



HITWH  
SE

## 第三章

# 排序与分治算法



**HITWH  
SE**

# 参考材料

**《Introduction to Algorithm》**

**Chapter 6, 7, 8, 9**

**《Introduction to the Design and  
Analysis of Algorithm》**

**Chapter 4**



- 3.1 分治算法的原理
- 3.2 基于分治思想的排序算法
- 3.3 Medians and Order Statistics
- 3.4 最邻近点对
- 3.5 线性时间排序算法
- 3.6 凸包问题
- 3.7 FFT
- 3.8 整数乘法



## 3.1 Divide-and-Conquer 原理

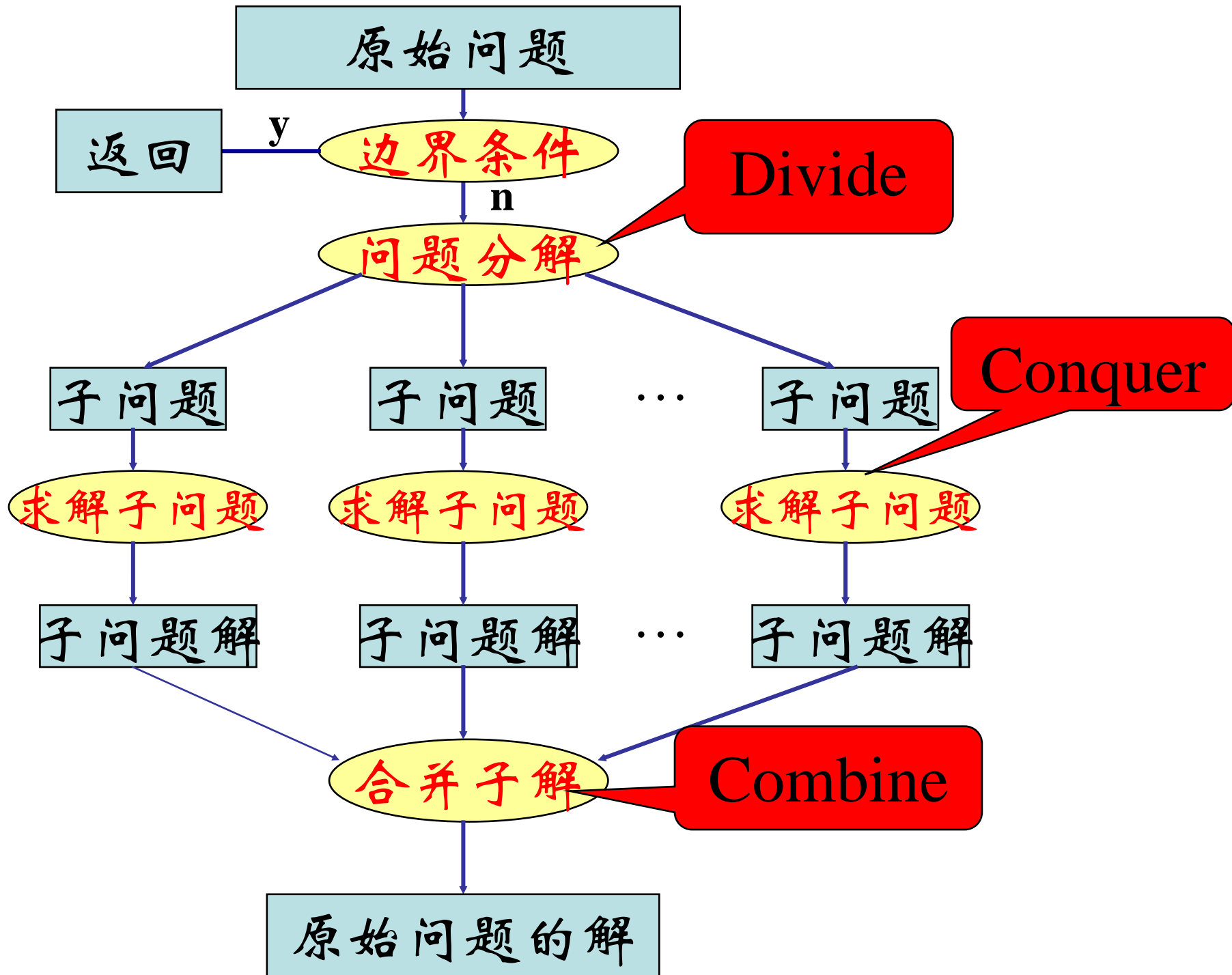
- Divide-and-Conquer 算法的设计
- Divide-and-Conquer 算法的分析



# Divide-and-Conquer算法的设计



- 设计过程分为三个阶段
  - Divide: 整个问题划分为多个子问题
    - 注意: 分解的这组子问题 $p_1, p_2, \dots, p_m$ 未必一定是相同的子问题, 即 $p_i$ 和 $p_j$ 可以是分别完成不同任务的子问题
  - Conquer: 求解各子问题(递归调用正设计的算法)
  - Combine: 合并子问题的解, 形成原始问题的解





# Divide-and-Conquer 算法的分析



- 分析过程
  - 建立递归方程
  - 求解
- 递归方程的建立方法
  - 设输入大小为  $n$ ,  $T(n)$  为时间复杂性
  - 当  $n < c$ ,  $T(n) = \theta(1)$



## – Divide阶段的时间复杂性

- 划分问题为 $a$ 个子问题。
- 每个子问题大小为 $n/b$ 。
- 划分时间可直接得到= $D(n)$

## – Conquer阶段的时间复杂性

- 递归调用
- Conquer时间= $aT(n/b)$

## – Combine阶段的时间复杂性

- 时间可以直接得到= $C(n)$

最后得到递归方程：

- $T(n) = \theta(1)$  if  $n \leq c$
- $T(n) = aT(n/b) + D(n) + C(n)$  if  $n > c$





HITWH  
SE

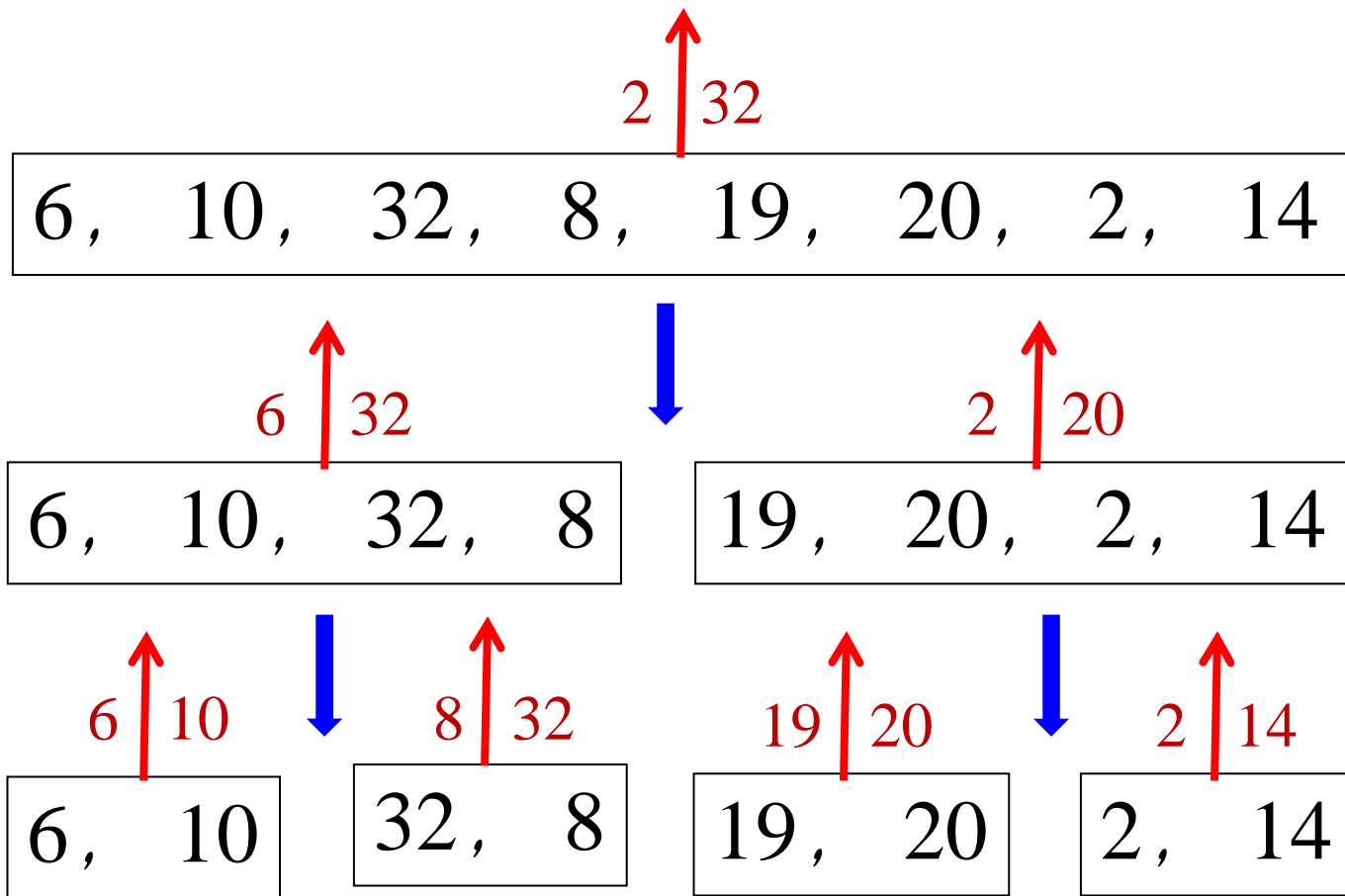
# 举例：最大最小值问题

输入：数组  $A[1, \dots, n]$

输出：A 中的 max 和 min

通常，直接扫描需要  $2n-2$  次比较操作

我们给出一个仅需  $3n/2-2$  次比较操作的算法





## 算法MaxMin(A)

输入: 数组 $A[i, \dots, j]$

输出: 数组 $A[i, \dots, j]$ 中的max和min

1. If  $j-i+1=1$  Then 输出 $A[i], A[i]$ , 算法结束
2. If  $j-i+1=2$  Then
3. If  $A[i] < A[j]$  Then 输出 $A[i], A[j]$ ; else 输出 $A[j], A[i]$ ;  
    算法结束
4.  $k \leftarrow (j+i)/2$
5.  $m_1, M_1 \leftarrow \text{MaxMin}(A[i:k]);$
6.  $m_2, M_2 \leftarrow \text{MaxMin}(A[k+1:j]);$
7.  $m \leftarrow \min(m_1, m_2);$
8.  $M \leftarrow \max(M_1, M_2);$
9. 输出 $m, M$



# 算法复杂性分析



$$T(1)=0$$

$$T(2)=1$$

$$T(n)=2T(n/2)+2 \text{ 这里的2是 } T(1) \text{ 和 } T(2)$$

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

$$= \dots$$

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

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

$$= n/2 + n - 2$$

$$= 3n/2 - 2$$

$$n=2^k$$

与Naïve算法相比，虽然同阶，但系数有所改进

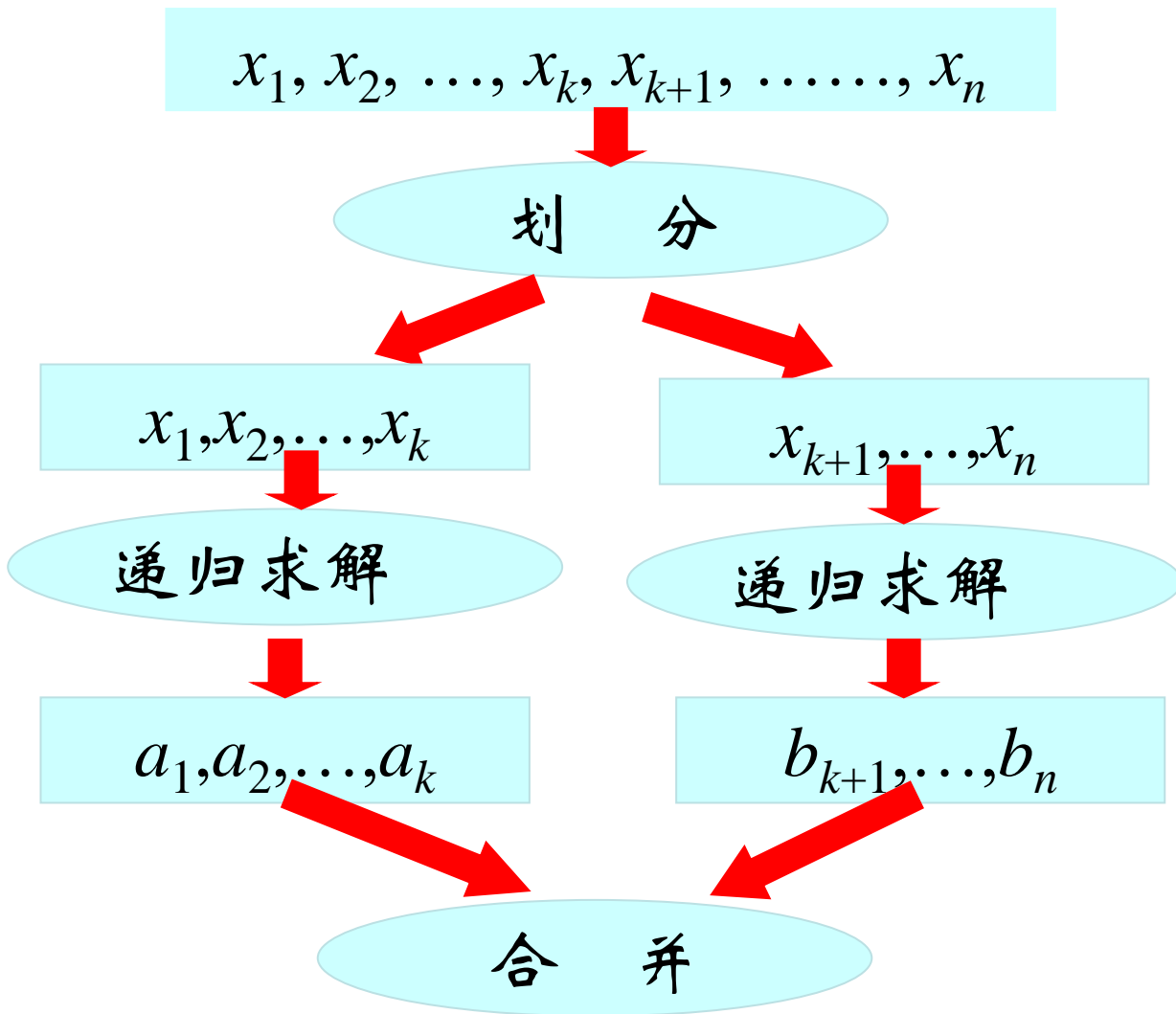


## 3.2 基于分治的排序算法

- *Quicksort* Algorithm
- 排序问题的下界



# 基于分治思想的排序算法



## 划分的策略

根据某一策略将数据集合划分成两个部分

Mergesort: 中间点

Quicksort: 任选一个划分点 $x$ , 利用 $x$ 的值将数据划分成两部分

## 合并策略

不同的划分策略对应不同的合并策略



## 3.2.1 *Quicksort*

- Idea of *Quicksort*
- *Quicksort* Algorithm
- Correctness Proof
- Performance Analysis
- Randomized *Quicksort* Algorithms



# Idea of *Quicksort*

- Divide-and-Conquer

- Divide:

- Partition  $A[p..r]$  into  $A[p..q]$  and  $A[q+1..r]$ .

$p$		$q-1$	$q$	$q+1$		$r$
-----	--	-------	-----	-------	--	-----

- $\forall x \in A[p...q], x \leq A[q], \forall y \in A[q+1...r], y > A[q]$ .
- $q$  is generated by partition algorithm.

- Conquer:

- Sort  $A[p...q-1]$  and  $A[q+1...r]$  using quicksort recursively

- Combine:

- Since  $A[p...q-1]$  and  $A[q+1...r]$  have been sorted, nothing to do

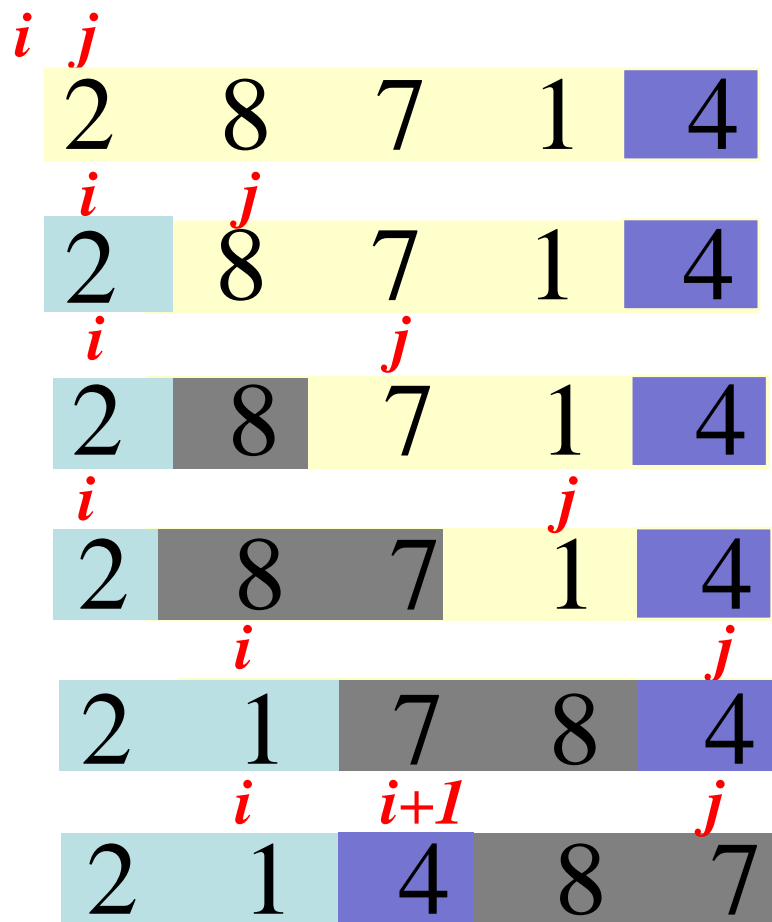




- 划分  $A[p..r]$

- 选择元素  $x$  作为划分点,  $x=A[r]$
- $x$  逐一与其它元素作比较

算法执行过程中,  
 $A$  被分成4个区域





Partition( $A, p, r$ )

$x \leftarrow A[r];$

$i \leftarrow p - 1;$

for  $j \leftarrow p$  to  $r - 1$

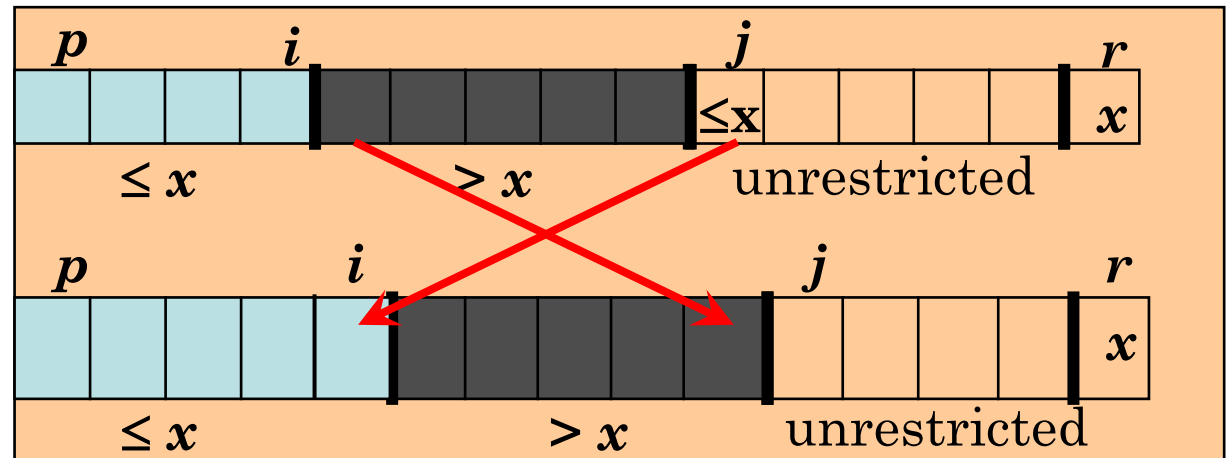
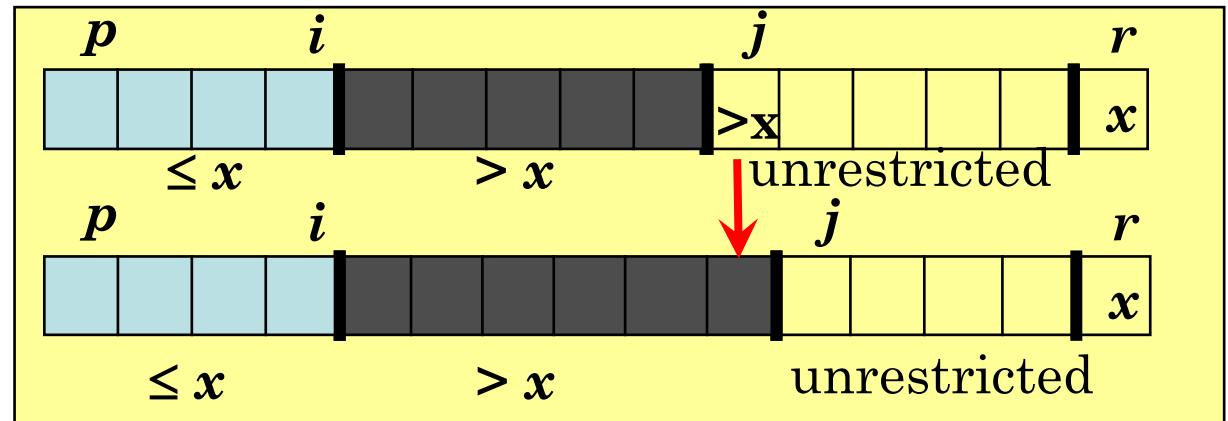
do if  $A[j] \leq x$

$i \leftarrow i + 1;$

exchange  $A[i] \leftrightarrow A[j];$

exchange  $A[i + 1] \leftrightarrow A[r];$

return  $i + 1;$



Running time:  $\Theta(n)$



# *Quicksort Algorithm*



Quicksort( $A, p, r$ )

**If**  $p < r$

**Then**  $q = \text{Partition}(A, p, r);$

    Quicksort ( $A, p, q-1$ );

    Quicksort ( $A, q+1, r$ );



- Loop Invariant(循环不变量方法)

证明主要结构是循环结构的算法的正确性

循环不变量：数据或数据结构的关键性质

依赖于具体的算法和算法特点

证明分三个阶段

(1) 初始阶段：循环开始前循环不变量成立

(2) 循环阶段：循环体每执行一次,循环不变量成立

(3) 终止阶段：算法结束后，循环不变量保证算法正确



HITWH

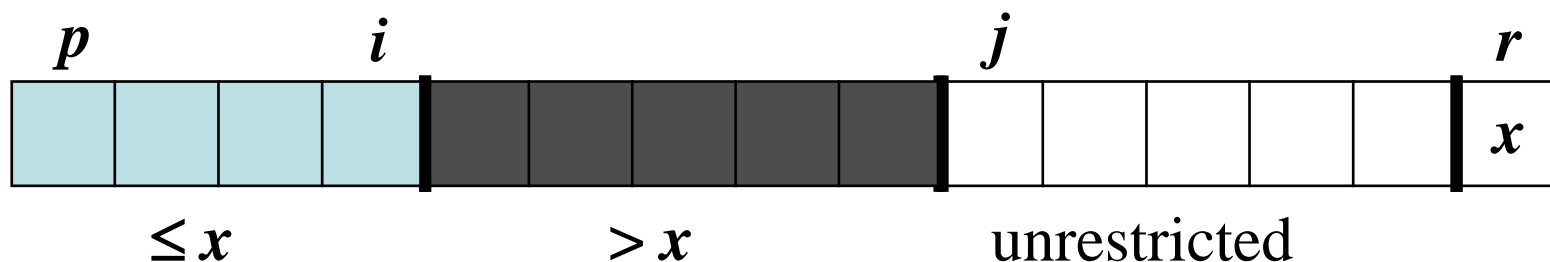
SE

这里所说的循环不变量就是指参考元素A[r]

