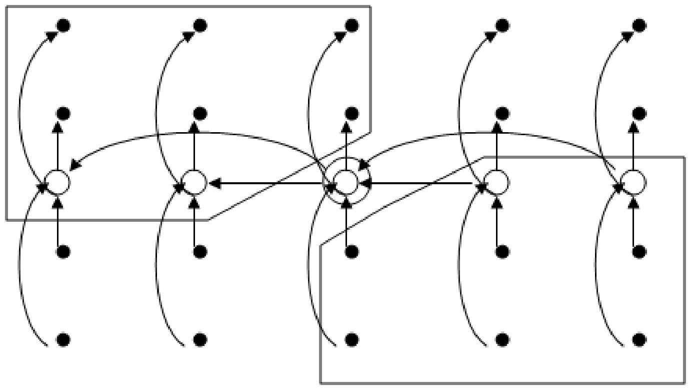
```
RSelect(a,p,r,k){     //选择a[p:r]中第k小数
    mid=RamdomizePartition(a,p,r);
    if( mid >= k)return(RSelect(a, p, mid,k));
    else return(RSelect(a, mid+1, r, k-mid);
}  //粗略时间分析: T(n) = T(9n/10) + O(n) = O(n)
```

改进选择算法：线性时间选择算法

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⌈n/5⌉组, 取出每组中位数(共⌈n/5⌉个),
2. 使用Select找这⌈n/5⌉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

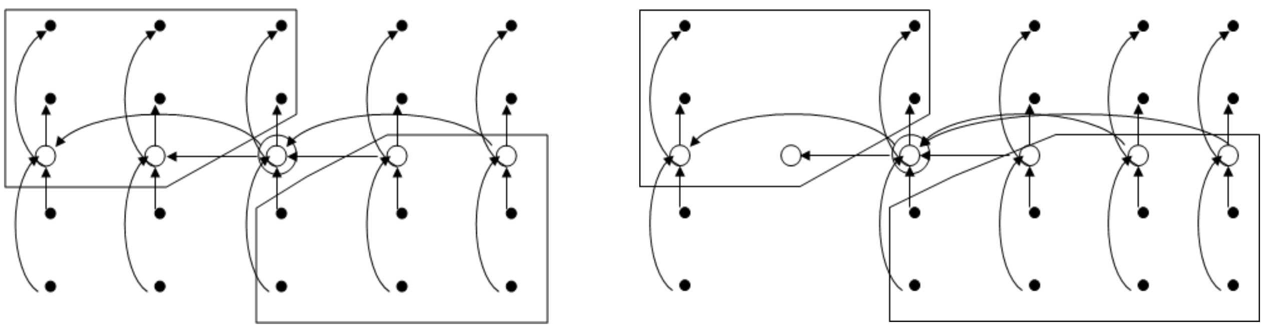


- 假设所有数互不相同
- 当n充分大, 至少有1/4的数<新基准, 1/4的数>新基准?

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

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] $3\lfloor (n-5)/10 \rfloor \geq n/4$

需要 $n \geq 75$.

说明：当n充分大, 至少有1/4的数<新基准, 1/4的数>新基准。

◆ 比基准小的有$\frac{1}{2}\left\lfloor \frac{n}{5} - 1 \right\rfloor$组，在每组中有2个元素小于本组的中位数，而每组的中位数都小于基准数，所以$\frac{3}{2}\left\lfloor \frac{n}{5} - 1 \right\rfloor$个小于基准；因此至少有$3\left\lfloor \frac{n-5}{10} \right\rfloor$个元素小于基准。

◆ 而当n≥75时$3\left\lfloor \frac{n-5}{10} \right\rfloor \geq \frac{n}{4}$，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)
}
```

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

本章内容

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

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

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

最大子段和

- 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),
- 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

```
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：$T(n)=O(n^3)$

最大子段和

```
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；

最大子段和: 分治算法

- 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

最大子段和: 分治算法

```
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;
       }
```

最大子段和: 分治算法

