

# 算法设计与分析

---

主讲人：权丽君

Email: *ljquan@suda.edu.cn*

苏州大学 计算机学院



# 第三讲 分治策略

## 内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法



# 第三讲 分治策略

内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法



# 分治法的基本思想

□ **基本思想**：当问题规模比较大而无法直接求解时，将原始问题分解为几个规模较小、但类似于原始问题的子问题，然后递归地求解这些子问题，最后合并子问题的解以得到原始问题的解。

□ **分治策略遵循三个步骤**：

1) **分解 ( Divide )**：将原问题分为若干个规模较小、相互独立，形式与原问题一样的子问题。

2) **解决 ( Conquer )**：递归地解各个子问题。若子问题足够小，则直接求解；否则“递归”地求解各个子问题，即继续将较大子问题分解为更小的子问题，然后重复上述计算过程。

3) **合并 ( Combine )**：将子问题的结果合并成原问题的解。

当子问题的规模足够大，需要进一步分解并递归求解时，这种情况称为**递归情况** (recursive case)；若子问题的规模变得足够小，不再需要再进一步分解了，这种情况称为**基本情况** (base case) (基本情况的子问题可以直接求解) 2021/10/21



# 分治算法的实例：归并排序



## □ 归并排序的基本思路：

- ① 分解：把  $n$  个元素分成各含  $n/2$  个元素的子序列；
- ② 解决：用归并排序算法对两个子序列递归地排序；
- ③ 合并：合并两个已排序的子序列以得到排序结果。

## □ 归并排序的过程描述：

```
MERGE-SORT( $A, p, r$ )
1   if  $p < r$ 
2       Then  $q \leftarrow \lfloor p+r/2 \rfloor$ 
3           MERGE-SORT( $A, p, q$ )
4           MERGE-SORT( $A, q+1, r$ )
5           MERGE( $A, p, q, r$ )
```

## □ 归并排序的时间分析：

2021/10/21

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

Soochow University



## • 递归式

- 使用递归式可以很自然地刻画分治算法的运行时间
- 等式或不等式，通过更小的输入上的函数值来描述一个函数

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$T(n) = T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + \Theta(n)$$

## • 三种求解递归式的方法

- 代入法：我们猜测一个界，然后用数学归纳法证明这个界是正确的。
- 递归树法：将递归式转换为一棵树，其结点表示不同层次的递归调用产生的代价。然后采用边界和技术来求解递归式。
- 主方法：可求解形如下面公式的递归式的界：

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- 其中  $a \geq 1, b > 1, f(n)$  是一个给定的函数。



# 第三讲 分治策略

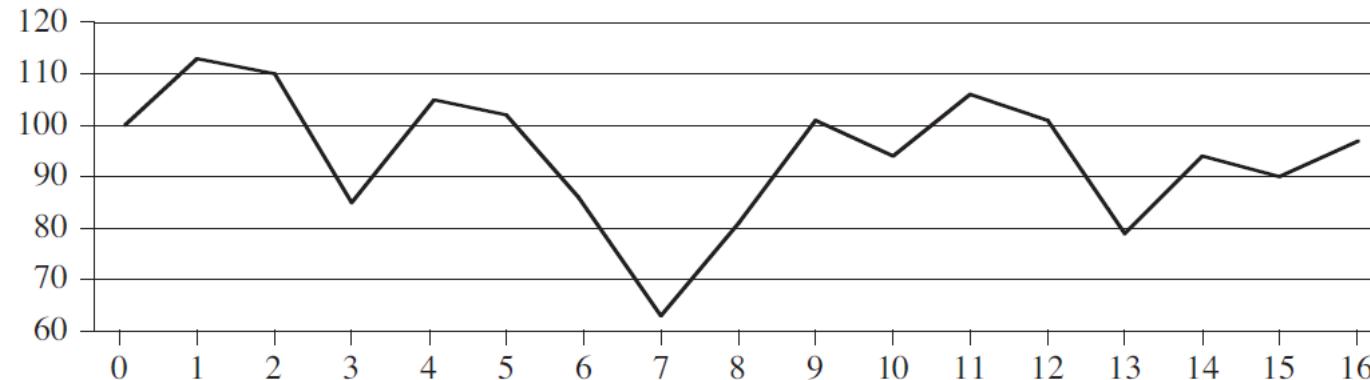
## 内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法



# 最大子数组问题

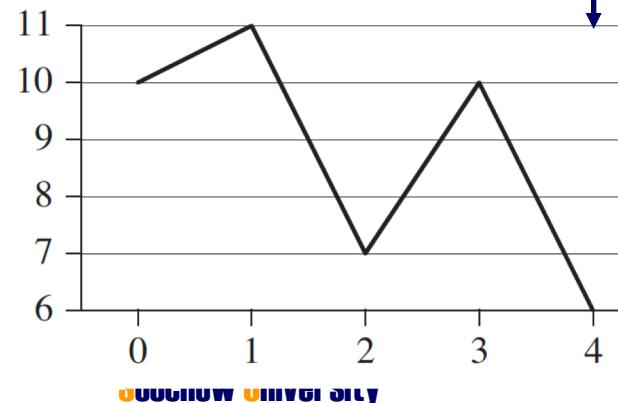
## □ 一个关于炒股的 story :



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

反例  
↓

□ 问：哪段时间最赚钱？  
即股市有起落，从哪天到哪天的收益最大呢？



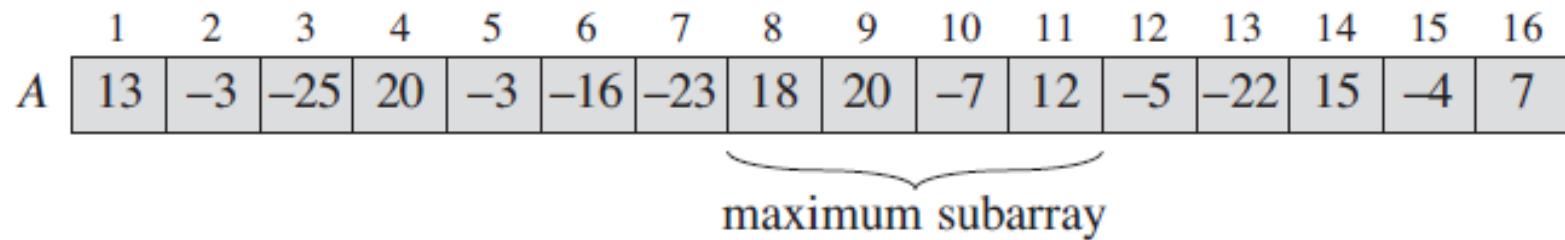


# 最大子数组问题



## □ 从问题定义到建模求解：

- ✓ 求解炒股问题的算法模型：**最大子数组问题**
- ✓ 已知数组  $A$ ，在  $A$  中寻找 “和” 最大的**非空连续子数组**



称这样的连续子数组为**最大子数组** (maximum subarray)

## □ 怎么求解？

- ✓ **方法一：暴力求解法 ( brute-force solution )**

搜索  $A$  的每一对起止下标区间的和，和最大的子区间就是最大子数组，时间复杂度： $\binom{n}{2} = \Theta(n^2)$



# 最大子数组问题



## 口 怎么求解？

### ✓ 方法二：使用分治策略求解

设当前要寻找子数组  $A[low..high]$  的最大子数组

**分治的基本思想是：**

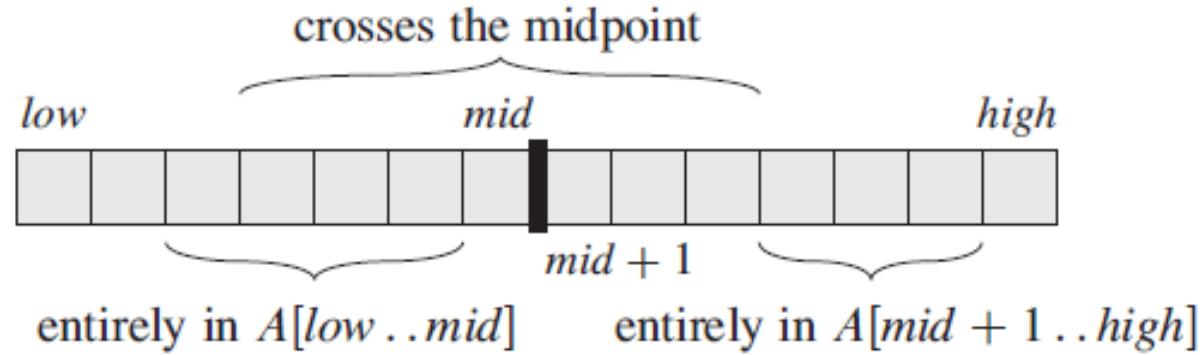
- 首先将子数组  $A[low..high]$  划分为两个规模尽量相等的子数组，分割点： $mid=(low+high)/2$ 。
- 然后分别求解  $A[low..mid]$  和  $A[mid+1..high]$  的最大子数组
  -



# 最大子数组问题



- 基于上述划分， $A[low..high]$  的连续子数组  $A[i..j]$  所处的位置必是下面三种情况之一：
  - 完全位于左子数组  $A[low..mid]$  中，因此  $low \leq i \leq j \leq mid$
  - 完全位于右子数组  $A[mid+1..high]$  中，因此  $mid \leq i \leq j \leq high$
  - 跨越了中点，因此  $low \leq i \leq mid < j \leq high$

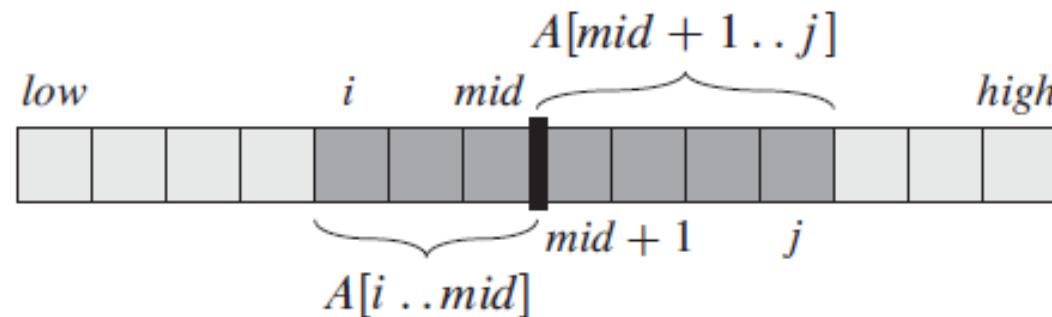




# 最大子数组问题



- $A[low..high]$  的**最大子数组**也是  $A[low..high]$  的连续子数组，所以  $A[low..high]$  的一个**最大子数组**所处的位置也必然是这三种情况之一。
- 即： $A[low..high]$  的这个“**最大子数组**”必然是：或者完全位于  $A[low..mid]$  中、或者完全位于  $A[mid+1..high]$  中、或者是跨越中点的所有子数组中和最大的那个。





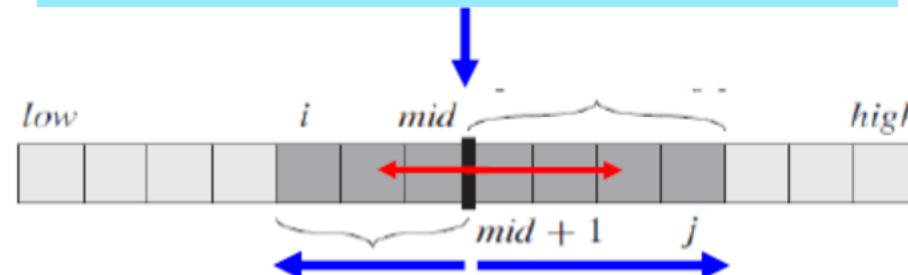
# 最大子数组问题



## □ 求解过程分析：

- 1 ) 对于完全位于 $A[low..mid]$  和  $A[mid+1..high]$  中的最大子数组，可以在这两个较小的子数组上用递归的方法进行求解。
- 2 ) 怎么寻找跨越中点的最大子数组呢？