Partition( $A, p, r$ ) $x \leftarrow A[r];$  $i \leftarrow p - 1;$ (3) for  $j \leftarrow p$  to  $r - 1$ (4) do if  $A[j] \leq x$ (5)  $i \leftarrow i + 1;$ (6) exchange  $A[i] \leftrightarrow A[j];$ exchange  $A[i + 1] \leftrightarrow A[r];$ return  $i + 1;$ 

# Correctness Proof

定义循环不变量:

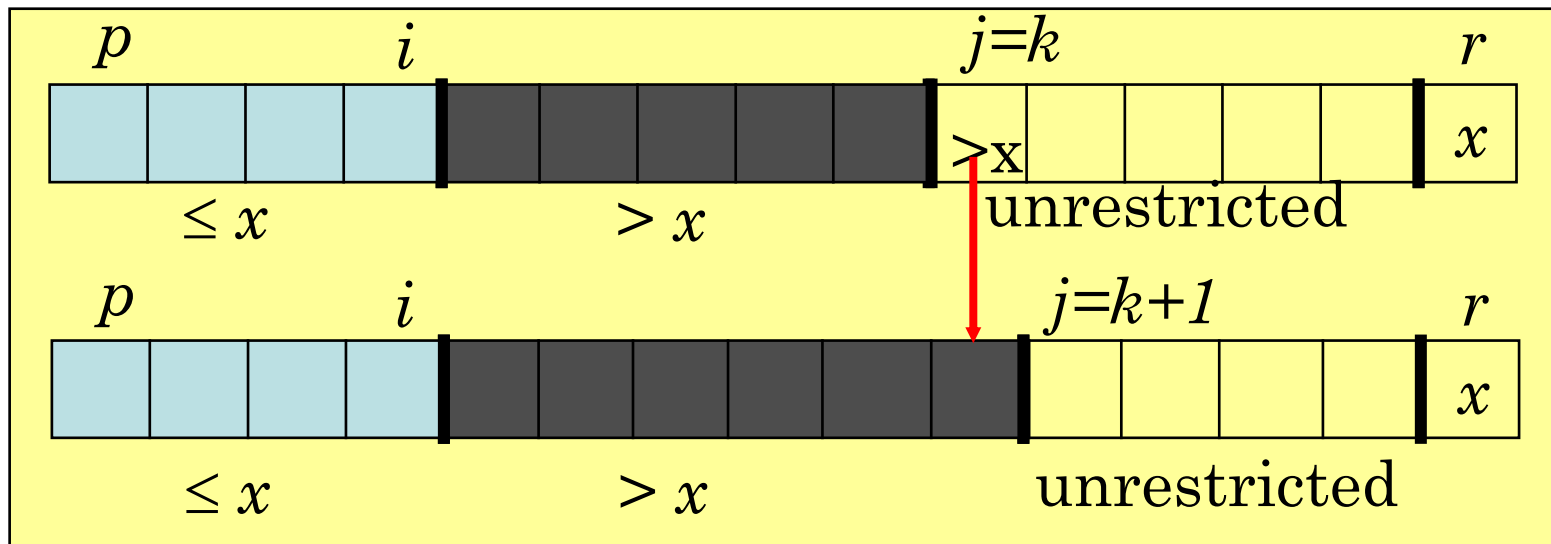
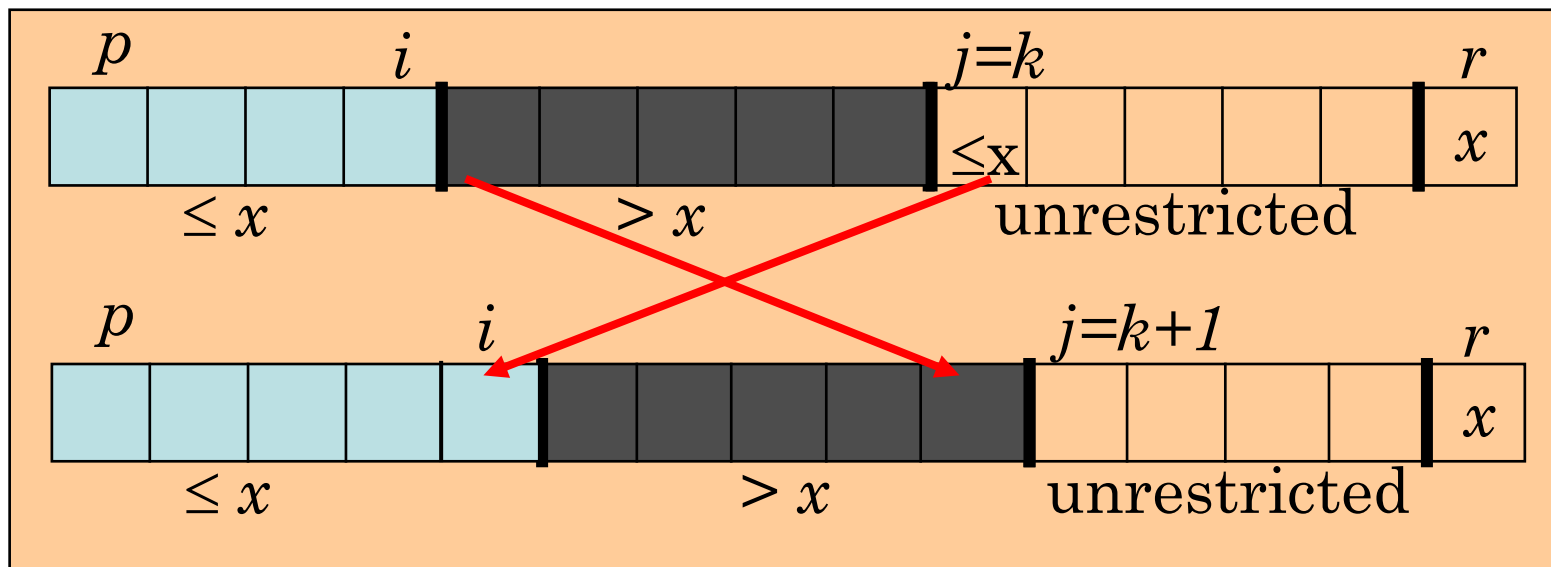
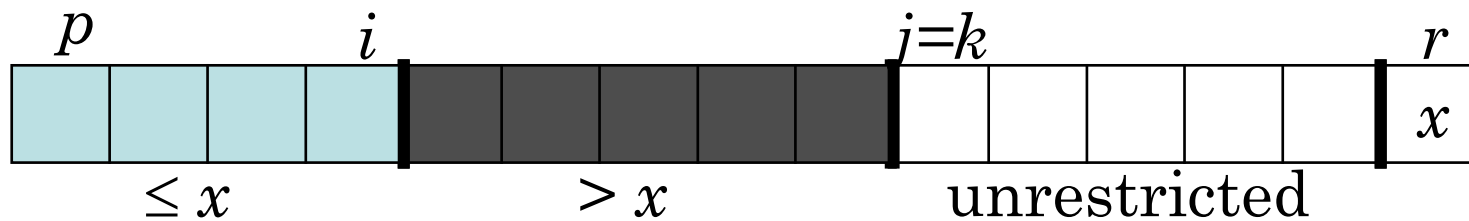
At the start of the loop of lines 3-6, for any  $k$ 1. if  $p \leq k \leq i$ , then  $A[k] \leq x$ .2. if  $i + 1 \leq k \leq j - 1$ , then  $A[k] > x$ .3. if  $k = r$ , then  $A[k] = x$ .- 初始阶段:  $j = p$ 算法迭代前:  $i = p - 1, j = p$ , 条件 1 和 2 为真. 算法第1行使得条件 3 为真. $i \quad p \quad j$ 

# 一 保持阶段

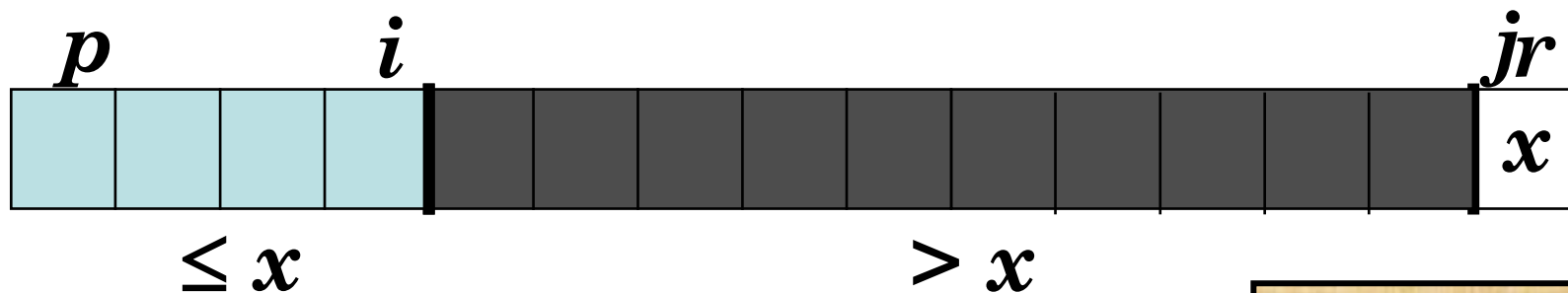
设 $j=k$ 时循环  
不变量成立.

往证 $j=k+1$ 时  
不变量成立.

```
Partition(A, p, r)
  x ← A[r];
  i ← p - 1;
  for j ← p to r - 1
    do if A[j] ≤ x
      i ← i + 1;
      exchange A[i] ↔ A[j];
  exchange A[i + 1] ↔ A[r];
  return i + 1;
```



## - 终止阶段



循环结束时,  $j=r$ , 产生三个集合:

1. 所有小于等于  $x$  的元素构成的集合.
2. 所有大于  $x$  的元素构成的集合.
3. 由元素  $x$  构成的集合.

## 算法结束时

最后一个步骤将  $A[r]$  与  $A[i+1]$  互换.

Partition( $A, p, r$ )

$x \leftarrow A[r];$

$i \leftarrow p - 1;$

for  $j \leftarrow p$  to  $r - 1$

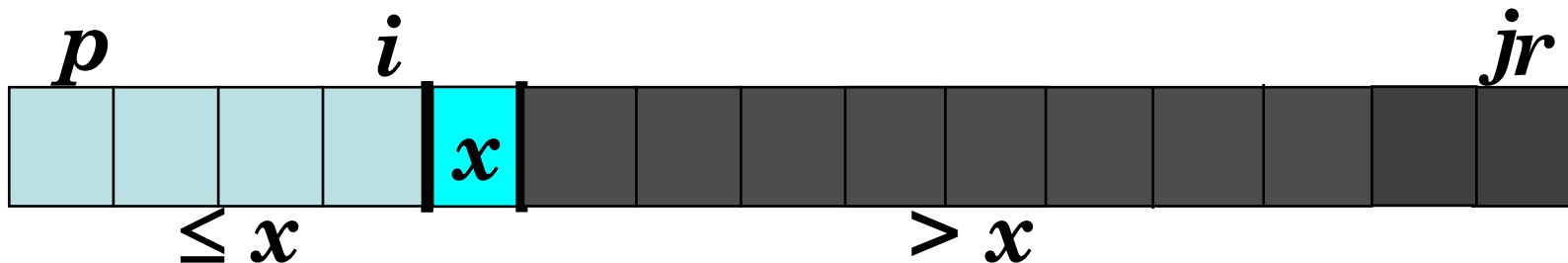
do if  $A[j] \leq x$

$i \leftarrow i + 1;$

exchange  $A[i] \leftrightarrow A[j];$

exchange  $A[i + 1] \leftrightarrow A[r];$

return  $i + 1;$





# Performance Analysis



- Time complexity of PARTITION:  $\theta(n)$
- Best case time complexity of *Quicksort*
  - Array in partition into 2 equal sets
  - $T(n) = 2T(n/2) + \theta(n)$
  - $T(n) = \theta(n \log n)$





# Performance Analysis



- Worst case time complexity of Quicksort

- Worst Case

- $|A[p..q-1]|=0, |A[q+1..r]|=n-1$



- The worst case happens in call to Partition Algorithm

- Worst case time complexity

- $T(n) = T(0) + T(n-1) + \theta(n) = T(n-1) + \theta(n) = \theta(n^2)$

左半部分个数为零，右半部分个数与左半部分相同，每次拆解排序都需要 $n$ ，按照以上拆解方法，需要拆解 $n$ 次，所以是 $n \cdot n = n^2$



HITWH  
SE

# Performance Analysis



What is the average time complexity?

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

**Why?**



# Performance Analysis

- 假如第一次划分后产生两个子序列，一个子序列包含 $s$ 个元素，另一个子序列包含 $n-1-s$ 个元素
- 一共有 $n$ 种可能的划分，即 $0 \leq s \leq n-1$ ，每种可能划分产生的概率为 $1/n$

$$\bullet \text{ 平均复杂性 } T(n) = \frac{1}{n} \sum_{s=0}^{n-1} (T(s) + T(n-1-s)) + cn$$

$$\frac{1}{n} \sum_{s=0}^{n-1} (T(s) + T(n-1-s)) = \frac{1}{n} (T(0) + T(n-1) + T(1) + T(n-2) + \cdots + T(n-1) + T(0))$$

由于 $T(0)=0$ ，有：
$$T(n) = \frac{1}{n} (2T(1) + 2T(2) + \cdots + 2T(n-1)) + cn$$

$$nT(n) = 2T(1) + 2T(2) + \cdots + 2T(n-1) + cn^2$$



$$nT(n) = 2T(1) + 2T(2) + \cdots + 2T(n-1) + cn^2$$

用 $n$ 替换为 $n-1$ 代入上式，有：

$$(n-1)T(n-1) = 2T(1) + 2T(2) + \cdots + 2T(n-2) + c(n-1)^2$$

两式相减： $nT(n) - (n-1)T(n-1) = 2T(n-1) + c(2n-1)$

$$nT(n) - (n+1)T(n-1) = c(2n-1)$$

$$\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = c \frac{(2n-1)}{(n+1)n} = c \frac{(3n - (n+1))}{(n+1)n} = c \left( \frac{3}{n+1} - \frac{1}{n} \right)$$

递归地：

$$\frac{T(n-1)}{n} - \frac{T(n-2)}{n-1} = c \left( \frac{3}{n} - \frac{1}{n-1} \right)$$

...

$$\frac{T(2)}{3} - \frac{T(1)}{2} = c \left( \frac{3}{3} - \frac{1}{2} \right)$$



# Performance Analysis



我们得到：

$$\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = c \left( \frac{3}{n+1} - \frac{1}{n} \right)$$

$$\frac{T(n-1)}{n} - \frac{T(n-2)}{n-1} = c \left( \frac{3}{n} - \frac{1}{n-1} \right)$$

...

$$\frac{T(2)}{3} - \frac{T(1)}{2} = c \left( \frac{3}{3} - \frac{1}{2} \right)$$

$$\frac{T(n)}{n+1} = 3c \left( \frac{1}{n+1} + \frac{1}{n} + \cdots + \frac{1}{3} \right) - c \left( \frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} \right) + \frac{T(1)}{2}$$

$$= 2c \left( \frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{3} + \frac{1}{2} + 1 \right) - \frac{7}{2}c + \frac{T(1)}{2} + \frac{3c}{n+1}$$

$$= 2cH_n - \frac{7}{2}c + \frac{T(1)}{2} + \frac{3c}{n+1}$$

$$T(n) = 2cnH_n + 2cH_n + \left( \frac{T(1)}{2} - \frac{7}{2}c \right)n + \left( \frac{T(1)}{2} - \frac{7}{2}c + 3c \right) = O(n \log n)$$



# Randomized Quicksort Algorithms

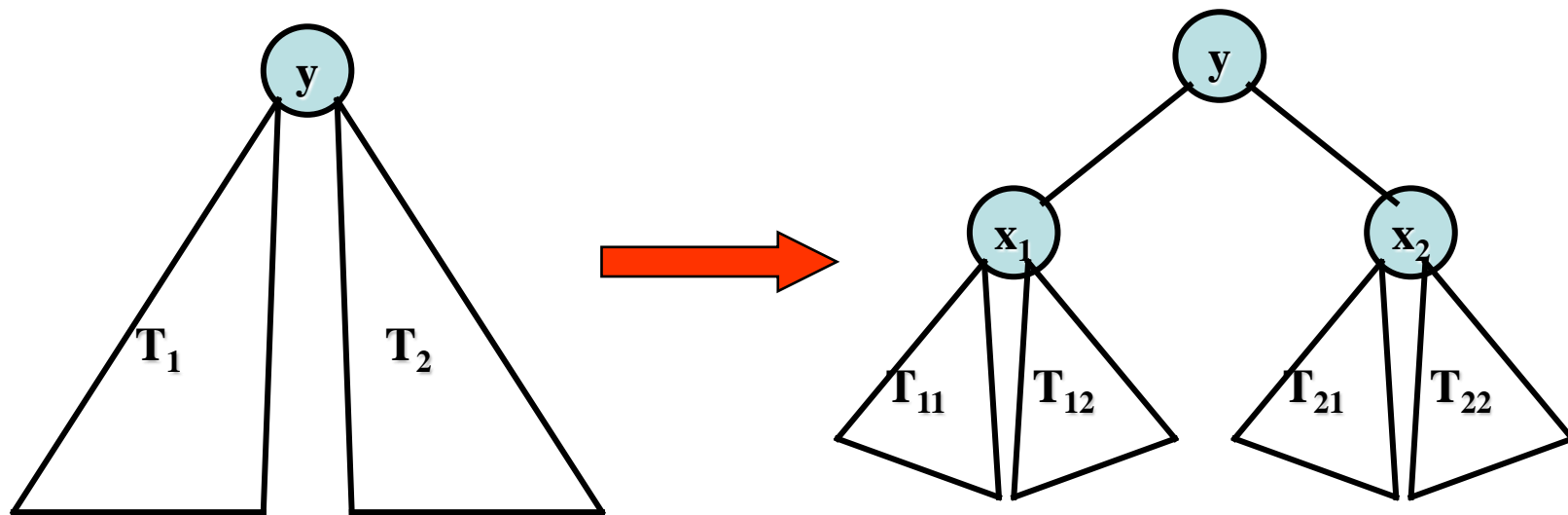


- **Randomized-Partition( $A, p, r$ )**
  1.  $i := \text{Random}(p, r)$
  2.  $A[r] \leftrightarrow A[i];$
  3. Return Partition( $A, p, r$ )
- **Randomized-Quicksort( $A, p, r$ )**
  1. **If**  $p < r$
  2. **Then**  $q := \text{Randomized-Partition}(A, p, r);$
  3.       Randomized-Quicksort( $A, p, q-1$ );
  4.       Randomized-Quicksort( $A, q+1, r$ ).



# 随机快速排序复杂性分析

- 我们可以用树表示算法的计算过程



- 我们可以观察到如下事实：
  - 一个子树的根必须与其子树的所有节点比较
  - 不同子树中的节点不可能比较
  - 任意两个节点至多比较一次



## • 基本概念

- $x_{(i)}$  表示  $A$  中 Rank 为  $i$  的元素 (第  $i$  小元素)

例如,  $x_{(1)}$  和  $x_{(n)}$  分别是最小和最大元素

- 随机变量  $X_{ij}$  定义如下:

$X_{ij}=1$  如果  $x_{(i)}$  和  $x_{(j)}$  在运行中被比较, 否则为 0

$X_{ij}$  是  $x_{(i)}$  和  $x_{(j)}$  的比较次数

- 算法的比较次数为  $\sum_{i=1}^n \sum_{j>i} X_{ij}$

- 算法的复杂性为  $T(n) = E\left[\sum_{i=1}^n \sum_{j>i} X_{ij}\right] = \sum_{i=1}^n \sum_{j>i} E[X_{ij}]$





$$T(n) = E\left[\sum_{i=1}^n \sum_{j>i} X_{ij}\right] = \sum_{i=1}^n \sum_{j>i} E[X_{ij}]$$

- 计算  $E[X_{ij}]$

- 设  $p_{ij}$  为  $x_{(i)}$  和  $x_{(j)}$  在运行中被比较的概率, 则

$$E[X_{ij}] = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij}$$

关键问题成为求解  $p_{ij}$



## • 求解 $p_{ij}$

•  $Z_{ij} = \{x_{(i)}, x_{(i+1)}, \dots, x_{(j)}\}$  是  $x_{(i)}$  和  $x_{(j)}$  之间元素集合,  
 $Z_{ij}$  在同一棵子树时,  $x_{(i)}$  和  $x_{(j)}$  才可能比较.

•  $x_{(i)}$  和  $x_{(j)}$  在执行中被比较, 需满足下列条件:

- $x_{(i)}$  是  $Z_{ij}$  中第一个被选择的子树根节点, 或者
- $x_{(j)}$  是  $Z_{ij}$  中第一个被选择的子树根节点

• 一棵子树所有点等可能地被选为划分点, 所以  $x_{(i)}$   
或  $x_{(j)}$  被选为划分点的概率 =  $2/|Z_{ij}| = 2/(j-i+1)$ .