```
    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
```

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

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

本章内容

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

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

$$d(p_s, p_t) = \min \{ d(u,v) \mid u \neq v \in P \}$$

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 $P = \{ p_1, p_2, \ldots, p_n \}$
- **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

4. 若 min > d(p_i,p_j),

O(1) **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 logn) 1. 排序: $p_{i_1} \le p_{i_2} \le \ldots \le p_{i_n}$.

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) 4. 若 $\min > d(p_{i_k}, p_{i_{k+1}})$

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

- 总时间 O(nlogn)
- 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. $\delta = \min \{ d_L, d_R, q-p \}$

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

最近点对—二维分治尝试

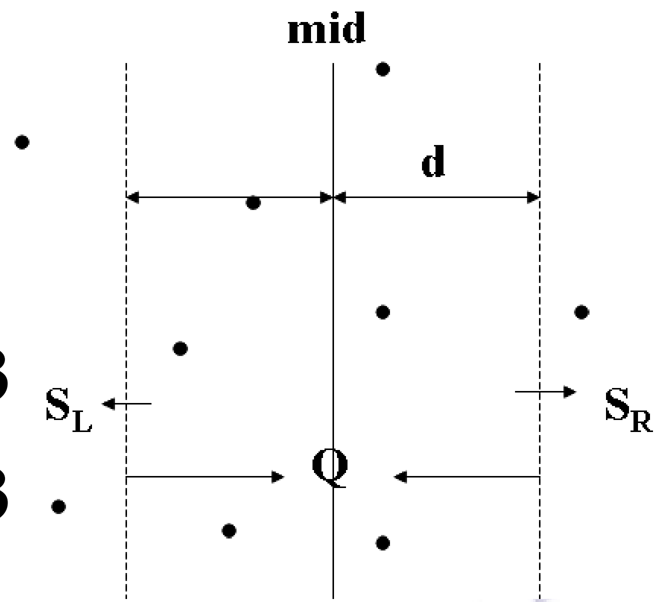
设点集合为S,