这样的子数组必然跨越中点  $mid$



从  $mid$  出发，分别向左和向右找出和最大的子区间，然后合并这两个区间即可得到跨越中点时的  $A[low..high]$  的最大子数组。



# 最大子数组问题

□ 以下2个过程用于求解最大子数组问题：

✓ 过程1：FIND-MAX-CROSSING-SUBARRAY，求跨越中点的最大子数组

FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )

```
1  left-sum = -∞  
2  sum = 0  
3  for i = mid downto low  
4      sum = sum + A[i]  
5      if sum > left-sum  
6          left-sum = sum  
7          max-left = i  
8  right-sum = -∞  
9  sum = 0  
10 for j = mid + 1 to high  
11     sum = sum + A[j]  
12     if sum > right-sum  
13         right-sum = sum  
14         max-right = j  
15 return (max-left, max-right, left-sum + right-sum)
```

搜索从  $mid$  开始向左的半个区间，  
找出左边最大的连续子数组的和

同理，搜索从  $mid+1$  开始向右的半个区间，找出右边最大的连续子数组的和

返回搜索的结果

✓ 时间复杂度： $(mid-low+1)+(high-mid)=high-low+1=n$

2021/10/21

14

Soochow University



# 最大子数组问题

- ✓ 过程2：FIND-MAXIMUM-SUBARRAY, 求最大子数组问题的分治算法

FIND-MAXIMUM-SUBARRAY( $A, low, high$ )

```
1 if  $high == low$ 
2     return ( $low, high, A[low]$ )           // base case: only one element
3 else  $mid = \lfloor (low + high)/2 \rfloor$ 
4     ( $left-low, left-high, left-sum$ ) =
5         FIND-MAXIMUM-SUBARRAY( $A, low, mid$ )
6     ( $right-low, right-high, right-sum$ ) =
7         FIND-MAXIMUM-SUBARRAY( $A, mid + 1, high$ )
8     ( $cross-low, cross-high, cross-sum$ ) =
9         FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )
10    if  $left-sum \geq right-sum$  and  $left-sum \geq cross-sum$ 
11        return ( $left-low, left-high, left-sum$ )
12    elseif  $right-sum \geq left-sum$  and  $right-sum \geq cross-sum$ 
13        return ( $right-low, right-high, right-sum$ )
14    else return ( $cross-low, cross-high, cross-sum$ )
```

求  $A[low..mid]$  的最大子数组

求  $A[mid+1..high]$  的最大子数组

求跨越中点的最大子数组

返回三个最大子子数组中的  
大者作为问题的解



# 最大子数组问题

## □ FIND-MAXIMUM-SUBARRAY的时间分析证明

令  $T(n)$  表示求解  $n$  个元素的最大子数组问题的执行时间

- 1 ) 当  $n = 1$  时 ,  $T(1) = \Theta(1)$  ;
- 2 ) 当  $n > 1$  时 , 对  $A[low..mid]$  和  $A[mid+1..high]$  两个子问题递归求解 , 每个子问题的规模是  $n/2$  , 所以每个子问题的时间为  $T(n/2)$  , 两个子问题递归求解的总时间是  $2T(n/2)$ 。
- 3 ) FIND-MAX-CROSSING-SUBARRAY 的时间是  $\Theta(n)$ 。

## □ 算法FIND-MAXIMUM-SUBARRAY执行时间 $T(n)$ 的递归式 为 :

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

2021/10/21                          16


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



# 第三讲 分治策略

## 内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法



# Strassen矩阵乘法



## 回顾一下矩阵运算

已知两个n阶方阵:  $A = (a_{ij})_{nxn}$ ,  $B = (b_{ij})_{nxn}$

### 1) 矩阵加法

$$C = A + B = (c_{ij})_{nxn}, \text{ 其中, } c_{ij} = a_{ij} + b_{ij}, i, j = 1, 2, \dots, n$$

时间复杂度:  $\Theta(n^2)$

### 2) 矩阵乘法

$$C = AB = (c_{ij})_{nxn}, \text{ 其中, } c_{ij} = \sum_{1 \leq k \leq n} a_{ik} b_{kj}, i, j = 1, 2, \dots, n$$

时间复杂度:  $\Theta(n^3)$ 。

共有 $n^2$ 个 $c_{ij}$ 需要计算, 每个 $c_{ij}$ 需要 $n$ 次乘运算,  $n-1$ 次加法



# Strassen矩阵乘法

## □ 朴素的矩阵乘法：

已知两个  $n$  阶方阵： $A = (a_{ij})_{n \times n}$ ,  $B = (b_{ij})_{n \times n}$ , 定义乘积  $C = A \cdot B$  中的元素：

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

实现两个  $n \times n$  矩阵乘的过程：

SQUARE-MATRIX-MULTIPLY( $A, B$ )

```
1   $n = A.\text{rows}$ 
2  let  $C$  be a new  $n \times n$  matrix
3  for  $i = 1$  to  $n$ 
4      for  $j = 1$  to  $n$ 
5           $c_{ij} = 0$ 
6          for  $k = 1$  to  $n$ 
7               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return  $C$ 
```

朴素的矩阵乘的计算  
时间是  $\Theta(n^3)$





# Strassen矩阵乘法

## □ 基于分治策略的矩阵乘算法：

设  $n=2^k$ ，两个  $n$  阶方阵为  $A=(a_{ij})_{n \times n}$ ,  $B=(b_{ij})_{n \times n}$

N: 若  $n \neq 2^k$ ，可通过在 A 和 B 中补0使之变成阶是2的幂的方阵

- ✓ 首先，将 A、B 和 C 分成4个  $n/2 \times n/2$  的子矩阵：

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- ✓ 可以将公式  $C=AB$  改写为：

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad \longleftrightarrow$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \end{aligned}$$

共有：8次 $(n/2) \times (n/2)$  矩阵乘  
4次 $(n/2) \times (n/2)$  矩阵加



# Strassen矩阵乘法

## ➤ SQUARE-MATRIX-MULTIPLY-RECURSIVE , 求矩阵乘的分治算法

SQUARE-MATRIX-MULTIPLY-RECURSIVE( $A, B$ )

```
1   $n = A.\text{rows}$ 
2  let  $C$  be a new  $n \times n$  matrix
3  if  $n == 1$ 
4       $c_{11} = a_{11} \cdot b_{11}$ 
5  else partition  $A, B$ , and  $C$  as in equations (4.9)
6       $C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21})$ 
7       $C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22})$ 
8       $C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21})$ 
9       $C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})$ 
10 return  $C$ 
```

$\Theta(n^2)$  ← + T(n/2)

复制矩阵，花费 $\Theta(n^2)$

使用下标，花费 $\Theta(1)$

N: 任意两个子矩阵块的乘可以沿用同样的规则：如果子矩阵的阶大于2，则将子矩阵分成更小的子矩阵，直到每个子矩阵只含一个元素为止。从而构造出一个递归计算过程。



# Strassen矩阵乘法



## □ **SQUARE-MATRIX-MULTIPLY-RECURSIVE的时间分析证明**

令  $T(n)$  表示两个  $n \times n$  矩阵相乘的计算时间

1 ) 当  $n = 1$  时 ,  $T(1) = \Theta(1)$  ;

2 ) 当  $n > 1$  时 , 8次  $n/2 \times n/2$  矩阵乘 , 花费时间为  $8T(n/2)$  ,  
4次  $n/2 \times n/2$  矩阵加 , 花费  $\Theta(n^2)$  时间。

## □ **算法SQUARE-MATRIX-MULTIPLY-RECURSIVE执行时间 $T(n)$ 的递归式为 :**

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

$\longrightarrow T(n) = \Theta(n^3)$



# Strassen矩阵乘法

## □ Strassen方法核心思想：

- 令递归树稍微不那么茂盛，即只递归进行 7 次而不是 8 个  $n/2 \times n/2$  矩阵乘

## □ Strassen方法具体步骤：

- 1 ) 按上述方法将输入矩阵 A、B 和输出矩阵 C 分解为  $n/2 \times n/2$  的子矩阵。采用下标计算方法，此步骤花费  $\Theta(1)$  时间。
- 2 ) 创建 10 个  $n/2 \times n/2$  的矩阵  $S_1, S_2, \dots, S_{10}$ ，每个矩阵保存步骤 1 ) 中创建的两个矩阵的和或差。花费时间为  $\Theta(n^2)$ 。
- 3 ) 用步骤 1 ) 中创建的子矩阵和步骤 2 ) 中创建的 10 个矩阵，递归地计算 7 个矩阵积  $P_1, P_2, \dots, P_7$ 。每个矩阵  $P_i$  都是  $n/2 \times n/2$ 。
- 4 ) 通过  $P_i$  矩阵的不同组合进行加减运算，计算出结果矩阵 C 的子矩阵  $C_{11}, C_{12}, C_{21}, C_{22}$ 。花费时间为  $\Theta(n^2)$ 。



# Strassen矩阵乘法

## □ Strassen方法细节：

$$S_1 = B_{12} - B_{22},$$

$$S_2 = A_{11} + A_{12},$$

$$S_3 = A_{21} + A_{22},$$

$$S_4 = B_{21} - B_{11},$$

$$S_5 = A_{11} + A_{22},$$

$$S_6 = B_{11} + B_{22},$$

$$S_7 = A_{12} - A_{22},$$

$$S_8 = B_{21} + B_{22},$$

$$S_9 = A_{11} - A_{21},$$

$$S_{10} = B_{11} + B_{12}.$$

$$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22},$$

$$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22},$$

$$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11},$$

$$P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11},$$

$$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22},$$

$$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22},$$

$$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}.$$

7次 $(n/2) \times (n/2)$  矩阵相乘

10次 $(n/2) \times (n/2)$  矩阵加减法

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

8次 $(n/2) \times (n/2)$  矩阵加减法



# Strassen矩阵乘法



## □ Strassen方法的时间分析证明

令  $T(n)$  表示两个  $n \times n$  矩阵的Strassen矩阵乘所需的计算

1 ) 当  $n = 1$  时 ,  $T(1) = \Theta(1)$  ;

2 ) 当  $n > 1$  时 , 7次  $(n/2) \times (n/2)$  矩阵乘 , 花费时间为  $7T(n/2)$  , 18次  $(n/2) \times (n/2)$  矩阵加减 , 花费  $\Theta(n^2)$  时间。

## □ Strassen方法执行时间 $T(n)$ 的递归式为 :

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases} \quad \xrightarrow{\hspace{1cm}} \quad T(n) = \Theta(n^{\lg 7}) \\ \approx \Theta(n^{2.81})$$



# 第三讲 分治策略

## 内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法
  - ✓ 代入法
  - ✓ 递归树法
  - ✓ 主方法



# 递归式求解方法

## □ 分治算法的计算时间表达式往往是递归式

- ✓ 如归并排序、最大子数组的分治算法运行时间 $T(n)$ 的递归式为：

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

边界条件

- ✓ Strassen矩阵乘法的运行时间 $T(n)$ 的递归式为：

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

递归方程

## □ 那么如何化简递归式，以得到形式简单的限界函数？

- ✓ 代换法
- ✓ 递归树法
- ✓ 主方法

N：递归式求解的结果是得到形式简单的渐近限界函数表示（即用O、 $\Omega$ 、 $\Theta$ 表示的函数式）



# 递归式求解方法

## □ 预处理——对表达式细节的简化

为便于处理，通常做如下假设和简化处理：

- (1) 运行时间函数  $T(n)$  的定义中，一般假定自变量为正整数；因为这样的  $n$  通常表示数据的个数。
- (2) 忽略递归式的边界条件，即  $n$  较小时函数值的表示；原因在于虽然递归式的解会随着  $T(1)$  值的改变而改变，但此改变不会超过常数因子，对函数的阶没有根本影响。
- (3) 对上取整、下取整运算做合理简化，如：