•  $x_{(i)}$  和  $x_{(j)}$  被进行比较的概率:

$$p_{ij} = 2/(j-i+1)$$



# 随机快速排序复杂性分析



• 现在我们有

$$\sum_{i=1}^n \sum_{j>i} E[X_{ij}] = \sum_{i=1}^n \sum_{j>i} p_{ij} = \sum_{i=1}^n \sum_{j>i} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^n \sum_{k=1}^{n-i} \frac{2}{k+1} \leq \sum_{i=1}^n \sum_{k=1}^{n-i} \frac{2}{k}$$

$$\leq 2 \sum_{i=1}^n \sum_{k=1}^n \frac{1}{k} = 2nH_n = O(n \log n)$$

定理. 随机排序算法的期望时间复杂性为  $O(n \log n)$



HITWH  
SE

## 3.2.2 排序问题的下界



- 问题的下界(lower bound of a problem)
  - 是用于解决该问题的任意算法所需要的最小时间复杂度
  - 问题难度的一种度量
    - 如果问题可由一个具有较低时间复杂性的算法解决，则该问题是简单的；否则是困难的
  - 通常指：最坏情况下界
- 问题的下界是**不唯一的**
  - 例如.  $\Omega(1)$ ,  $\Omega(n)$ ,  $\Omega(n \log n)$  都是排序的下界
  - 只有  $\Omega(n \log n)$  是有意义的
  - 下界应尽可能地高，达到上限
  - 下界的分析都是经过严格理论分析和证明，而非纯粹猜测



# 问题的下界的意义



- 如果一个问题的最高下界是  $\Omega(n \log n)$  而当前最好算法的时间复杂性是  $O(n^2)$ .
  - 我们可以寻找一个更高的下界.
  - 我们可以设计更好的算法.
  - 下界和算法都是可能改进的.
- 如果一个问题的下界是  $\Omega(n \log n)$  且算法的时间复杂性是  $O(n \log n)$ , 那么这个算法是**最优的**

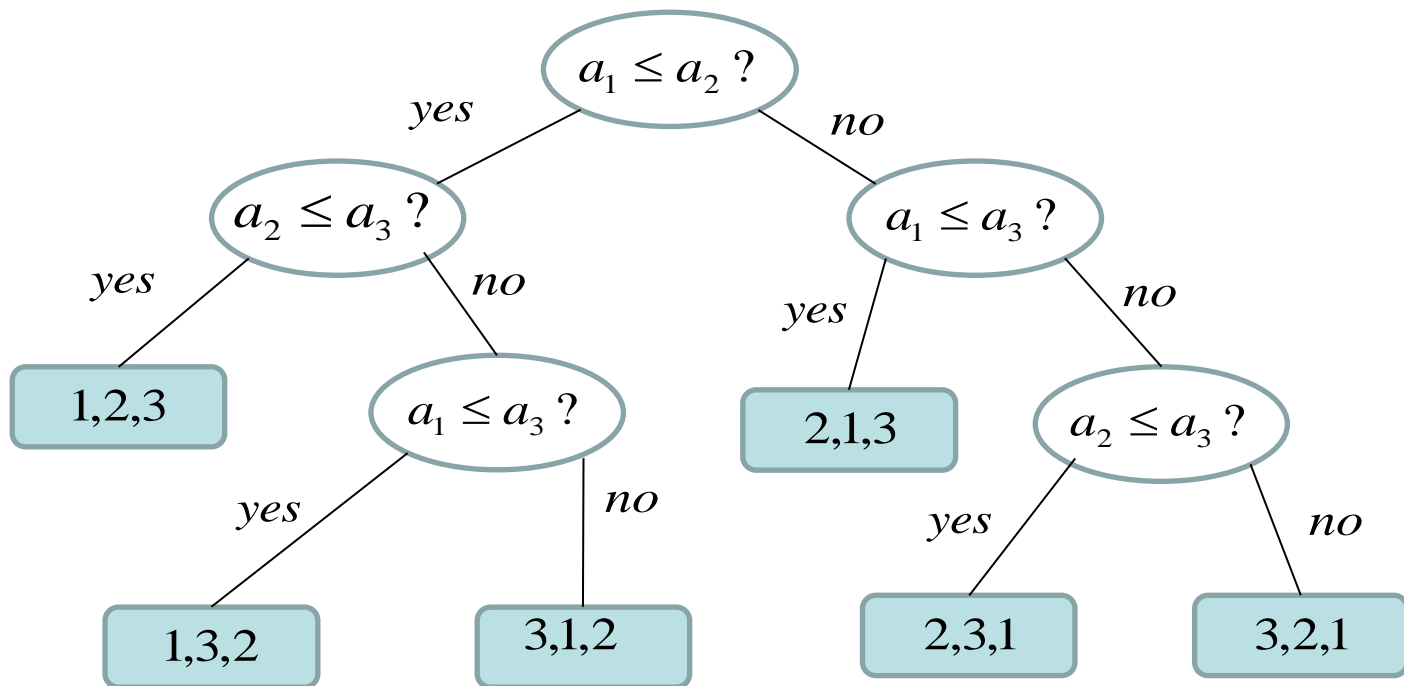


# 排序的下界

- 通常，基本操作是比较和交换的排序算法可以用一个二叉决策树描述

- 通过忽略比较以外的细节来抽象表示比较排序算法
- 每个内节点表示一个比较操作  $a_i \leq a_j$ ;
- 所有被排序元素的全排列是树的叶节点;

对于特定输入数据集的排序过程，对应于从树的根结点到叶子节点的一条路径





# 排序的下界



- $n$  个元素有  $n!$  种不同排列
- 其排序过程对应于一个高度为  $h$ ，具有  $n!$  个叶子节点的二叉决策树。
- 由于高度为  $h$  的二叉树至多有  $2^h$  个叶子节点
- 则有  $2^h \geq n!$

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

即：  $h \geq \lg(n!) = \Omega(n \lg n)$

排序的下界是： $\Omega(n \log n)$





## 3.3 Medians and Order Statistics

- Decrease and Conquer 原理
- Selection Problem



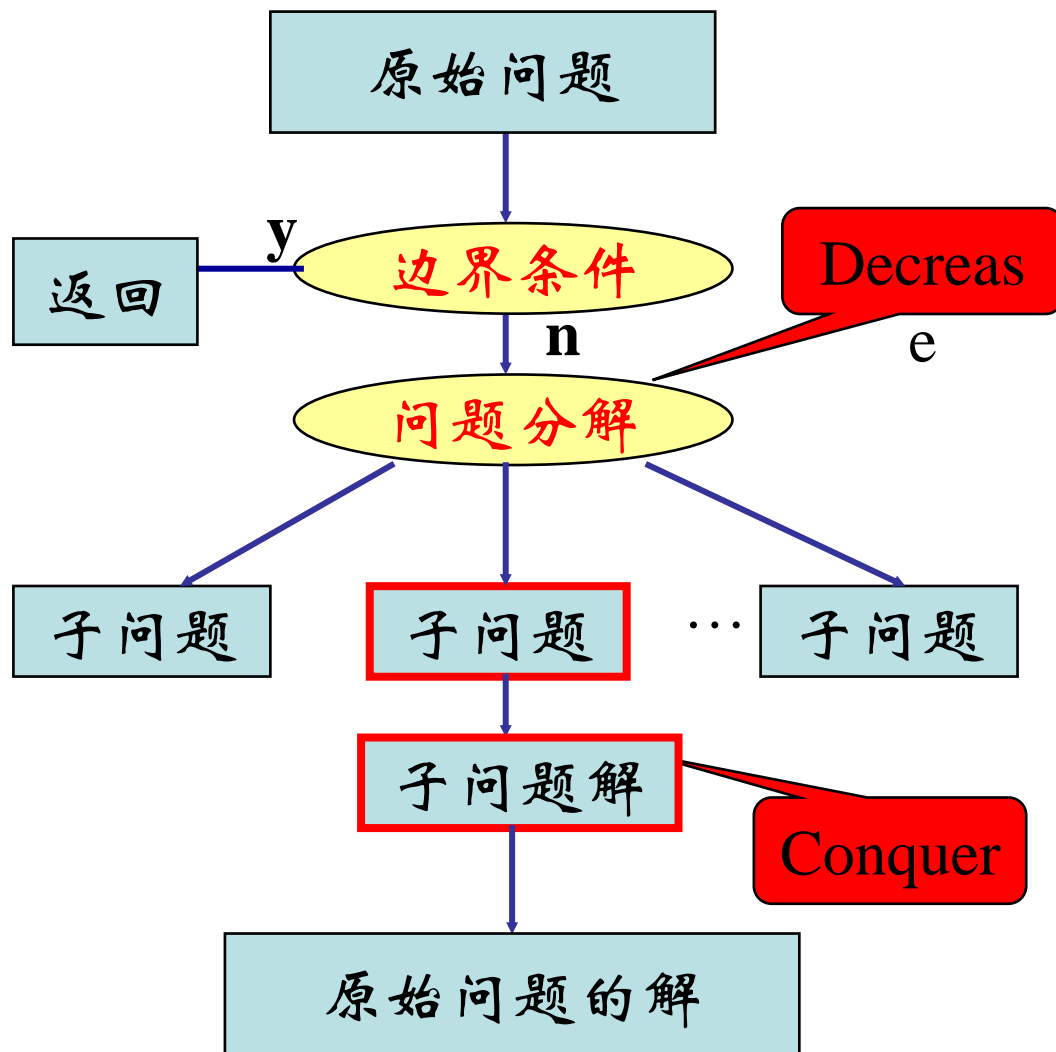
# Decrease and Conquer 原理

- 原始问题划分为若干子问题，将原始计算问题转化为其中某一个子问题的计算问题

例如：折半查找

$$T(n) = T(\lfloor n/2 \rfloor) + 1$$

- 非常有效的一种方法，通常用于解决优化问题





# Decrease and Conquer 原理

- 与 Divide and Conquer 的不同
  - 分治方法：递归求解每一个子问题，然后通过合并各个子问题的解最后得到原始问题的解
  - 减治方法：仅通过求解某一个子问题的解得到原始问题的解



# Medians and Order Statistics



- The  $i^{th}$  order statistic problem
  - Input: set  $S$  of  $n$  (distinct) elements, and a number  $k$ .
  - Output: element  $x$  in  $S$  that is greater than exactly  $k-1$  elements in  $S$ .
  - Special order statistics
    - The  $1^{st}$  order statistic is the *minimum* in  $S$
    - The  $n^{th}$  order statistic is the *maximum* in  $S$
    - The *median* in  $S$  is at
      - $(n+1)/2$  when  $n$  is odd
      - $n/2$  and  $n/2+1$  when  $n$  is even



# Selection Problem

- **Problem**

- Input: set  $A$  of  $n$  (distinct) elements, and a number  $k$ .
- Output: element  $x$  in  $A$  that is greater than exactly  $k-1$  elements in  $A$ , i.e. the  $k^{th}$  smallest element.

The straightforward algorithm:

step 1: Sort the  $n$  elements

step 2: Locate the  $k^{th}$  element in the sorted list.

Time complexity:  $O(n \log n)$



# Algorithm of Selection Problem

- **Main Idea**

- $S = \{a_1, a_2, \dots, a_n\}$
- Let  $p \in S$ , 用  $p$  将  $S$  划分为 3 个子集合  $S_1, S_2, S_3$ :
  - $S_1 = \{a_i \mid a_i < p, 1 \leq i \leq n\}$
  - $S_2 = \{a_i \mid a_i = p, 1 \leq i \leq n\}$
  - $S_3 = \{a_i \mid a_i > p, 1 \leq i \leq n\}$
- 3 种情况:
  - 若  $|S_1| > k$ , 则在集合  $S_1$  中搜索第  $k$  小的元素.
  - 否则, 若  $|S_1| + |S_2| > k$ , 则  $p$  是  $S$  中第  $k$  小的元素.
  - 否则, 在  $S_3$  中搜索第  $(k - |S_1| - |S_2|)$  小的元素.



# Algorithm of Selection Problem

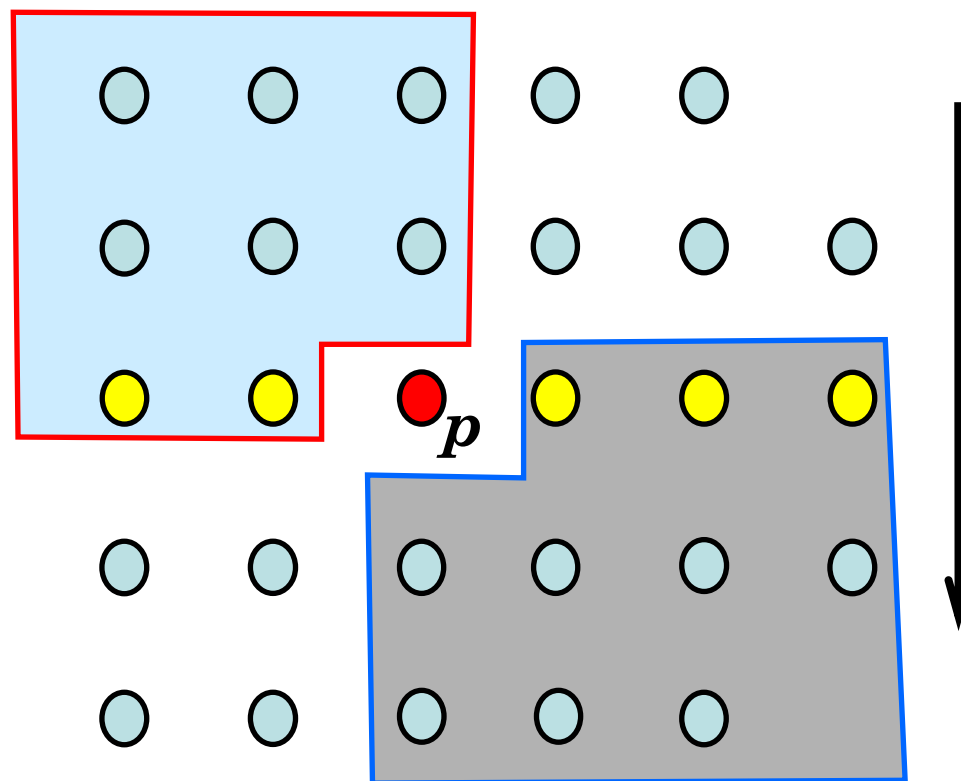
- 如何选择 $p$ ?

- 把 $n$ 个元素划分成 $\lceil \frac{n}{5} \rceil$ 个组

- (每组有5个元素.)

- 排序每一个组

- 找到元素 $p$ : 它是 $\lceil n/5 \rceil$ 个组的中位数的中位数





# Algorithm of Selection Problem

## 算法步骤:

$O(n)$

Step 1: 划分  $S$  为  $\lceil n/5 \rceil$  个组. 每组包含5个元素. 若最后一组不足5个元素, 则用  $\infty$  补足.

Step 2: 排序每组5个元素, 并确定每一分组的中位数.  $O(n)$

Step 3: 计算  $\lceil n/5 \rceil$  个中位数的中位数  $p$ .  $T(n/5)$

Step 4:  $p$  将  $S$  划分为三个子集合  $S_1, S_2$  及  $S_3$ ,  $S_1$  中元素均小于  $p$ ,  $S_2$  中元素均等于  $p$ ,  $S_3$  中元素均大于  $p$ .  $O(n)$

Step 5: 若  $|S_1| \geq k$ , 则递归地在  $S_1$  中搜索第  $k$  小元素;

否则, 若  $|S_1| + |S_2| \geq k$ , 则  $p$  即为  $S$  中第  $k$  小元素;

否则, 令  $k' = k - |S_1| - |S_2|$ , 递归地在  $S_3$  中搜索第  $k'$  小元素.

?

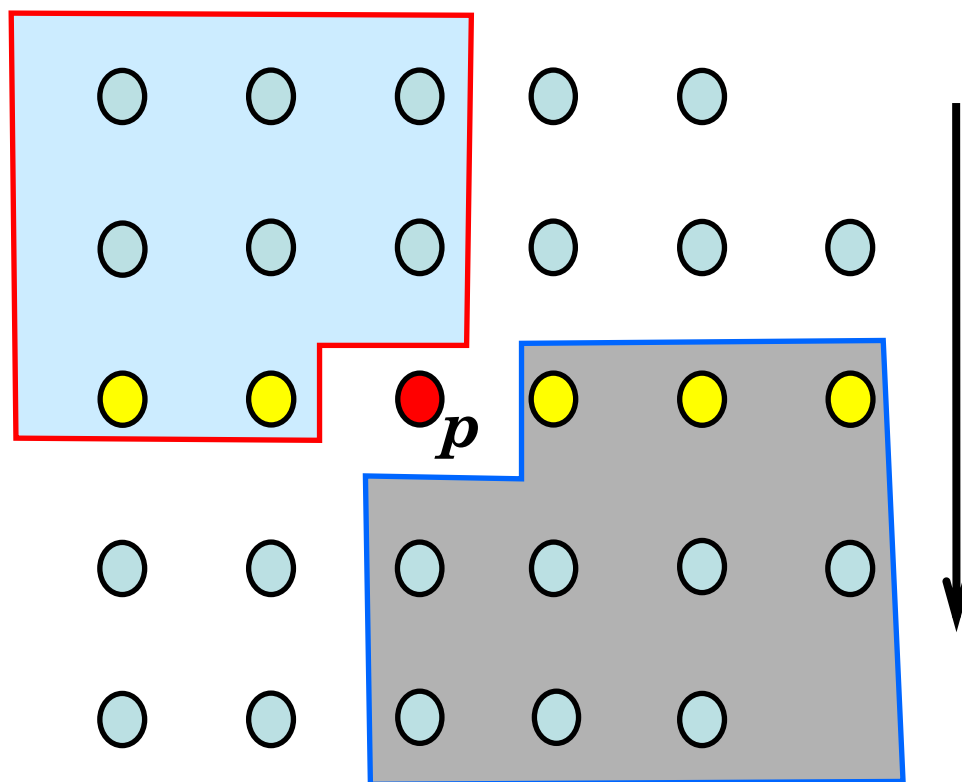




# Performance Analysis (粗略)

Each 5-element subset is sorted in non-decreasing sequence.

“大约”至少  
 $|S|/4$  个元素小于  
等于  $p$



“大约”至少  $|S|/4$   
个元素大于等于  $p$



# Algorithm of Selection Problem

## 算法步骤:

$O(n)$

Step 1: 划分  $S$  为  $\lceil n/5 \rceil$  个组. 每组包含5个元素. 若最后一组不足5个元素, 则用  $\infty$  补足.

Step 2: 排序每组5个元素, 并确定每一分组的中位数.  $O(n)$

Step 3: 计算  $\lceil n/5 \rceil$  个中位数的中位数  $p$ .  $T(n/5)$

Step 4:  $p$  将  $S$  划分为三个子集合  $S_1, S_2$  及  $S_3$ ,  $S_1$  中元素均小于  $p$ ,  $S_2$  中元素均等于  $p$ ,  $S_3$  中元素均大于  $p$ .  $O(n)$

Step 5: 若  $|S_1| \geq k$ , 则递归地在  $S_1$  中搜索第  $k$  小元素;

否则, 若  $|S_1| + |S_2| \geq k$ , 则  $p$  即为  $S$  中第  $k$  小元素;

否则, 令  $k' = k - |S_1| - |S_2|$ , 递归地在  $S_3$  中搜索第  $k'$  小元素.

$T(3n/4)$



- 算法复杂性分析

$$T(n) = T(3n/4) + T(n/5) + O(n)$$

$$\text{Let } T(n) = a_0 + a_1n + a_2n^2 + \dots, a_1 \neq 0$$

$$T(3n/4) = a_0 + (3/4)a_1n + (9/16)a_2n^2 + \dots$$

$$T(n/5) = a_0 + (1/5)a_1n + (1/25)a_2n^2 + \dots$$

$$T(3n/4 + n/5) = T(19n/20) = a_0 + (19/20)a_1n + (361/400)a_2n^2 + \dots$$

$$T(3n/4) + T(n/5) = a_0 + a_0 + (19/20)a_1n + (241/400)a_2n^2 + \dots$$

$$\leq a_0 + T(19n/20)$$

$$\Rightarrow T(n) \leq cn + T(19n/20)$$



# Performance Analysis (粗略)

$$\begin{aligned} \Rightarrow T(n) &\leq cn + T(19n/20) \\ &\leq cn + (19/20)cn + T((19/20)^2n) \\ &\quad \vdots \\ &\leq cn + (19/20)cn + (19/20)^2cn + \dots + (19/20)^p cn + T((19/20)^{p+1}n) , \\ &\quad (19/20)^{p+1}n \leq 1 \leq (19/20)^p n \\ &= cn (1 + 19/20 + (19/20)^2 + \dots + (19/20)^p) + b \\ &= \frac{1 - (\frac{19}{20})^{p+1}}{1 - \frac{19}{20}} cn + b \\ &\leq 20 cn + b \\ &= O(n) \end{aligned}$$



## • Time complexity analysis

- 比用于划分的元素 $x$ 更大的元素数量至少是:

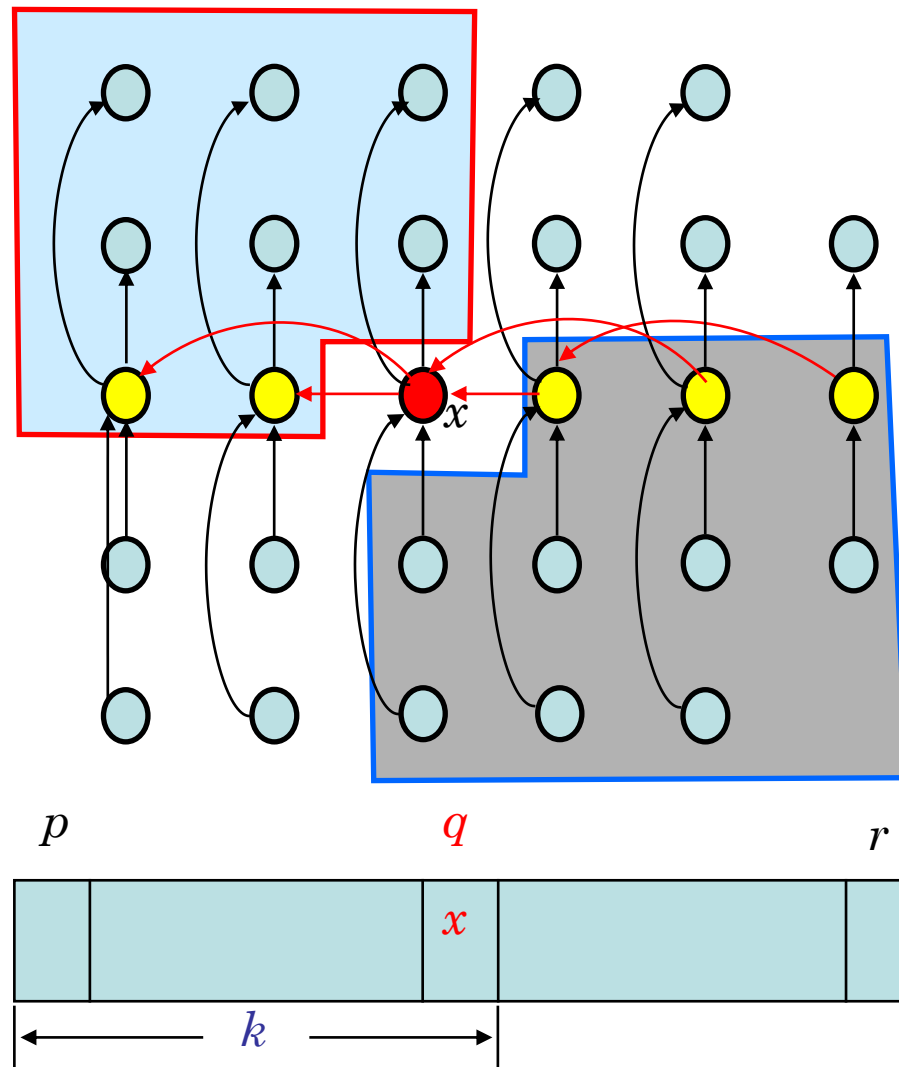
$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

- 因此, 最坏情况下, 比 $x$ 更小的元素的数量最多是:

$$n - ((3n/10) - 6) = 7n/10 + 6.$$

- 同理, 比 $x$ 更大的元素的数量最多也是:

$$7n/10 + 6$$





1. Divide  $n$  elements in  $A$  into  $\lceil n/5 \rceil$  groups of 5 elements each, at most one group has  $(n \bmod 5)$  elements.

$O(n)$

2. Find median of each group by sorting first.

$O(n)$

3. Use Select recursively to find the median  $x$  of the  $\lceil n/5 \rceil$  medians. In case of having two medians, take the lower.

$T(\lceil n/5 \rceil)$

4. Exchange  $x$  with the last element in  $A$  and apply Partition subroutine. Let  $k$  be the number of elements on the low side of the partition including  $x$ .

$O(n)$

5. If  $i = k$ , return  $x$ . Otherwise, use Select recursively to find the  $i^{\text{th}}$  smallest element on the low side if  $i \leq k$ , or the  $(i - k)^{\text{th}}$  smallest element on the high side if  $i > k$ .