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

分O(n), 治2T(n/2), 合$O(n^2)$

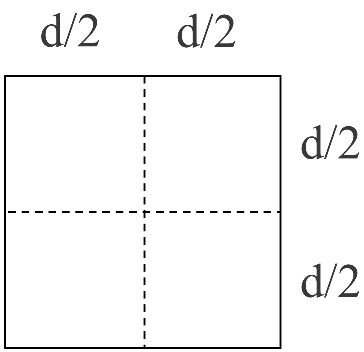
根据分治主定理 $T(n) = O(n^2)$

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



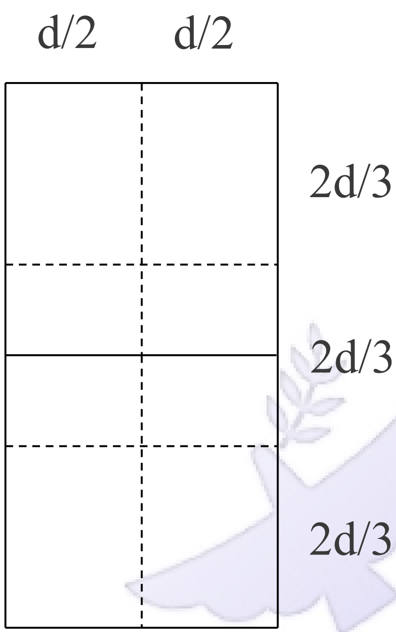
鸽巢(抽屉)原理的简单应用

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.



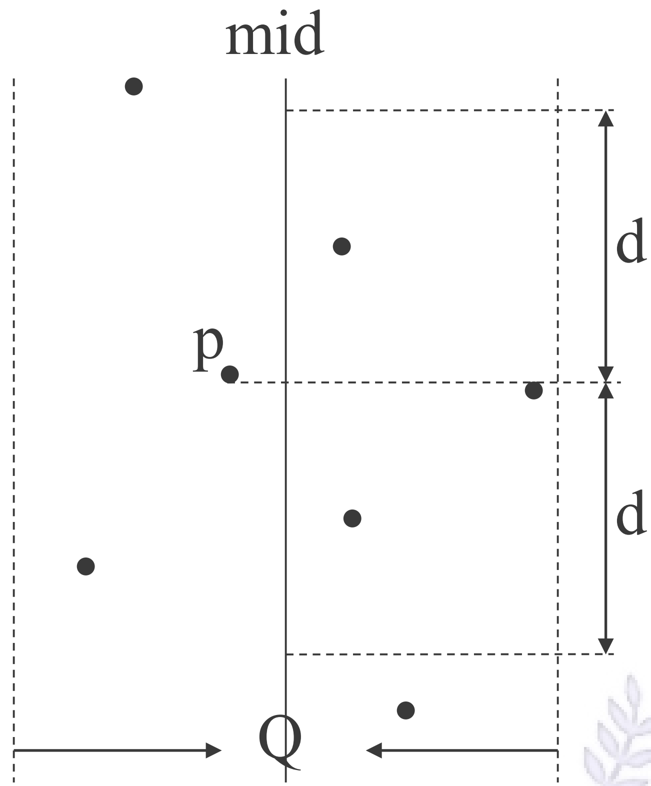
任取一个d×2d矩形内点集A,
若A中任意两点距离都≥d, 则A中点数≤6.
A中任意两点的距离小于等于:

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



方案一: Q左右分开

Q右侧中与p距离 < d 的点数 ≤ 6



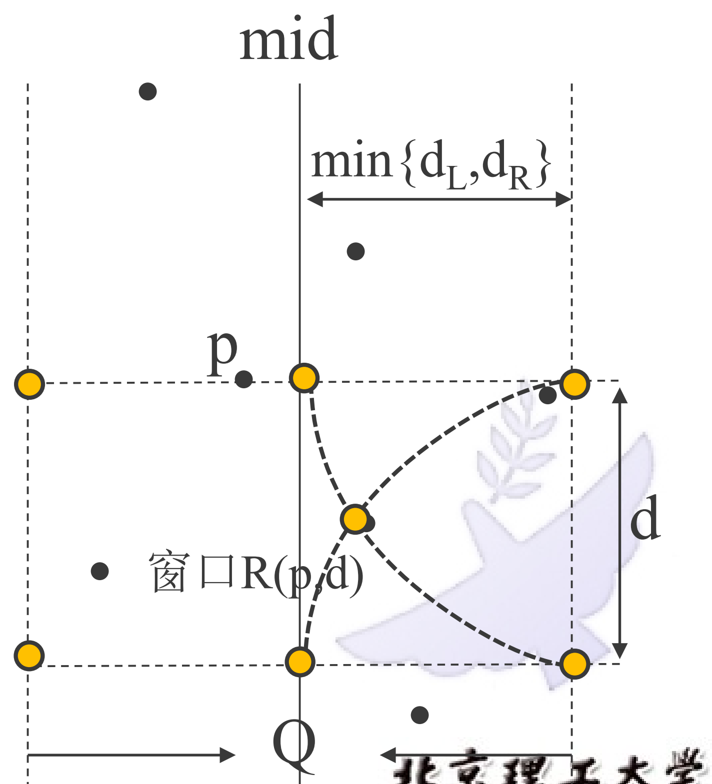
方案二: 检查p下方的点

- 定义窗口

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

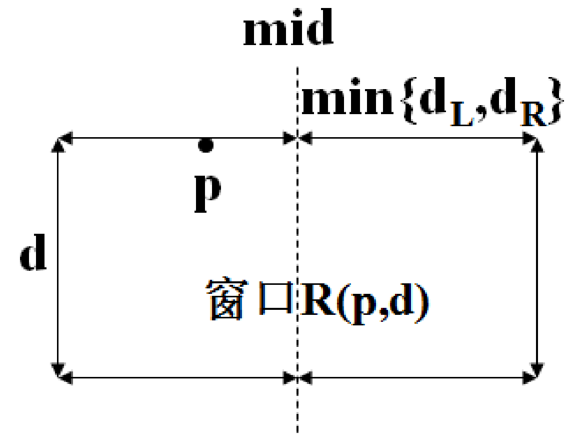
- Q中p下方与p距离 ≤ d 的点一定在R(p,d)中

而且点数 ≤ 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 $S_L \le_x S_R$.
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}



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

步4O(nlogn);步7-8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?

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

最近点对--排序放到分治前