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + f(n)$$

通常忽略上、下取整函数，就可写作以下简单形式：

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



# 代入法

## □ 代换法求解递归式要点：

1. 猜测解的形式。
2. 用数学归纳法找出使解真正有效的常数，并证明解是正确的。

例： $T(n)=2T(\lfloor n/2 \rfloor)+n$

解：(1) 猜测其解为  $T(n)=O(n\lg n)$

(2) 证明选择常数  $c > 0$ ，可有  $T(n) \leq cn\lg n$ 。

假定此上界对所有正数  $m < n$  都成立的，对于  $m = \lfloor n/2 \rfloor$ ，有  
 $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$ 。将其带入递归式，得到

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

其中，只要  $c \geq 1$ ，最后一步都会成立。



- 上面的过程证明了当  $n$  足够大时猜测的正确性，但对**边界值** 是否成立呢？  
即：  $T(n) \leq cn\log n$  的结论对于较小的  $n$  成立吗？
- 分析：事实上，对  $n=1$ ，上述结论存在问题：
  - (1) 作为边界条件，我们有理由假设  $T(1)=1$ ；
  - (2) 但对  $n=1$ ，带入表达式有： $T(1) \leq c \cdot 1 \cdot \log 1 = 0$ ，与  $T(1)=1$  不相符。
- 归纳证明的基础不成立，怎么处理？
- **从  $n_0$  的性质出发**：只需要存在常数  $n_0$ ，使得  $n \geq n_0$  时结论成立即可，所以  $n_0$  不一定取 1。



所以，这里不取 $n_0=1$ ，而取 $n_0=2$ ，用 $T(2)$ 、 $T(3)$ 代替 $T(1)$ 作为归纳证明中的边界条件：

(1) 依然合理地假设 $T(1)=1$ 。

(2) 研究什么样的 $c$ 使得 $T(2)$ 、 $T(3)$ 可以满足 $T(n) \leq cn\log n$ 。

(即使得  $T(2) \leq 2c\log 2$  且  $T(3) \leq 3c\log 3$  成立)

- 将 $T(1)=1$ 带入递归式，有： $T(2)=4$ ， $T(3)=5$
- 要使 $T(2) \leq 2c\log 2$  和 $T(3) \leq 3c\log 3$  成立，只要 $c \geq 2$ 即可。
- 综上所述，取常数 $c \geq 2$ ，结论 $T(n) \leq cn\log n$ 成立。

命题得证。



# 代入法

## □ 应用数学归纳法要求解对边界条件成立：

- ✓ 一般通过证明边界条件符合归纳证明的基本情况。但可能出现**归纳证明基本情况不能满足**的问题。
- ✓ 解决办法：**扩展边界条件**。渐近界只要求  $n \geq n_0$  即可，故可以选择适当的  $n_0$ ，让  $T(n_0)$  代替作为归纳证明中的边界条件，使递归假设对很小的  $n$  也成立。



## 如何猜测递归式的解呢？

- 遗憾的是，并不存在通用的方法来猜测递归式的正确解。
- 1 ) 主要靠经验
- ◆ 尝试1：看有没有形式上类似的表达式，以此推测新递归式解的形式。
- ◆ 尝试2：先猜测一个较宽的界，然后再缩小不确定范围，逐步收缩到精确的渐近界。
- ◆ 避免盲目推测： $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ ，猜测  $T(n) = O(n)$   
$$T(n) \leq c\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 = cn + 1$$

原因：并未证出一般形式  $T(n) \leq cn$  成立。 ( $cn + 1 < cn$ )



## ➤ 必要的时候要做一些技术处理

### ➤ 1 ) 去掉一个低阶项

- 新的猜测是  $T(n) \leq cn - d$  ,  $d$ 是一个大于0的一个常数
- $T(n) \leq c \left( \left\lfloor \frac{n}{2} \right\rfloor - d \right) + c \left( \left\lceil \frac{n}{2} \right\rceil - d \right) + 1 = cn - 2d + 1 \leq cn - d$
- 只要  $d \geq 1$ ,此事就成立

### ➤ 2 ) 变量代换：对陌生的递归式做些简单的代数变换，使之变成较熟悉的形式。



例：设有递归式  $T(n) \leq 2T(\lfloor \sqrt{n} \rfloor) + \log n$

分析：原始形态比较复杂

(1) 做代数代换：令  $m = \log n$ ，则  $n = 2^m$ ,  $\sqrt{n} = 2^{m/2}$

(2) 忽略下取整，直接使用  $\sqrt{n}$  代替  $\lfloor \sqrt{n} \rfloor$

得：

$$T(2^m) \leq 2T(2^{m/2}) + m$$

再设  $S(m) = T(2^m)$ ，得以下形式递归式：

$$S(m) \leq 2S(m/2) + m$$

从而获得形式上熟悉的递归式。

根据前面的一些讨论，可得新的递归式的上界是：

$$O(m \log m)$$

再将  $S(m)$ 、 $m = \log n$  带回  $T(n)$ ，有，

$$\begin{aligned} T(n) &= T(2^m) \\ &= S(m) = O(m \log m) \\ &= O(\log n \log \log n) \end{aligned}$$

这里， $m = \log n$



# 代入法

## □ 代换法求解步骤小结：

- ✓ 猜测递归式没有通用的方法
- ✓ 需要经验、创新性
- ✓ 试探法
- ✓ 递归树
- ✓ 证明较宽松的上下界，然后再缩小不确定性区间

## □ 注意事项：

- ✓ 对更小的值做更强的假设
- ✓ 避免陷阱
- ✓ 适当时进行变量替换



# 递归树法

- 画递归树有助于猜想递归式的解
- 当用递归式表示分治算法的运行时间时，递归树的方法尤其有用
- 递归树中，每一个节点都代表着递归函数调用集合中一个子问题的代价。将树中每一层内的代价相加得到一个每层代价的集合，再将每层的代价相加得到递归式所有层次的总代价
- 但递归树法不够严谨，使用递归树产生好的猜测时，通常需要容忍小量的“不良量”
  - ✓ floor, ceiling忽略
  - ✓  $n$  经常假设为某个整数的幂次方
- 通常作法：用递归树法生成好的猜测，然后可用代入法验证猜测是否正确



# 递归树法

## 基于递归树的时间分析

- **节点代价**：在递归树中，每个节点有求解相应（子）问题的代价(cost，这里指除递归以外的其它代价)。
- **层代价**：每一层各节点代价的和。
- **总代价**：整棵树的各层代价之和
- **目 标**：利用树的性质，获取对递归式解的猜测，然后用代换法或其它方法加以验证。



## 口例：求解 $T(n) = 3T(\lfloor n/4 \rfloor) + \theta(n^2)$

**准备性工作**：为简单起见，对一些细节做必要、合理的简化和假设，这里为：

### (1) 去掉底函数的表示

- 理由：底函数和顶函数对递归式求解并不“重要”。

### (2) 假设n是4的幂，即 $n=4^k$ , $k=\log_4 n$ 。

- 一般，当证明 $n=4^k$ 成立后，再加以适当推广，就可以把结论推广到n不是4的幂的一般情况了。

### (3) 展开 $\theta(n^2)$ ，代表递归式中非重要项。

- 假设其常系数为 $c$ ,  $c>0$ ，从而去掉 $\theta$ 符号，转变成 $cn^2$ 的形式，便于后续的公式化简。

**最终得以下形式的递归式：**  $T(n) = 3T(n/4) + cn^2$

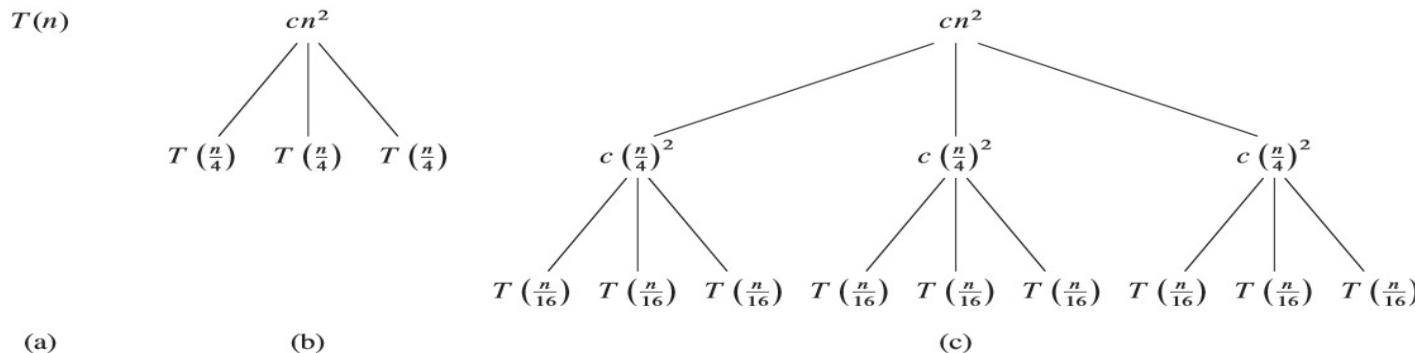


# 递归树法

□ 例：求解  $T(n) = 3T(n/4) + cn^2$

关键：1 ) 树的深度如何确定？

2 ) 每个节点的代价—>每层的代价—>总代价



- a) 对原始问题  $T(n)$  的描述。
- b) 第一层递归调用的分解情况， $cn^2$  是顶层计算除递归以外的代价， $T(n/4)$  是分解出来的规模为  $n/4$  的子问题的代价，总代价  $T(n) = 3T(n/4) + cn^2$ 。
- c) 第二层递归调用的分解情况。 $c(n/4)^2$  是三棵二级子树除递归以外的代价。



# 递归树法

例：求解  $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$

1) 树的深度：子问题的规模按 $1/4$ 的方式减小，在递归树中深度为 $i$  的节点子问题的大小为 $n/4^i$ 。

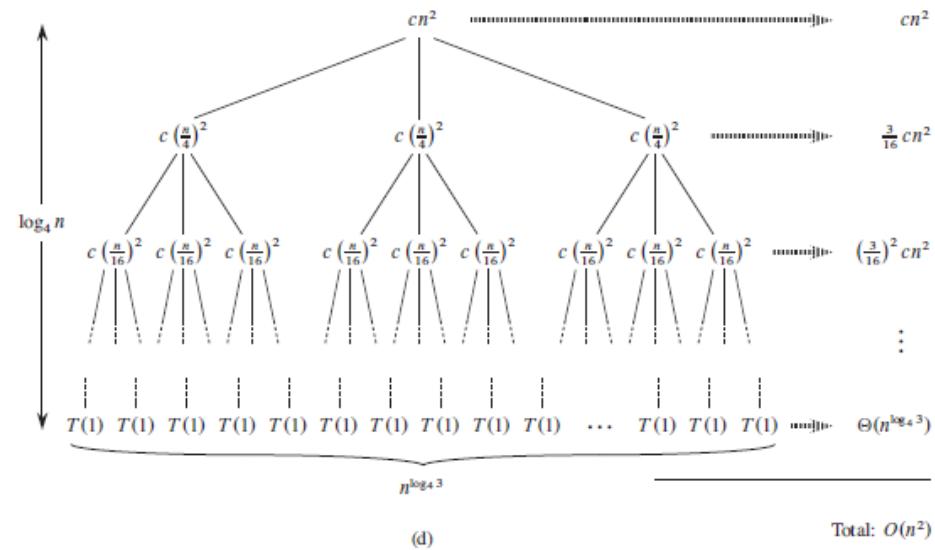
➤ 当 $n/4^i=1$ 时，子问题规模仅为1，达到边界值。

➤ 所以，

- 节点分布层： $0 \sim \log_4 n$
- 树共有 $\log_4 n + 1$ 层
- 从第2层起，每一层上的节点数为上层节点数的3倍
- 深度为 $i$  的层节点数为 $3^i$ 。