$T(7n/10 + 6)$

$$T(n) \leq \begin{cases} \theta(1) & \text{if } n \leq 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n > 140 \end{cases}$$



- Now we have

$$T(n) \leq \begin{cases} \theta(1) & \text{if } n \leq 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n > 140 \end{cases}$$

- Using inductive method, we can prove  $T(n) \leq cn$  for some  $c$  and  $n > 140$ .
- Thus, the worst case time complexity is  $T(n) = O(n)$ .



$$\begin{aligned} T(n) &= \Theta(1) && \text{if } n \leq c \\ T(n) &= aT(n/b) + D(n) + C(n) && \text{if } n > c \end{aligned}$$

## 3.4 Finding the closest pair of points

优化combine阶段, 降低 $T(n) = aT(n/b) + f(n)$ 中的 $f(n)$





输入：Euclidean空间上的n个点的集合 $Q$

输出：  $A, B \in Q$ ,

$$Dis(A, B) = \text{Min}\{Dis(P_i, P_j) \mid P_i, P_j \in Q\}$$

$Dis(P_i, P_j)$  是Euclidean距离：

如果  $P_i = (x_i, y_i)$ ,  $P_j = (x_j, y_j)$ , 则

$$Dis(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$



- 利用排序的算法

- 算法

- 把Q中的点排序



- 通过有序集合的线性扫描找出最近点对

- 时间复杂性

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



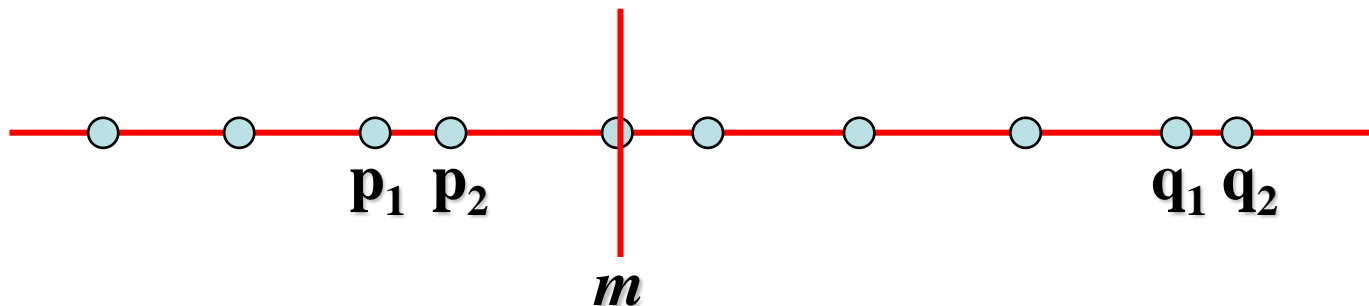
- Divide-and-conquer 算法

边界条件:

1. 如果  $Q$  中仅包含 2 个点, 则返回这个点对;

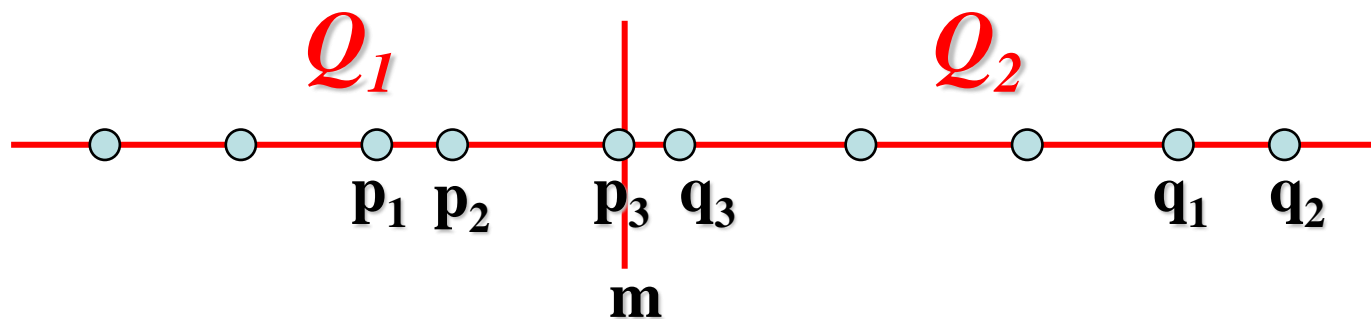
Divide:

2. 求  $Q$  中点的中位数  $m$ ;





3. 用 $Q$ 中点坐标中位数 $m$ 把 $Q$ 划分为两个大小相等的子集合 $Q_1=\{x \in Q \mid x \leq m\}$ ,  $Q_2=\{x \in Q \mid x > m\}$

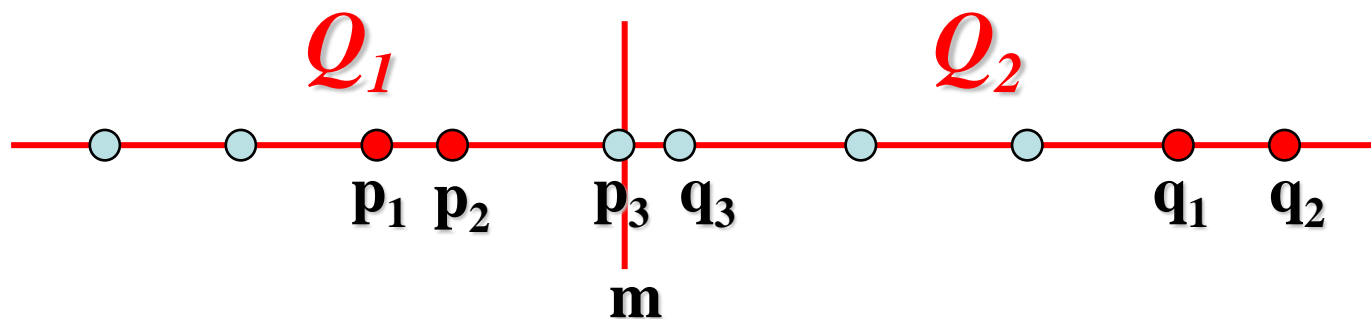


Conquer:

4. 递归地在 $Q_1$ 和 $Q_2$ 中找出最接近点对  
 $(p_1, p_2)$ 和 $(q_1, q_2)$



Merge:



5. 在  $(p_1, p_2)$ 、 $(q_1, q_2)$  和 某个  $(p_3, q_3)$  之间选择最接近点对  $(x, y)$ , 其中  $p_3$  是  $Q_1$  中最大点,  $q_3$  是  $Q_2$  中最小点。

$(x, y)$  是  $Q$  中最接近点对



- 时间复杂性

- Divide阶段需要 $O(n)$ 时间
- Conquer阶段需要 $2T(n/2)$ 时间
- Merge阶段需要 $O(n)$ 时间
- 递归方程

$$T(n) = O(1) \quad n = 2$$

$$T(n) = 2T(n/2) + O(n) \quad n \geq 3$$

- 用Master定理求解 $T(n)$

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

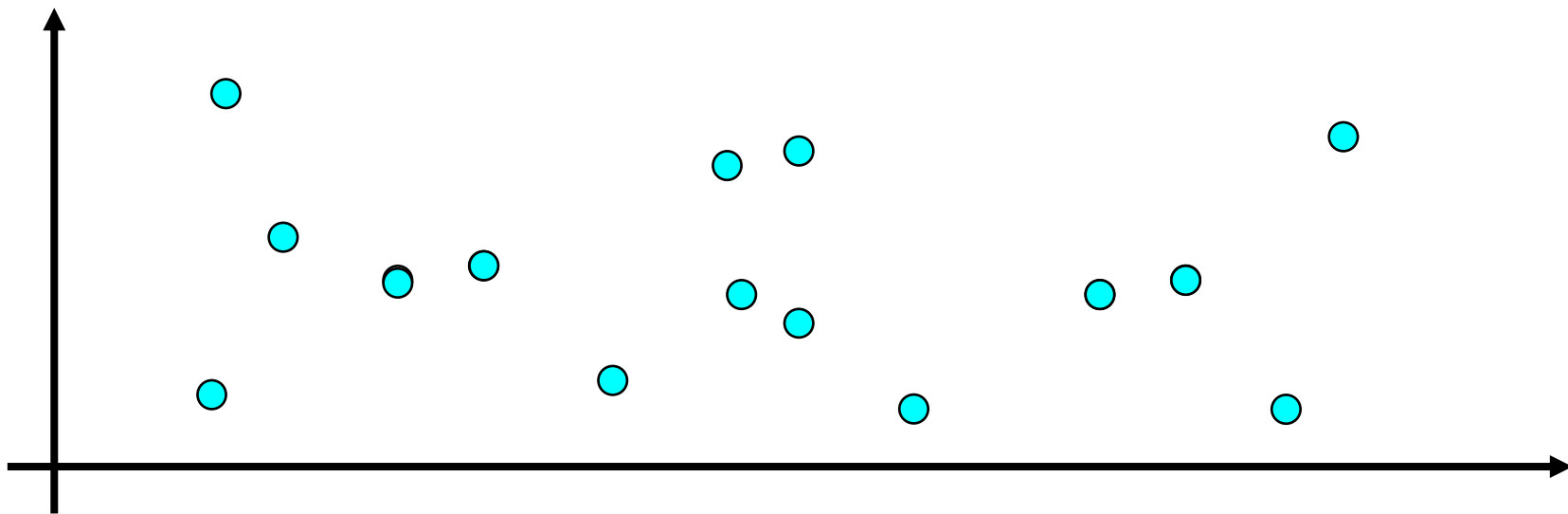


- Divide-and-conquer 算法

Assume:  $Q$  中点已经分别按  $x$  坐标和  $y$  坐标排序  
后存储在  $X$  和  $Y$  中.

边界条件:

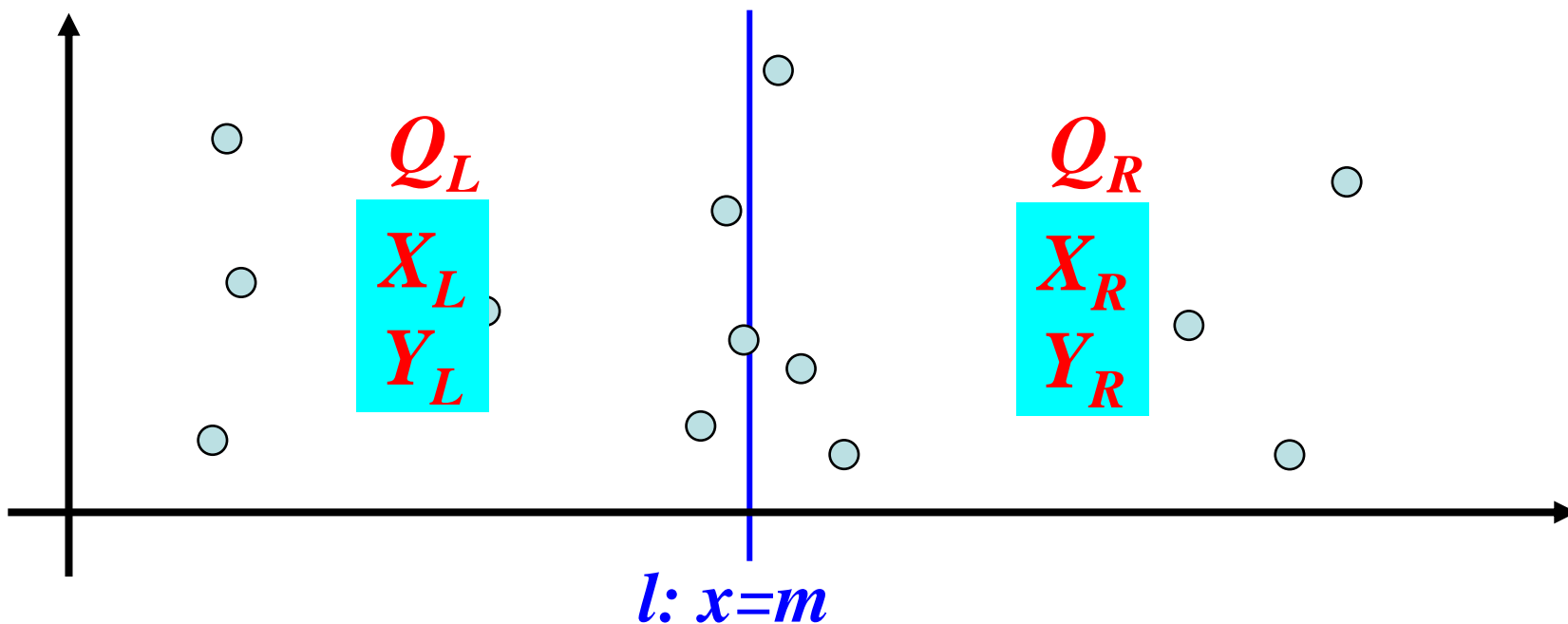
1. 如果  $Q$  中仅包含 3 个点, 则返回最近点对, 结束;





## Divide:

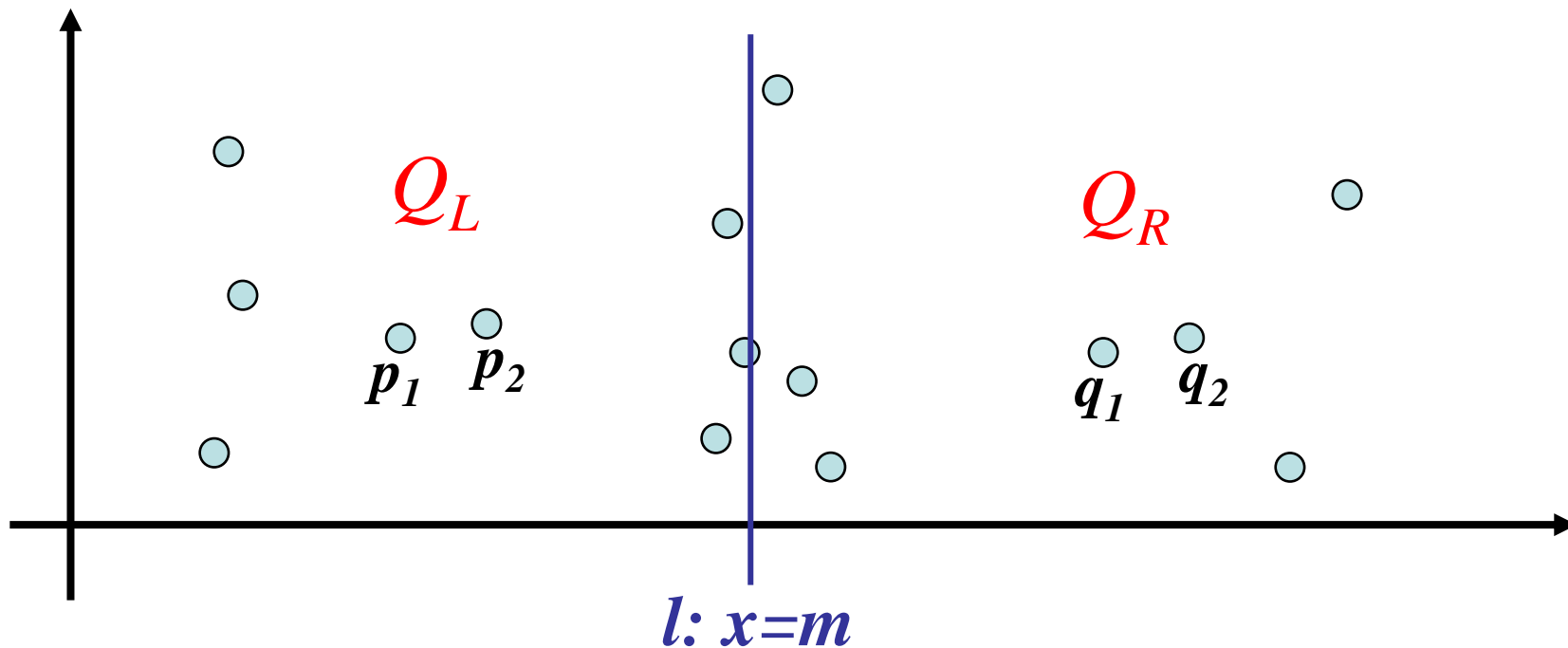
2. 计算 $Q$ 中各点 $x$ -坐标的中位数 $m$ ;
3. 用垂线 $l: x=m$ 把 $Q$ 划分成两个大小相等的子集合 $Q_L$ 和 $Q_R$ ,  $Q_L$ 中点在 $l$ 左边,  $Q_R$ 中点在 $l$ 右边;
4. 把 $X$ 划分为 $X_L$ 和 $X_R$ ; 把 $Y$ 划分为 $Y_L$ 和 $Y_R$ ;







HITWH  
SE

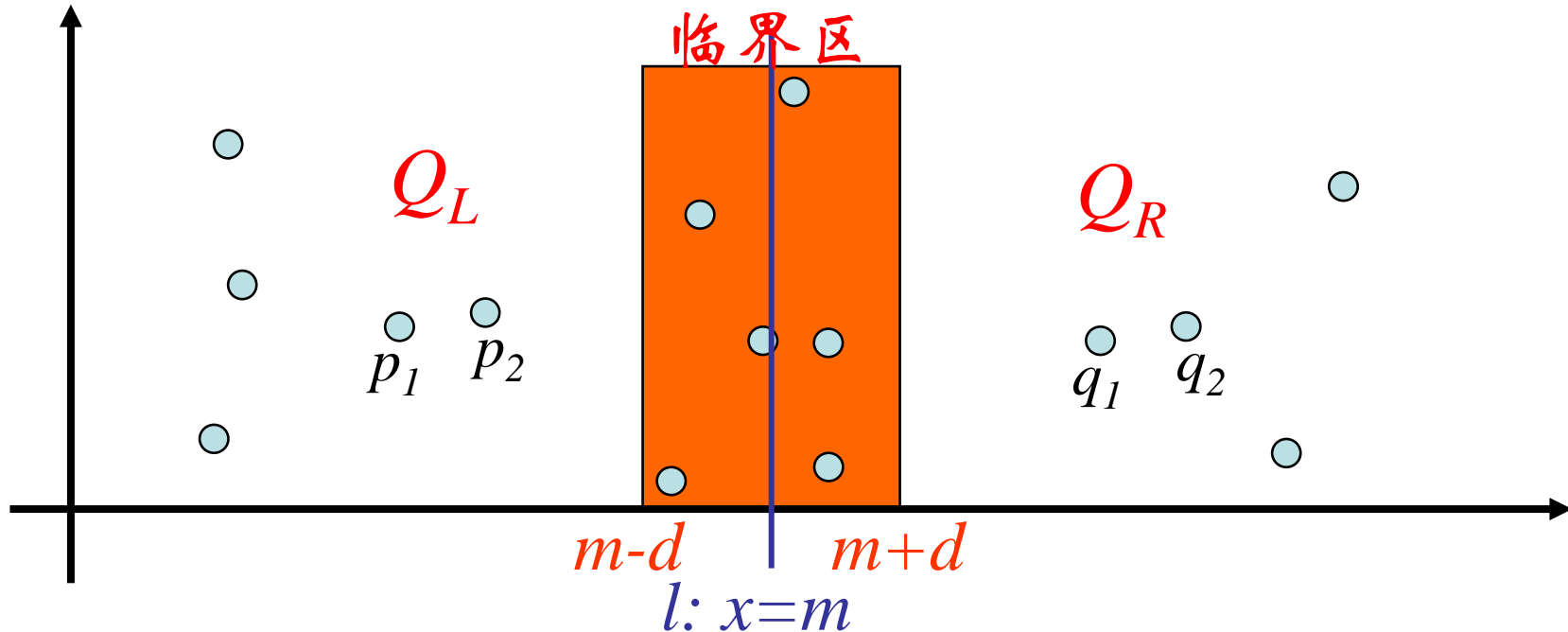


Conquer:

5. 递归地在  $Q_L$ 、 $Q_R$  中找出最近点对:

$$(p_1, p_2) \in Q_L, (q_1, q_2) \in Q_R$$

6.  $d = \min\{Dis(p_1, p_2), Dis(q_1, q_2)\};$



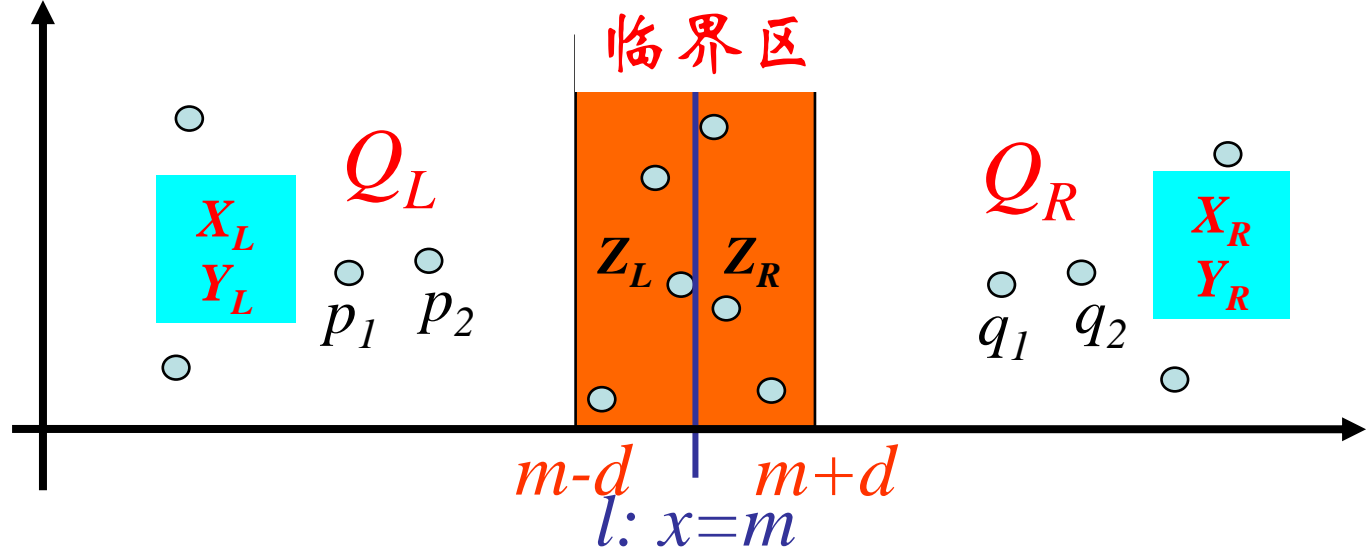
## Merge:

1. 在临界区查找距离小于  $d$  的最近点对  $(p_l, q_r)$ ,  $p_l \in Q_L$ ,  $q_r \in Q_R$ ;
2. 若找到, 则  $(p_l, q_r)$  是  $Q$  中最近点对, 否则  $(p_1, p_2)$  和  $(q_1, q_2)$  中距离最小者为  $Q$  中最近点对.