- 初始化S按y坐标升序, 求x坐标中位数mid
- 根据mid将S放入另一数组(S_L,S_R), S_L(S_R)各按*y*坐标升序
- 例: **mid=100**

…,(120,10), (95,20), (105,20), (85,30), (97,50), (93, 60), (107,80), (103, 90), …

得到 S_L = {(95,20), (85,30), (97,50),(93, 60), …},

S_R = {(120,10), (105,20), (107,80), (103, 90), …},

- 按y坐标升序归并S_L,S_R($\cap$Q)得Q, 则Q按y坐标升序
- 例如: **d=10**, 对y坐标升序执行归并得到 Q:

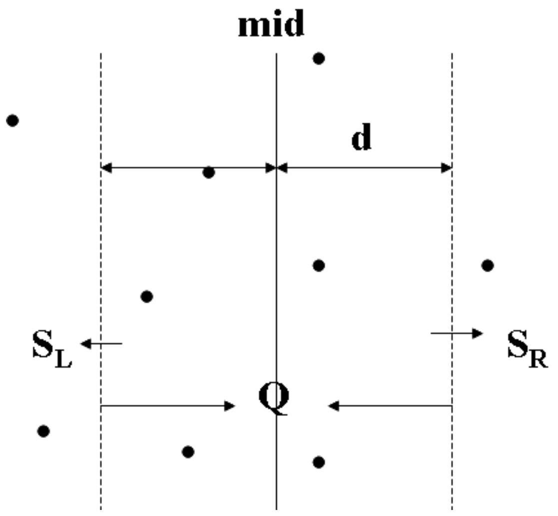
~~(120,10),~~ (95,20), (105,20), ~~(85,30),~~ (97,50), (93, 60), (107,80), (103, 90)

- 归并时间**O(n)**.

最近点对--合并时间改进二

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



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

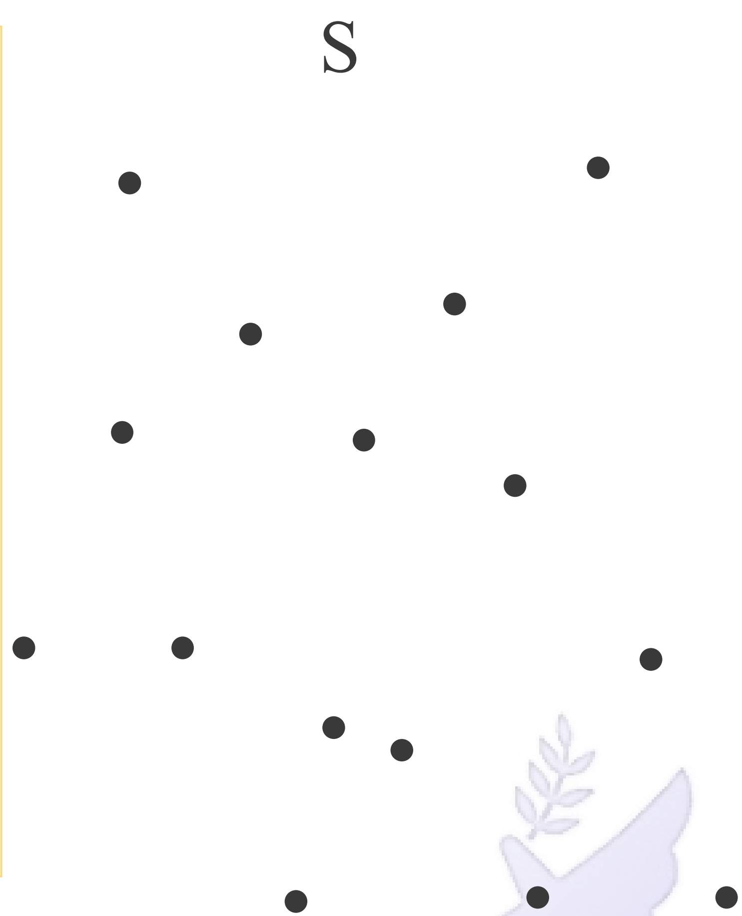
O(nlogn)

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

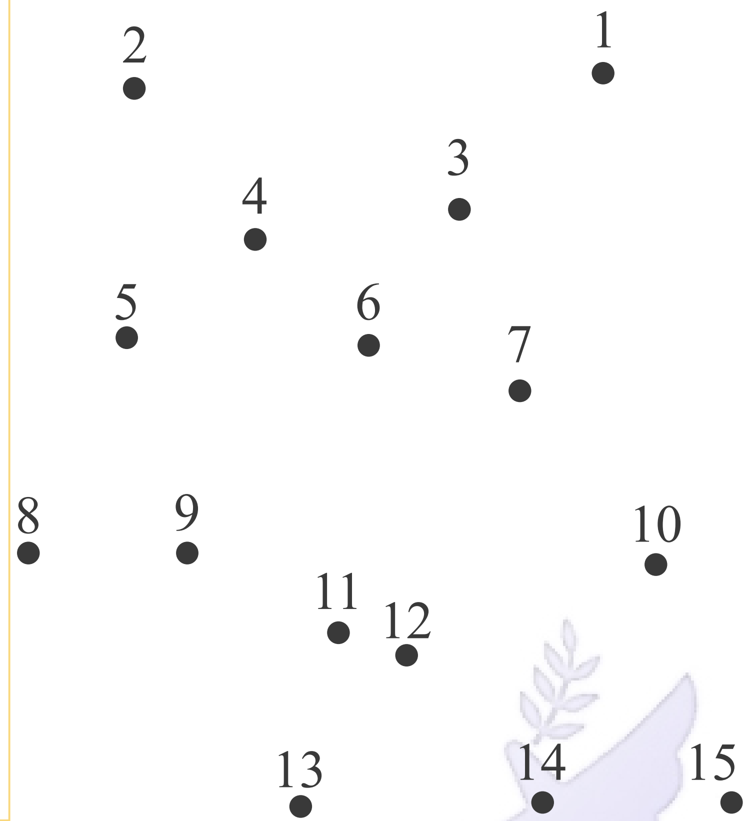


算法图示--预处理

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

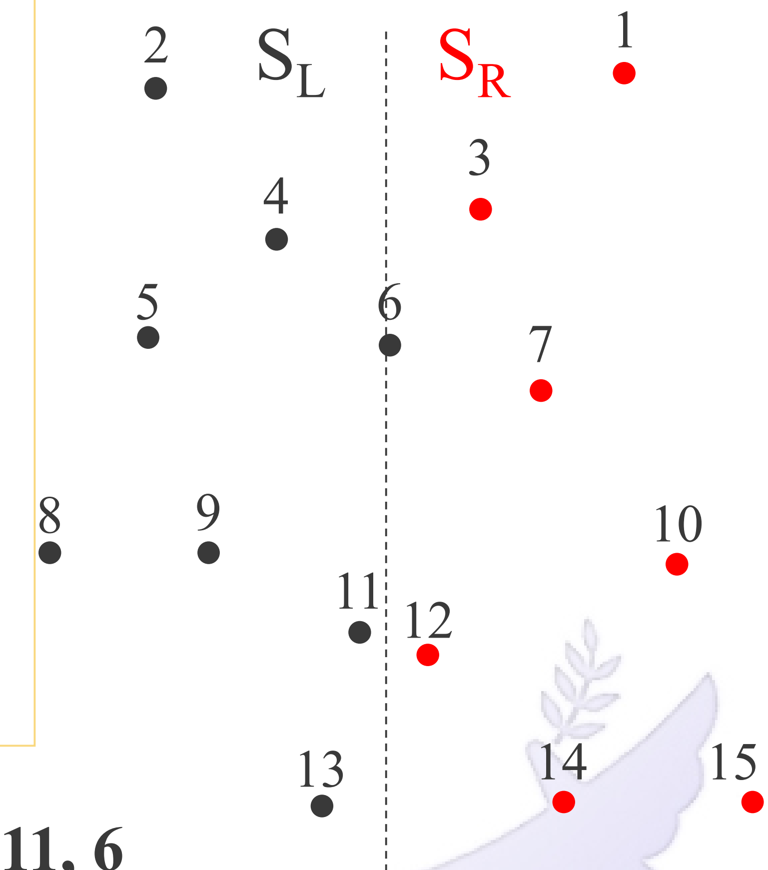


算法图示--分

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



左侧按x顺序为: 8, 5, 2, 9, 4, 13,11, 6

右侧按x顺序为: 12, 3, 7, 12, 1, 10, 15