2) 代价计算

- ✓ 深度为 $i$ 的每个结点的代价为： $c(n/4)^2$
- ✓ 对 $i=0, 1, 2, \dots, \log_4 n - 1$ ，深度为 $i$ 的所有结点的总代价为： $3^i c(n/4)^2 = (3/16)^i cn^2$
- ✓ 最底层结点为 $n^{\log_4 3}$ ，每个结点代价为 $T(1)$ ，总代价为 $n^{\log_4 3} T(1)$ ，即 $\Theta(n^{\log_4 3})$





# 递归树法

口 例：求解  $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$

所有层次的代价之和，即整棵树的代价：

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

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

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

无限递减几何级数

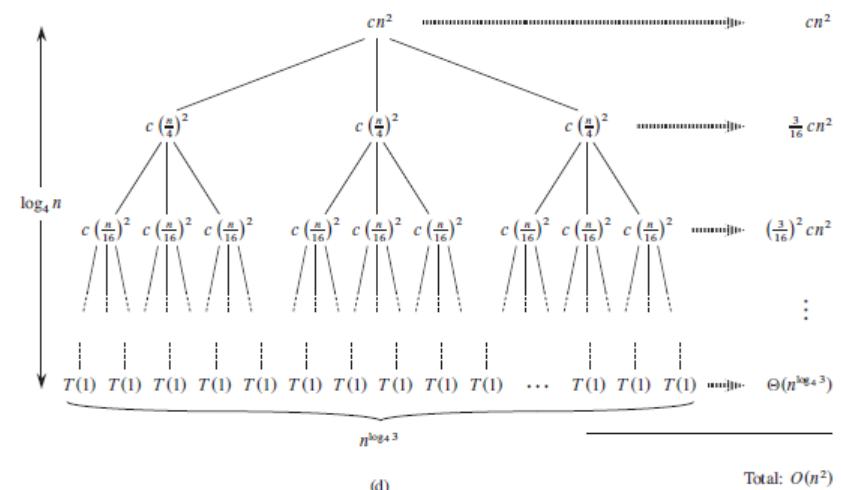
$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) < \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

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

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

$$= O(n^2)$$

至此，获得  $T(n)$  解的一个猜测：  $T(n)=O(n^2)$ ，成立吗？



- 对于实数  $x \neq 1$ ，和式  $\sum_{k=0}^n x^k = 1 + x + x^2 + \dots + x^n$  是一个几何级数（等比数列），其值为  $\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$
- 当和是无穷的且  $|x| < 1$  时，得到无穷递减几何级数，此时

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



## 用代换法证明猜测的正确：

- 将  $T(n) \leq dn^2$  作为归纳假设， $d$ 是待确定的常数，带入推论证明 过程，有

$$\begin{aligned} T(n) &= 3T(\lfloor n/4 \rfloor) + \Theta(n^2) \\ &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \quad c \text{是引入的另一个常量} \end{aligned}$$

显然，要使得  $T(n) \leq dn^2$  成立，只要  $d \geq (16/13)c$  即可。所以， $T(n) \leq dn^2$  的猜测成立

定理得证（边界条件的讨论略）。

另：  $O(n^2)$  是  $T(n)$  的一个紧确界，为什么？

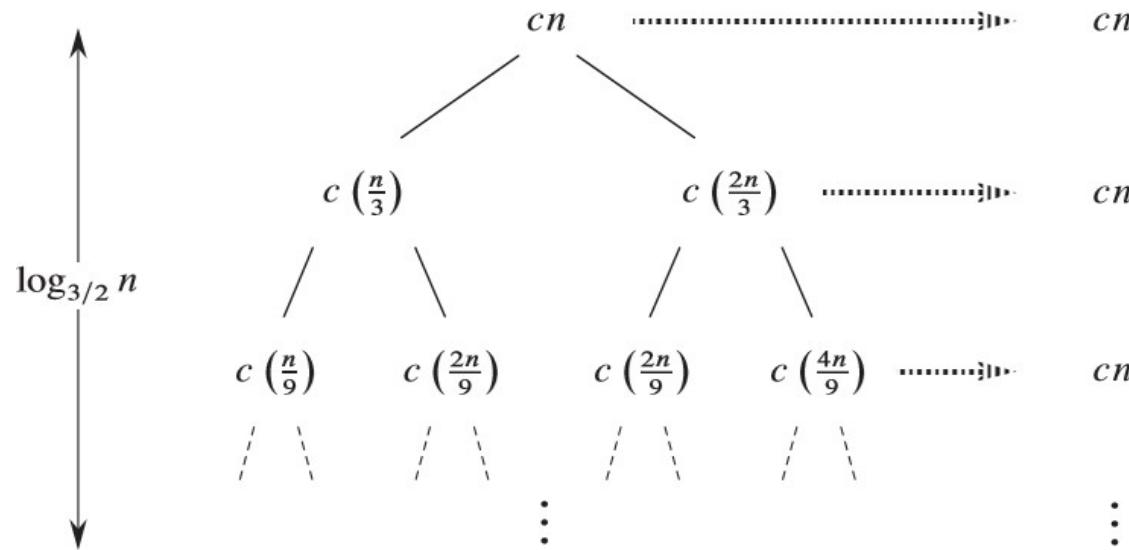


例 求表达式  $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$  , ( 这里 , 表达式中直接省略了下取整和上取整函数 ) :

进一步地 , 引入常数  $c$ , 展开  $O(n)$  , 得

$$T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$

递归树为 :



Total:  $O(n \lg n)$



## 分析：

### ■ 该树并不是一个完全的二叉树。

- 从根往下，越来越多的内节点在左侧消失(1/3分叉上)，因此每层的代价并不都是 $cn$ ，而是 $\leq cn$ 的某个值。

### ■ 树的深度：

- 在上述形态中，最长的路径是最右侧路径，由

$n \rightarrow (2/3)n \rightarrow (2/3)^2n \rightarrow \dots \rightarrow 1$  组成。

- 当 $k = \log_{3/2} n$ 时， $(3/2)^k/n = 1$ ，所以树的深度为 $\log_{3/2} n$ 。

### ■ 递归式解的猜测：

- 至此，我们可以合理地猜测该树的总代价至多是层数乘以 每层的代价，并鉴于上面关于层代价的讨论，我们可以假设递归式的上界为：

$$O(cn \log_{3/2} n) = O(n \log n)$$

注：这里，我们假设每层的代价为 $cn$ 。事实上， $cn$ 为每层代价的上界，这一假设是合理的细节简化处理。



## 猜测的证明：证明 $O(n \log n)$ 是递归式的上界

即证明： $T(n) \leq d n \log n$ ， $d$ 是待确定的合适正常数。

$$\begin{aligned} T(n) &\leq T(n/3) + T(2n/3) + cn \\ &\leq d(n/3) \log(n/3) + d(2n/3) \log(2n/3) + cn \\ &= (d(n/3) \log n - d(n/3) \log 3) + (d(2n/3) \log n - d(2n/3) \log 3/2) + cn \\ &= dn \log n - d((n/3) \log 3 + (2n/3) \log(3/2)) + cn \\ &= dn \log n - d((n/3) \log 3 + (2n/3) \log 3 - (2n/3) \log 2) + cn \\ &= dn \log n - dn(\log 3 - 2/3) + cn \leq dn \log n \end{aligned}$$

成立吗？

上式成立的条件是： $d \geq c / (\log 3 - (2/3))$

所以猜测正确，递归式的解得证。



# 主方法

- 主方法是求解如下形式的递归式

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

其中  $a \geq 1$  和  $b > 1$  是常数， $f(n)$  是一个渐近正的函数。

- 主方法要求针对三种情况，但这样很容易确定许多递归式的解，甚至可以不需要计算。
- 在上面的递归式中，因为  $n/b$  可能不是整数，可以用  $\lceil n/b \rceil$  和  $\lfloor n/b \rfloor$  来代替  $n/b$ 。这种代替不会对递归式的渐近行为产生影响。
- 实际上，在分析分治算法的运行时间时，经常略去下取整和上取整函数，以方便对递归式的分析。



# 主方法

□ 主方法的正确性依赖于如下的主定理

## 定理 4.1 (主定理)

假设  $a \geq 1$  和  $b > 1$  为常数， $f(n)$  为一函数， $T(n)$  是定义在非负整数上的递归式： $T(n) = aT(n/b) + f(n)$ ，其中将  $n/b$  解释为  $\lceil n/b \rceil$  和  $\lfloor n/b \rfloor$ 。那么  $T(n)$  可能有如下的渐近界：

1. 若对某个常数  $\varepsilon > 0$  有  $f(n) = O(n^{\log_b a - \varepsilon})$ ，则  $T(n) = \Theta(n^{\log_b a})$ 。
2. 若  $f(n) = O(n^{\log_b a})$ ，则  $T(n) = \Theta(n^{\log_b a} \lg n)$ 。
3. 若对某个常数  $\varepsilon > 0$  有  $f(n) = \Omega(n^{\log_b a + \varepsilon})$ ，且对某个常数  $c > 1$  和所有足够大的  $n$  有  $af(n/b) \leq cf(n)$ ，则  $T(n) = \Theta(f(n))$ 。



# 主方法

$$T(n) = a T(n/b) + f(n),$$

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \lg n) & \text{if } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ and } af(n/b) \leq c f(n) \end{cases} \quad \left. \right\} \exists \varepsilon > 0, c > 1$$

口将函数  $f(n)$  与函数  $n^{\log_b a}$  进行比较，两个函数较大者决定了递归式的解：

- ✓ 若函数  $n^{\log_b a}$  更大，如情况1，则  $T(n)=\Theta(n^{\log_b a})$ ；
- ✓ 若函数  $f(n)$  更大，如情况3，则  $T(n)=\Theta(f(n))$ ；
- ✓ 若两个函数大小相当，如情况2，则  $T(n)=\Theta(n^{\log_b a} \lg n)$ 。

口使用主方法技术细节：

- ✓ 情况1， $f(n)$  要多项式意义上小于  $n^{\log_b a}$
- ✓ 情况3， $f(n)$  要多项式意义上大于  $n^{\log_b a}$ ，且满足  $af(n/b) \leq c f(n)$



口上面三种情况并未覆盖所有可能的 $f(n)$ ，即case1&2和case2&3之间的缝隙。

口若递归式中的 $f(n)$ 与  $n^{\log_b a}$  的关系不满足上述性质：

- ◆  $f(n)$  小于等于  $n^{\log_b a}$ ，但不是多项式地小于。
- ◆  $f(n)$  大于等于  $n^{\log_b a}$ ，但不是多项式地大于。

口情况3中要求的正则条件不成立

则不能用主方法求解该递归式。



# 主方法



## 口 使用主方法举例

例1 :  $T(n)=9T(n/3)+n$

解 :  $a = 9, b = 3, f(n) = n \Rightarrow n^{\log_b a} = n^{\log_3 9} = n^2 = \Theta(n^2)$   
 $\Rightarrow f(n) = O(n^{\log_3 9 - \varepsilon}), \text{ where } \varepsilon = 1 \Rightarrow T(n) = \Theta(n^2)$

例2 :  $T(n)=T(2n/3)+1$

解 :  $a = 1, b = 3/2, f(n) = 1 \Rightarrow n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$   
 $\Rightarrow f(n) = \Theta(n^{\log_{3/2} 1}) = \Theta(1) \Rightarrow T(n) = \Theta(\lg n)$