关键是  $(p_l, q_r)$  的搜索方法及其搜索时间



HITWH  
SE



•  $(p_l, q_r)$  搜索算法

1.  $Z_L = \{Q_L \text{ 中左临界区点}\};$   
 $Z_R = \{Q_R \text{ 中右临界区点}\};$



## • 时间复杂性

$$O(6n) = O(n)$$

## • $(p_l, q_r)$ 搜索算法

1.  $Z_L = \{Q_L \text{ 中左临界区点}\};$

$Z_R = \{Q_R \text{ 中右临界区点}\};$

2. For  $\forall p(x_p, y_p) \in Z_L$  Do

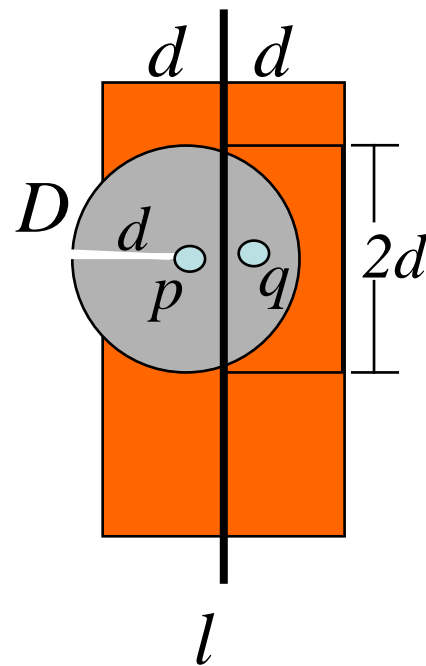
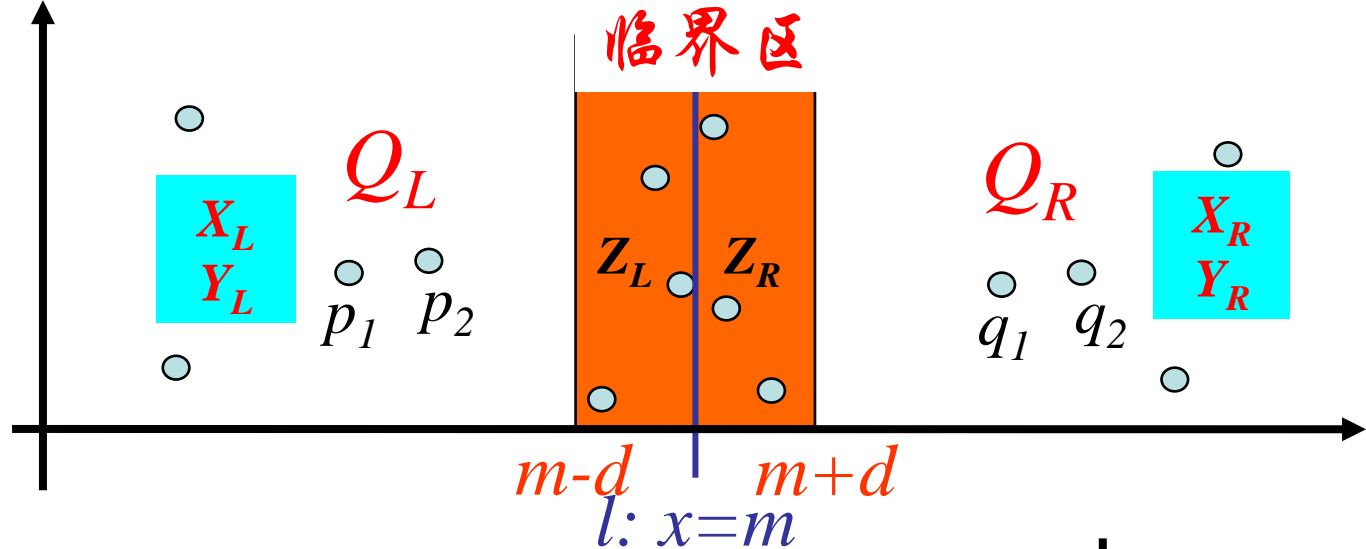
3. For  $\forall q(x_q, y_q) \in Z_R$  Do  $(y_p - d \leq y_q \leq y_p + d)$

*\*这样点至多6个\**

4. If  $Dis(p, q) < d$

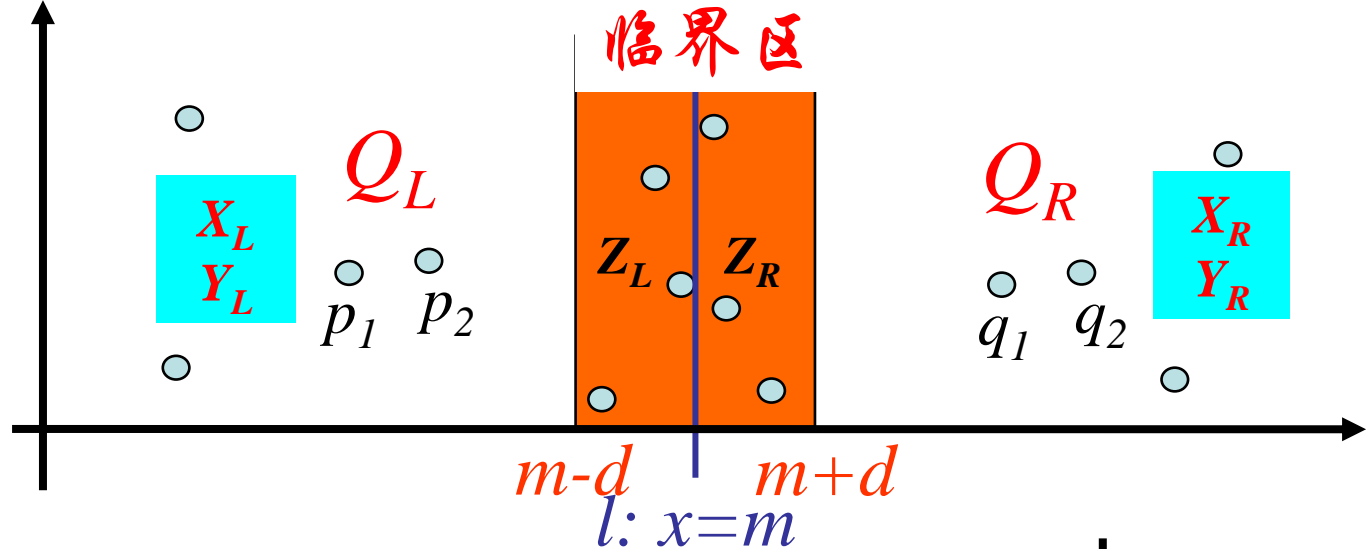
5. Then  $d = Dis(p, q)$ , 记录  $(p, q)$ ;

6. 如果  $d$  发生过变化, 与最后的  $d$  对应的点对即为  $(p_l, q_r)$ , 否则不存在  $(p_b, q_r)$ .





• 时间复杂性  
 $O(6n) = O(n)$



•  $(p_l, q_r)$  搜索算法

1.  $Z_L = \{Q_L \text{ 中左临界区点}\};$

$Z_R = \{Q_R \text{ 中右临界区点}\};$

2. For  $\forall p(x_p, y_p) \in Z_L$  Do

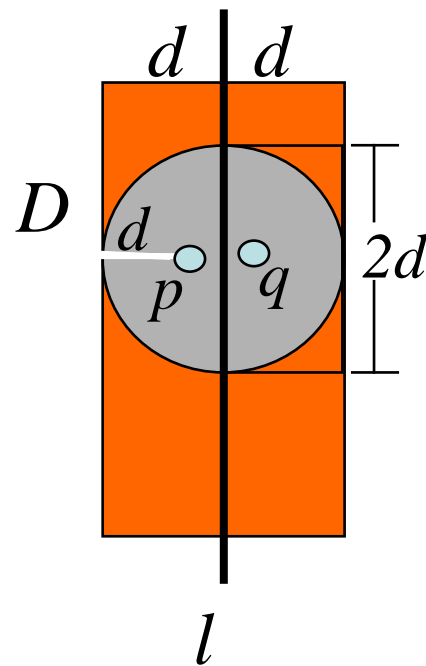
3. For  $\forall q(x_q, y_q) \in Z_R$  Do  $(y_p - d \leq y_q \leq y_p + d)$

$\backslash$  \*这样点至多6个\*  $\backslash$

4. If  $Dis(p, q) < d$

5. Then  $d = Dis(p, q)$ , 记录  $(p, q)$ ;

6. 如果  $d$  发生过变化, 与最后的  $d$  对应的点对即为  $(p_l, q_r)$ , 否则不存在  $(p_b, q_r)$ .





- 时间复杂性

- Divide阶段需要 $O(n)$ 时间
- Conquer阶段需要 $2T(n/2)$ 时间
- Merge阶段需要 $O(n)$ 时间
- 递归方程

$$T(n) = O(1) \quad n \leq 3$$

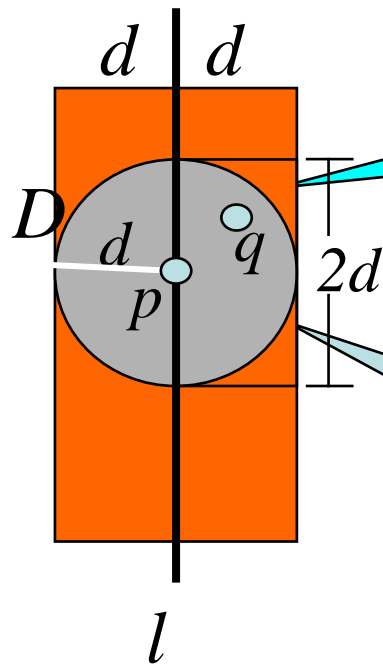
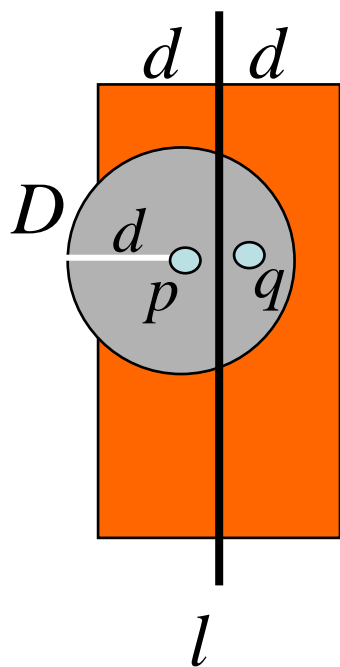
$$T(n) = 2T(n/2) + O(n) \quad n > 3$$

- 用Master定理求解 $T(n)$

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

## $(p_l, q_r)$ 的搜索时间:

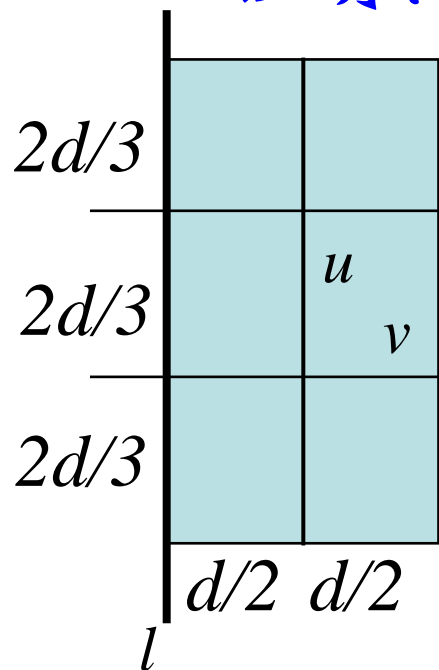
- 若  $(p, q)$  是最近点对而且  $p \in Q_L, q \in Q_R$ ,  $\text{dis}(p, q) < d$ ,  $(p, q)$  只能在下图的区域  $D$ .
- 若  $p$  在分割线  $l$  上, 包含  $(p, q)$  的区域  $D$  最大, 嵌于  $d \times 2d$  的矩形 ( $p$ -右邻域) 中, 如下图所示.





定理1. 对于左临界区中的每个点 $p$ ,  
 $p$ -右邻域中至多包含6个点。

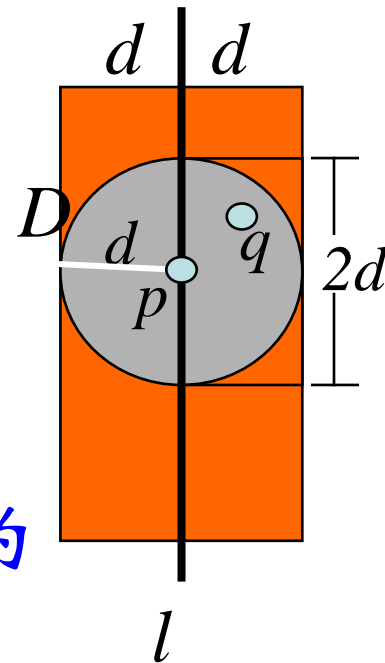
证明：把 $p$ -右邻域划分为6个 $(d/2) \times (2d/3)$ 的矩形。



若 $p$ -右邻域中点数大于6, 由鸽巢原理, 至少有一个矩形中有两个点. 设为 $u$ 、 $v$ , 则

$$(x_u - x_v)^2 + (y_u - y_v)^2 \leq (d/2)^2 + (2d/3)^2 = 25d^2/36$$

即 $Dis(u, v) \leq 5d/6 < d$ , 与 $d$ 的定义矛盾。







- **Assume:**

*Q* 中点已经分别按  $x$  坐标和  $y$  坐标  
排序后存储在  $X$  和  $Y$  中.

1.  $X$  = 按  $x$  排序  $Q$  中点;
2.  $Y$  = 按  $y$  排序  $Q$  中点;
3. FindCPP( $X$ ,  $Y$ ).

时间复杂度 =  $O(n \log n) + T(\text{FindCPP}) = O(n \log n)$



## 3.5 Sorting in Linear Time

- Counting Sort Algorithm
- Radix Sort Algorithm
- Bucket Sort Algorithm



# Counting Sort



$k$  is the biggest element in this set, representing the upper bound of the set

- Input:  $A[1..n]$  ,  $0 \leq A[i] \leq k$  for  $1 \leq i \leq n$
- Output:  $B[1..n] = \text{sorted } A[1..n]$
- Idea
  - Use  $C[0..k]$  to compute the position of each  $A[i]$
  - Put each  $A[i]$  for  $i=n$  to  $1$  into  $B[C[A[i]]]$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3
	1	2	3	4	5	6	7	8
B								
	1	2	3	4	5	6	7	8
B							3	
	1	2	3	4	5	6	7	8
B		0					3	
	1	2	3	4	5	6	7	8
B		0				3	3	
	1	2	3	4	5	6	7	8
B		0		2		3	3	
	1	2	3	4	5	6	7	8
B	0	0		2		3	3	
	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

	0	1	2	3	4	5
C	2	0	2	3	0	1
	0	1	2	3	4	5
C	2	2	4	7	7	8
	0	1	2	3	4	5
C	2	2	4	6	7	8
	0	1	2	3	4	5
C	1	2	4	6	7	8
	0	1	2	3	4	5
C	1	2	4	5	7	8
	0	1	2	3	4	5
C	1	2	3	5	7	8
	0	1	2	3	4	5
C	0	2	3	5	7	8



- Algorithm and Time complexity

```
for  $i \leftarrow 0$  to  $k$ 
```

```
do  $C[i] \leftarrow 0$ ;
```

$O(k)$

```
for  $j \leftarrow 1$  to  $length[A]$ 
```

```
do  $C[A[j]] \leftarrow C[A[j]] + 1$ ;
```

$O(n)$

```
for  $i \leftarrow 1$  to  $k$ 
```

```
do  $C[i] \leftarrow C[i] + C[i-1]$ ;
```

$O(k)$

```
for  $j \leftarrow length[A]$  downto 1
```

```
do begin
```

```
     $B[C[A[j]]] \leftarrow A[j]$ ;
```

```
     $C[A[j]] \leftarrow C[A[j]] - 1$ ;
```

$O(n)$

***Time Complexity =  $O(n+k)$***

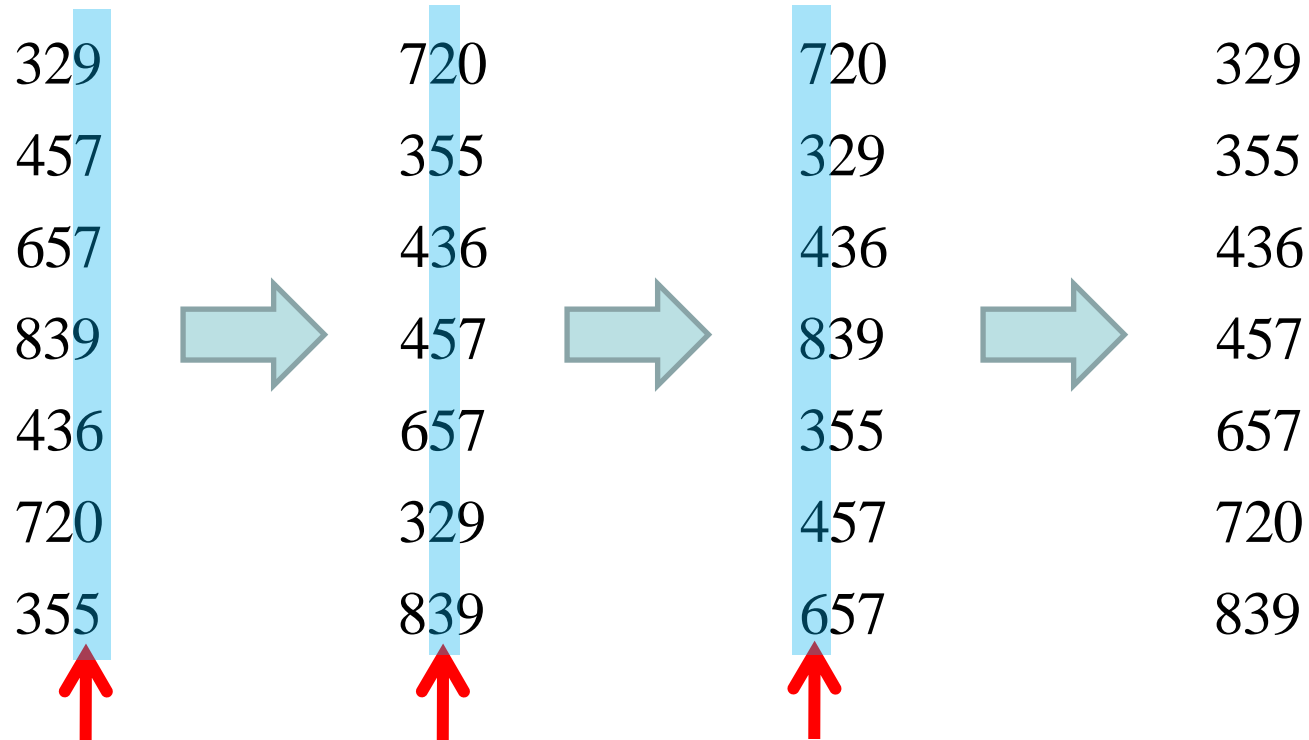


- Counting Sort 算法性质
  - Counting sort doesn't sort in place
  - Counting sort is **stable**
    - That is, the same values appear in the output array in the same order as they do in the input array.
  - Problems
    - $A[i]$  must be integer.
    - $k$  should be small



# Radix Sort Algorithm

- Idea of Radix sort algorithm
  - Use stable sort algorithm
  - Sort the  $n$   $d$ -digit elements from the lowest digit to the highest digit





- Radix sort algorithm

Input: Array  $A$ , each element is a number of  $d$  digit.

Radix-Sort( $A, d$ )

for  $i \leftarrow 1$  to  $d$  do

    use a stable sort to sort array  $A$  on digit  $i$ ;

- Time complexity of Radix sort algorithm

- Using Counting sort algorithm,  $0 \leq A[i] \leq k$

- The time complexity is  $O(d(n+k))$

- Problems





把每一个数字分成 $b/r$ 份，每一个子数字长 $r$ ，然后使用基数排序，对于 $r$ 位的数字，要开辟 $2^r$ 的大小的存储空间

- Extension of Radix sort

- Input:  $n$   $b$ -binary-digit number, any  $r \leq b$

- Radix sort can sort these numbers in  $\Theta((b/r)(n+2^r))$

- Why

- View each number as  $d = \lceil b/r \rceil$  digits of  $r$  bits each.

- Each digit is an integer in the range  $0$  to  $2^r - 1$

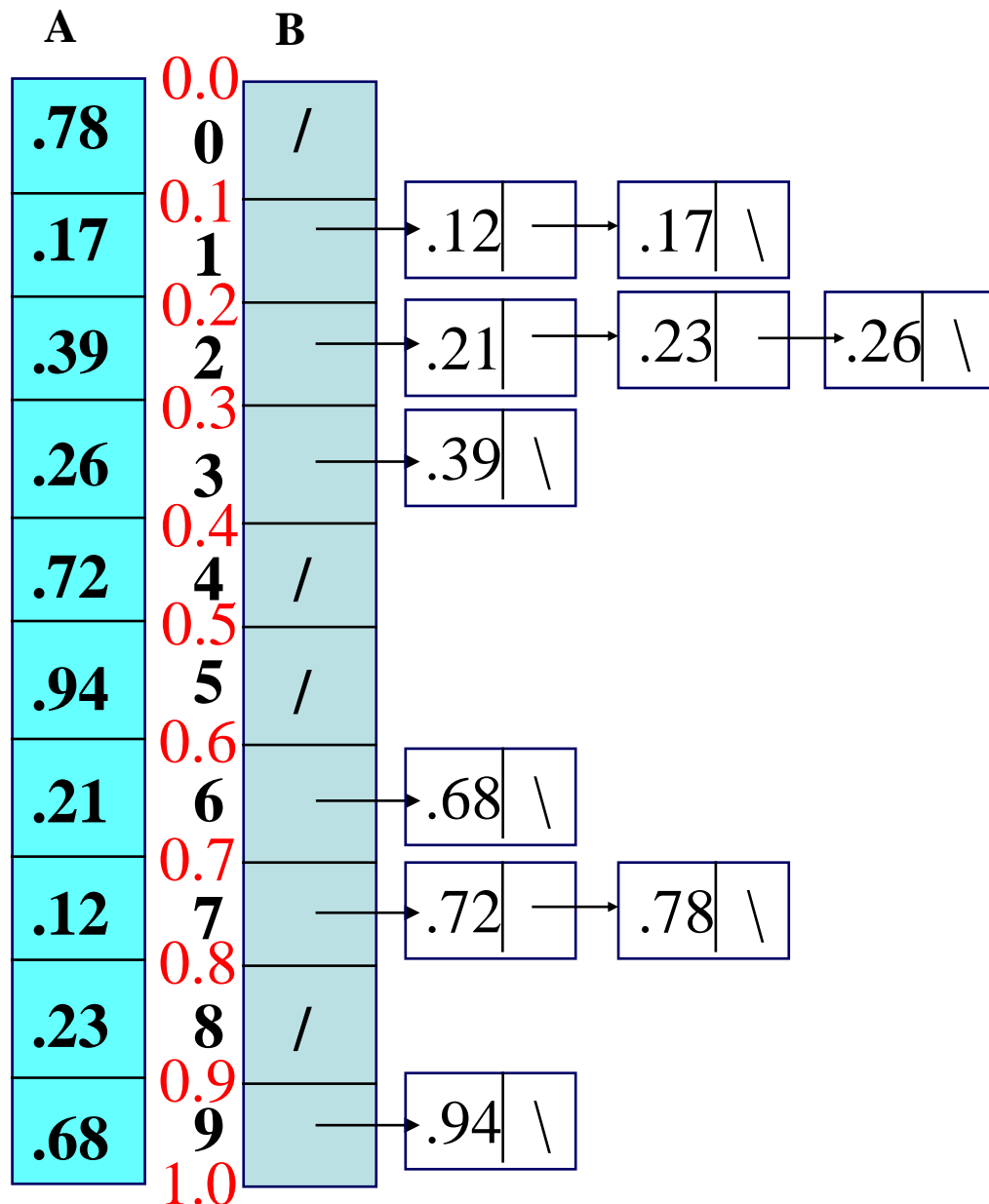
- Use counting sort with  $k = 2^r - 1$

- How about  $b=500$ ,  $r=100$ ?



