HIT
CS&E

第三章

分治算法的
设计与分析原理

程思瑶
海量数据计算研究中心

参考资料

R.C.T.Lee, S.S.Tseng, R.C.Chang, and Y.T.Tsai,

Introduction to
the Design and Analysis of Algorithms

Chapter 4

Introduction to Algorithms

Chapter 6~9

提要

3.1 分治算法原理
3.2 整数乘法
3.3 最近点对发现算法
3.4 凸包(convex hull)构建算法
3.5 分位数选择算法
3.6 快速排序

3.1 分治算法原理

- 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
  - Conquer：求解各子问题(递归调用算法)
  - Combine：合并子问题的解, 形成原始问题的解

M-Sort(*A*, *p*, *r*)

$q=p+(r-p+1)/2$;

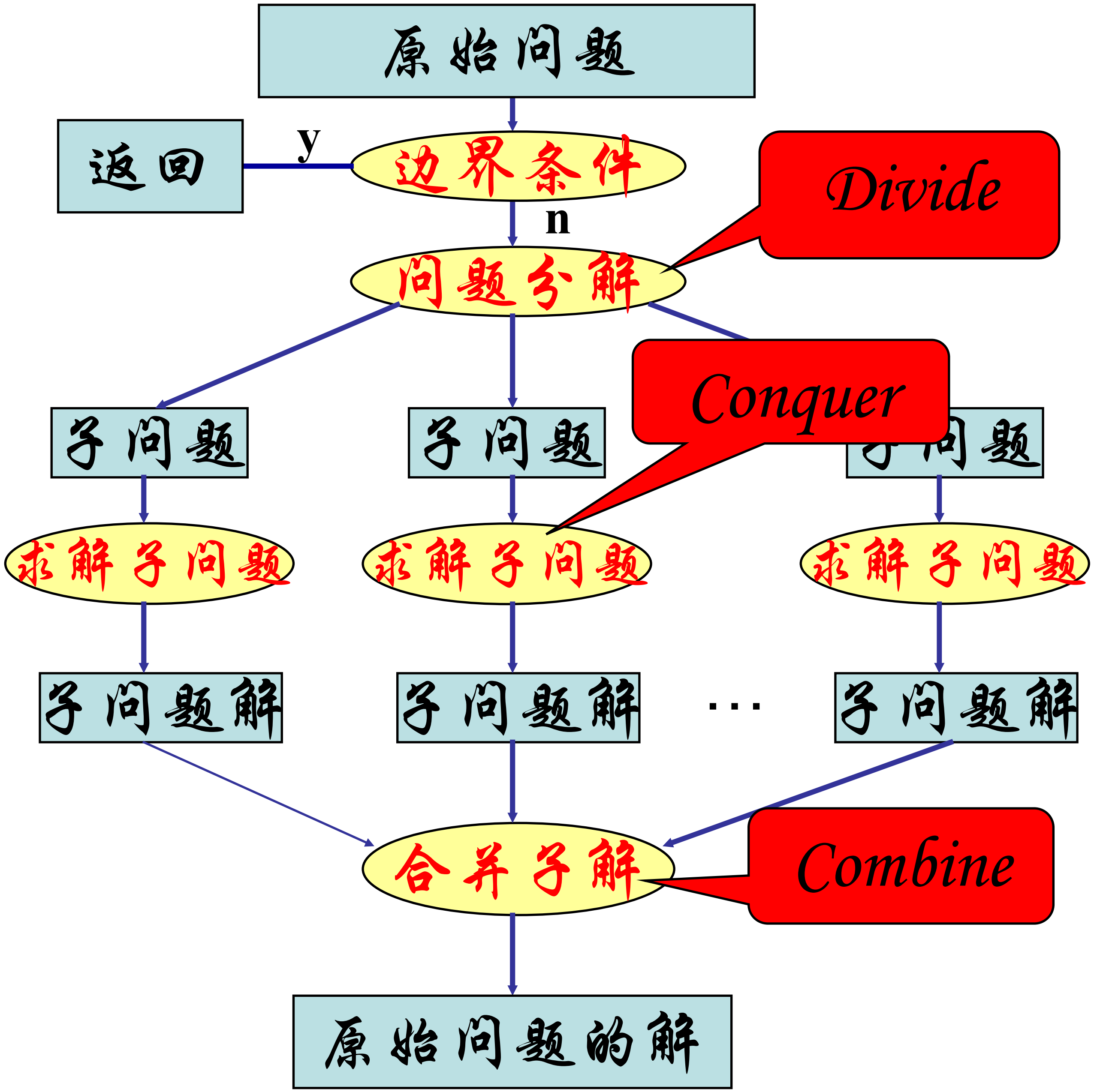
If $p<r$

Then

  B=M-sort(*A*, *p*, *q-1*);

  C=M-sort(*A*,*q*,*r*);

Merge(C, B).



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$=C(n)$

优化

最后得到递归方程：

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

HIT
CS&E

$$T(n)=\theta(1) \qquad \text{if } n \le c$$
$$T(n)=aT(n/b)+D(n)+C(n) \qquad \text{if } n > c$$

3.2 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时间复杂性为$O(n^2)$，
我们给出一个复杂性为$O(n^{1.59})$的算法。

算法的思想

	n/2位	n/2位
X=	A	B

	n/2位	n/2位
Y=	C	D

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

T(*n*)=4

时间复杂性

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

$T(n)=3T(n/2)+O(n)$ if $n>1$

使用Master定理

$T(n)=O(n^{log3})$

$=O(n^{1.59})$

- 算法

1. 计算A-B和D-C
2. 计算n/2位乘法AC、BD、(A-B)(D-C)
3. 计算M=(A-B)(D-C)+AC+BD
4. N=AC左移n位，M=M左移n/2位
5. 计算XY=N+M+BD

$$T(n)=\theta(1) \quad \text{if } n \le c$$
$$T(n)=aT(n/b)+D(n)+C(n) \quad \text{if } n > c$$

3.3 最近点对发现算法

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

问题定义

输入：Euclidean空间上的n个点的集合Q

输出： $A, B \in Q$,

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

$Dis(P_i, P_j)$是Euclidean距离：

如果Ｓ$_i=(x_i, y_i), P_j=(x_j, y_j)$，则

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

一维空间算法

- 利用排序的算法
  - 算法
    - 把Q中的点排序

    - 通过排序集合的线性扫描找出最近点对
  - 时间复杂性
    - $T(n)=O(nlogn)$

一维空间算法(续)

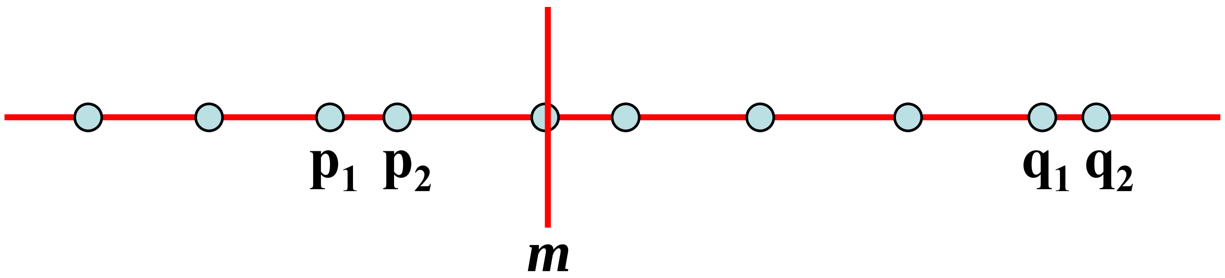
- Divide-and-conquer算法

边界条件:

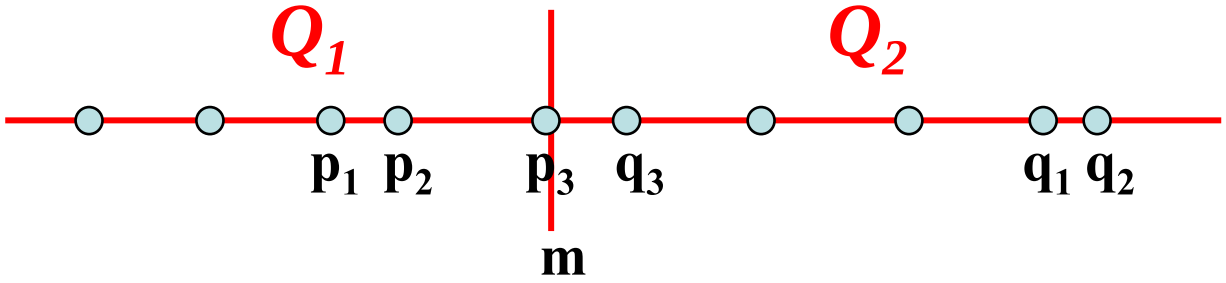
1. 如果Q中仅包含2个点，则返回这个点对；

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$Q_1=\{x\in Q \mid x\le m\}$，$Q_2=\{x\in Q \mid x>m\}$

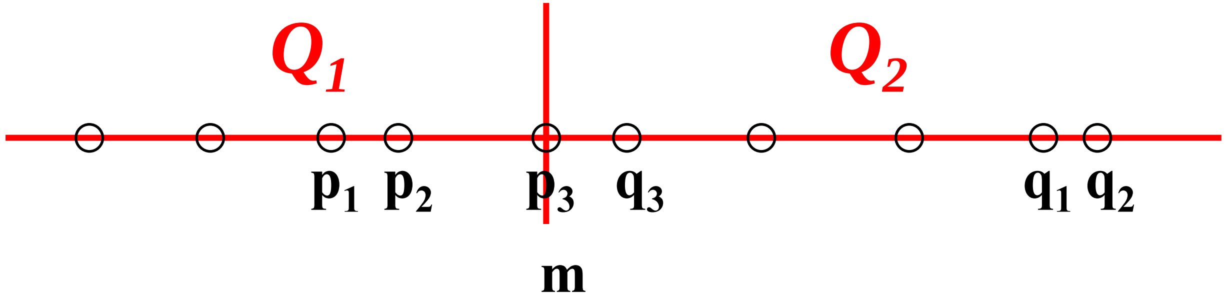


Conquer:

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$O(1)$时间
  - 递归方程

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

  - 用Master定理求解$T(n)$

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

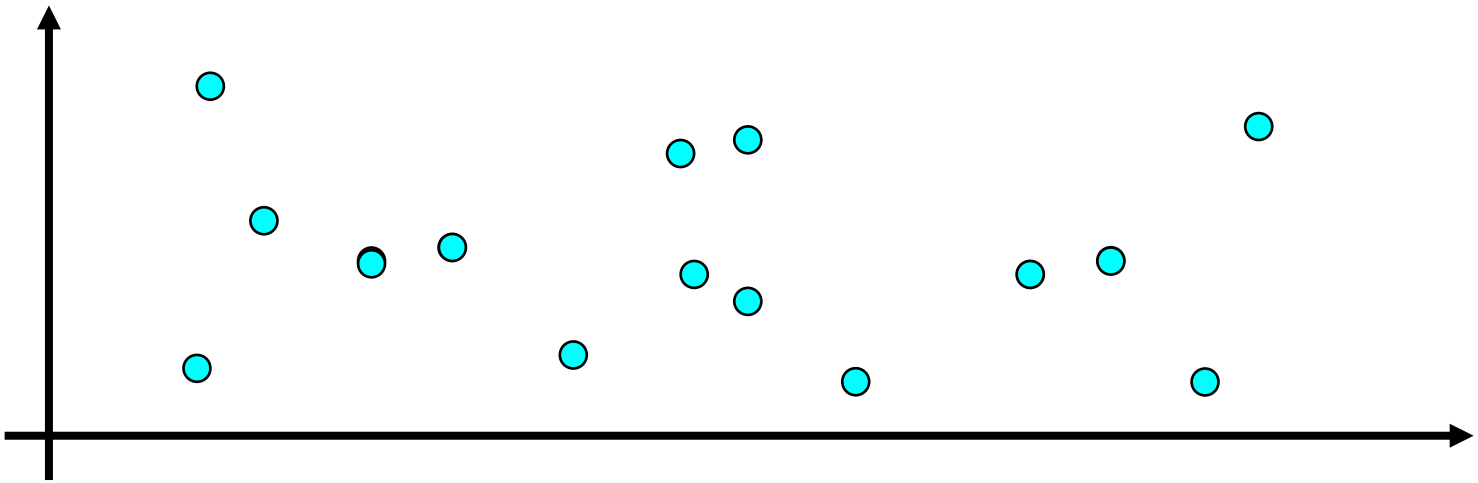
二维空间算法

- 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.

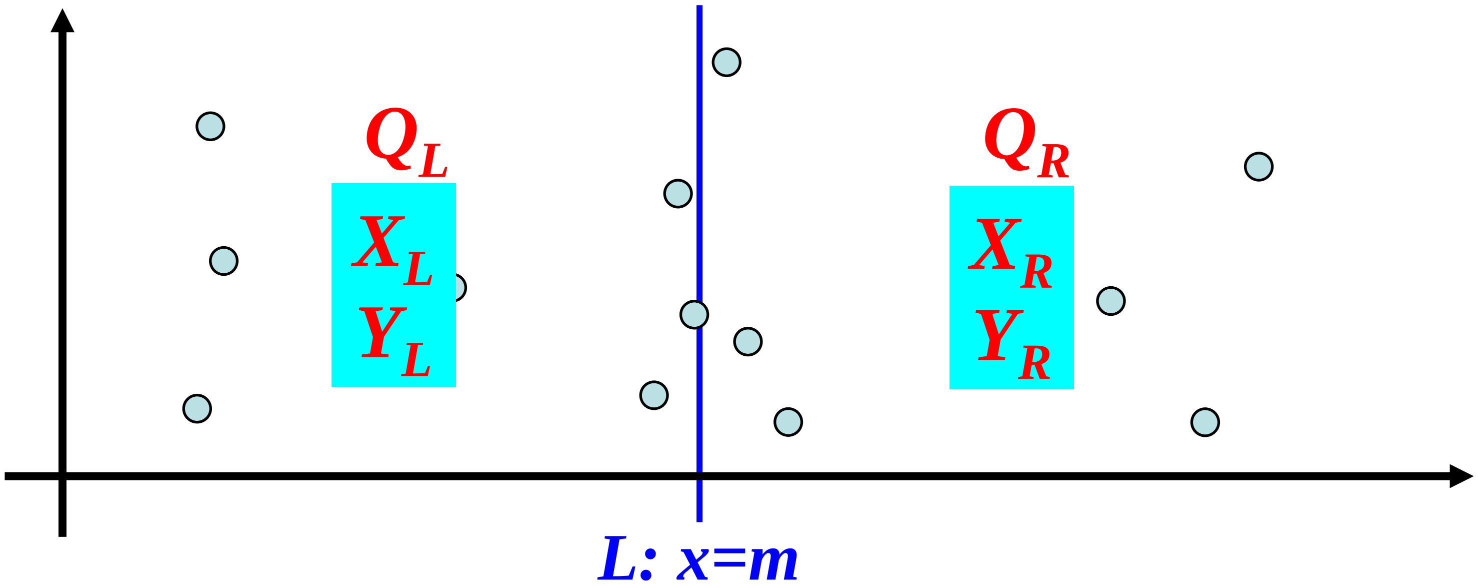
边界 条件:

1. 如果Q中仅包含3个点,则返回最近点对,结束;

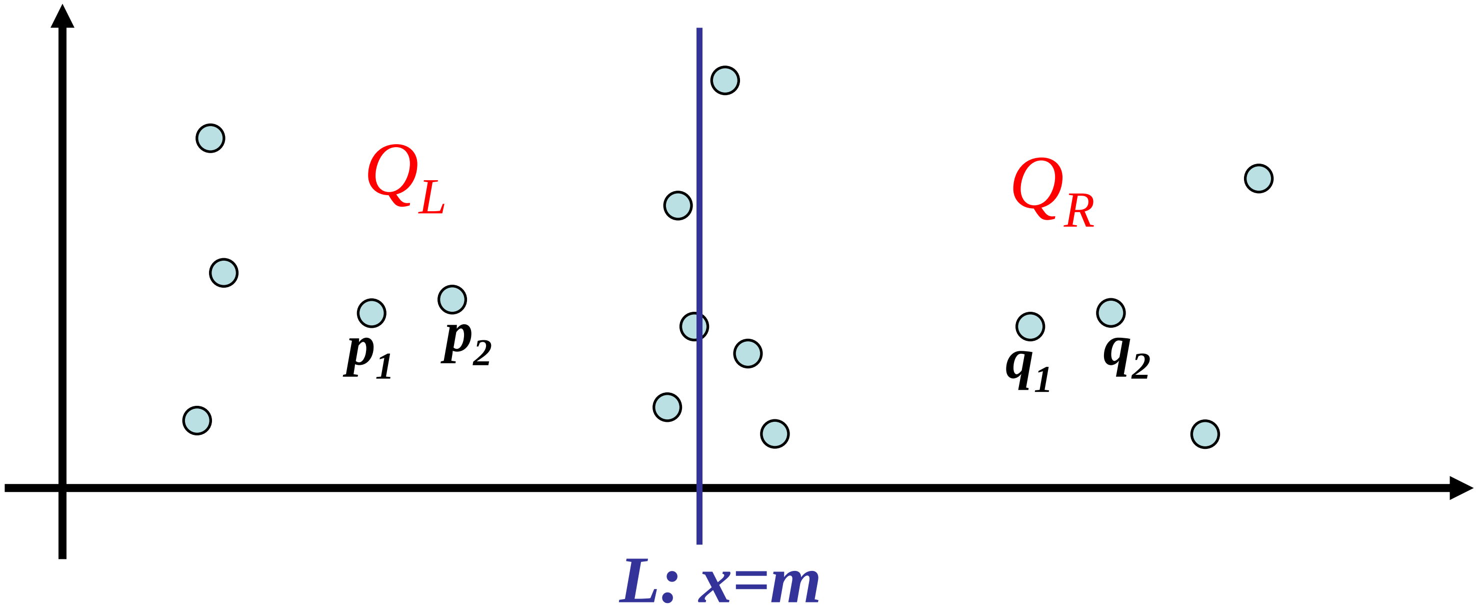


Divide:

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；



HIT
CS&E

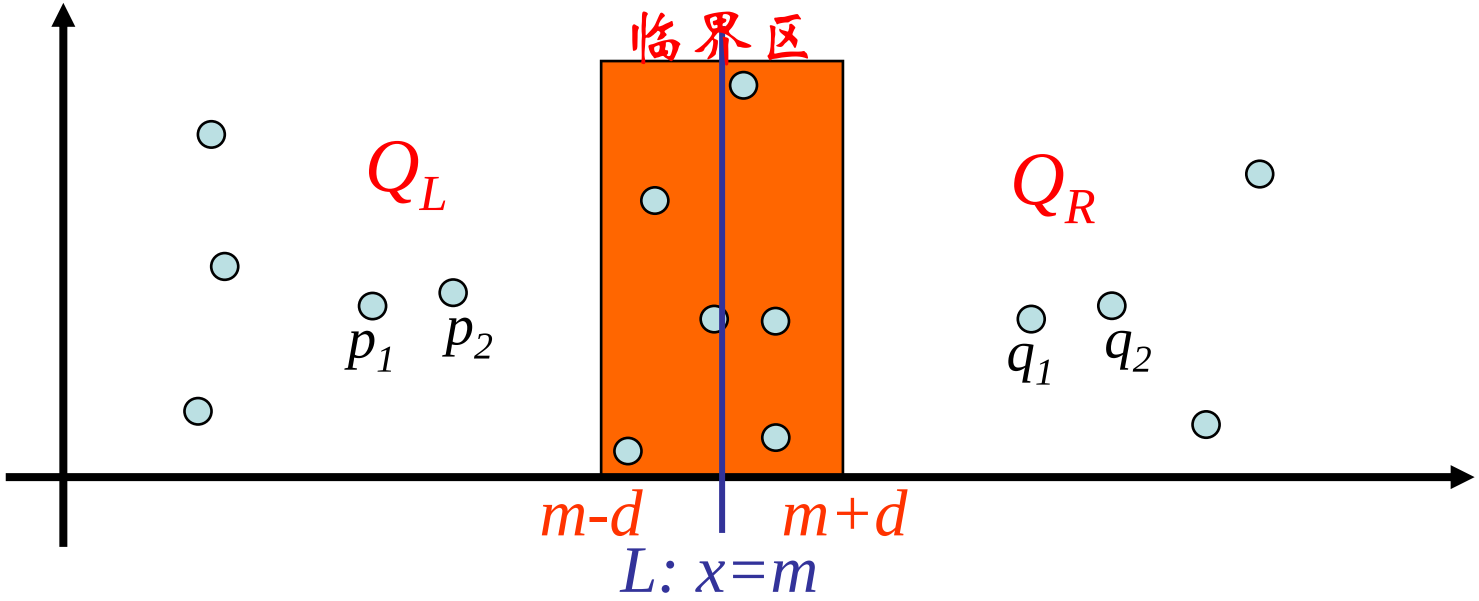


Conquer:

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

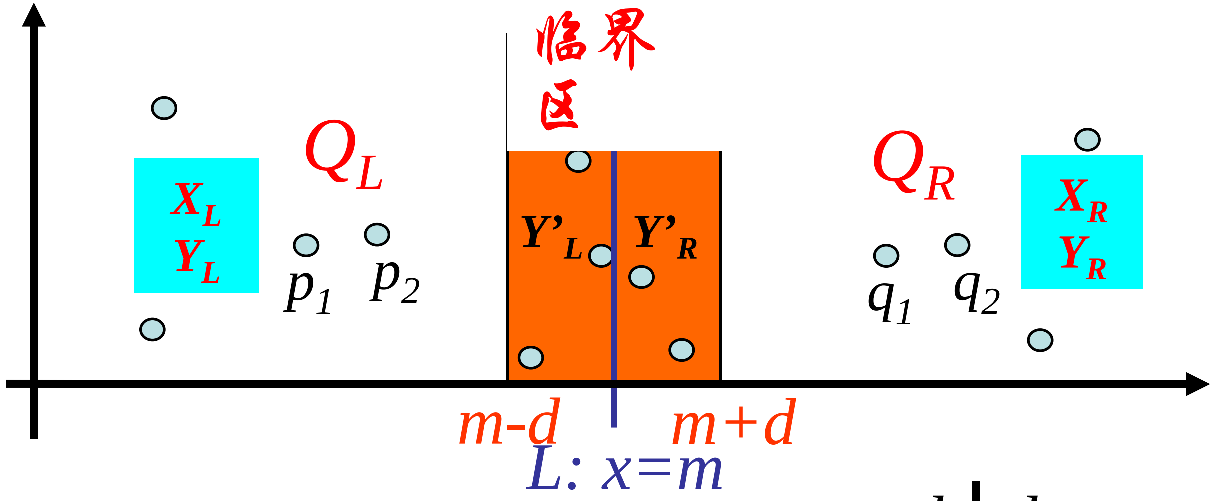


Merge:

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)是临界区中最近点对，否则(p_1, p_2)和(q_1, q_2) 中距离最小者为Q中最近点对.

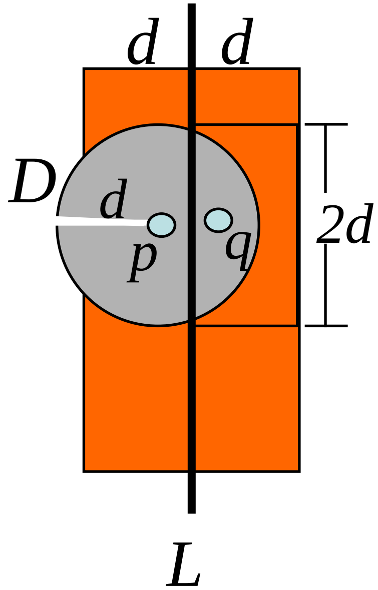
关键是(p_l, q_r)的搜索方法及其搜索时间

- 时间复杂性
  $O(6n)=O(n)$



- (p_l, q_r)搜索算法

1. $Q'_L = Q_L$ - {非临界区点};
   $Q'_R = Q_R$ - {非临界区点};
2. For $\forall p(x_p, y_p) \in Q'_L$ Do
3. For $\forall q(x_q, y_q) \in Q'_R$ $(y_p - d \le y_q \le y_p + d)$ Do
   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_l, 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

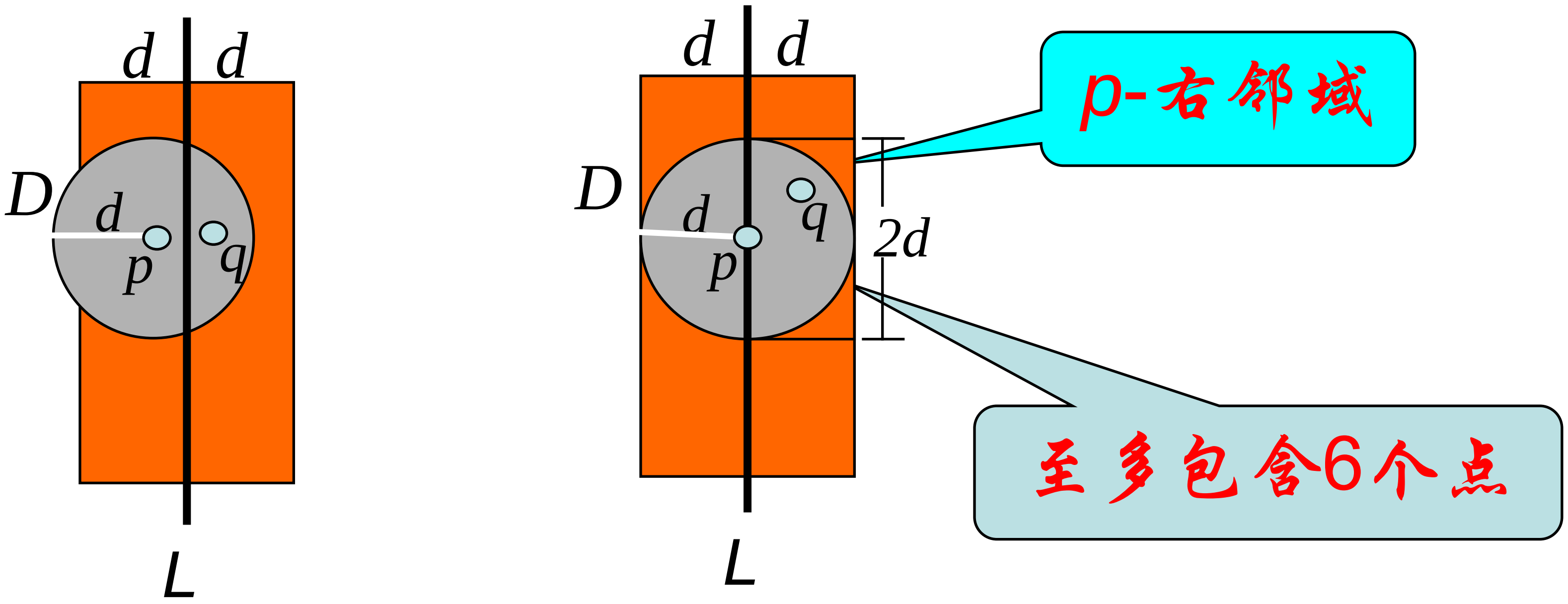
$$T(n) = \begin{cases} O(1) & n \le 3 \\ 2T(n/2) + O(n) & n > 3 \end{cases}$$

  - 用Master定理求解$T(n)$

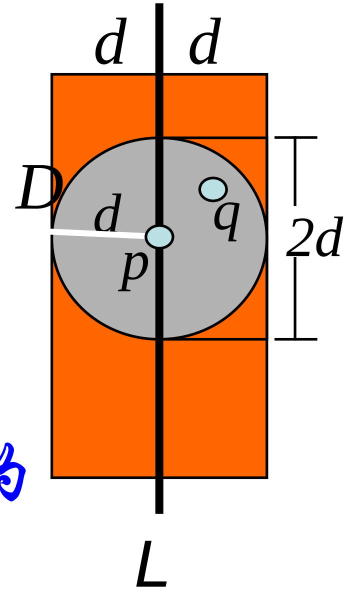
$$T(n) = O(n\log n)$$

(p_l, q_r)的搜索时间：

- 若(p, q)是最近点对而且$p\in Q_L$, $q\in Q_R$, $dis(p, q)<d$，(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.



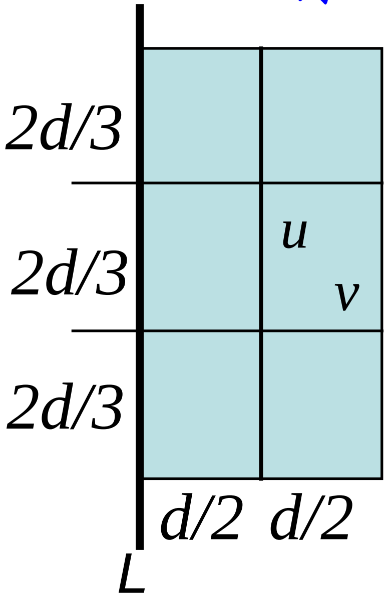
HIT
CS&E



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

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

即$Dis(u, v)\le 5d/6<d$, 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).

时间复杂性=$O(n\log n)$+T(FindCPP)=$O(n\log n)$

扩展到三维空间或更高维空间如何?