例3 :  $T(n)=3T(n/4)+n\lg n$

解 :  $a = 3, b = 4, f(n) = n \lg n \Rightarrow n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$   
 $\Rightarrow f(n) = \Omega(n^{\log_4 3 + \varepsilon}), \text{ where } \varepsilon \approx 0.2,$   
 $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n) \text{ for } c = 3/4$



# 主方法

## 口 使用主方法举例

例4 :  $T(n)=2T(n/2)+n\lg n$

解 :  $a = 2, b = 2, f(n) = n \lg n \Rightarrow n^{\log_b a} = n$

上述情况可能错误的应用情况3，因为  $f(n)=n\lg n$  渐近大于  $n^{\log_b a}=n$ 。但是它并不是多项式意义上的大于，对任意正常数 $\varepsilon$ ，比值  $f(n)/n^{\log_b a}=(n\lg n)/n$  都渐近小于  $n^\varepsilon$ 。因此，递归式落入了情况2和情况3的间隙。





- 归并算法的其它经典应用
- 第7章 快速排序
- 第9章 中位数和顺序统计量
- 第33章 寻找最近点对
- 第30章 多项式与快速傅立叶变换



# 寻找最近点对



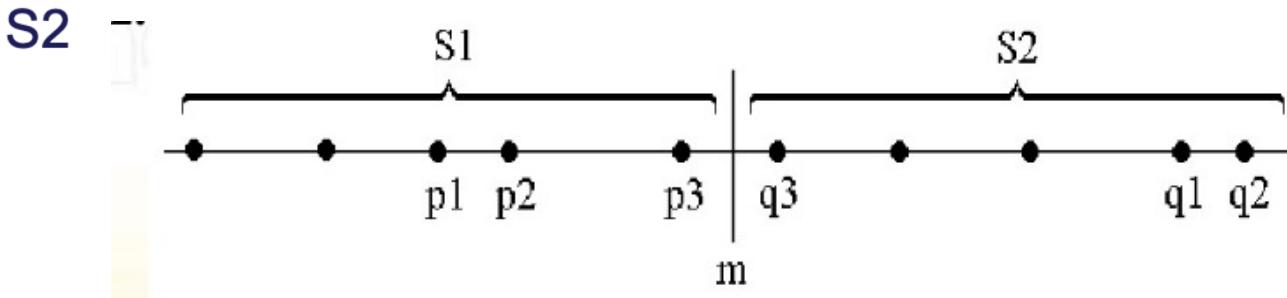
- Closest pair. 给定平面中的  $n$  个点，找到它们之间具有最小欧几里得距离的一对。
  - 基本几何图元。
    - ✓ 图形、计算机视觉、地理信息系统、分子建模、空中交通管制。
    - ✓ 最近邻的特例，欧几里得 MST，Voronoi。
      - ↑ *fast closest pair inspired fast algorithms for these problems*
  - Brute force. Check all pairs of points  $p$  and  $q$  with  $\Theta(n^2)$  comparisons.
  - 1-D version.  $O(n \log n)$  easy if points are on a line.
  - Assumption. No two points have same  $x$  coordinate.
    - ↑ *to make presentation cleaner*



# 1-D Closest Pair



- ❑ 分治法
- ❑ Divide the points S into two sets S1, S2 by some x-coordinate so that  $p < q$  for all  $p \in S1$ , and  $q \in S2$ .
- ❑ Recursively compute closest pair ( $p_1, p_2$ ) in  $S1$  and ( $q_1, q_2$ ) in  $S2$

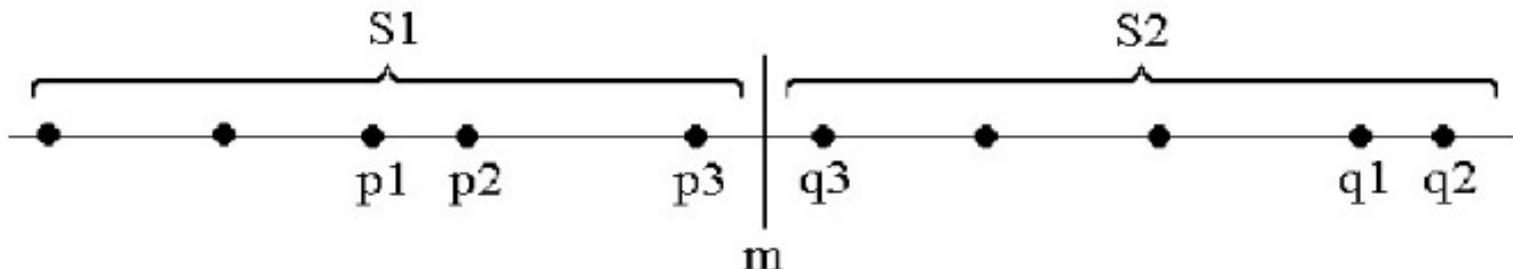


- ❑ Let  $d$  be the smallest separation found so far:
  - ✓  $d = \min(|p_2-p_1|, |q_2-q_1|)$



# 1-D Closest Pair

[



- The closest pair is  $(p_1, p_2)$ , or  $(q_1, q_2)$ , or some  $(p_3, q_3)$  where  $p_3 \in S_1$  and  $q_3 \in S_2$
- **Key Observation:** if  $(p_3, q_3)$  is the closest pair and  $m$  is the dividing coordinate, then  $p_3, q_3$  must be within  $d$  of  $m$  (**why?**)
  - ✓ How many points of  $S_1$  can lie in the interval  $(m-d, m]$ ?
  - ✓ By definition of  $d$ , at most one.
  - ✓ Same holds for  $S_2$ .
  - ✓ Then,  $P_3$  must be the rightmost point of  $S_1$  and  $q_3$  the leftmost point of  $S_2$



# 1-D Closest Pair



Closest\_Pair\_1D (S[l ... r])

//Input: A subarray S[l ... r] of a given array S[0 ... n-1] of real numbers

//Output: The distance between the closest pair of numbers

if  $i = l$  return  $\infty$

else if  $r - l = 1$  return  $S[r] - S[l]$

else

$M \leftarrow$  the median of S

$k \leftarrow \text{partition}(S, l, r, M)$

$p \leftarrow \max(S[l \dots k]); q \leftarrow \min(S[k+1 \dots r])$

return  $\min\{\text{Cloest_Pair_1D}(S[l \dots k],$

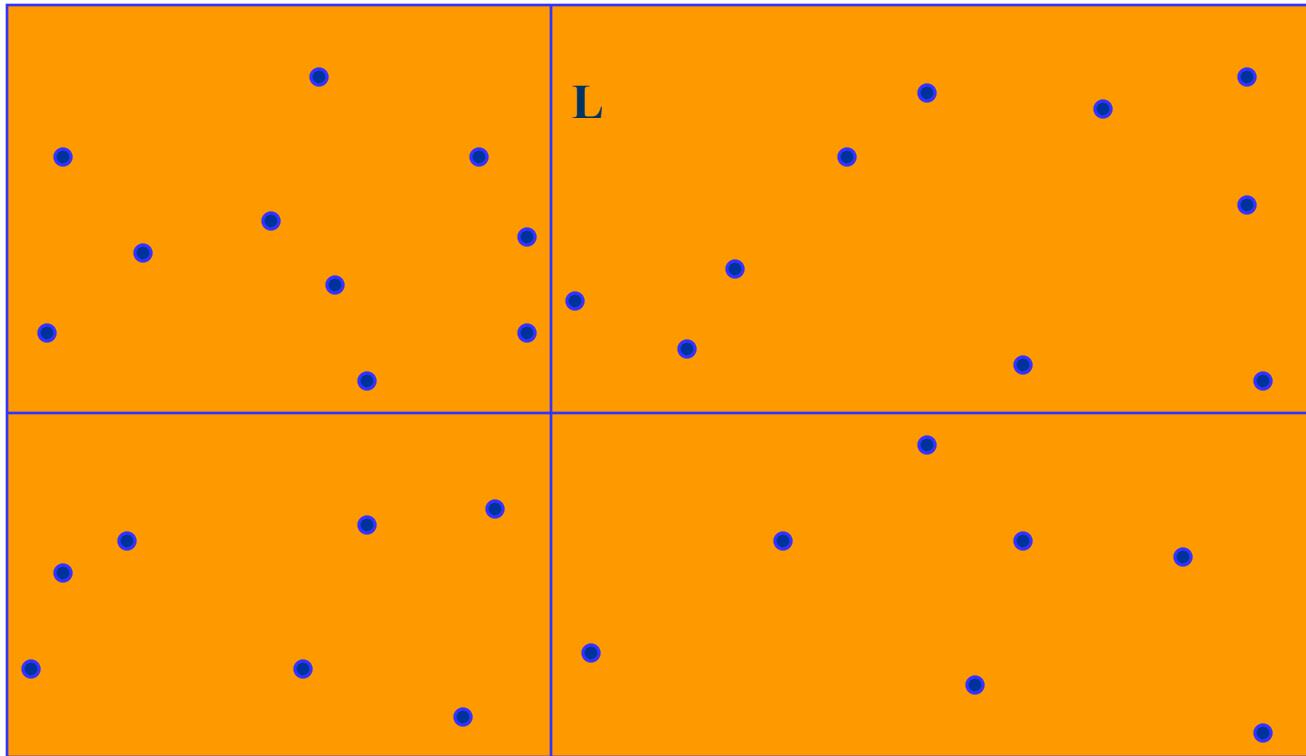
Cloest\_Pair\_1D(S[k+1 \dots r]), q-p\}

- ❑ Recurrence is  $T(n) = 2T(n/2) + O(n)$ , which solves to  $T(n) = O(n \log n)$



# 2D Closest Pair of Points: First Attempt

- Divide. Sub-divide region into 4 quadrants.



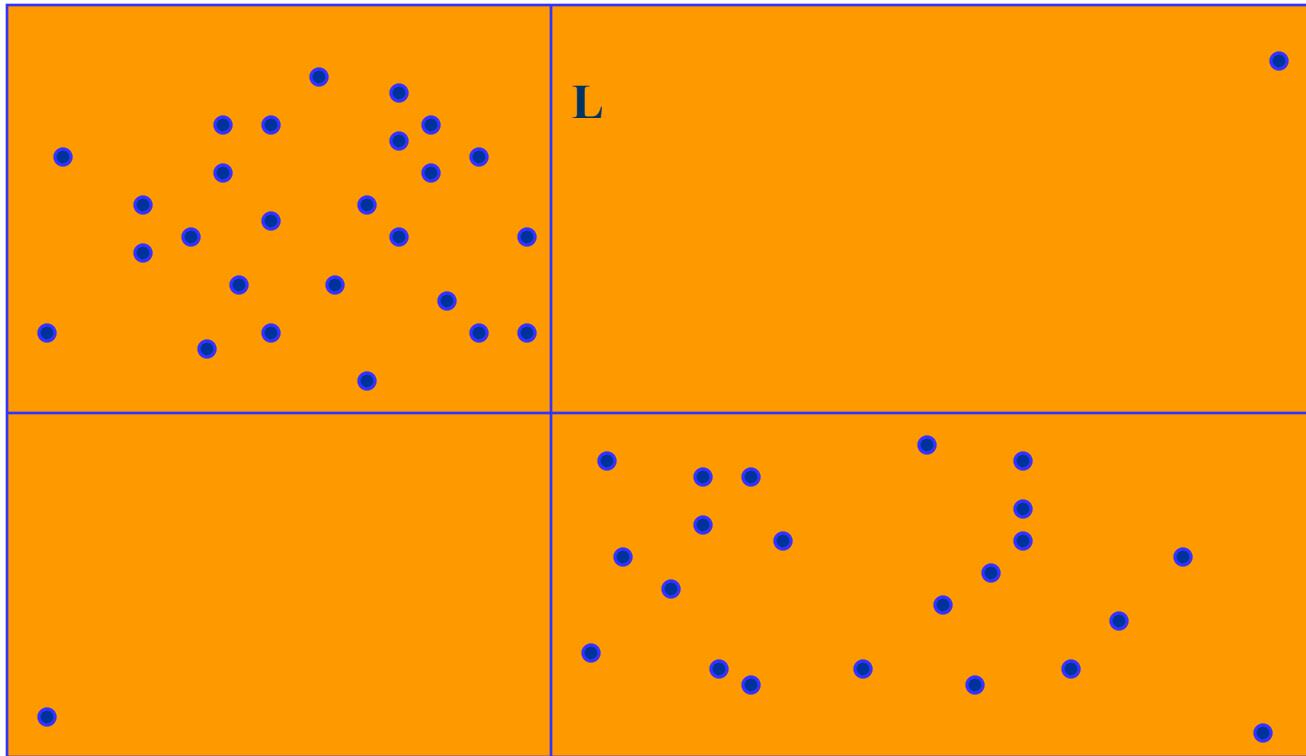


# 2D Closest Pair of Points: First Attempt



59

- ❑ Divide. Sub-divide region into 4 quadrants.
- ❑ Obstacle. Impossible to ensure  $n/4$  points in each piece.