# Bucket Sort

- **Assumption of Bucket Sort**
  - Input is elements uniformly distributed in  $[0, 1)$  independently
- **Idea of Bucket Sort**
  - Divide  $[0, 1)$  into  $n$  equal-sized bucket
  - Distribute the input into the  $n$  bucket
  - Sort the numbers in each bucket
  - List all the sorted numbers in each bucket in order

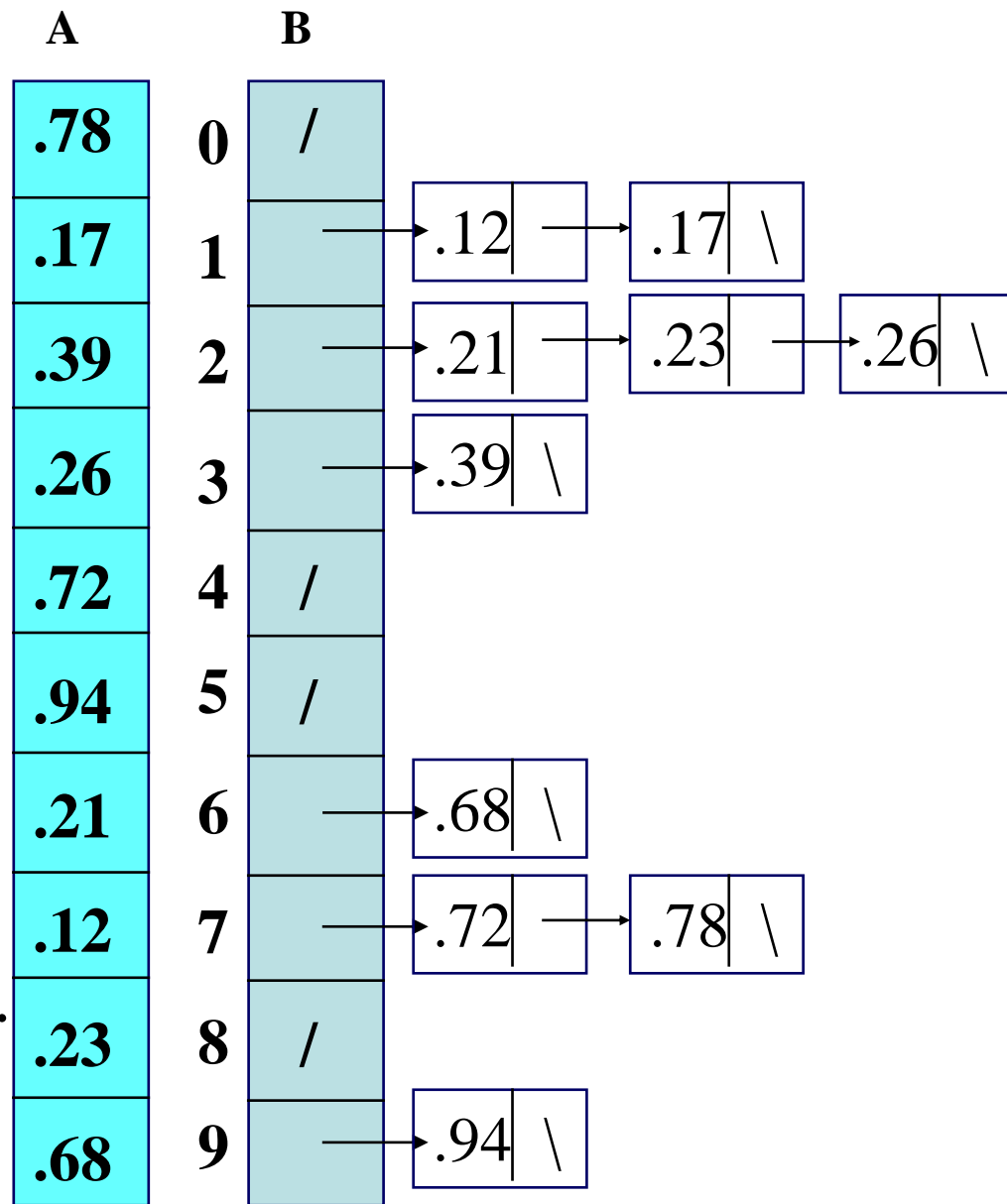




## • Bucket Sort Algorithm

Bucket-Sort( $A$ )

1.  $n = \text{length}[A]$ ;
2. For  $i = 1$  To  $n$  Do
3.    Insert  $A[i]$  into list  $B[\lfloor nA[i] \rfloor]$ ;
4. For  $i = 0$  To  $n-1$  Do
5.    Sort list  $B[i]$  with insert sort;
6. concatenate lists  $B[0], \dots, B[n-1]$ .





- **Time complexity**

- Let the random variable  $n_i = |B[i]|$
- The time complexity:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

- Since  $E[n_i^2] = 2 - 1/n$

$$\begin{aligned} E[T(n)] &= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \\ &= \Theta(n) + O(n(2 - 1/n)) = \Theta(n) \end{aligned}$$



**HITWH**  
**SE**

## **3.6 Finding the convex hull**



输入：平面上的 $n$ 个点的集合 $Q$

输出：CH( $Q$ ):  $Q$ 的convex hull

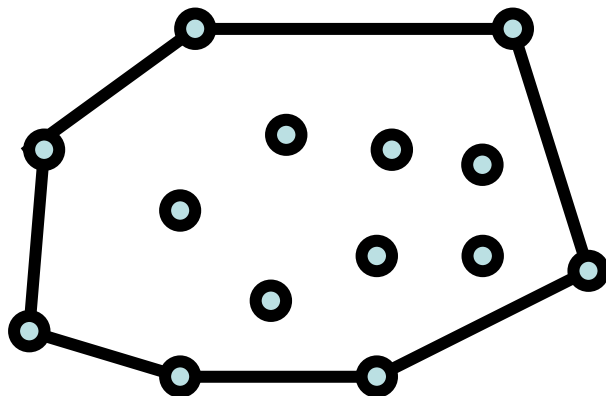
$Q$ 的convex hull是一个最小凸多边形  
 $P$ ,  $Q$ 的点或者在 $P$ 上或者在 $P$ 内

凸多边形 $P$ 是具有如下性质多边形:  
连接 $P$ 内任意两点的边都在 $P$ 内



- 基本思想

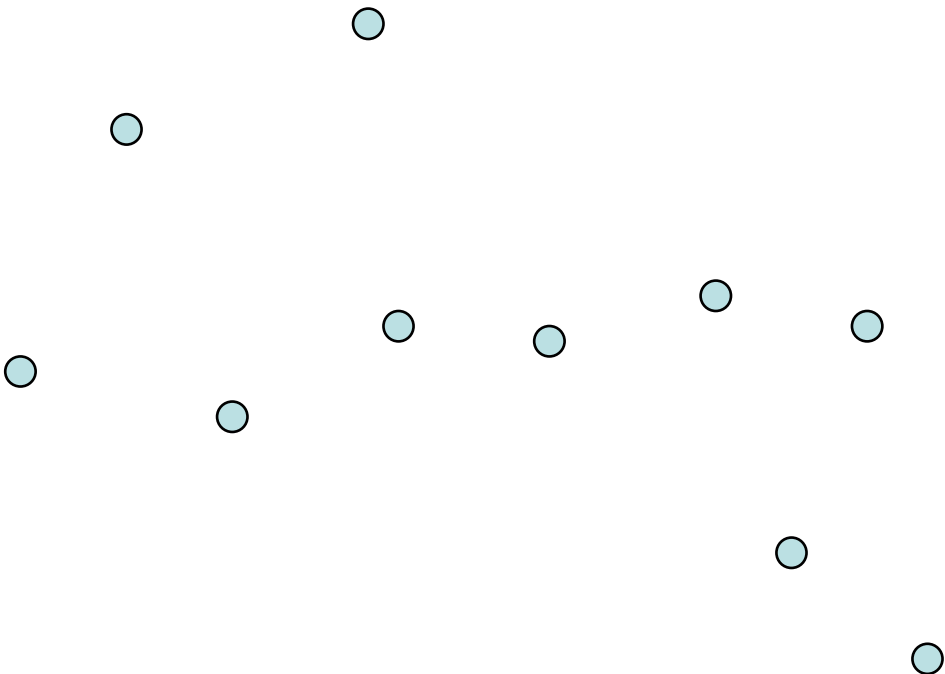
- 当沿着Convex hull逆时针漫游时，总是向左转
- 在极坐标系下按照极角大小排列，然后逆时针方向漫游点集，去除非Convex hull顶点(非左转点)。