3.4 凸包(convex hull)构建算法

问题定义

输入：平面上的n个点的集合Q

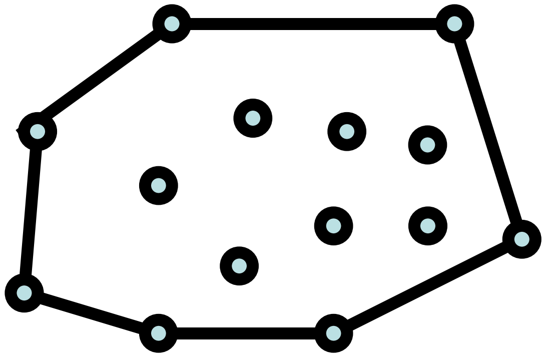
输出：CH(Q): Q的convex hull

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

Graham-Scan算法

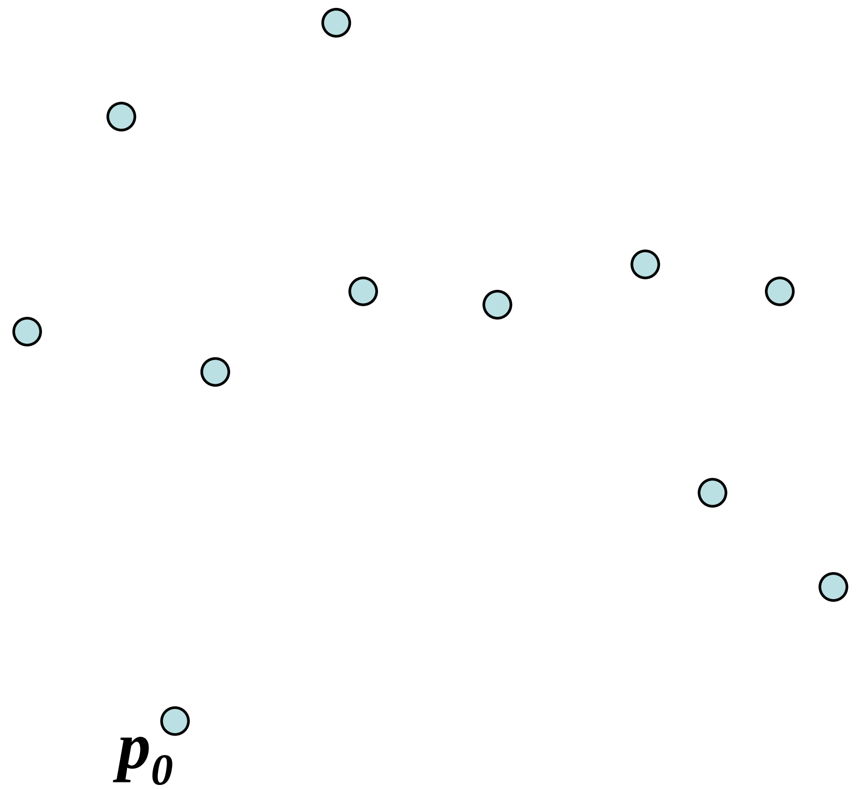
- 基本思想
  - 当沿着Convex hull逆时针漫游时，总是向左转
  - 在极坐标系下按照极角大小排列，然后逆时针方向漫游点集，去除非Convex hull顶点(非左转点)。