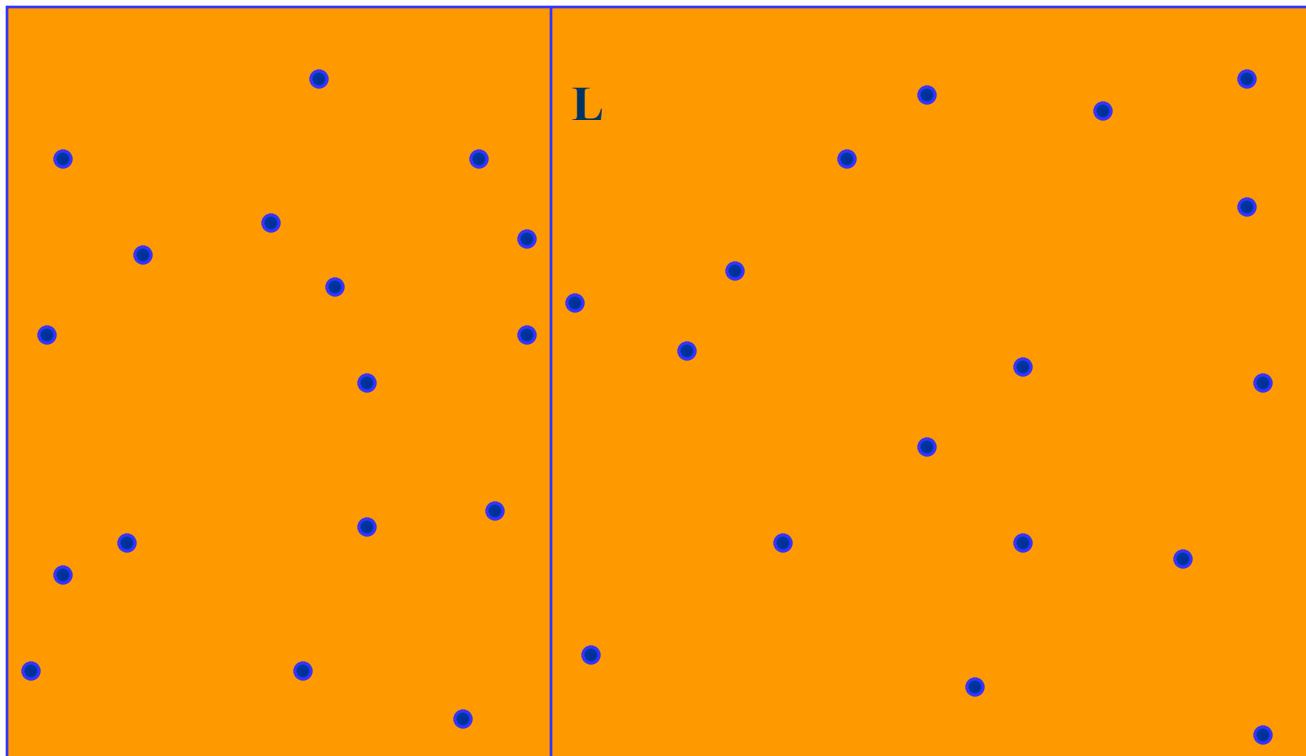


# 2D Closest Pair of Points



## Algorithm.

- ✓ Divide: draw vertical line L so that roughly  $\frac{1}{2}n$  points on each side.



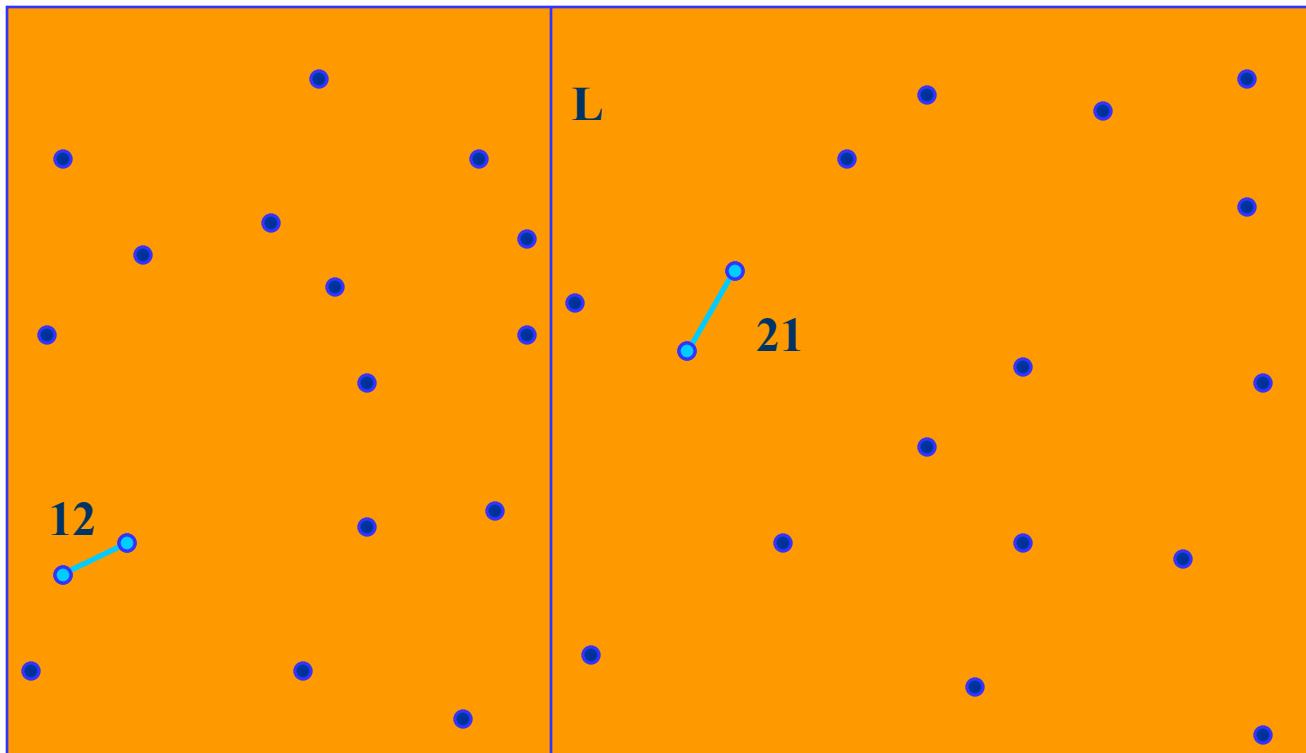


# 2D Closest Pair of Points



## Algorithm.

- ✓ Divide: draw vertical line L so that roughly  $\frac{1}{2}n$  points on each side.
- ✓ Conquer: find closest pair in each side recursively.



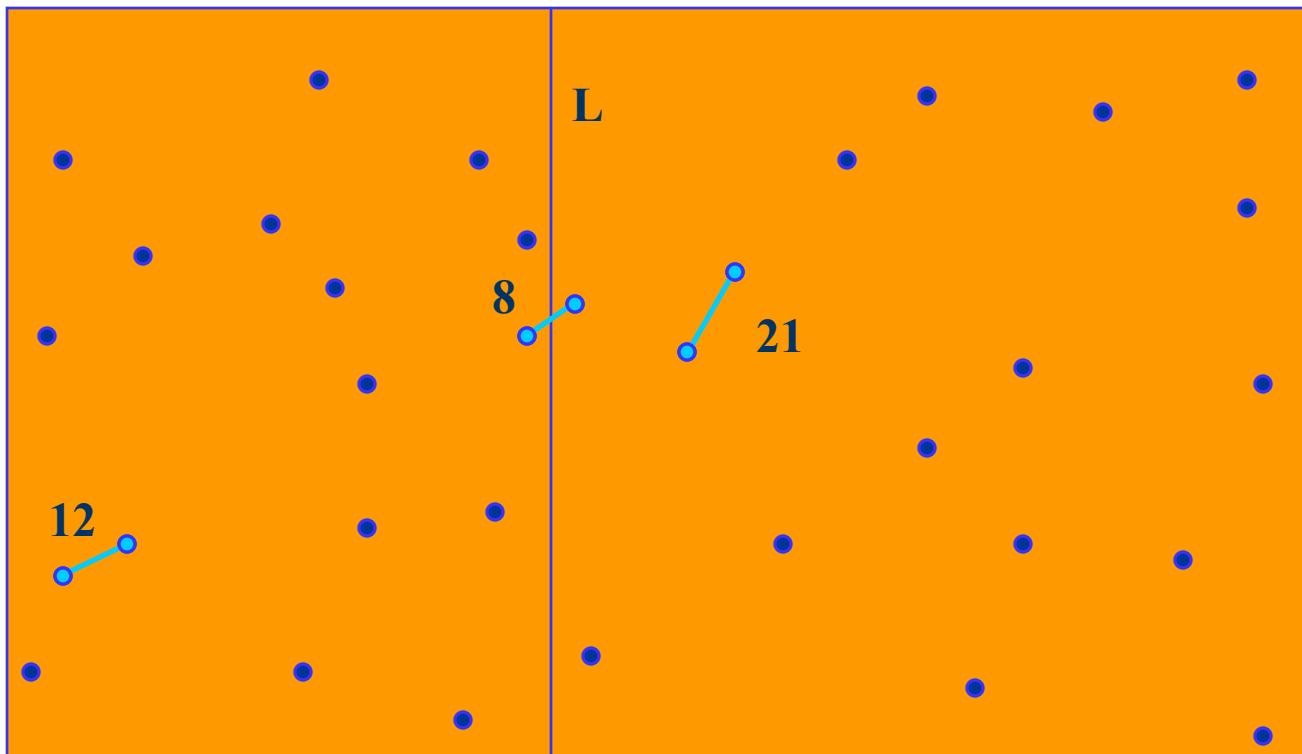


# 2D Closest Pair of Points



## Algorithm.

- ✓ Divide: draw vertical line L so that roughly  $\frac{1}{2}n$  points on each side.
- ✓ Conquer: find closest pair in each side recursively. ← seems like  $\Theta(n^2)$
- ✓ Combine: find closest pair with one point in each side.
- ✓ Return best of 3 solutions.