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)

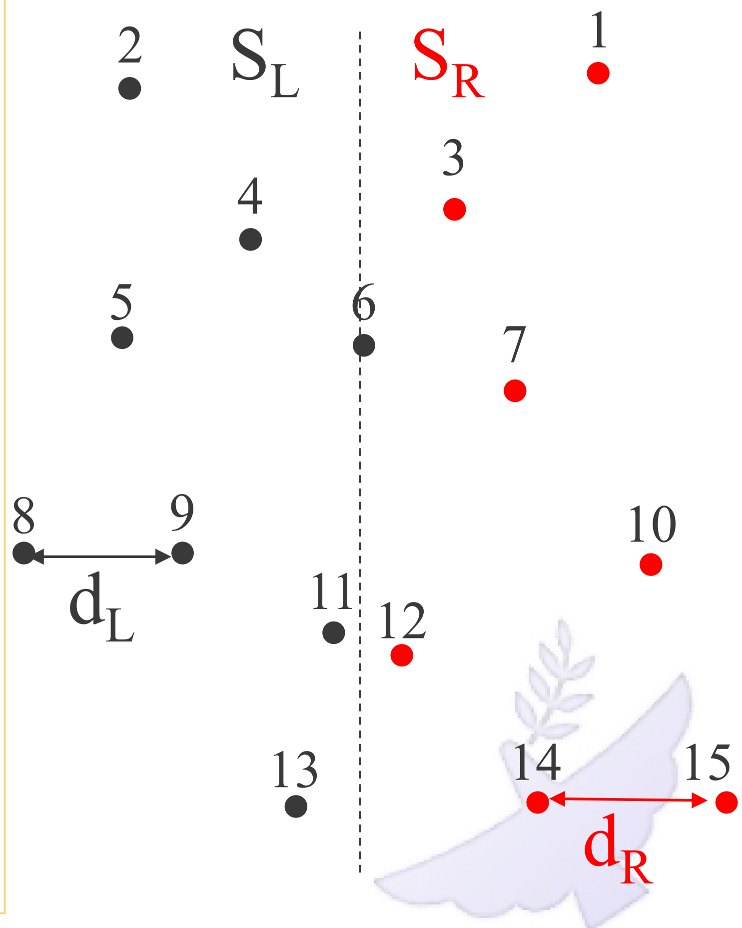
3. 合: $d = \min \{ d_L, d_R \}$

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

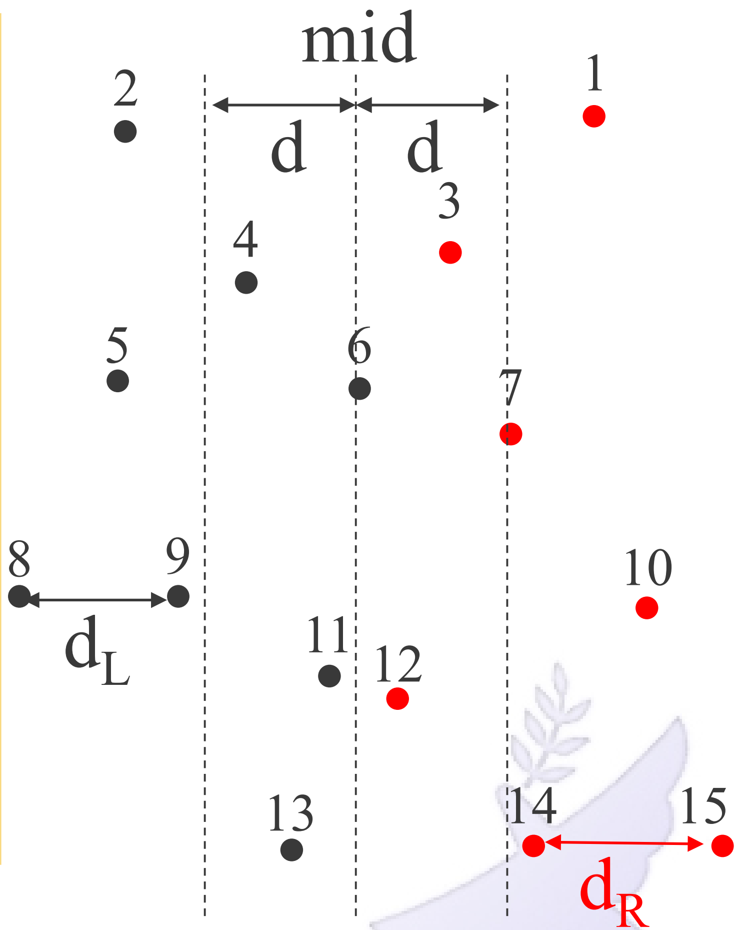


算法图示--合3

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

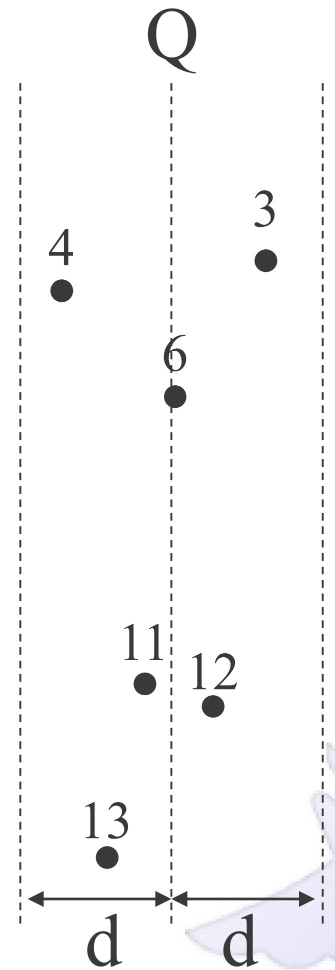


算法图示--合4

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



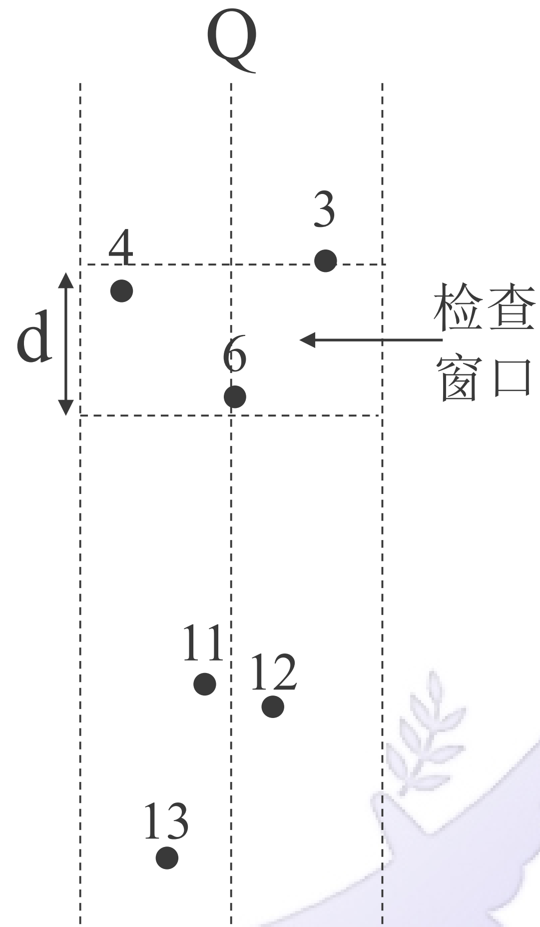
北京理工大学

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

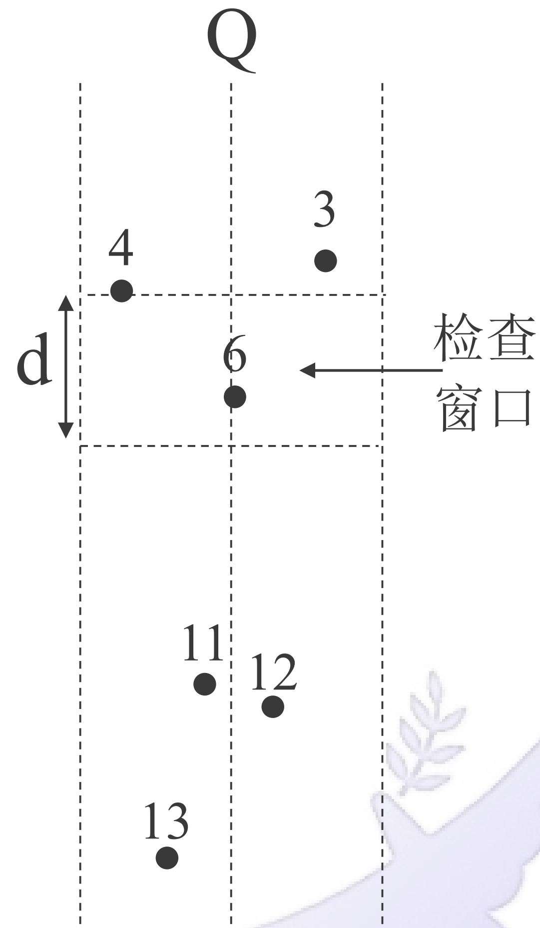


算法图示--合67:p_4

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

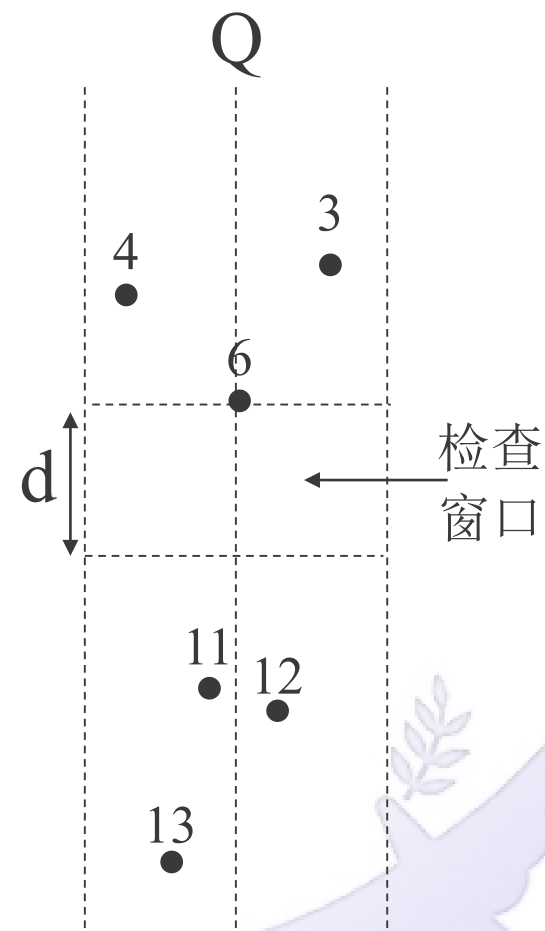


算法图示--合67:p_6

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

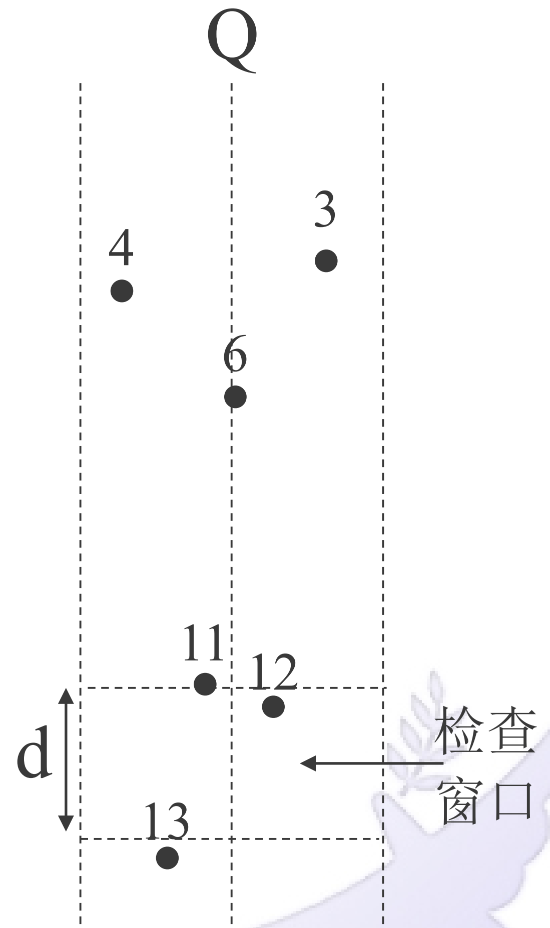


算法图示--合6:p_{11}

设有平面点集S

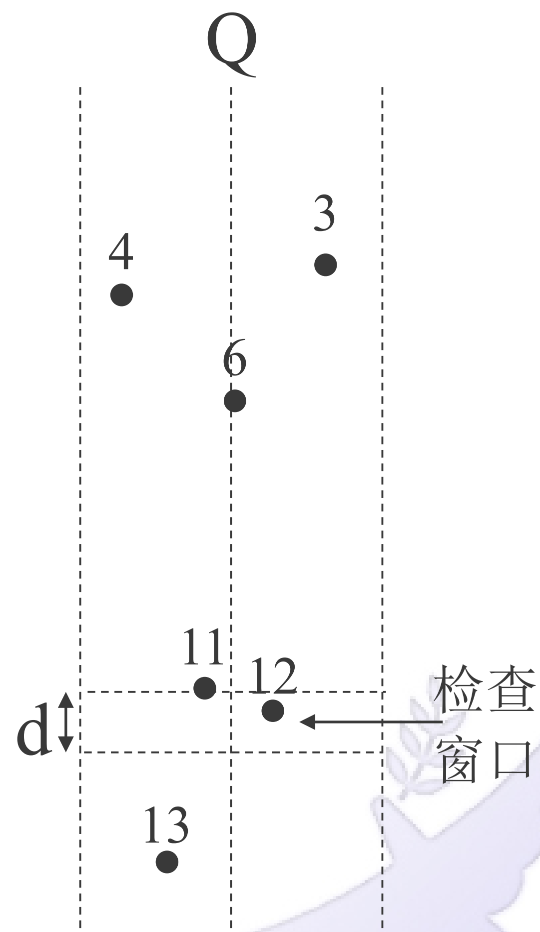
按y坐标递减(预处理)

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

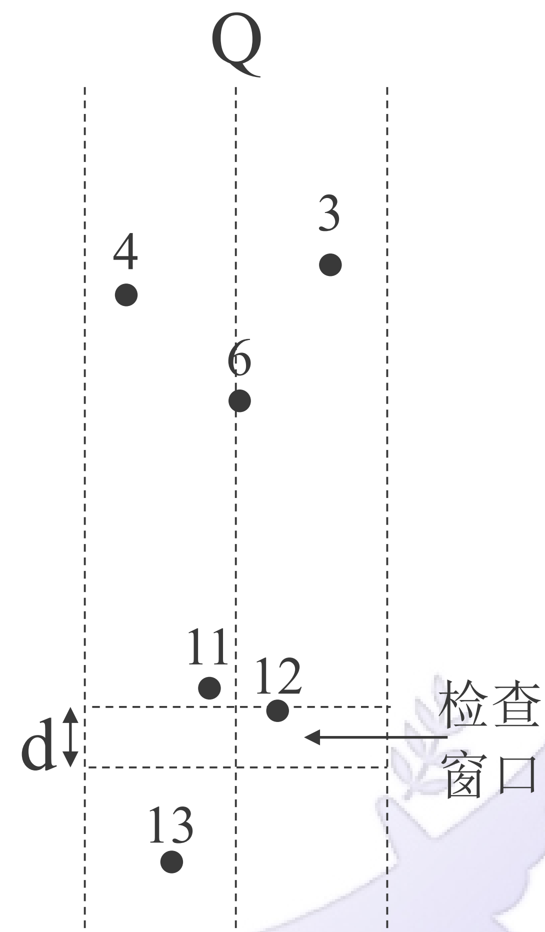


算法图示--合67:p_{12}

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