HITWH  
SE

# 第1步



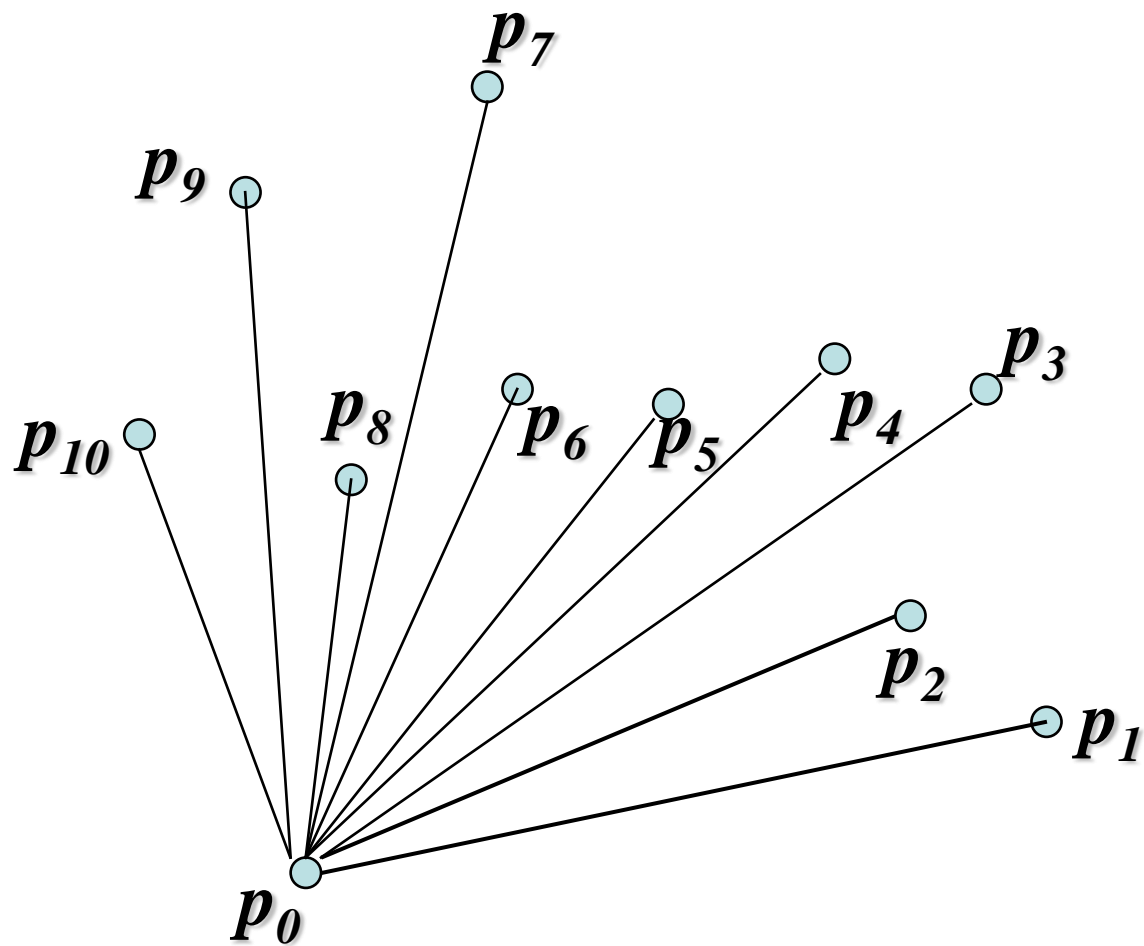
$p_0$





HITWH  
SE

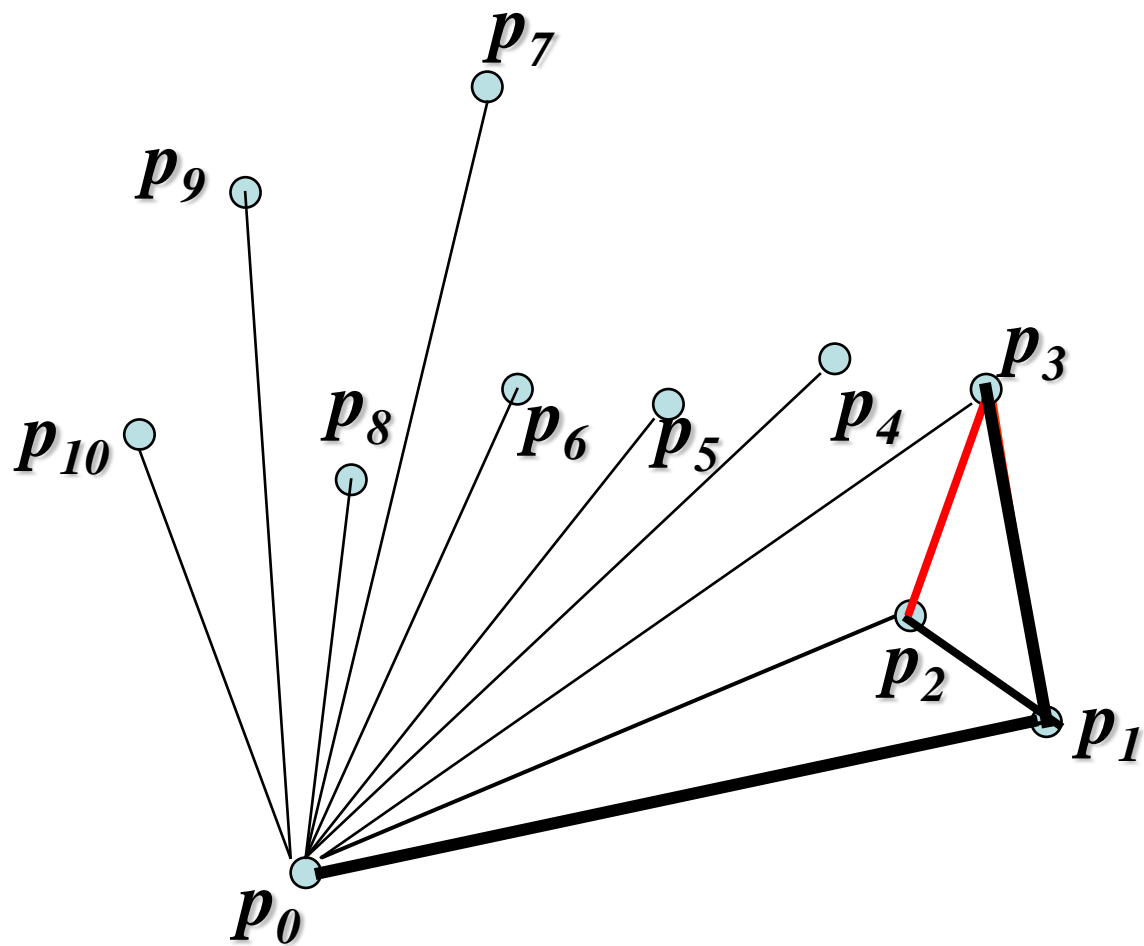
## 第2步





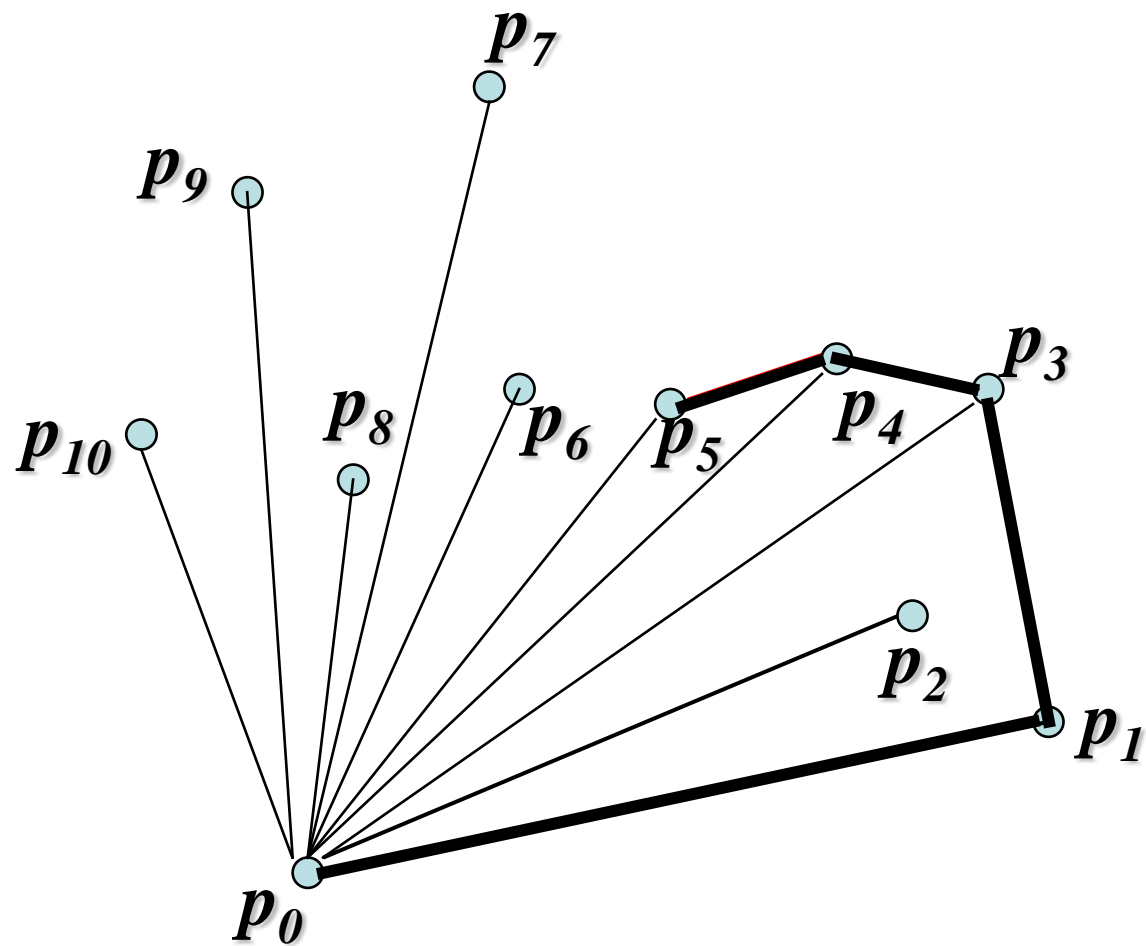
HITWH  
SE

# 第3步



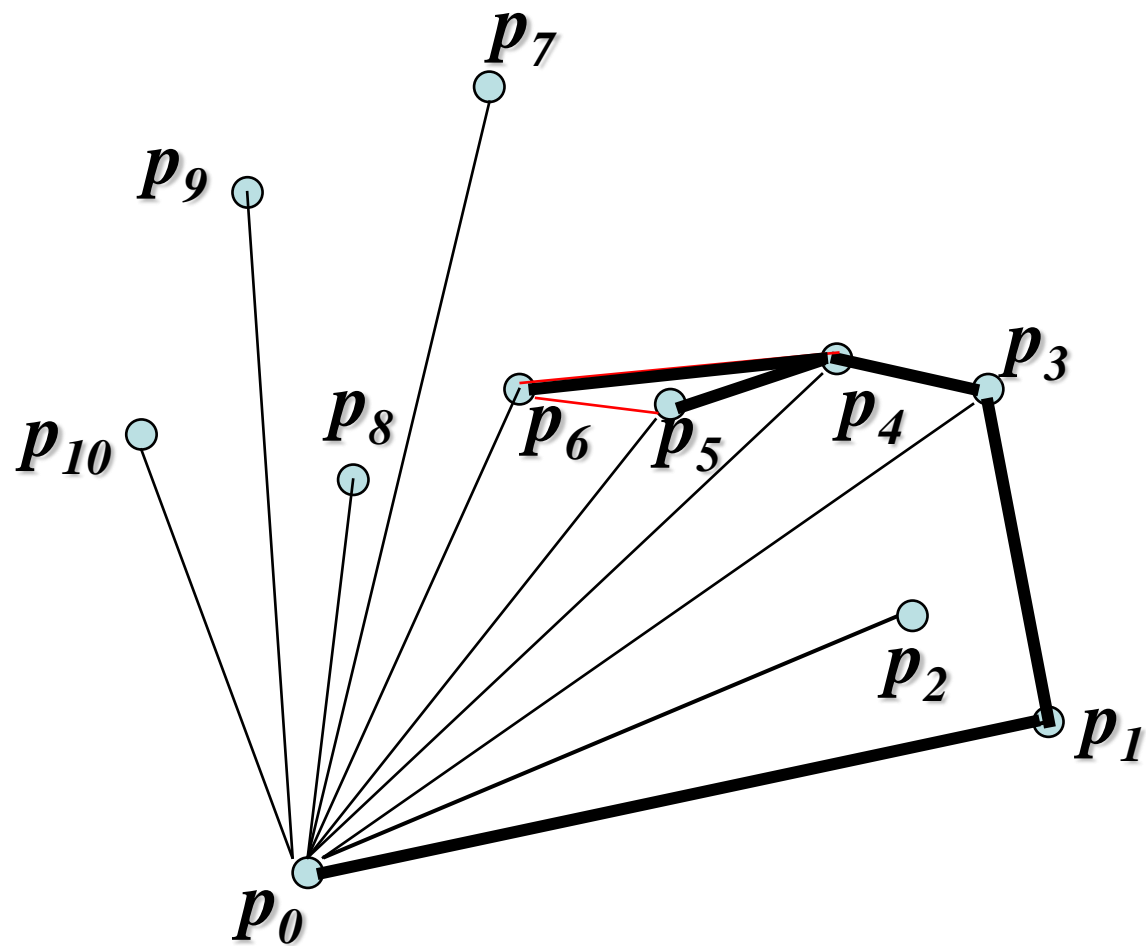


HITWH  
SE



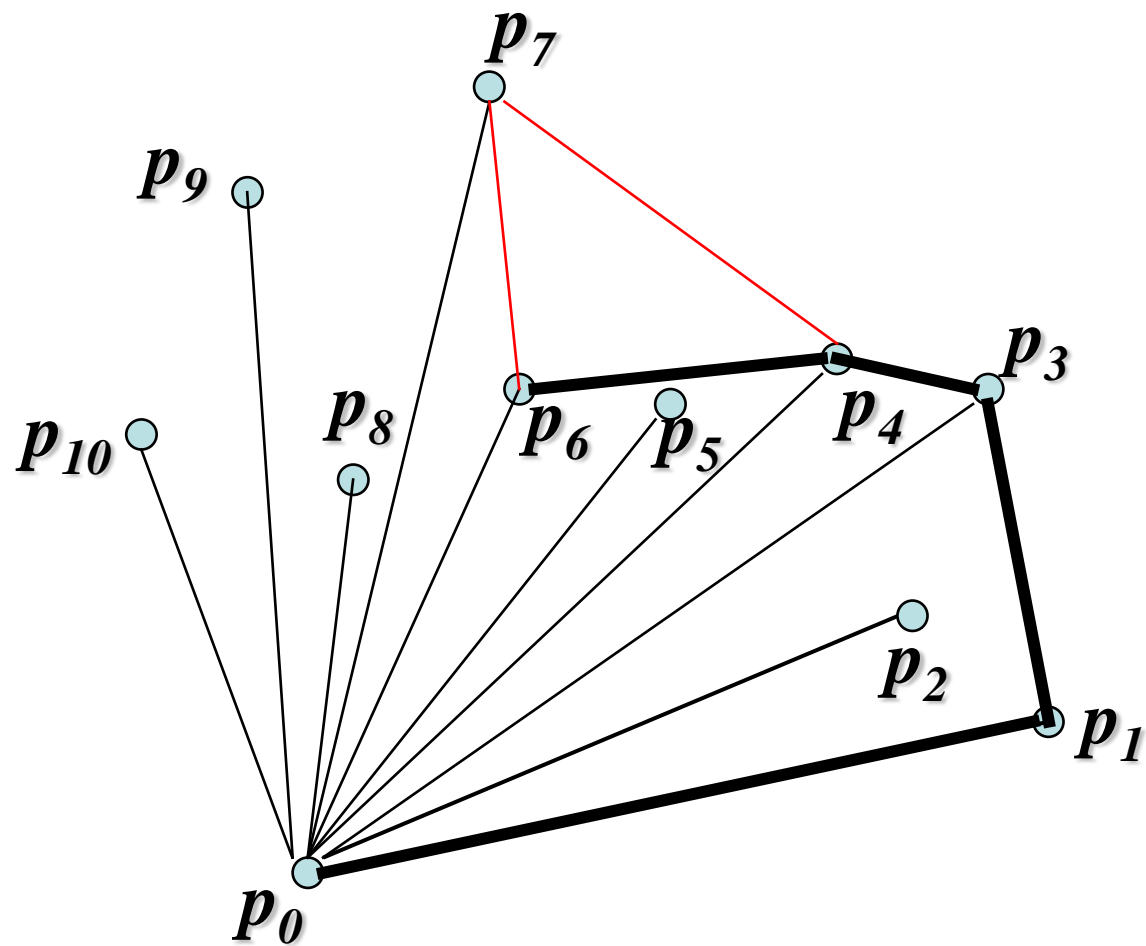


HITWH  
SE



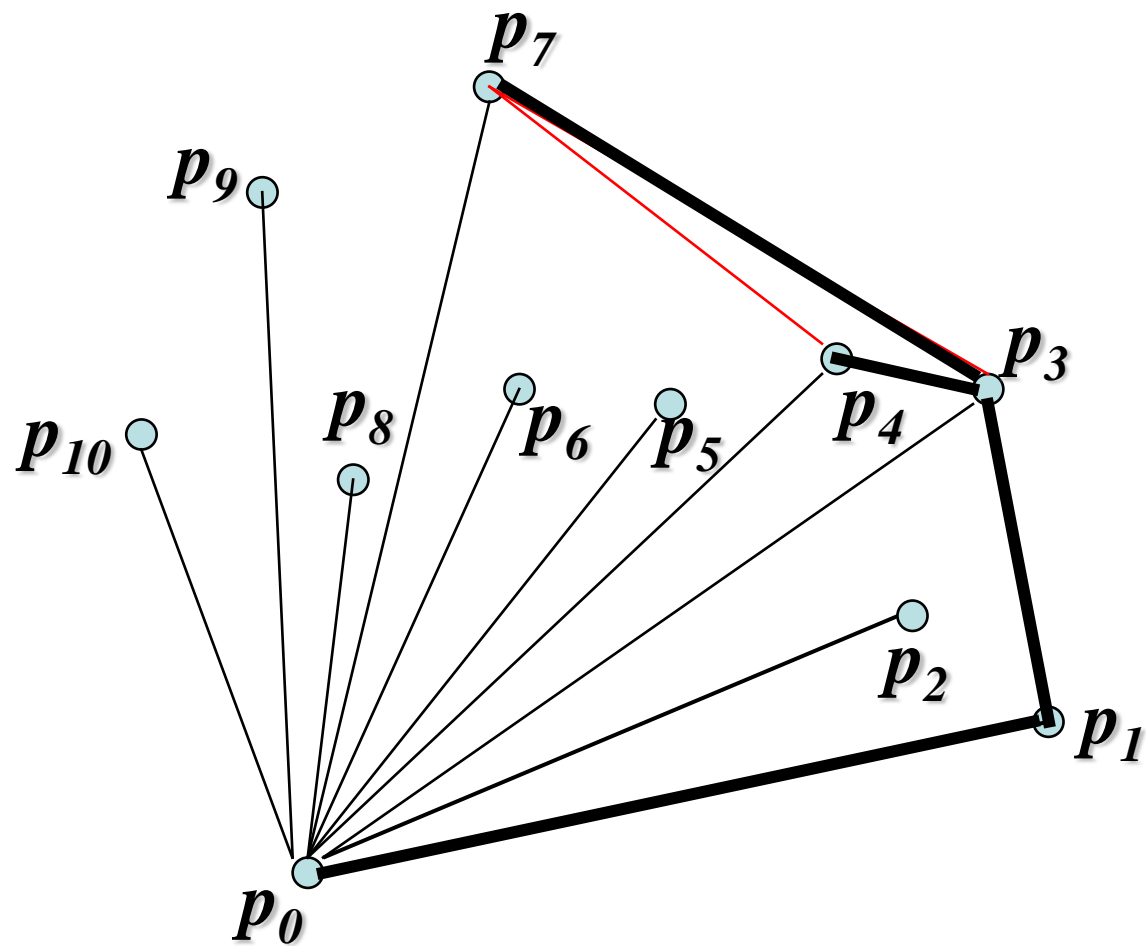


HITWH  
SE



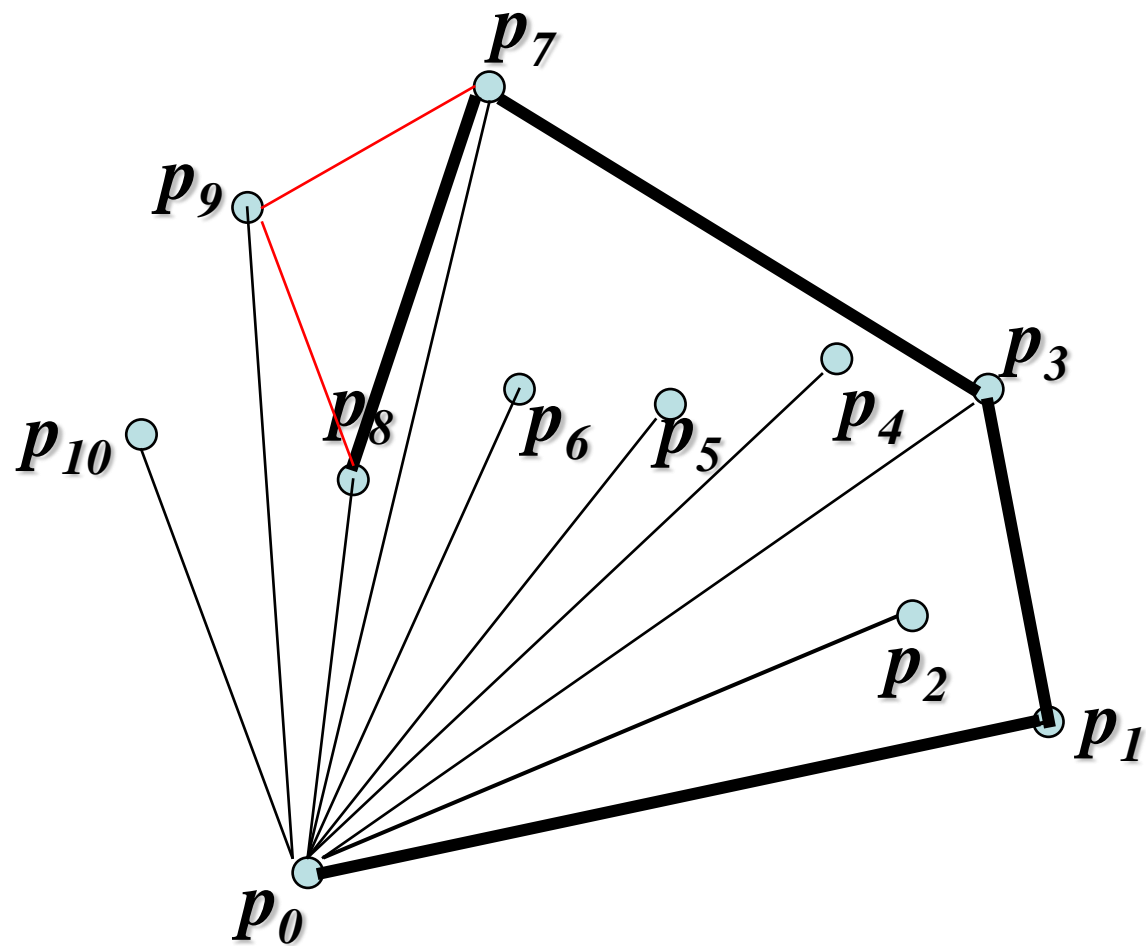


HITWH  
SE



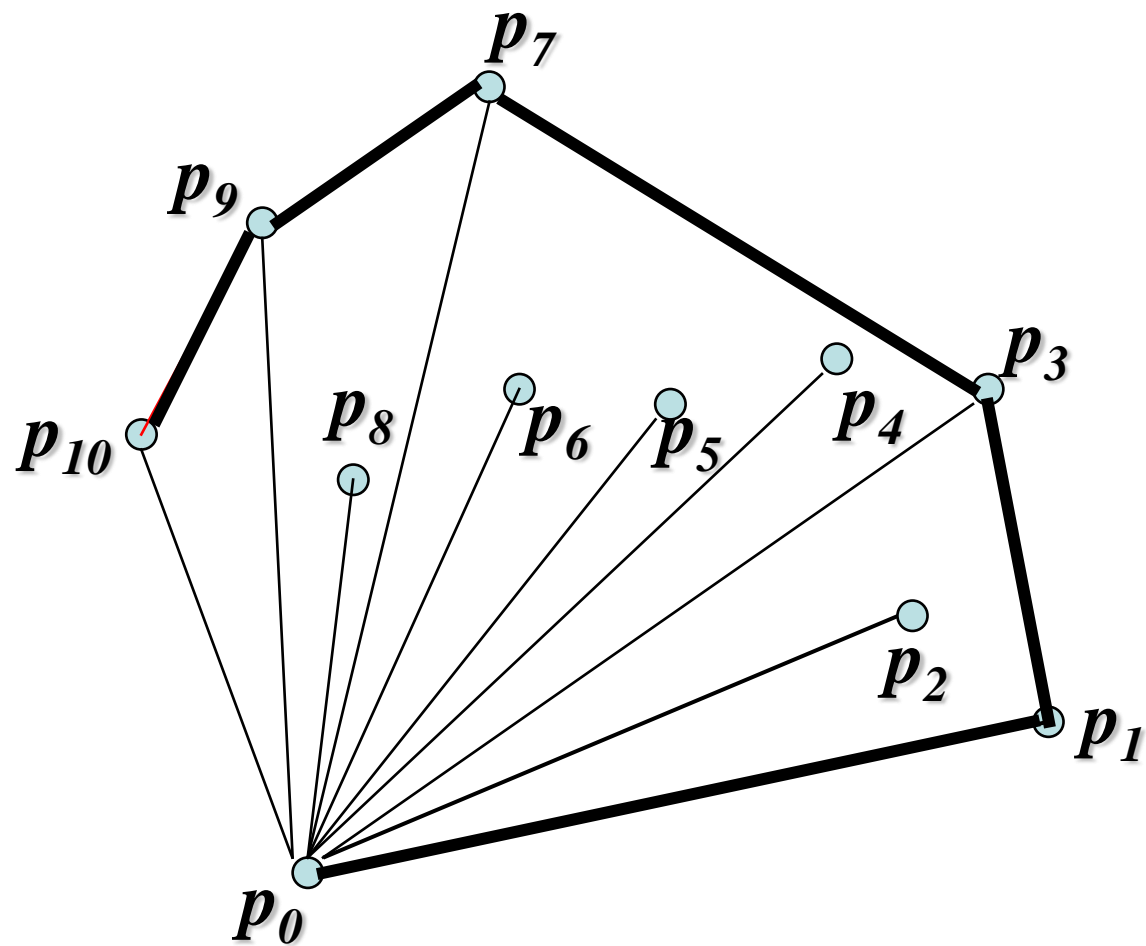


HITWH  
SE





HITWH  
SE

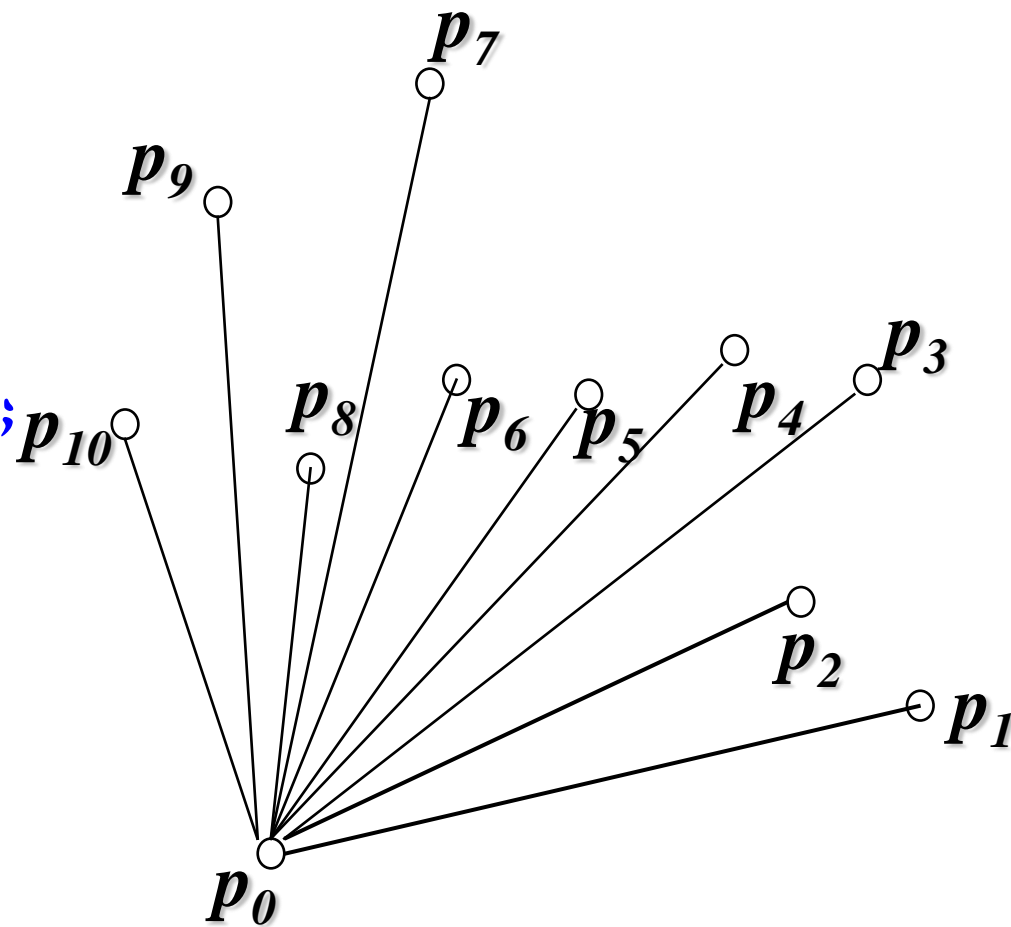




# 算法Graham-Scan( $Q$ )

/\* 栈 $S$ 从底到顶存储按逆时针  
方向排列的CH( $Q$ )顶点 \*/

1. 求 $Q$ 中 $y$ -坐标值最小的点 $p_0$ ;
2. 按照与 $p_0$ 极角(逆时针方向)  
大小排序 $Q$ 中其余点,  
结果为 $\langle p_1, p_2, \dots, p_n \rangle$ ;
3. Push  $p_0, p_1, p_2$  into  $S$ ;
4. FOR  $i=3$  TO  $n$  DO
5.     While Next-to-top( $S$ )、Top( $S$ )和 $p_i$ 形成非左移动 Do
6.         Pop( $S$ );
7.     Push( $p_i, S$ );
8. Rerurn  $S$ .





## • 时间复杂性 $T(n)$

1. 求  $Q$  中  $y$ -坐标值最小的点  $p_0$ ;
2. 按照与  $p_0$  极角(逆时针方向)大小排序  $Q$  中其余点, 结果为  $\langle p_1, p_2, \dots, p_n \rangle$ ;
3. Push  $p_0, p_1, p_2$  into  $S$ ;
4. **FOR**  $i=3$  **TO**  $n$  **DO**
5.     **While** Next-to-top( $S$ )、Top( $S$ )  
              和  $p_i$  形成非左移动 **Do**
6.             Pop( $S$ );
7.     Push( $p_i, S$ );
8. Rerurn  $S$ .

- 第1步需要  $O(n)$  时间
- 第2步需要  $O(n \log n)$  时间
- 第3步需要  $O(1)$  时间
- 第4-7步需要  $O(n)$  时间
  - 因为每个点至多进栈一次出栈一次, 每次需要常数计算时间
- $T(n) = O(n \log n)$



- 正确性分析

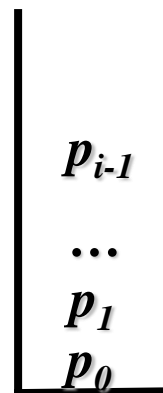
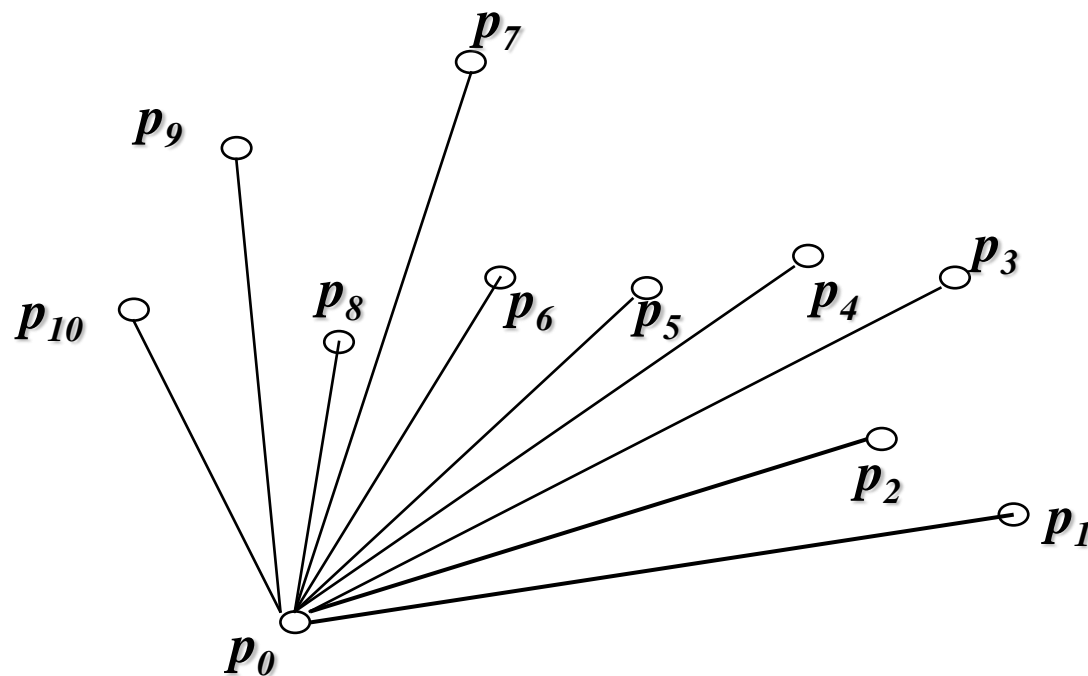
定理. 设 $n$ 个二维点的集合 $Q$ 是Graham-Scan算法的输入,  $|Q| \geq 3$ , 算法结束时, 栈 $S$ 中自底到顶存储 $CH(Q)$ 的顶点 (按照逆时针顺序) .

证明: 使用循环不变量方法

```
3. Push  $p_0, p_1, p_2$  into  $S$ ;  
4. FOR  $i=3$  TO  $n$  DO  
5.   While Next-to-top( $S$ )、Top( $S$ )  
      和  $p_i$  形成非左移动 Do  
6.     Pop( $S$ );  
7.   Push( $p_i, S$ );
```

## Loop invariant

在处理第  $i$  个顶点之前, 栈  $S$  中自底到顶存储  $CH(Q_{i-1})$  的顶点.



栈  $S$

```
3. Push  $p_0, p_1, p_2$  into  $S$ ;  
4. FOR  $i=3$  TO  $n$  DO  
5.   While Next-to-top( $S$ )、Top( $S$ )  
      和  $p_i$  形成非左移动 Do  
6.     Pop( $S$ );  
7.   Push( $p_i, S$ );
```

### 循环不变量

在处理第  $i$  个顶点之前, 栈  $S$   
自底到顶存储  $CH(Q_{i-1})$  的顶点.

## • Proof by induction

### – Initialization: (第3步)

- 处理  $i=3$  之前, 栈  $S$  中包含 了  $Q_{i-1}=Q_2=\{p_0, p_1, p_2\}$  中的顶点, 这三个点形成了一个  $CH$ . 循环不变量为真.

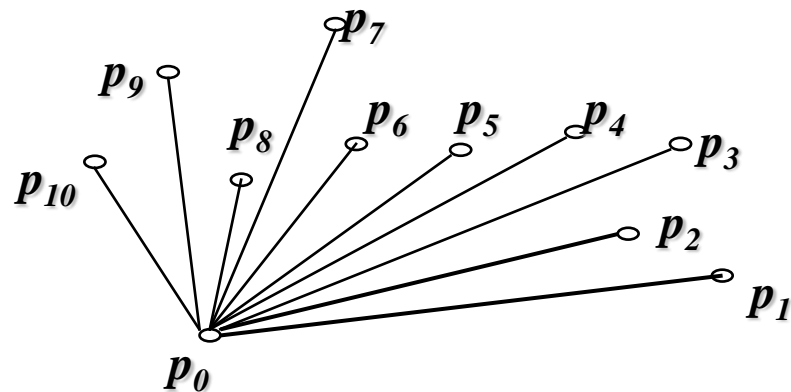
### – Maintenance:

- 设在处理第  $i (i \geq 3)$  个顶点之前, 循环不变量为真, 即: 栈  $S$  中自底到顶存储  $CH(Q_{i-1})$  的顶点.
- 往证:  
算法执行5~7步之后, 栈  $S$  中自底到顶存储  $CH(Q_i)$  的顶点.

```

3. Push  $p_0, p_1, p_2$  into  $S$ ;
4. FOR  $i=3$  TO  $n$  DO
5.   While Next-to-top( $S$ )、Top( $S$ )
      和  $p_i$  形成非左移动 Do
6.     Pop( $S$ );
7.   Push( $p_i, S$ );

```



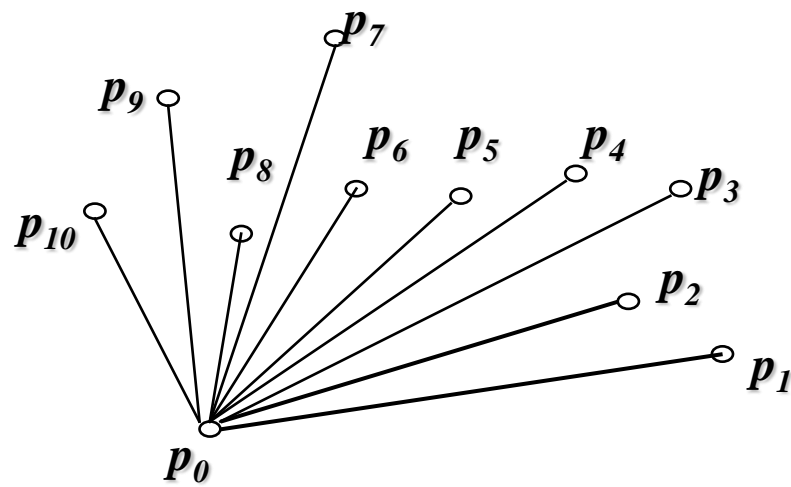
• 往证：算法执行5~7步后，栈 $S$ 中自底到顶存储 $CH(Q_i)$ 的顶点

- 5~6步while循环执行结束后，第7步将 $p_i$ 压入栈之前，设栈顶元素为 $p_j$ ，次栈顶元素为 $p_k$ ，则此时，**栈中包含了与for循环的第 $j$ 轮迭代后相同的顶点，即 $CH(Q_j)$** ，因为第 $j$ 轮迭代后循环不变量为真。
- 执行第7步之后， $p_i$ 入栈，则栈 $S$ 中包含了 $CH(Q_j \cup \{p_i\})$ 中的顶点，且这些点仍按逆时针顺序，自底向上出现在栈中。 **$CH(Q_j \cup \{p_i\}) = CH(Q_j)$ ?**
- 对于任意一个在第 $i$ 轮迭代中被弹出的栈顶点 $p_t$ ，设 $p_r$ 为紧靠 $p_t$ 的次栈顶点， $p_t$ 被弹出当且仅当 $p_r, p_t, p_i$ 构成非左移动。因此， $p_t$ 不是 $CH(Q_i)$ 的一个顶点，即 **$CH(Q_i - \{p_t\}) = CH(Q_i)$** ，以此类推，可知第 $i$ 轮迭代中被弹出的点都不是 $CH(Q_i)$ 的一个顶点。
- 设 $P_i$ 为for循环第 $i$ 轮迭代中被弹出的所有点的集合，则有 **$CH(Q_i - P_i) = CH(Q_i)$** 。
- 对于 **$j+1 \leq t \leq i-1$** ，由于第 $t$ 轮迭代后循环不变量为真，可知 $P_t$ 不在 $CH(Q_t)$ 中，继而必然不在 $CH(Q_i)$ 中，因此 **$CH(Q_i) = CH(Q_i - P_i) = CH(Q_i - P_i - \bigcup_{t=j+1}^{i-1} P_t)$** 。
- 又 **$Q_i - P_i - \bigcup_{t=j+1}^{i-1} P_t = Q_j \cup \{p_i\}$** ，故有 **$CH(Q_j \cup \{p_i\}) = CH(Q_i - P_i - \bigcup_{t=j+1}^{i-1} P_t) = CH(Q_i)$**
- 即得到：一旦将 $p_i$ 压入栈后，栈 $S$ 中恰包含 $CH(Q_i)$ 中的顶点，且按照逆时针顺序，自底向上排列。

```

3. Push  $p_0, p_1, p_2$  into  $S$ ;
4. FOR  $i=3$  TO  $n$  DO
5.   While Next-to-top( $S$ )、Top( $S$ )
      和  $p_i$  形成非左移动 Do
6.     Pop( $S$ );
7.   Push( $p_i, S$ );

```



– Termination:

- $i=n+1$ , 栈  $S$  中自底到顶存储  $CH(Q_n)$  的顶点, 算法正确.