如何判断左转？

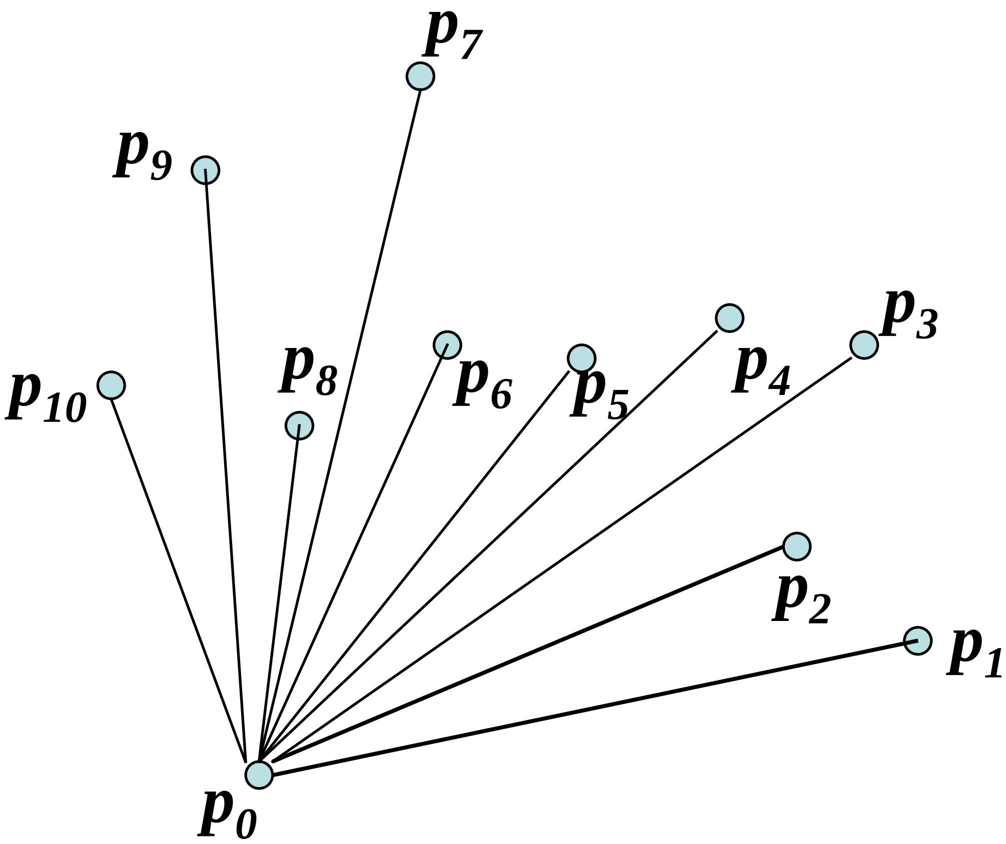
HIT
CS&E

第1步



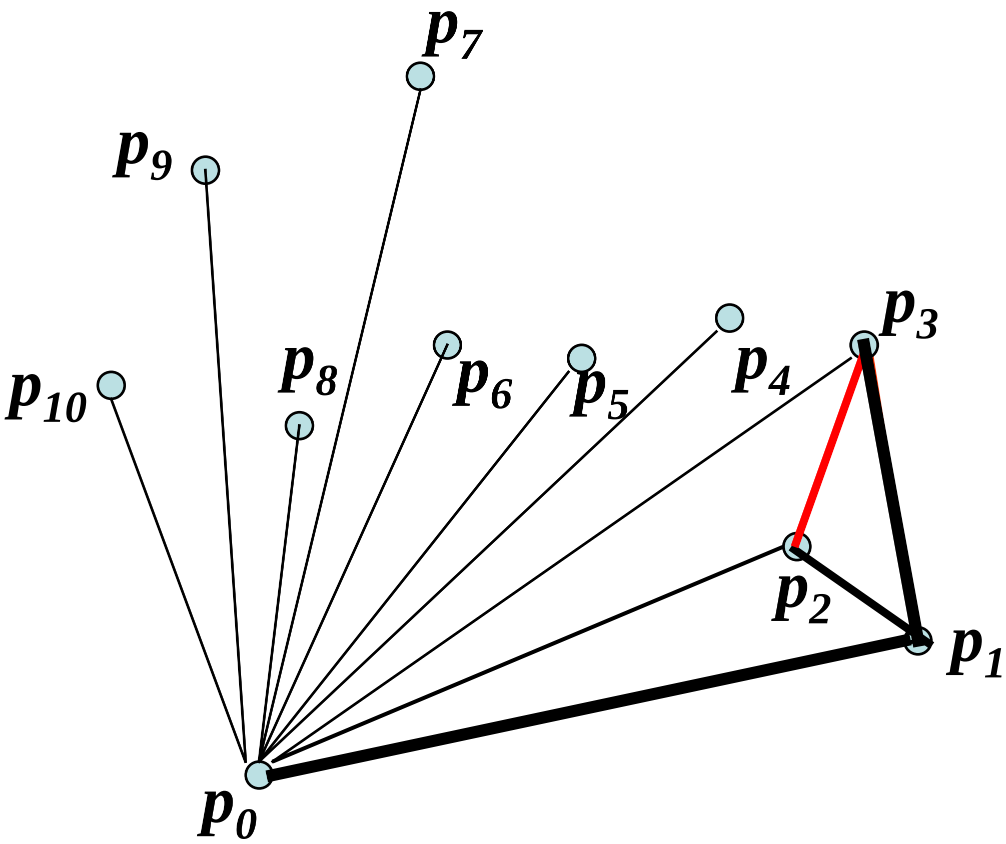
HIT
CS&E

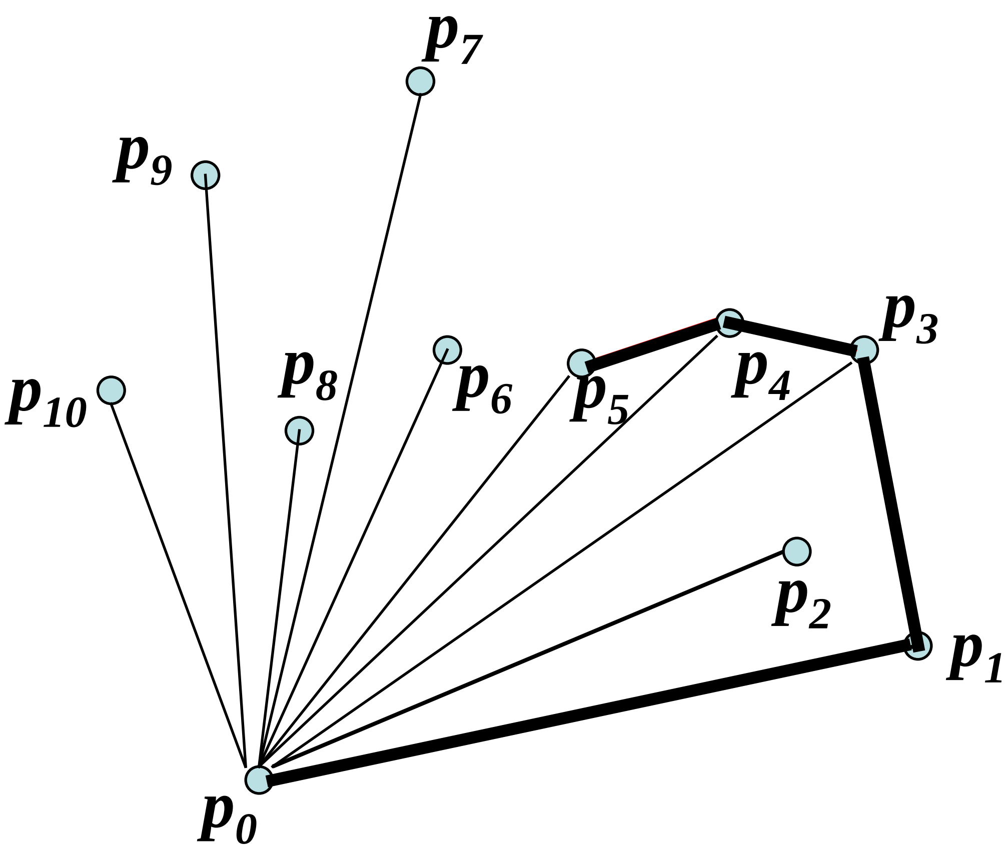
第2步

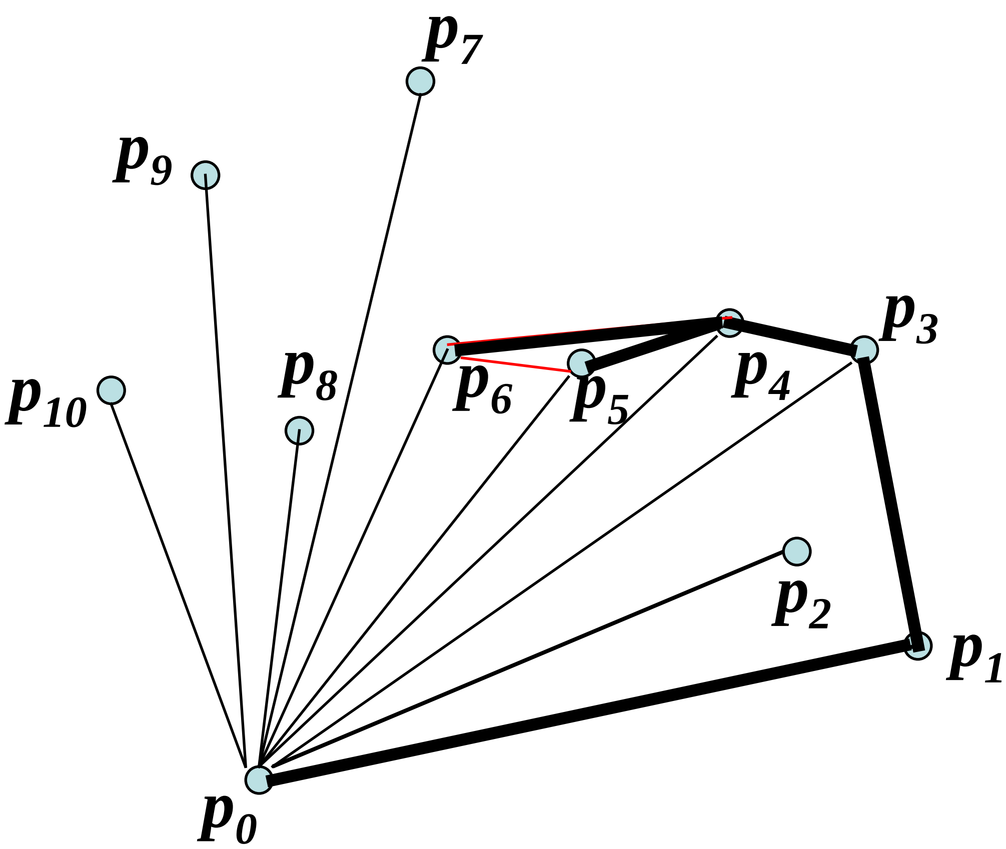


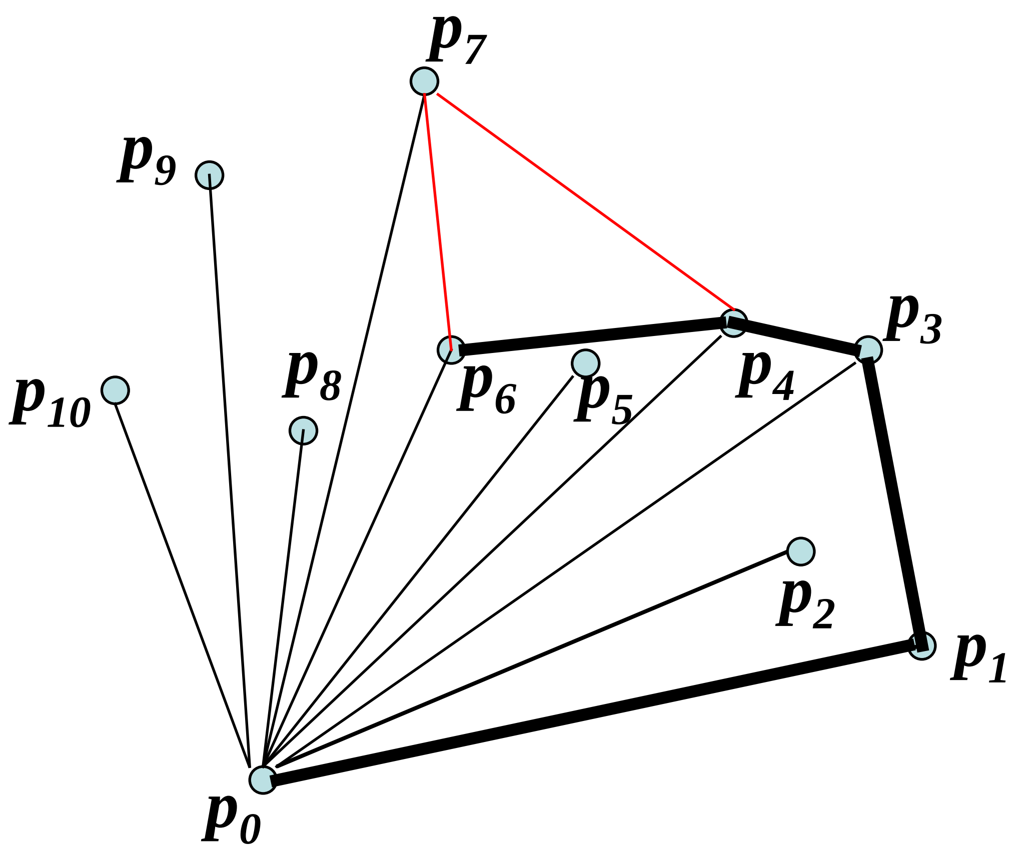
HIT
CS&E

第3步

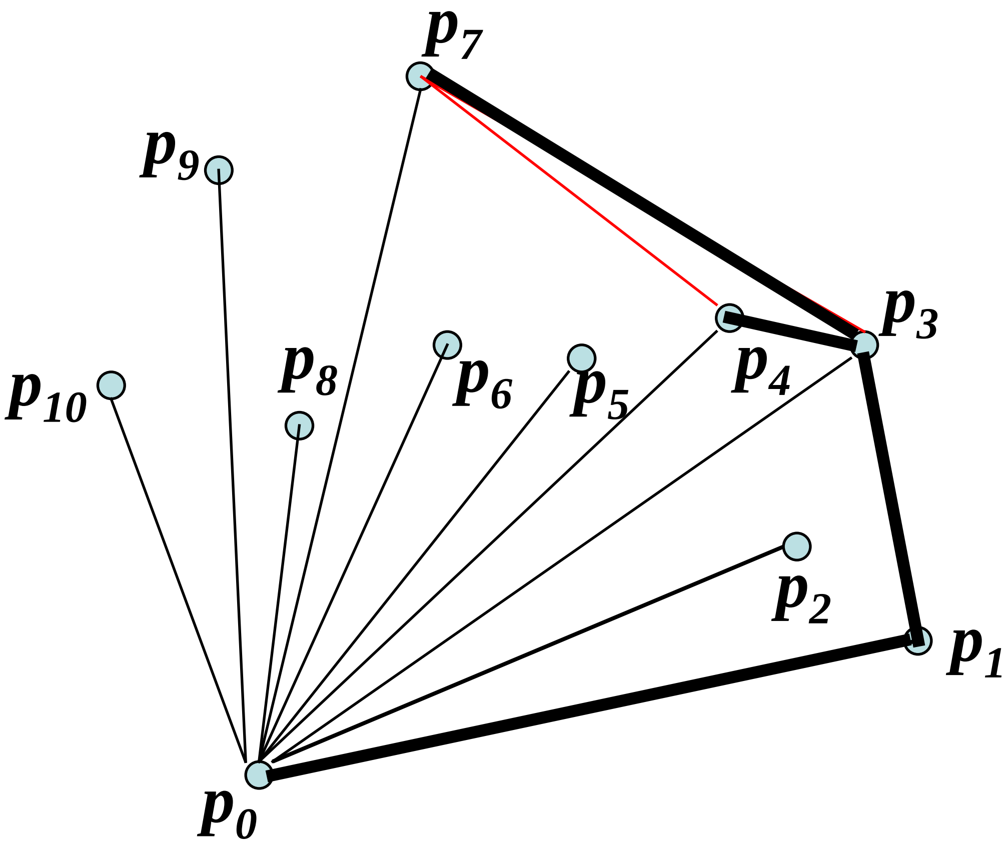


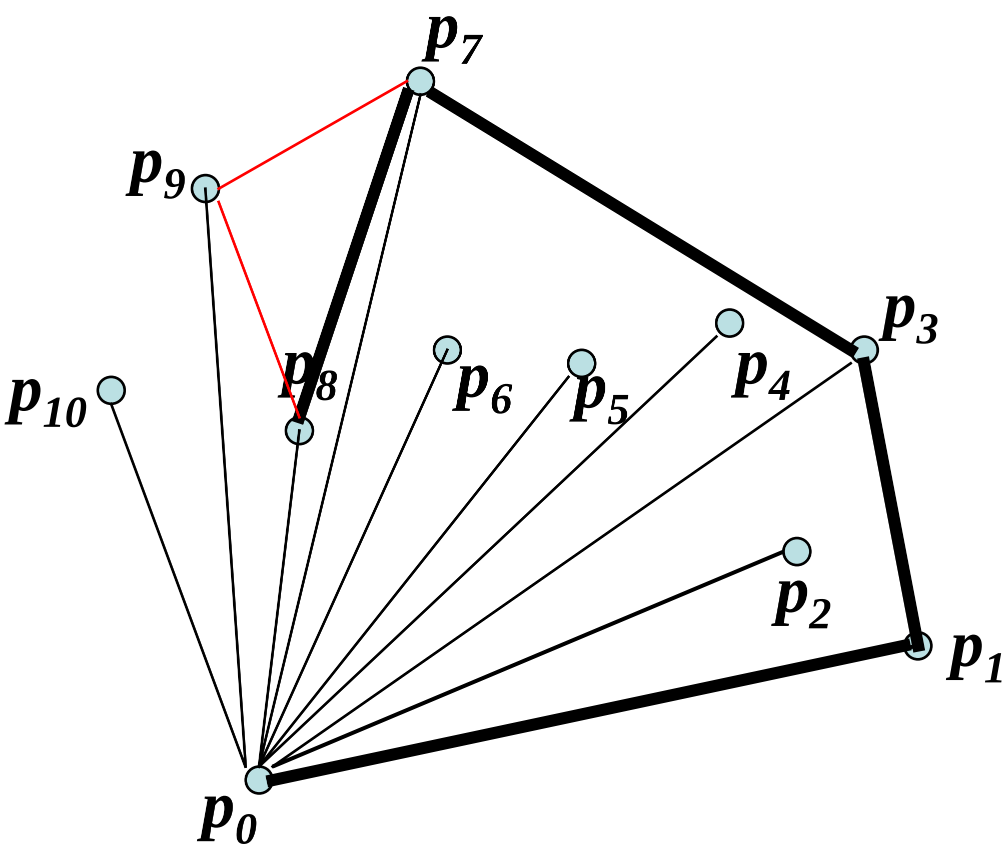

p_7
p_9
p_3
p_{10}
p_8
p_6
p_5
p_4
p_2
p_1
p_0

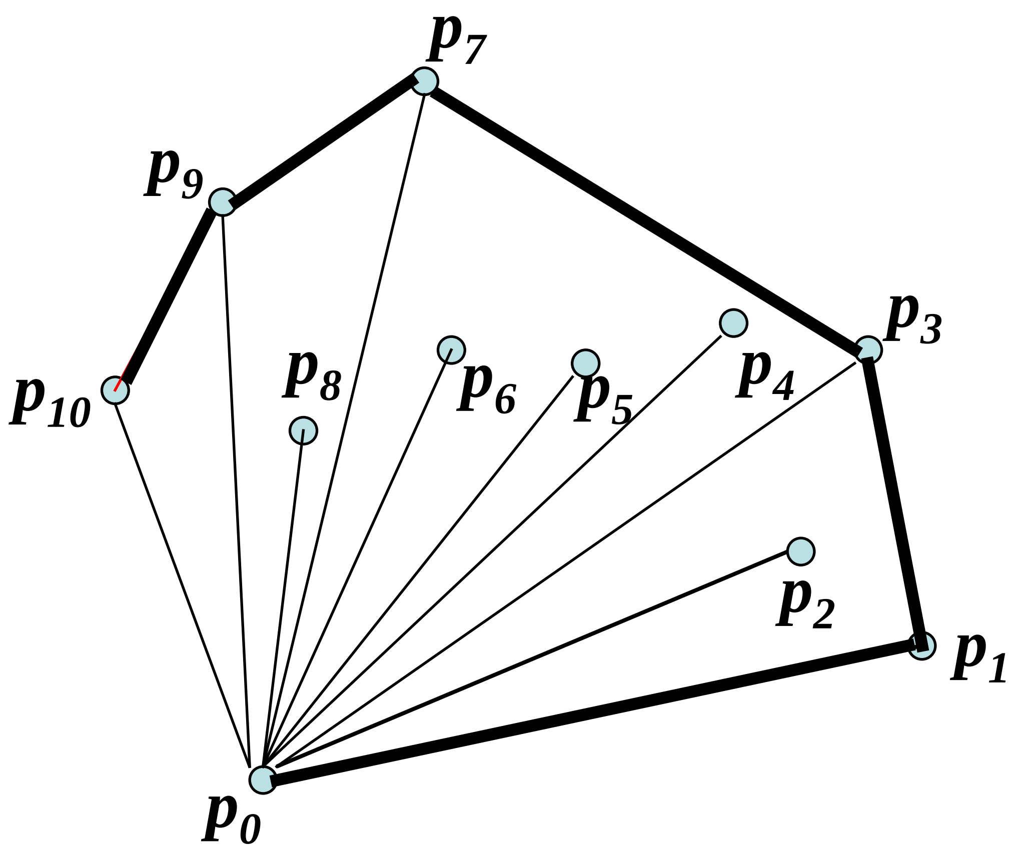

p_7
p_9
p_3
p_8
p_6
p_5
p_4
p_{10}
p_2