```
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

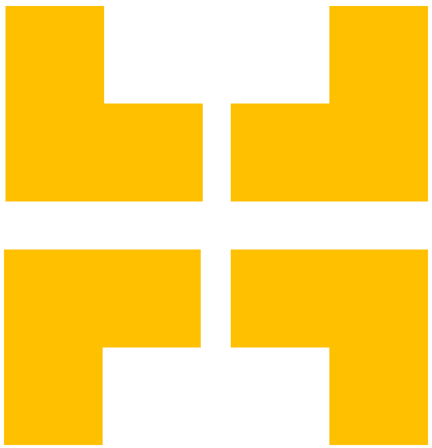
```
int main()
{
    int n;
    scanf("%d",&n);
    PointX *X = new PointX [n];
    float xx,yy;
    for(int i = 0; i < n; i++)  //输入数组X
    {
        scanf("%f %f",&xx,&yy);
        X[i].ID = i; X[i].x = xx; X[i].y = yy;  //给点编号
    }
    PointX& a; PointX& b; float& d;
    Cpair2(X, n, a, b, d);
 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

最近点对程序

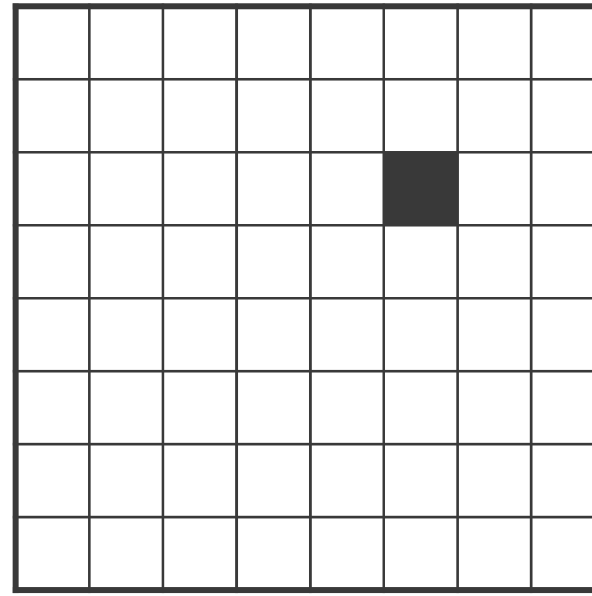
```
void closest(PointX X[], PointY Y[], PointY Z[], int l,
                    int r, PointX& a, PointX& b, float& d)
{   if( r - l <= 2) {直接计算; return;}          //2点和3点的情形
    int m = ( l + r ) / 2; int f = l, g = m + 1;  //多于3点的情形，用分治法
    for( int i = l; i <= r; i++) if( Y[i].p > m ) Z[g++] = Y[i]; 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