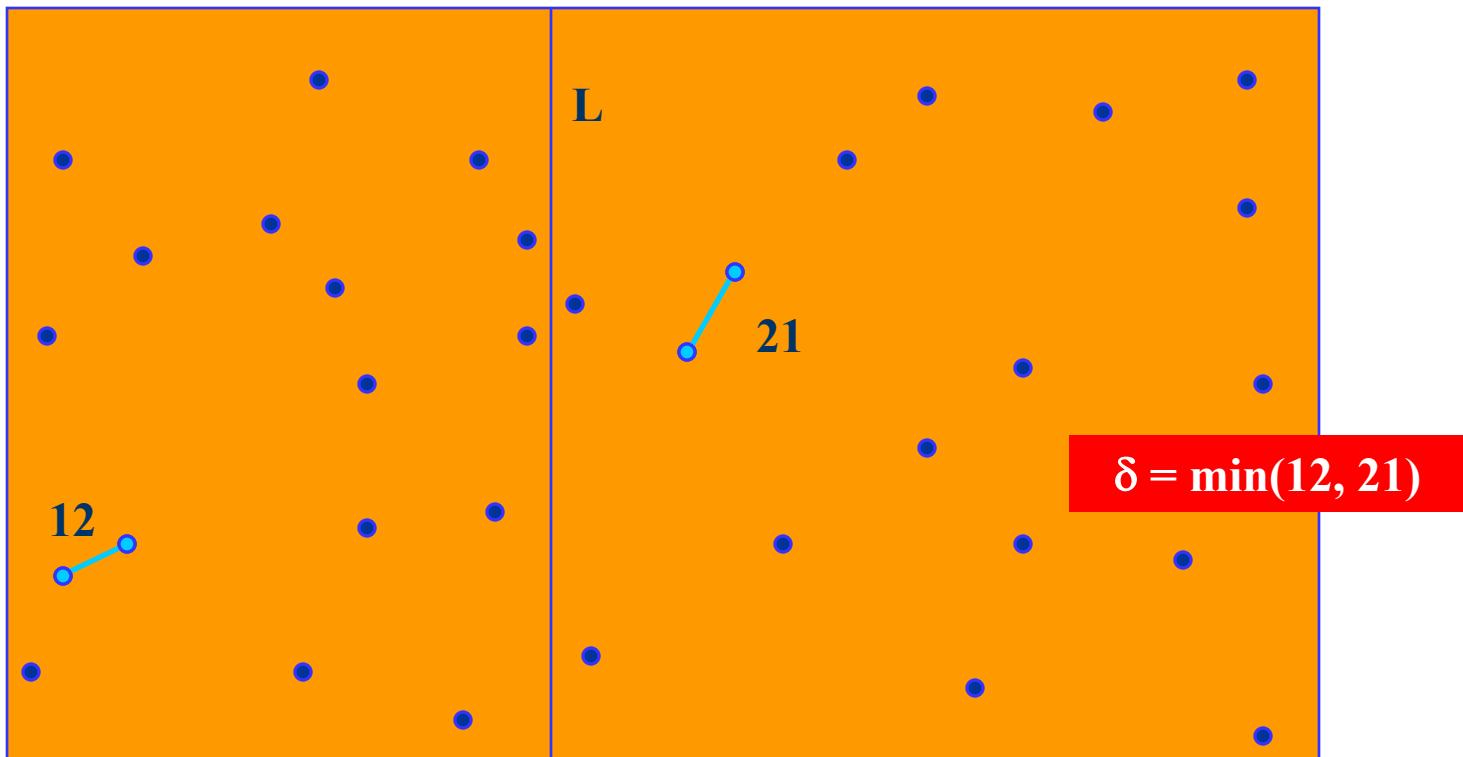




# 2D Closest Pair of Points



- Find closest pair with one point in each side, assuming that distance  $< \delta$ .

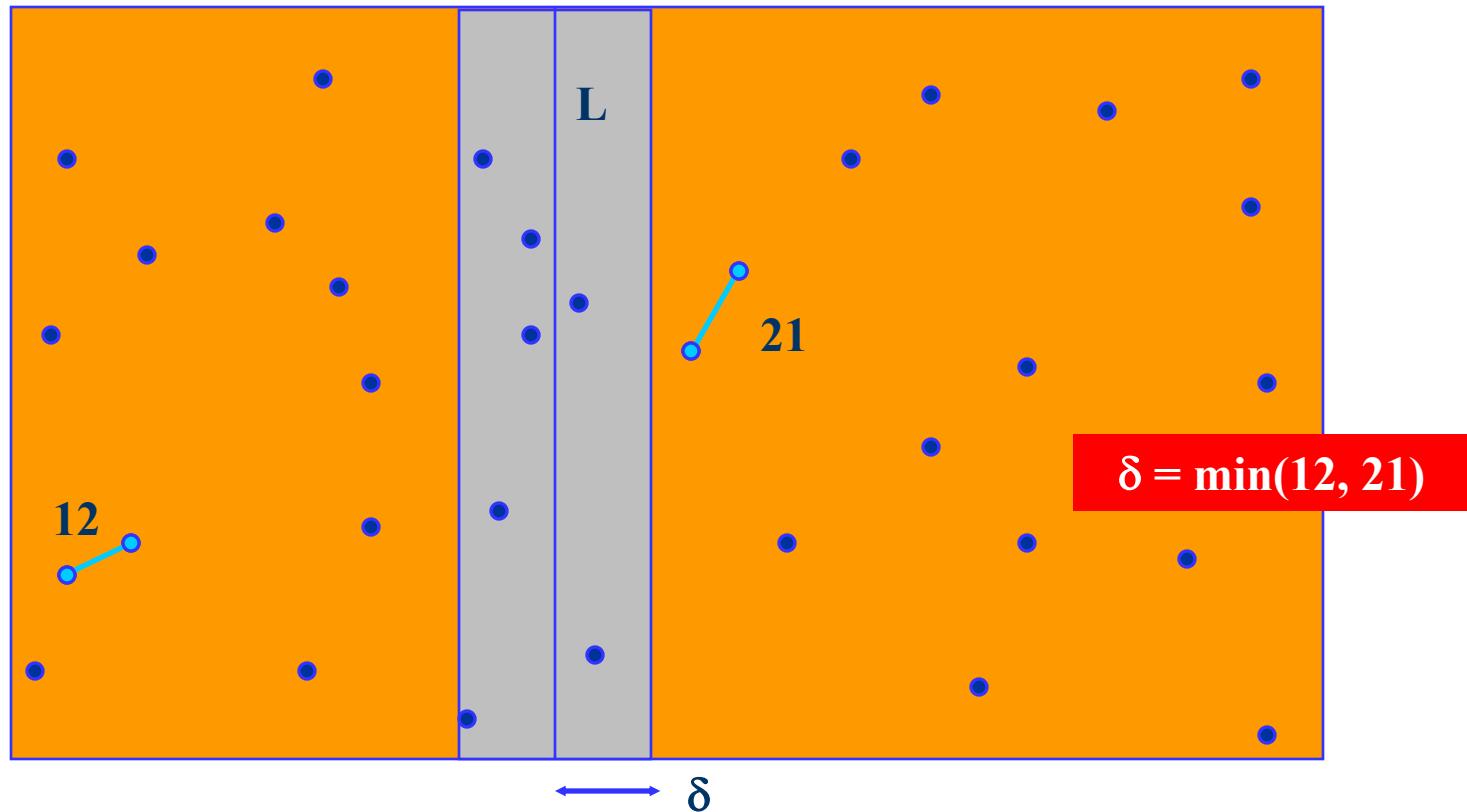




# 2D Closest Pair of Points



- ❑ Find closest pair with one point in each side, **assuming that distance  $< \delta$ .**
  - ✓ Observation: only need to consider points within  $\delta$  of line L.



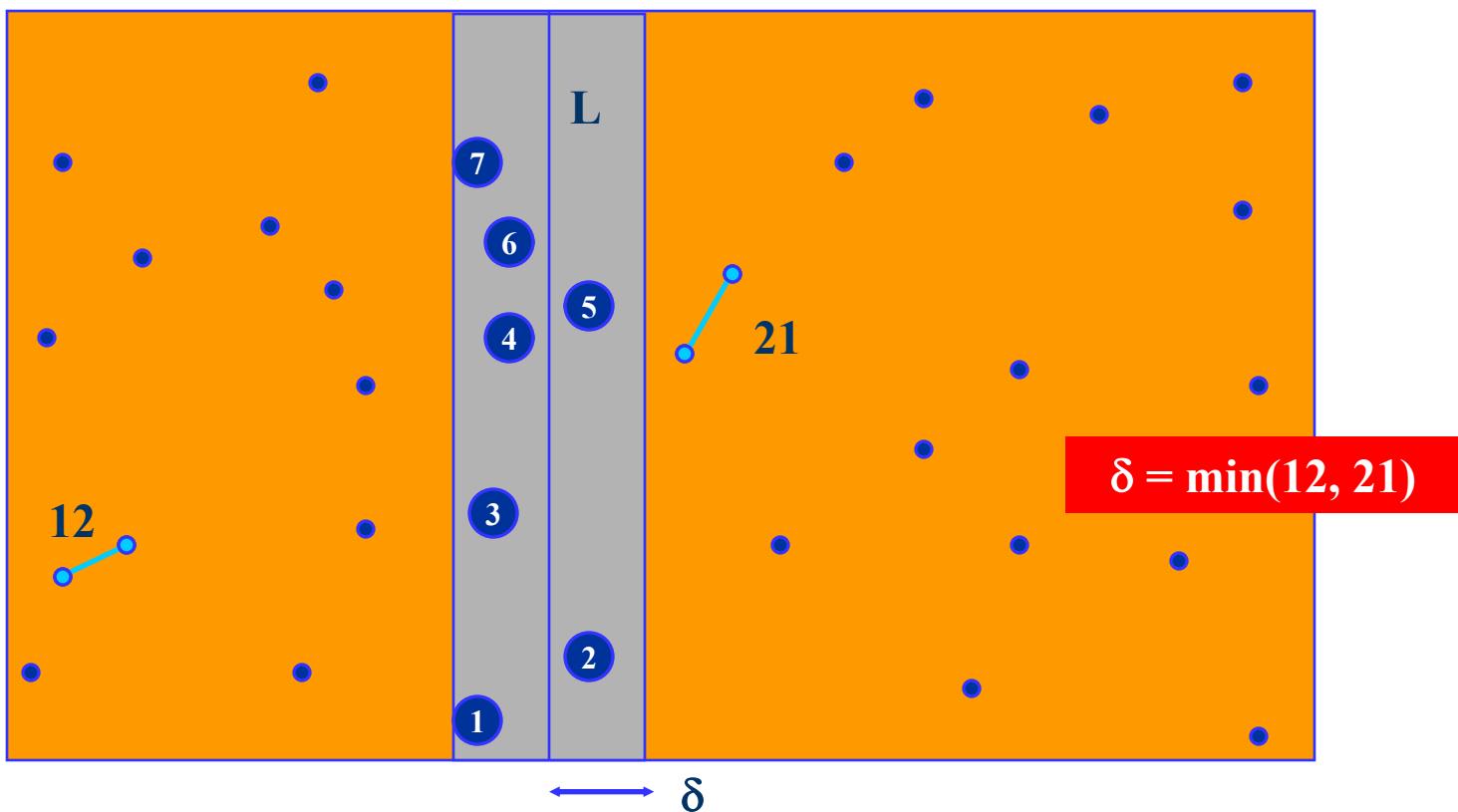


# 2D Closest Pair of Points



□ Find closest pair with one point in each side, **assuming that distance  $< \delta$ .**

- ✓ Observation: only need to consider points within  $\delta$  of line L.
- ✓ Sort points in  $2\delta$ -strip by their y coordinate.