p_1
p_0


p_7
p_9
p_3
p_8
p_6
p_5
p_4
p_{10}
p_2
p_1
p_0

HIT
CS&E



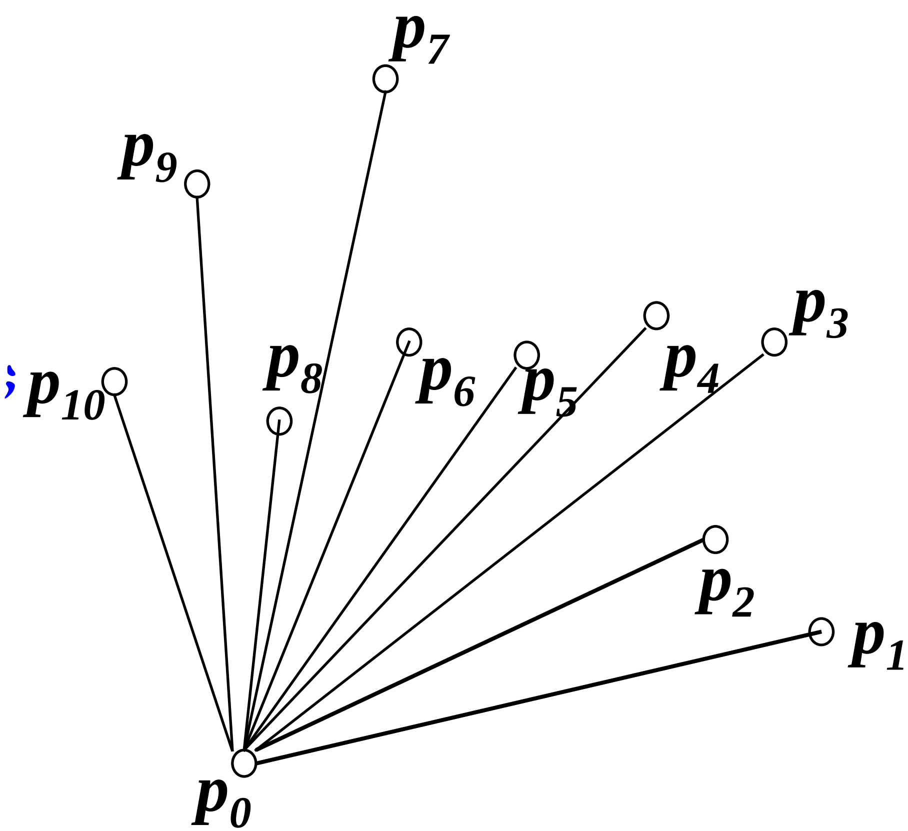

p_7
p_9
p_3
p_8
p_6
p_5
p_4
p_{10}
p_2
p_1
p_0


p7
p9
p3
p4
p8
p6
p5
p10
p2
p1
p0

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<$p_1, p_2, \ldots, p_n$>;
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$<p_1, p_2, \ldots, p_n>$;
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. Return 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）.