}   }  }
```

棋盘覆盖

L型骨牌



$2^k \times 2^k$棋盘



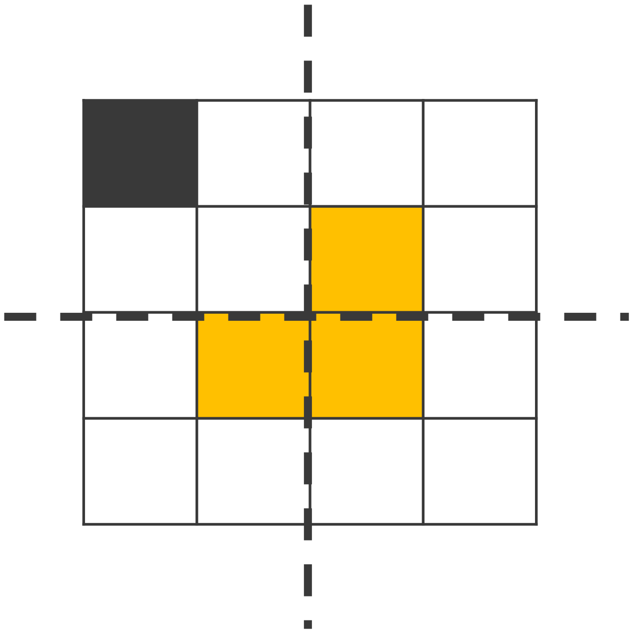
3	3	4	4	8	8	9	9
3	2	2	4	8	7	7	9
5	2	6	6	10		7	11
5	5	6	1	10	10	11	11
13	13	14	1	1	18	19	19
13	12	14	14	18	18	17	19
15	12	12	16	20	17	17	21
15	15	16	16	20	20	21	21

输入: k, 代表$2^k \times 2^k$棋盘,$k>1$

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

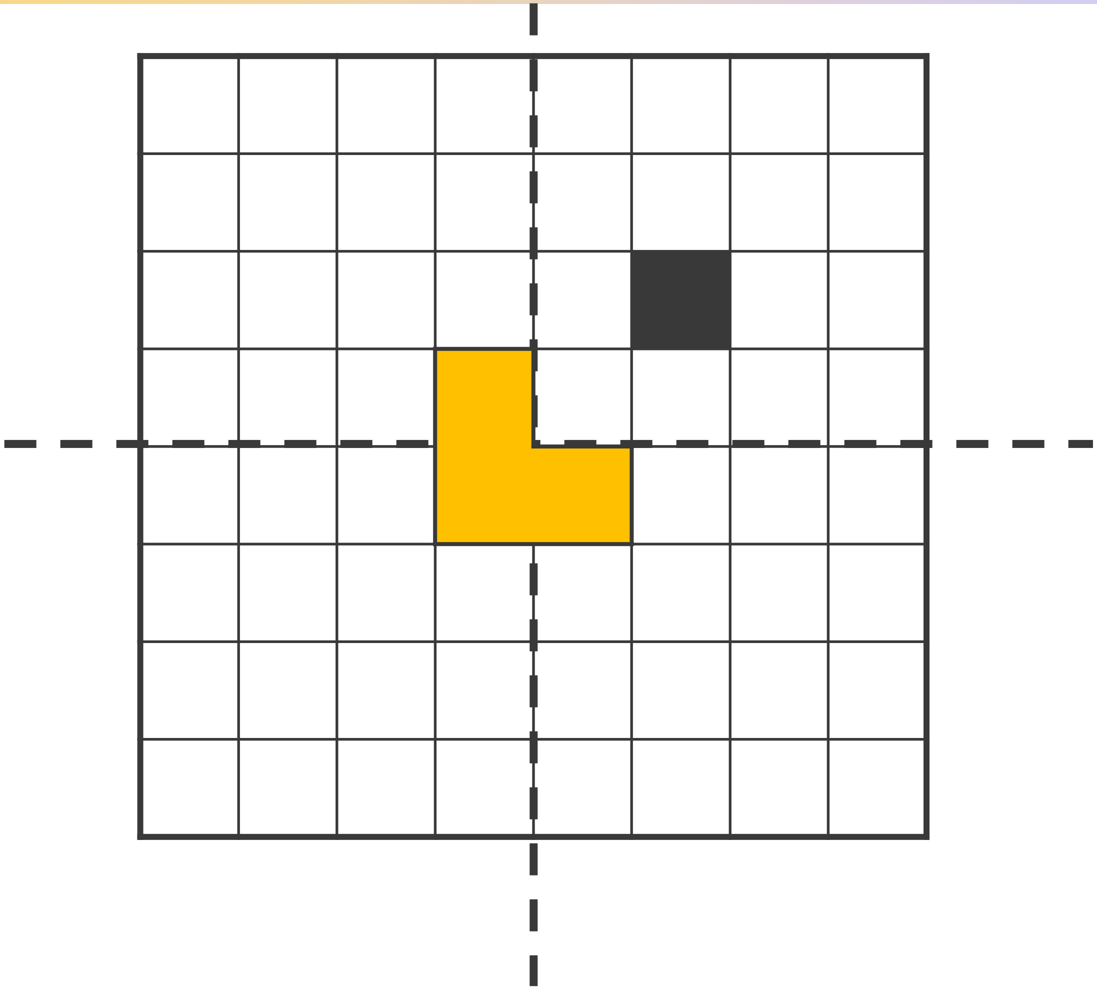
$2^1 \times 2^1$棋盘



$2^2 \times 2^2$棋盘

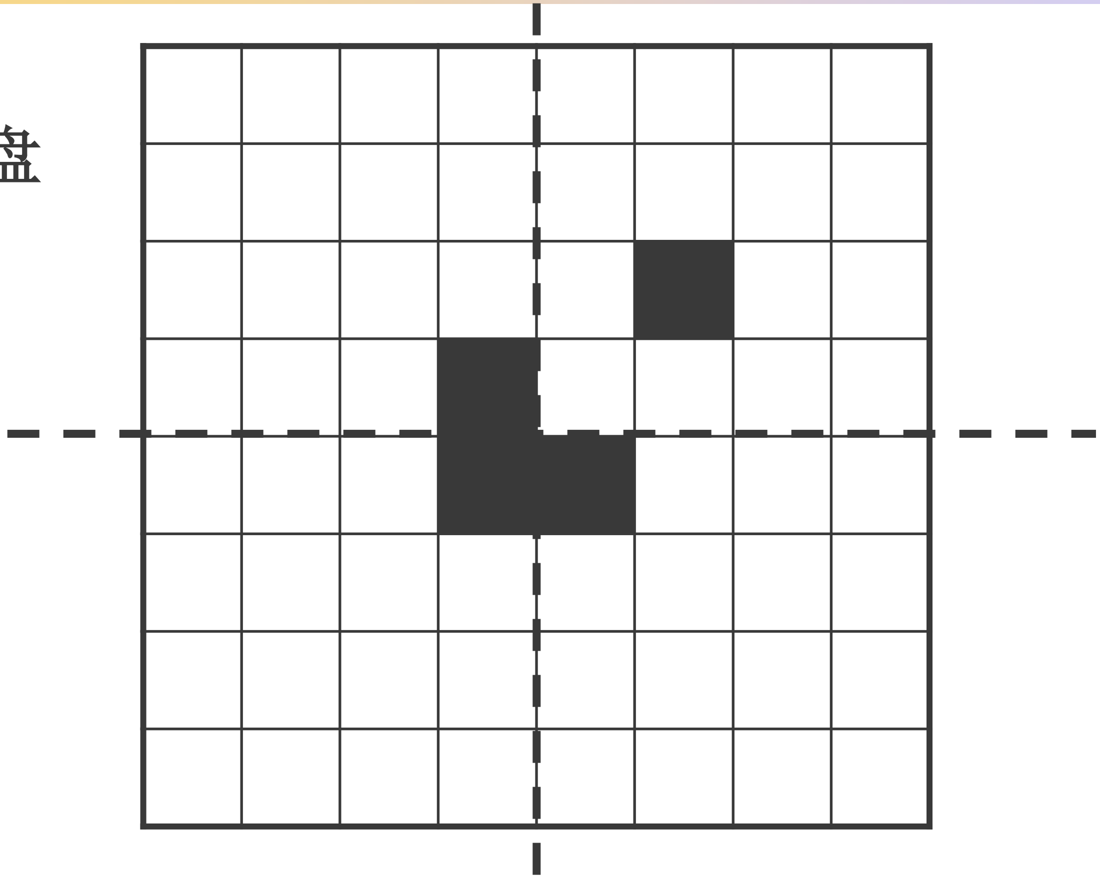
分治: 递归构造

$2^3 \times 2^3$棋盘



分治: 递归构造

$2^k \times 2^k$棋盘



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

编程变量设计

- tr 棋盘中左上角方格行号
- tc 棋盘中左上角方格列号
- dr 特殊方格行号
- dl 特殊方格列号
- size 棋盘的行数或列数, 初始=2^k.
- tile 正在赋值的L型骨牌的编号, 初始0
- Board[i][j] = t, 方格(i,j)被第t号骨牌覆盖
- void ChessBoard(int tr, int tc, int dr, int dc, int size)

程序设计