证毕.



# Divide-and-conquer 算法

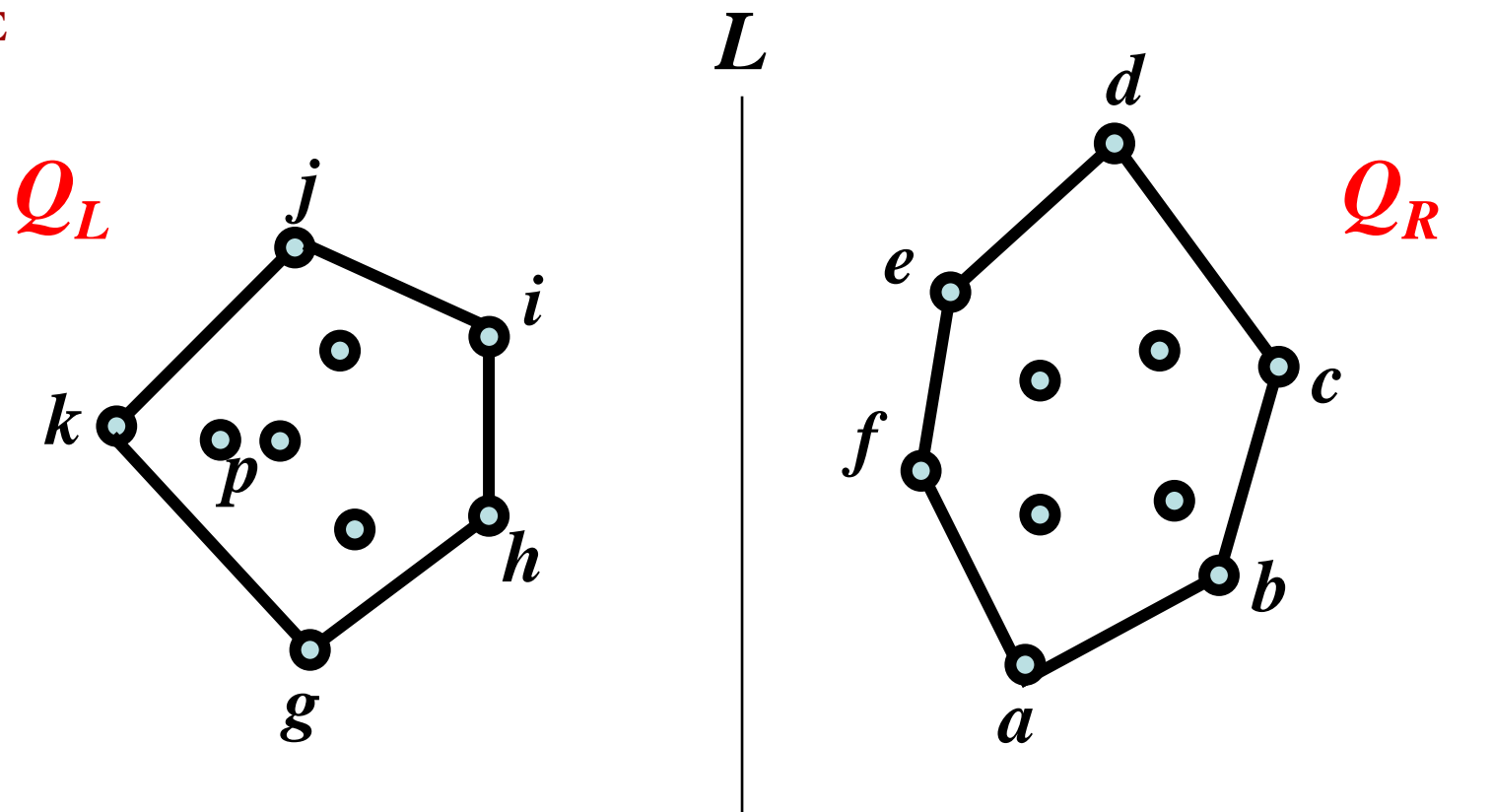
边界条件：(时间复杂性为  $O(1)$ )

1. 如果  $|Q| < 3$ , 算法停止;
2. 如果  $|Q| = 3$ , 按照逆时针方向输出  $CH(Q)$  的顶点;
3. 如果  $|Q| < 5$ , 使用 Graham-Scan 求  $CH(Q)$ ;

Divide: (使用  $O(n)$  算法求中值)

1. 选择一个垂直于  $x$ -轴的直线把  $Q$  划分为基本相等的两个集合  $Q_L$  和  $Q_R$ ,  $Q_L$  在  $Q_R$  的左边;





**Conquer:** (时间复杂度为  $2T(n/2)$ )

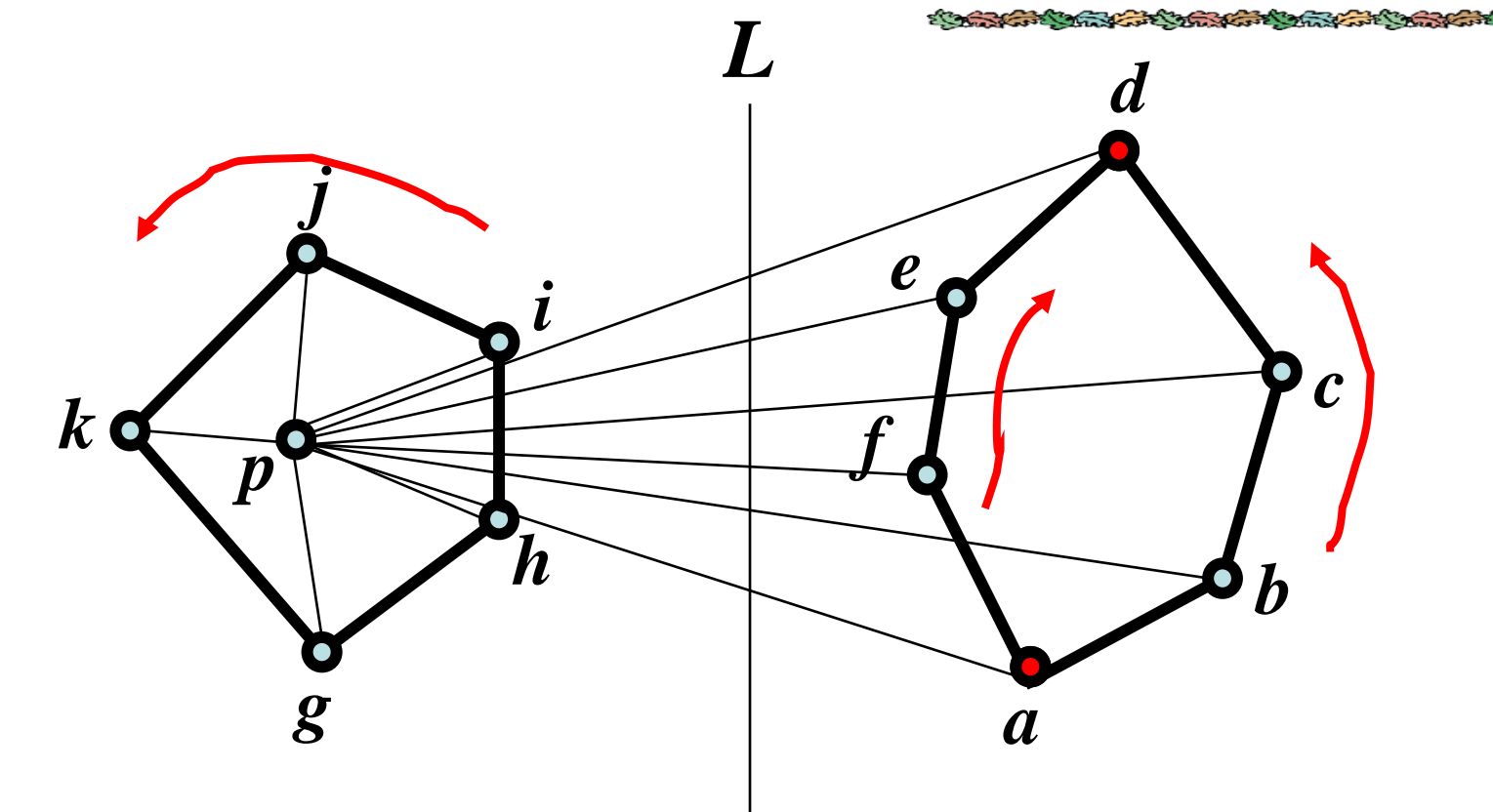
1. 递归地为  $Q_L$  和  $Q_R$  构造  $CH(Q_L)$  和  $CH(Q_R)$ ;



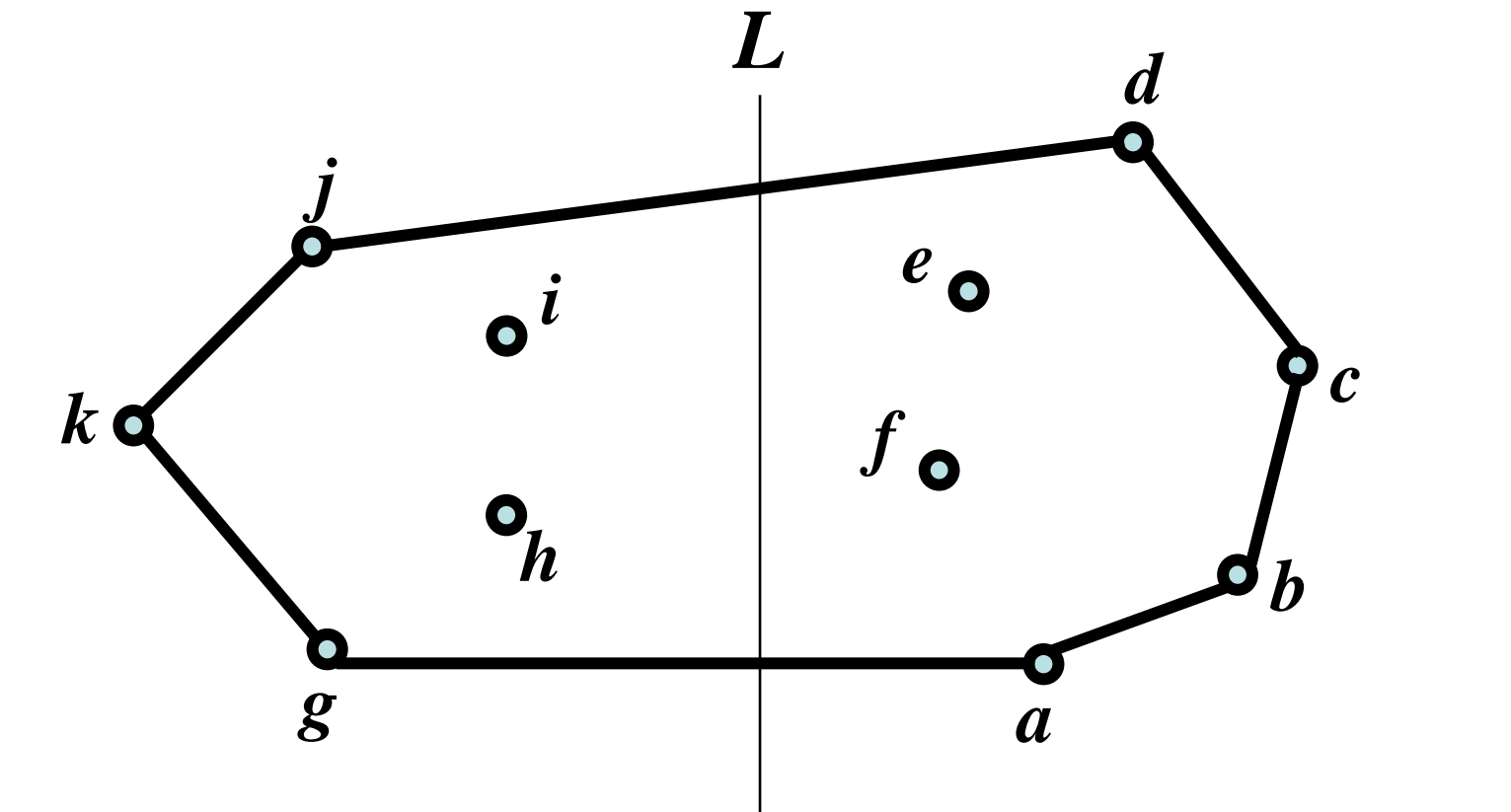
HITWH  
SE

Merge:

我们先通过一个例子来看Merge的思想



3个序列:  $\langle g, h, i, j, k \rangle$ ,  $\langle a, b, c, d \rangle$ ,  $\langle f, e \rangle$   
合并以后:  $\langle g, h, a, b, f, c, e, d, i, j, k \rangle$



选择p点为y坐标最小的点  
在合并的序列上运行Graham-Scan, 排序部  
分合并三个有序序列即可



**Merge:**(时间复杂性为 $O(n)$ )

1. 找到 $Q_L$ 和 $Q_R$ 中 $y$ 坐标最小的点 $p$  (假设在 $Q_L$ 中);
2. 在 $CH(Q_R)$ 中找与 $p$ 的极角最大和最小顶点 $u$ 和 $v$ ;
3. 构造如下三个点序列:
  - (1) 按逆时针方向排列的 $CH(Q_L)$ 的所有顶点,
  - (2) 按逆时针方向排列的 $CH(Q_R)$ 从 $v$ 到 $u$ 的顶点,
  - (3) 按顺时针方向排列的 $CH(Q_R)$ 从 $v$ 到 $u$ 的顶点;
4. 合并上述三个序列;
5. 在合并的序列上应用Graham-Scan.



- Preprocessing 阶段
  - $O(1)$
- Divide 阶段(使用  $O(n)$  算法求中值)
  - $O(n)$
- Conquer 阶段
  - $2T(n/2)$
- Merge 阶段
  - $O(n)$



- 总的时间复杂性

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

- 使用Master定理

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



HITWH  
SE

## 3.7 快速傅里叶变换





输入:  $a_0, a_1, \dots, a_{n-1}$ ,  $a_i$  是实数,  $(0 \leq i \leq n-1)$

输出:  $A_0, A_1, \dots, A_{n-1}$ , 使得,

$$A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi i j k}{n}}$$

其中:

(2)  $0 \leq j \leq n-1$

(3)  $e$  是自然对数的底数

(4)  $i = \sqrt{-1}$  是虚数单位

蛮力法利用定义计算每个  $A_j$ , 时间复杂度为  $\Theta(n^2)$



$$A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi ijk}{n}} \quad \text{令 } w_n = e^{\frac{2\pi i}{n}}, \quad \text{有: } A_j = \sum_{k=0}^{n-1} a_k w_n^{jk}$$

$$\begin{aligned} A_j &= a_0 + a_1 w_n^j + a_2 w_n^{2j} + a_3 w_n^{3j} + a_4 w_n^{4j} + \dots + a_{n-2} w_n^{(n-2)j} + a_{n-1} w_n^{(n-1)j} \\ &= (a_0 + a_2 w_n^{2j} + a_4 w_n^{4j} + \dots + a_{n-2} w_n^{(n-2)j}) \\ &\quad + (a_1 w_n^j + a_3 w_n^{3j} + a_5 w_n^{5j} + \dots + a_{n-1} w_n^{(n-1)j}) \\ &= (a_0 + a_2 w_n^{2j} + a_4 w_n^{4j} + \dots + a_{n-2} w_n^{(n-2)j}) \\ &\quad + w_n^j (a_1 + a_3 w_n^{2j} + a_5 w_n^{4j} + \dots + a_{n-1} w_n^{(n-2)j}) \end{aligned}$$



# 算法的数学基础

$$A_j = (a_0 + a_2 w_n^{2j} + a_4 w_n^{4j} + \dots + a_{n-2} w_n^{(n-2)j})$$

奇数输入项的FFT

$$+ w_n^j (a_1 + a_3 w_n^{2j} + a_5 w_n^{4j} + \dots + a_{n-1} w_n^{(n-2)j})$$

偶数输入项的FFT

由于  $w_n^2 = w_{n/2}$ , 以及  $w_n^{n+k} = w_n^k$ ,  $e^{i \cdot \text{seta}} = \cos(\text{seta}) + i \cdot \sin(\text{seta})$

$$A_j = (\underbrace{a_0 + a_2 w_{n/2}^j + a_4 w_{n/2}^{2j} + \dots + a_{n-2} w_{n/2}^{(n-2)j}}_{B_j})$$

将问题划分为  
两个子问题

$$+ w_n^j (\underbrace{a_1 + a_3 w_{n/2}^j + a_5 w_{n/2}^{2j} + \dots + a_{n-1} w_{n/2}^{(n-2)j}}_{C_j})$$

于是,  $A_j = B_j + w_n^j C_j$  还可证明,  $A_{j+n/2} = B_j + w_n^{j+n/2} C_j$



# 分治算法过程



划分：将输入拆分成 $a_0, a_2, \dots, a_{n-2}$ 和 $a_1, a_3, \dots, a_{n-1}$ 。

递归求解：递归计算 $a_0, a_2, \dots, a_{n-2}$ 的变换 $B_0, B_1, \dots, B_{n/2-1}$

递归计算 $a_1, a_3, \dots, a_{n-1}$ 的变换 $C_0, C_1, \dots, C_{n/2-1}$

合并：根据 $A_j = B_j + C_j \cdot W_n^j$  ( $j < n/2$ ) 和  $A_j = B_{j-n/2} + C_{j-n/2} \cdot W_n^j$  ( $n/2 \leq j < n-1$ ) 依次求得 $A_0, A_1, \dots, A_{n-1}$ 。



# 分治算法过程

例如：计算8个点 $a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$ 的FFT.

计算4个点  
 $a_0, a_2, a_4, a_6$ 的FFT.

计算4个点  
 $a_1, a_3, a_5, a_7$ 的FFT.

计算2个点  
 $a_0, a_4$ 的FFT.

计算2个点  
 $a_2, a_6$ 的FFT.

计算2个点  
 $a_1, a_5$ 的FFT.

计算2个点  
 $a_3, a_7$ 的FFT.



HITWH  
SE

# 算法及复杂性分析

算法 FFT( $a_0, a_2, \dots, a_{n-1}, n$ )

输入:  $a_0, a_1, \dots, a_{n-1}, n=2^k$

输出:  $a_0, a_1, \dots, a_{n-1}$  的傅里叶变换  $A_0, \dots, A_{n-1}$

1.  $W \leftarrow \exp(2\pi i/n)$ ;

2. If ( $n=2$ ) Then

$T(n) = \theta(1)$  If  $n=2$

3.  $A_0 \leftarrow a_0 + a_1$ ;

$T(n) = 2T(n/2) + \theta(n)$  If  $n > 2$

4.  $A_1 \leftarrow a_0 - a_1$ ;

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

5. 输出  $A_0, A_1$ , 算法结束;

6.  $B_0, B_1, \dots, B_{n/2-1} \leftarrow \text{FFT}(a_0, a_2, \dots, a_{n-2}, n/2)$ ;

7.  $C_0, C_1, \dots, C_{n/2-1} \leftarrow \text{FFT}(a_1, a_3, \dots, a_{n-1}, n/2)$ ;

8. For  $j=0$  To  $n/2-1$

9.  $A_j \leftarrow B_j + C_j \cdot W^j$ ;

10.  $A_{j+n/2} \leftarrow B_j - C_j \cdot W^j$ ;

11. 输出  $A_0, A_1, \dots, A_{n-1}$ , 算法结束;



HITWH  
SE

$$\begin{aligned} T(n) &= \theta(1) && \text{if } n \leq c \\ T(n) &= aT(n/b) + D(n) + C(n) && \text{if } n > c \end{aligned}$$

## 3.8 整数乘法

优化划分阶段, 降低  $T(n) = aT(n/b) + f(n)$  中的  $a$



# 问题定义

输入：n位二进制整数X和Y

输出：X和Y的乘积

通常，计算 $X*Y$ 时间复杂度为 $O(n^2)$ ，  
我们给出一个复杂度为 $O(n^{1.59})$ 的算法。





$$X = \begin{array}{|c|c|} \hline \text{n/2位} & \text{n/2位} \\ \hline A & B \\ \hline \end{array}$$

$$Y = \begin{array}{|c|c|} \hline \text{n/2位} & \text{n/2位} \\ \hline C & D \\ \hline \end{array}$$

$$\begin{aligned} XY &= (A2^{n/2} + B)(C2^{n/2} + D) \\ &= AC2^n + AD2^{n/2} + BC2^{n/2} + BD \\ &= AC2^n + ((A-B)(D-C) + AC + BD)2^{n/2} + BD \end{aligned}$$

时间复杂性

$$T(n) = \theta(1)$$

if  $n=1$

$$T(n) = 3T(n/2) + O(n) \quad \text{if } n > 1 \quad \text{使用 Master 定理}$$

$$T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

如此计算需要

$$T(n) = 4T(n/2) + O(n) = O(n^2)$$



HITWH  
SE

针对如下问题，设计分治算法。

输入：包含 $n$ 个整数的集合 $S$ ，整数 $x$ ；

输出：判定 $S$ 中是否存在两个和为 $x$ 的元素。



给定一棵树 $T$ ，其中每个顶点 $n$ 的代价记为 $c(n)$ ，树 $T$ 中一个顶点集合 $S$ 的代价为 $C(S) = \sum_{n \in S} c(n)$ 。记树 $T$ 的顶点集合为 $V(T)$ ， $T$ 的切面定义如下：

一个顶点集合 $S$ 称为树 $T$ 的一个切面，如果 $S$ 满足以下两个条件：

(1)  $S \subseteq V(T)$ ;

(2) 从 $T$ 的根顶点到任意叶子顶点的路径上有且只有一个顶点属于 $S$ 。

请设计算法，求解 $T$ 的一个切面 $S$ 使得 $C(S)$ 最小，并分析算法的时间开销。