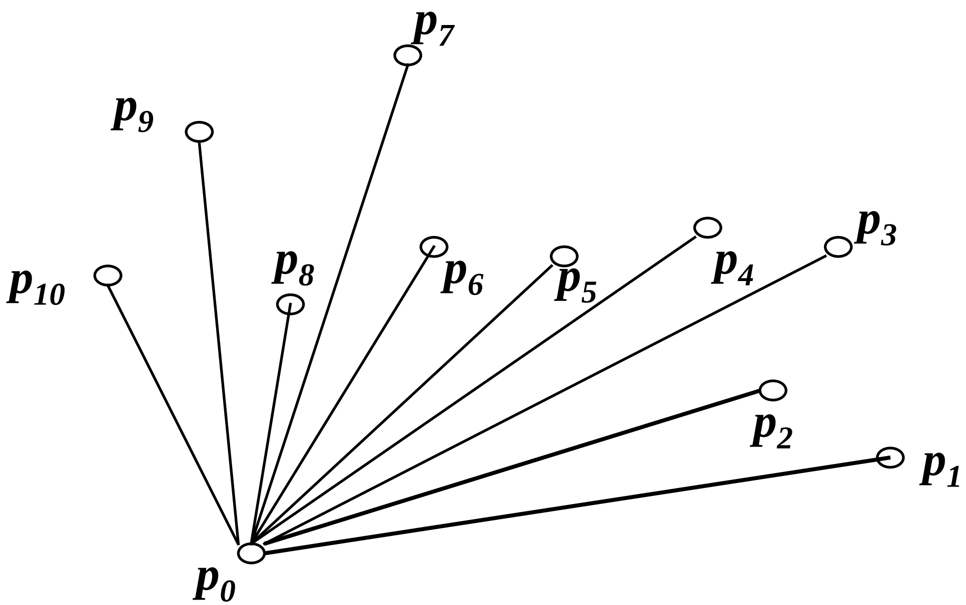
证明: 使用循环不变量方法

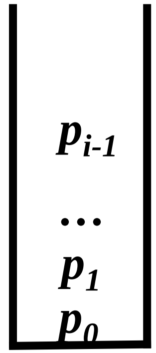
- 循环不变量方法
  - 主要结构为循环的算法的正确性证明的通用方法.
- 主要步骤
  - 确定循环不变量P
    - 定义在算法所操作的数据上的一个谓词(关键性质)
  - Initialization，即循环开始前，P成立
  - Maintenance：证明循环体每执行一次之后P仍然成立
  - Termination：证明循环结束后，P成立

```
3. Push p0, p1, p2 into S;
4. FOR i=3 TO n DO
5.    While Next-to-top(S)、Top(S)
            和pi形成非左移动 Do
6.                Pop(S);
7.     Push(pi, 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);

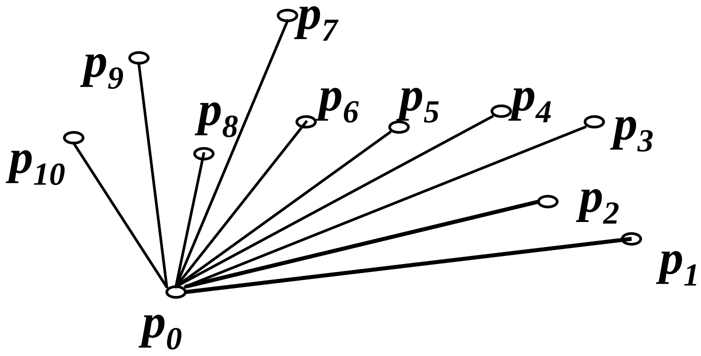
循环不变量

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$i(i\geq3)$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：

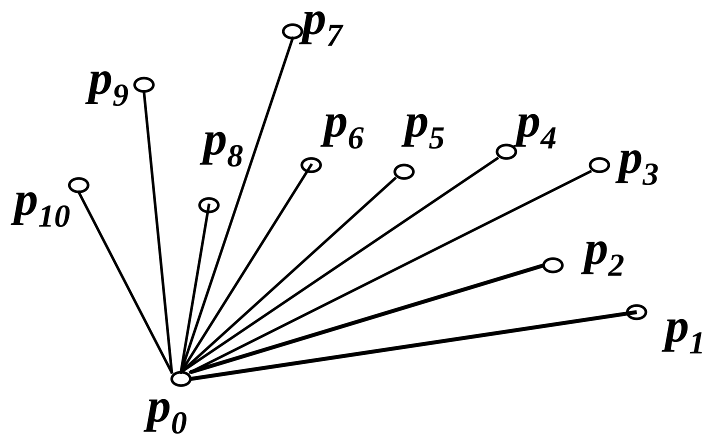
      算法执行5~7步之后，栈S中自底到顶存储CH(Q_i)的顶点.

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)，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.
  - 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)。
  - 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)
  - 又 $Q_i - P_i = Q_j \cup \{p_i\}$，故有CH($Q_j \cup \{p_i\}$)= CH($Q_i - P_i$)= 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 p0, p1, p2 into S;
4.  FOR  i=3  TO  n  DO
5.     While  Next-to-top(S)、Top(S)
                和pi形成非左移动   Do
6.                  Pop(S);
7.      Push(pi, 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.

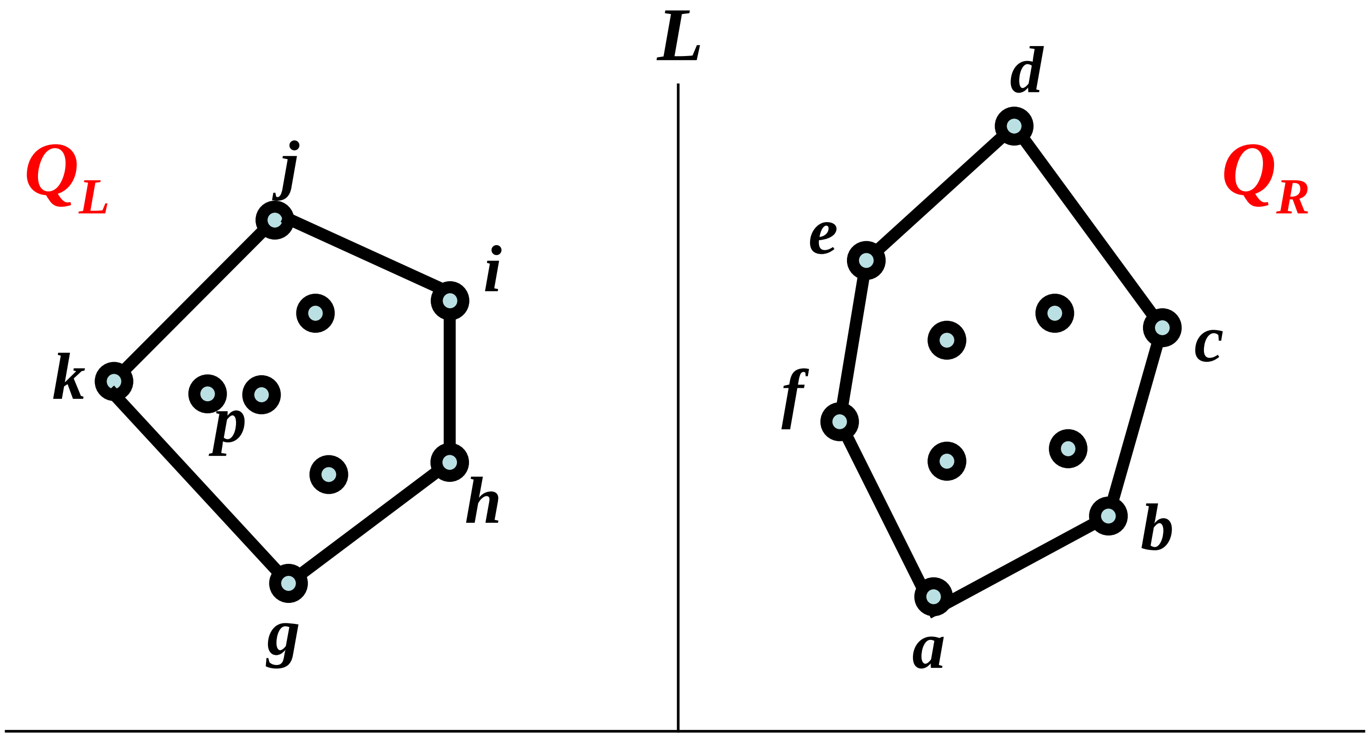
Divide-and-conquer算法

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;

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;



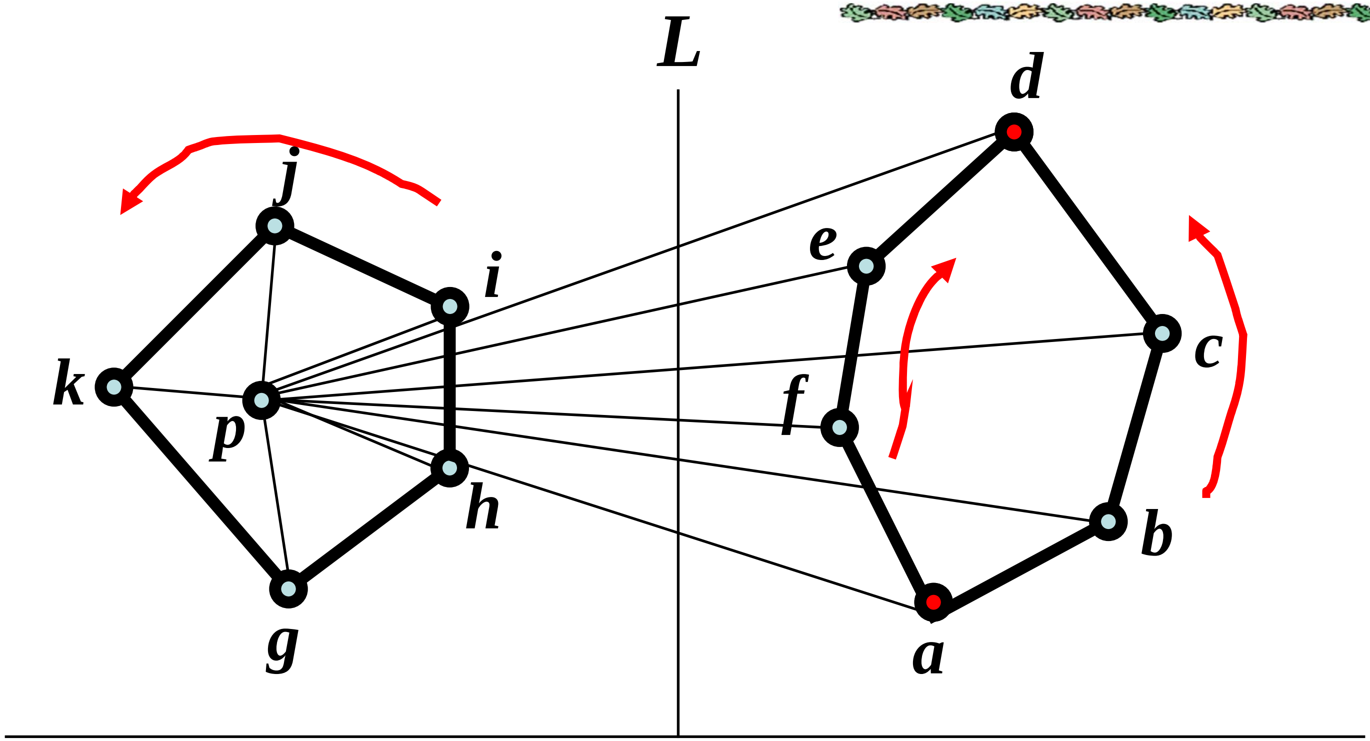
Conquer: (时间复杂性为$2T(n/2)$)

1. 递归地为Q_L和Q_R构造$CH(Q_L)$和$CH(Q_R)$;

Merge:

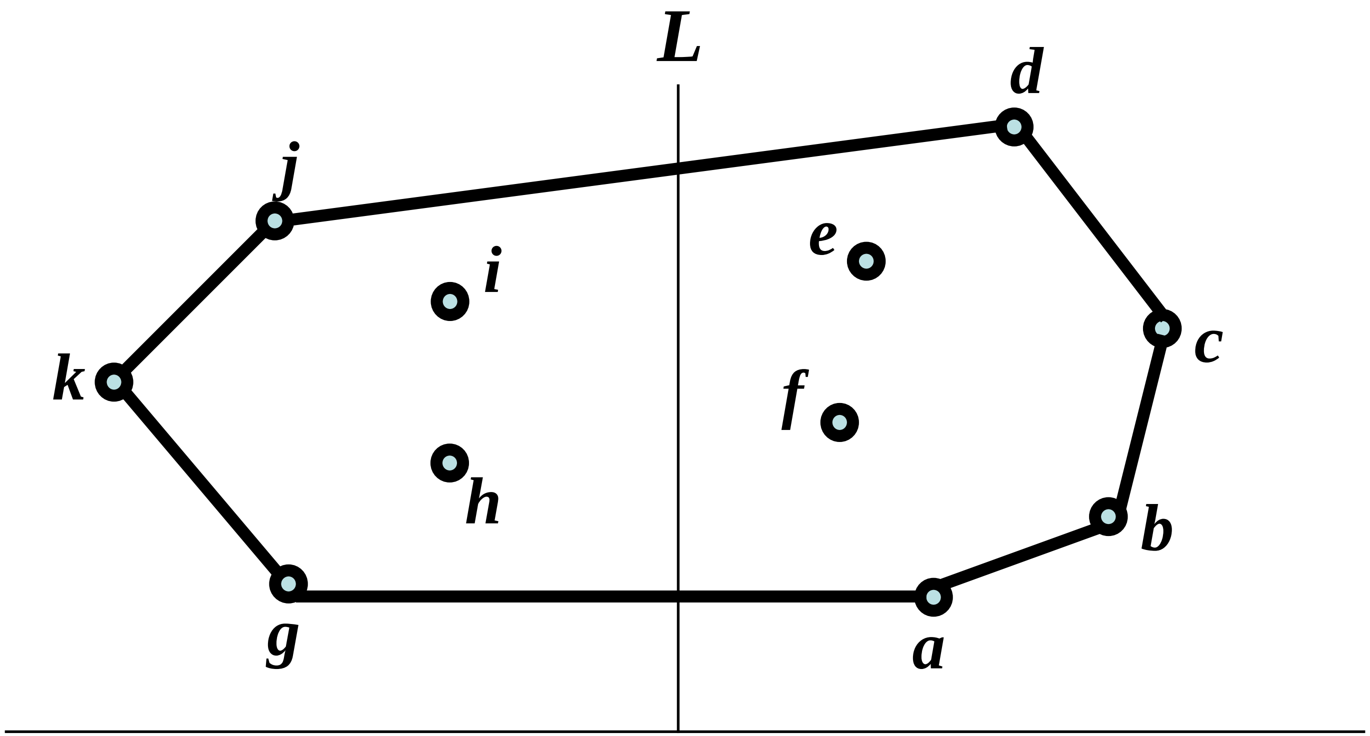
我们先通过一个例子来看Merge的思想

Merge实例



3个序列：<g, h, i, j, k>, <a, b, c, d>, <f, e>

合并以后：<g, h, a, b, f, c, e, d, i, j, k>



Graham-Scan on <*g*, *h*, *a*, *b*, *f*, *c*, *e*, *d*, *i*, *j*, *k*>

Merge:(时间复杂性为$O(n)$)

1. 找一个Q_L的内点p;
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u到v的顶点,
   (3) 按顺时针方向排列的CH(Q_R)从u到v的顶点;
4. 合并上述三个序列;
5. 在合并的序列上应用Graham-Scan.

时间复杂性

- 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

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

3.5 分位数选择算法

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 *median* problem: to find the $\left\lceil \frac{n}{2} \right\rceil$-th smallest element

The straightforward algorithm:

step 1: Sort the n elements

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

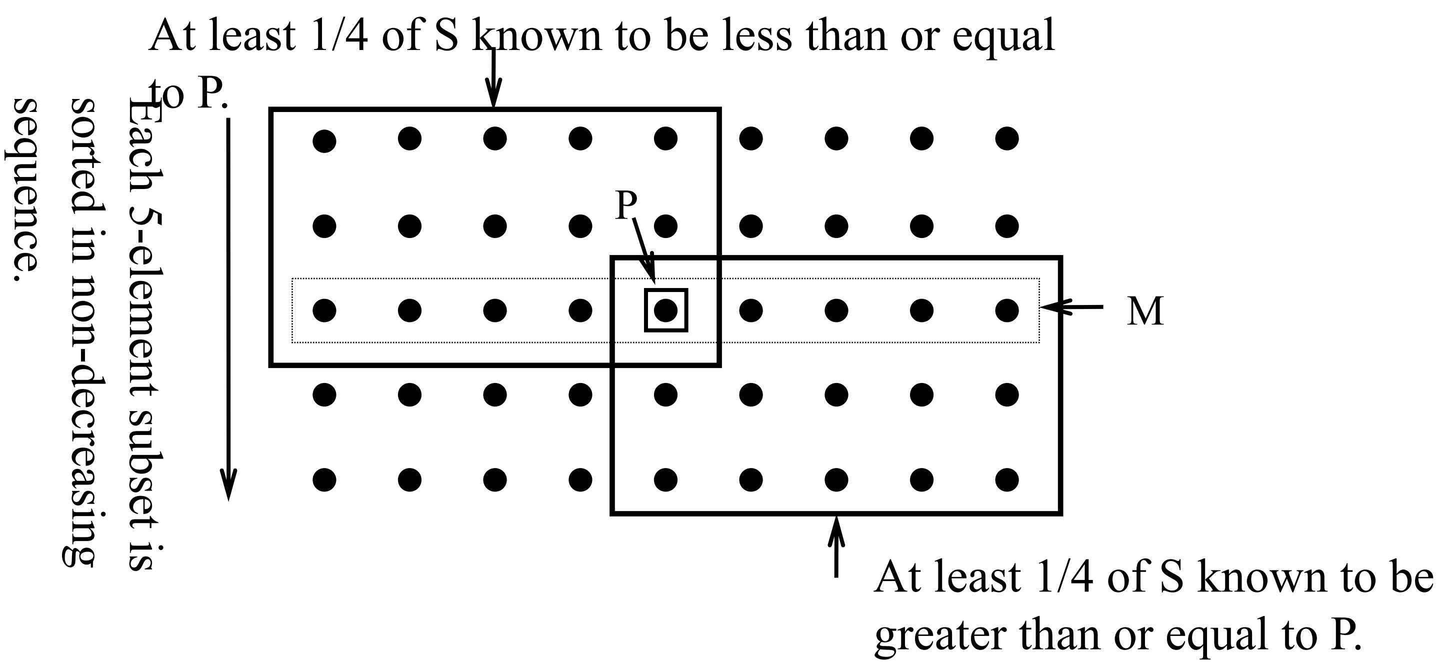
Time complexity: O($n$log$n$)

Selection Problem

- **Main Idea**
  - $S=\{a_1, a_2, \ldots, a_n\}$
  - Let $p \in S$, use p to partition S into 3 subsets S_1 , S_2 , S_3:
    - $S_1=\{ a_i \mid a_i < p , 1 \le i \le n\}$