```
void ChessBoard(int tr, int tc, int dr, int dc, int size)
{ if (size<2)  return;
  int t = tile ++,        // L型骨牌编号
  s=size/2;               // 子问题棋盘大小
  if (dr < tr + s && dc < tc + s)  //特殊方格位于左上棋盘
    { Board[tr + s - 1][tc + s] = t;      // 记录t号骨牌
      Board[tr + s][tc + s - 1] = t;
      Board[tr + s][tc + s] = t;
      ChessBoard ( tr, tc, dr, dc, s);  // 覆盖其余部分
      ChessBoard (tr, tc+s, tr+s-1, tc+s, s);
      ChessBoard(tr+s, tc, tr+s, tc+s-1, s);
      ChessBoard(tr+s, tc+s, tr+s, tc+s, s); }
   … //右上, 左下, 右下
}
```

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

加4

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

本章小结和作业

习题 8, 9, 25

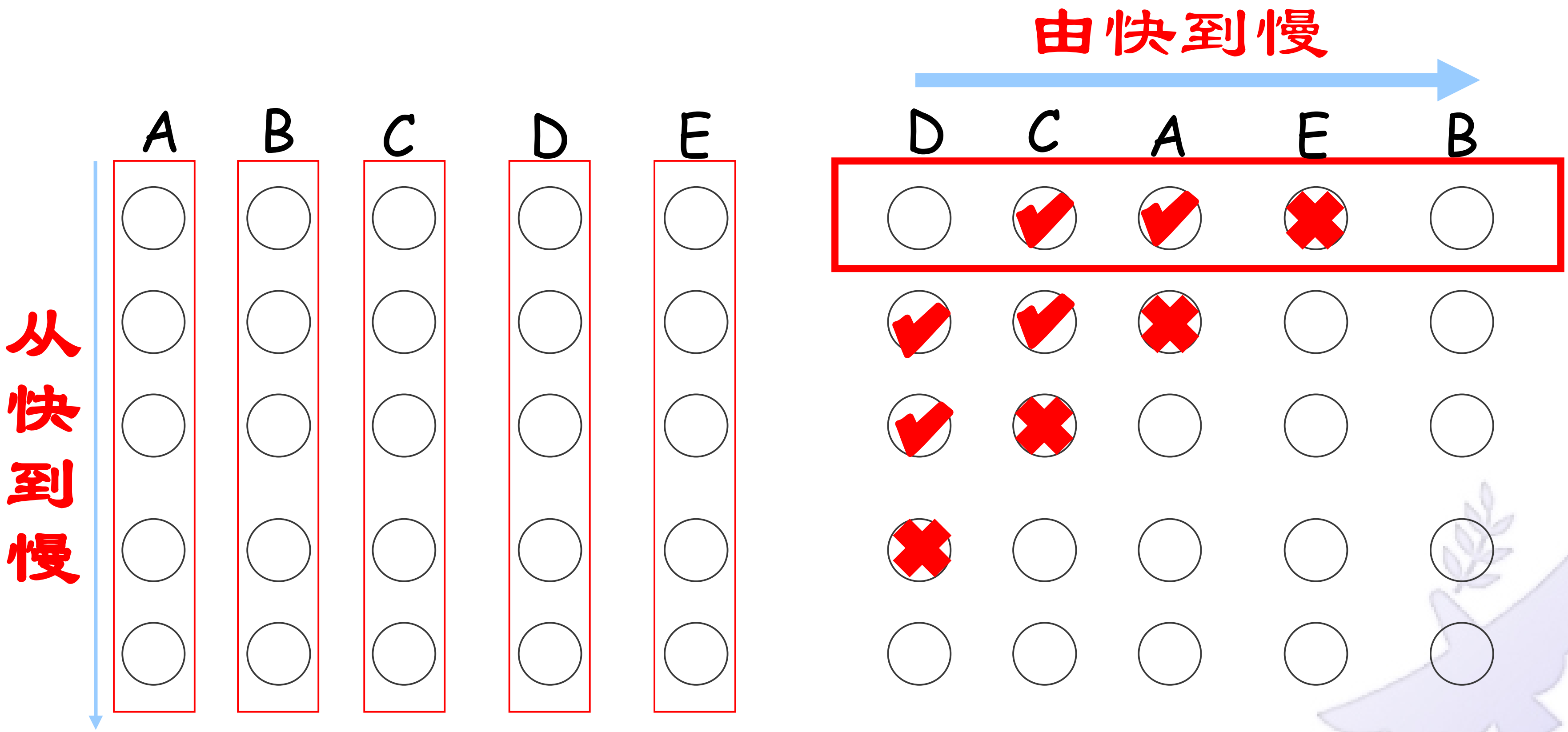
- **2-8. 设n个不同的整数排好序后存于T[1:n]中. 若存在一个下标i，$1\le i \le n$, 使得T[i]=i. 设计一个有效算法找到这个下标. 要求算法在最坏情况下的计算时间O(log n).**
- **2.9 设T[0:n-1]是n个元素的数组. 对任一元素x, 设S(x)={ i | T[i]=x}.当|S(x)|>n/2时, 称x为主元素. 设计一个线性时间算法, 确定T[0:n-1]是否有一个主元素.**
- **2.25 在线性时间选择算法中, 输入元素被划分为5个一组, 如果将它们划分为7个一组, 该算法仍然是线性时间算法吗? 划分成3个一组又怎样?**

习题1

- 现给出4根电缆，长度分别为8.02、7.43、 4.57、 5.39，要你把它们分割出11根等长的电缆，每根电缆的最大长度是多少？
- 解：先分析例子
  - 每1m一根，则可分出：8+7+4+5=24根
  - 每3m一根，则可分出：2+2+1+1=6根
- 二分法
  - 范围1：[0, +∞]
  - 范围2：[0, 最长电缆长度]
  - 范围3：[0, 所有电缆长度之和/11] ✔

习题2

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



5组

习题3

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

习题3

- 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

附录内容

- 分治主定理([M]Page37)
- 一些符号和公式

附录：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

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

注:[M]中为大O记号, 无详细证明.

由n是b的幂的简单情况猜公式

设整数$a\geq 1$, $b\geq 2$,
且 $a=b^k$, (注 $\log_b a = k$)

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

设$n = b^m$,

$T(n) = a\,(\,a\,T(n/b^2) + n^k/b^k) + n^k$, //迭代2次

$= a^2\,T(n/b^2) + 2n^k$, // $a=b^k$,

$= a^m\,T(n/b^m) + m\times n^k$, //迭代$m$次

$= b^{mk} + m\times n^k$, //$a=b^k$,

$= n^k\,(m+1)$, //$n=b^m$, $m=\log_b n$

$= \Theta(n^k \log n)$ // $k=\log_b a$

严格证明方法一:数学归纳法

设整数$a \geq 1$, $b \geq 2$,
且 $a=b^k$, (注 $\log_b a = k$)

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

先将T扩展到任意实数 $x \geq 1$, $T(x) = T(\lceil x \rceil)$,

再改为证明$T(x) = \Theta(x^k \log x)$. 取大数N.

假设对任意 $x \leq N$, $T(x) < c\, x^k \log x$, (设$c > 2/\log b$)

则对 $N < x \leq bN$,

$$\begin{aligned} T(x) &< a\, T(x/b) + 2x^k, && \text{//迭代1次} \\ &< a\, c(x/b)^k \log(x/b) + 2x^k, && \text{//归纳假设} \\ &= c\, x^k \log x - x^k(\, c \log b - 2\,), && //a=b^k, \\ &< c\, x^k \log x\,, \end{aligned}$$

下界证明类似. $a<b^k$ 证明类似.

$a > b^k$ 的情况

设整数$a\ge1$, $b\ge2$, $$T(n)=\begin{cases}1 & n=1\\ aT(n/b)+n^k & n>1\end{cases}$$

将T扩展到任意实数 $x\ge1$, $T(x) = T(\lceil x\rceil)$.

记$\gamma=\log_b a>k$. 证明$T(x) = \Theta(x^\gamma)$,

假设对任意 $x \le N$, $T(x) < c\, x^\gamma - 2b^k x^k/(a-b^k)$

则对 $N < x \le bN$,

$T(x) < aT(x/b) + 2x^k$, //迭代1次

$< ac(x/b)^\gamma - 2ab^k(x/b)^k/(a-b^k) + 2x^k$, //归纳假设

$= c\, x^\gamma - 2b^k x^k/(a-b^k)$ //$a=b^\gamma$,

下界证明类似.

严格证明方法二:直接计算([C])

设整数$a\geq 1$, $b\geq 2$, 且 $a>b^k$, (设$k\geq 1$)

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

固定n, 定义$n_0=n$, $n_i=\lceil n_{i-1}/\mathrm{b}\rceil$, $1\leq i\leq \mathrm{m}$, $n_m\leq \mathrm{M}$,

$\mathrm{T}(n) = a\,\mathrm{T}(n_1) + n_0^k$, //迭代1次

$= a^m\,\mathrm{T}(n_m) + \Sigma_{i=0}^{m-1}\, a^i\, n_i^k$, //迭代$m$次

$= a^m + \Sigma_{i=0}^{m-1}\, a^i\, n_i^k$, //(*)

由 $n_{i-1}/b \leq n_i \leq n_{i-1}/b+1$,

得 $n/b^i \leq n_i \leq n/b^i + b^{-i+1}+\ldots+b+1$

由$b\geq 2$得 $n/b^i \leq n_i \leq n/b^i + 2$

再由 $\mathrm{M}/\mathrm{b} \leq n_m \leq \mathrm{M}$

设M较大得 $\log_b(n/\mathrm{M}) \leq \mathrm{m} \leq \log_b(n/\mathrm{M})+1$

严格证明方法二:直接计算续

由上页(*)式

$$T(n)=a^{m}+\Theta\left(n^{k}\frac{(ab^{-k})^{m}-1}{ab^{-k}-1}\right)$$

$$=a^{m}+\Theta\left(a^{m}(nb^{-m})^{k}\right)$$

$$=\Theta(a^{m})=\Theta(n^{\log_b a})$$

附录：一些符号和公式

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

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

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

2020!末尾多少个0？

503个0

一些符号和公式

- 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.

一些符号和公式

- 对数

$\lg 2 = 0.301$

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

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

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

性质：

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

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

一些符号和公式

- 证明：

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

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

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

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

一些符号和公式

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

一些符号和公式

◆阶乘

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

$n! = o(n^n)$，即$n!$严格比n^n小

$n! = \omega(2^n)$，即$n!$严格比2^n大

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

一些符号和公式

◆ 求和

等比级数的求和公式

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

一些符号和公式

◆ 调和级数

$$\sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$$

1/1+1/2+1/3+…+1/n = ln n +γ

欧拉常数γ≈0.5772156…

END