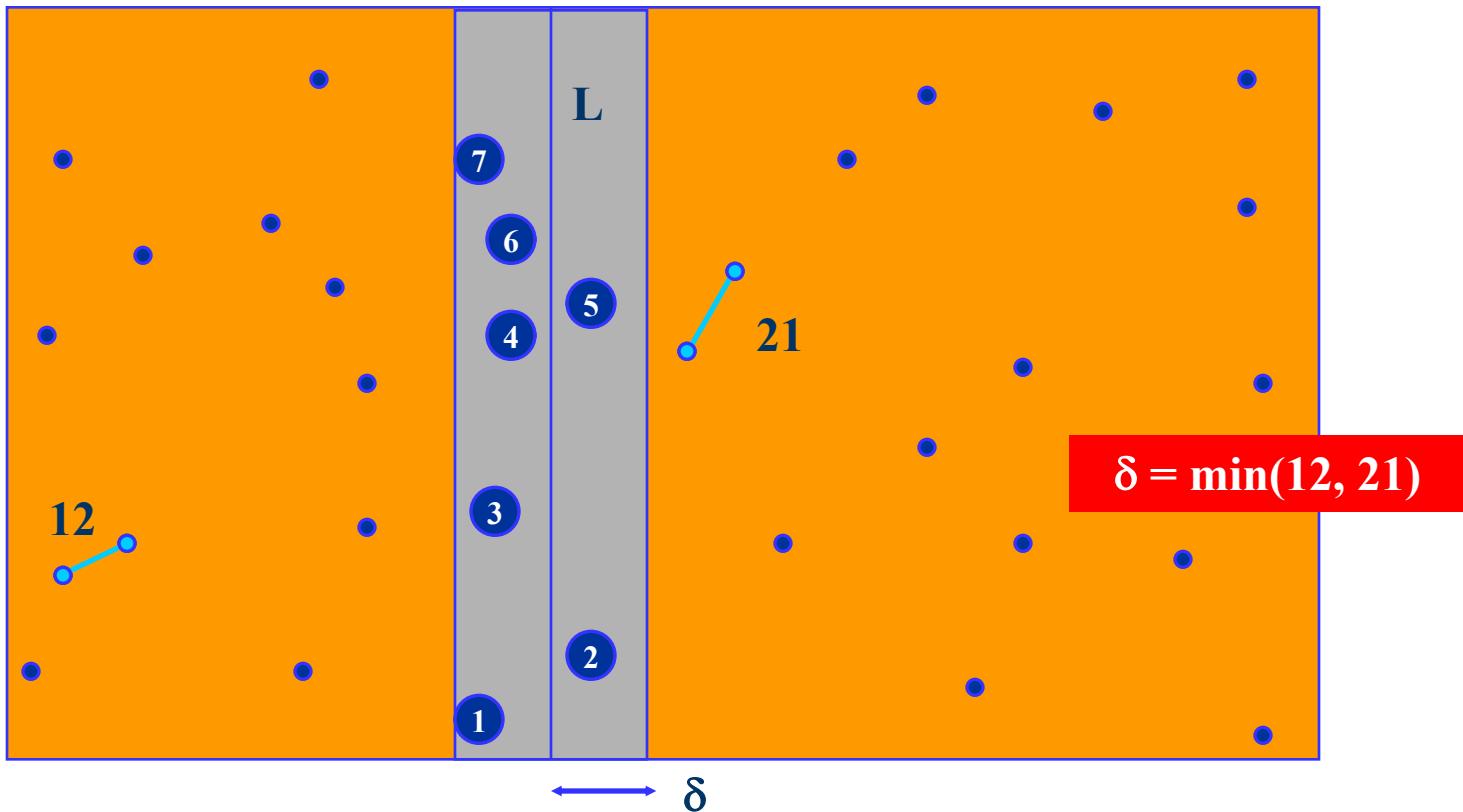


# 2D Closest Pair of Points

□ Find closest pair with one point in each side, assuming that

distance  $< \delta$ .

- ✓ Observation: only need to consider points within  $\delta$  of line L.
- ✓ Sort points in  $2\delta$ -strip by their y coordinate.
- ✓ For each point in P1, only check distances of those within 6 positions in sorted list!

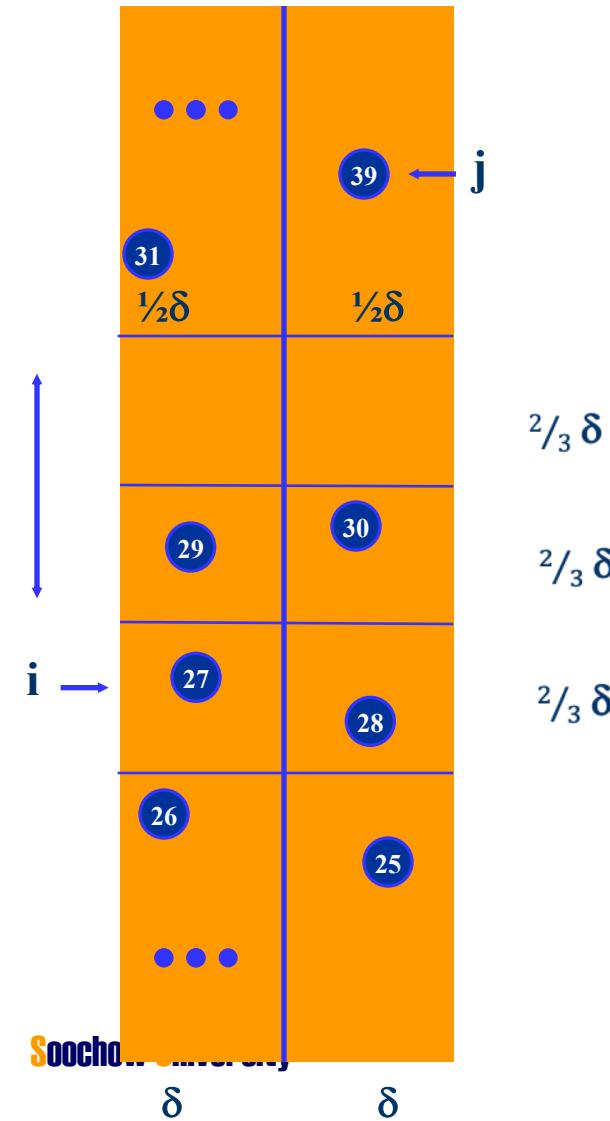




# 2D Closest Pair of Points



- Def. Let  $s_i$  be the point in the  $2\delta$ -strip, with the  $i^{\text{th}}$  smallest  $y$ -coordinate.
- Claim. If  $|i - j| \geq 6$ , then the distance between  $s_i$  and  $s_j$  is at least  $\delta$ .
- Pf.
  - ✓ No two points lie in same  $\frac{1}{2}\delta$ -by- $\frac{2}{3}\delta$  box.

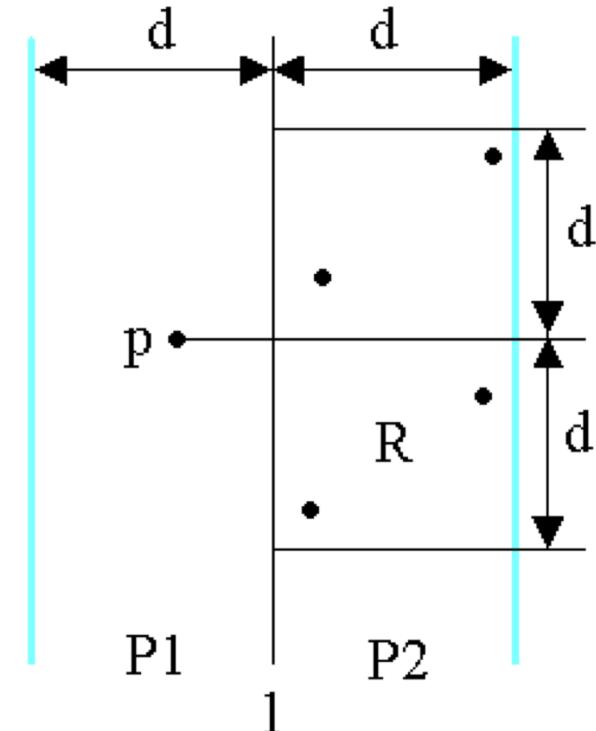




# 2D Closest Pair of Points



- 考虑P1中任意点p它若与P2中的点q构成最接近点对的候选者则必有 $\text{distance}(p,q) < d$ 。满足这个条件的P2中的点一定路在一个 $d^*2d$ 的矩形R中
- P2中任何2个S中的点的距离都不小于d。由此可以推出矩形R中最少只有6个S中的点。





# 2D Closest Pair of Points

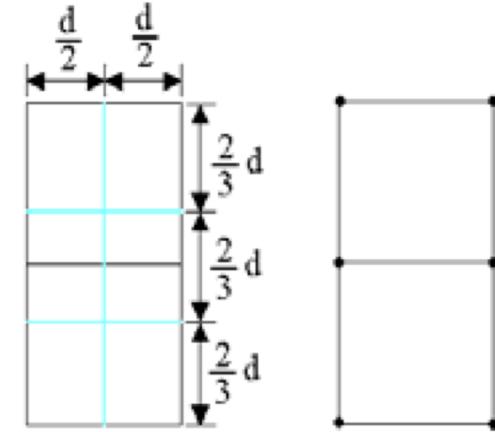


□ 证明:将矩形R的长为 $2d$ 的边3等分, 将它的长为 $d$ 的边2等分, 由此导出6个 $(d/2) \times (2d/3)$ 的矩形。

若矩形R中有多于6个S中的点, 则由鸽舍原理易知至少有一个 $(d/2) \times (2d/3)$ 的小矩形中有2个以上S中的点。

□ 设 $u, v$ 是位于同一小矩形中的2个点, 则令他们之间距离表示为  
:  $\text{distance}(u, v)$ , 计算公式如下

$$\begin{aligned} & (x(u) - x(v))^2 + (y(u) - y(v))^2 \\ & \leq \left(\frac{d}{2}\right)^2 + \left(\frac{2d}{3}\right)^2 = \frac{25}{36}d^2 \end{aligned}$$



(a)

(b)



# 2D Closest Pair Algorithm

```
Closest-Pair(p1, ..., pn) {
```

**Compute** separation line L such that half the points  
are on one side and half on the other side.

**O(n log n)**

```
    δ1 = Closest-Pair(left half)
```

**2T(n / 2)**

```
    δ2 = Closest-Pair(right half)
```

```
    δ = min(δ1, δ2)
```

**Delete** all points further than δ from separation line L

**O(n)**

**Sort** remaining points by y-coordinate.

**O(n log n)**

**Scan** points in y-order and compare distance between  
each point and next 6 neighbors. If any of these  
distances is less than δ, update δ.

**O(n)**

```
    return δ.
```

```
}
```





## ❑ Running time.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

❑ Q. Can we achieve  $O(n \log n)$ ?

❑ A. Yes. Don't sort points in strip from scratch each time.

- ✓ Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- ✓ Sort by **merging** two pre-sorted lists.

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$



# Closest Pair Algorithm



```
Closest-Pair(P, X, Y) {
```

Compute  $P_{left}$  和  $P_{right}$ , 根据  $X$  的中位数;

$O(n)$

Compute  $X_{left}$  和  $X_{right}$ , 根据  $P_{left}$  和  $P_{right}$ ;

$O(n)$

Compute  $Y_{left}$  和  $Y_{right}$ , 根据  $P_{left}$  和  $P_{right}$ ;

$O(n)$

```
     $\delta_1 = \text{Closest-Pair}(P_{left}, X_{left}, Y_{left})$ 
```

$2T(n/2)$

```
     $\delta_2 = \text{Closest-Pair}(P_{right}, X_{right}, Y_{right})$ 
```

```
     $\delta = \min(\delta_1, \delta_2)$ 
```

Delete all points further than  $\delta$  from separation line L

$O(n)$

remaining points Sorted by y-coordinate.

Scan  $X_{left}$  points in y-order and compare distance between each point and next 6 neighbors in  $X_{right}$ . If any of these distances is less than  $\delta$ , update  $\delta$ .

$O(n)$

```
return  $\delta$ .
```

```
}
```



# 谢谢 !

## Q & A

作业 : 4.2-1 4.2-2

4.3-6

4.4-4

4.5-3

写2个快排版本比较之