    - $S_2=\{ a_i \mid a_i = p , 1 \le i \le n\}$
    - $S_3=\{ a_i \mid a_i > p , 1 \le i \le n\}$
  - 3 cases:
    - If $|S_1| > k$, then the k^{th} smallest element of S is in S_1, prune away S_2 and S_3.
    - if $|S_1| + |S_2| > k$, then p is the k^{th} smallest element of S.
    - Else, the k^{th} smallest element of S is the $(k - |S_1| - |S_2|)$-*th* smallest element in S_3, prune away S_1 and S_2.

Selection Problem

- **How to select *p*?**
  - The *n* elements are divided into $\left\lceil \frac{n}{5} \right\rceil$ subsets (Each subset has 5 elements.)



HIT
CS&E

Selection Problem

算法步骤:

R +q ,

Step 1: Divide S into $\lceil n/5 \rceil$ subsets. Each subset contains five elements. Add some dummy ∞ elements to the last subset if n is not a net multiple of S.

Step 2: Sort each subset of elements. R +q ,

Step 3: Find the element p which is the median of the medians of the $\lceil n/5 \rceil$ subsets. W +***n*/5** ,

Step 4: Partition S into S_1, S_2 and S_3, which contain the elements less than, equal to, and greater than p, respectively. R +q ,

Step 5: If $|S_1| \geq k$, then discard S_2 and S_3 and solve the problem that selects the k^{th} smallest element from S_1 during the next iteration;

else if $|S_1| + |S_2| \geq k$ then p is the k^{th} smallest element of S; W +**3*n*/4** ,

otherwise, let $k' = k - |S_1| - |S_2|$, solve the problem that selects the k'^{th} smallest element from S_3 during the next iteration.

Selection Problem

- 算法复杂性分析

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

Let $T(n) = a_0 + a_1 n + a_2 n^2 + \ldots\ , a_1 \neq 0$

$T(3n/4) = a_0 + (3/4)a_1 n + (9/16)a_2 n^2 + \ldots$

$T(n/5) = a_0 + (1/5)a_1 n + (1/25)a_2 n^2 + \ldots$

$T(3n/4 + n/5) = T(19n/20) = a_0 + (19/20)a_1 n + (361/400)a_2 n^2 + \ldots$

$T(3n/4) + T(n/5) \leq a_0 + T(19n/20)$

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

HIT
CS&E

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

$$\le cn + (19/20)cn + \mathrm{T}((19/20)^2 n)$$

$$\vdots$$

$$\le cn + (19/20)cn + (19/20)^2 cn + \ldots + (19/20)^p cn + T((19/20)^{p+1} n) \ ,$$

$$(19/20)^{p+1} n \le 1 \le (19/20)^p n$$

$$= \frac{1-(\frac{19}{20})^{p+1}}{1-\frac{19}{20}} cn + b$$

$$\le 20\, cn + b$$

$$= O(n)$$

3.6 快速排序

- Idea of Quicksort
- Quicksort Algorithm
- Correctness Proof
- Performance Analysis

Idea of Quicksort

- Divide-and-Conquer
  - Divide:
    - Partition *A[p..r]* into *A[p..q]* and *A[q+1..r]*.

s		t04	t	t.4		u

    - $\forall x \in A[p...q], x \leq A[q], \ \forall y \in A[q+1...r], y > A[q]$.
    - *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

Quicksort Algorithm

TXLFNVRUW+D/s/u,

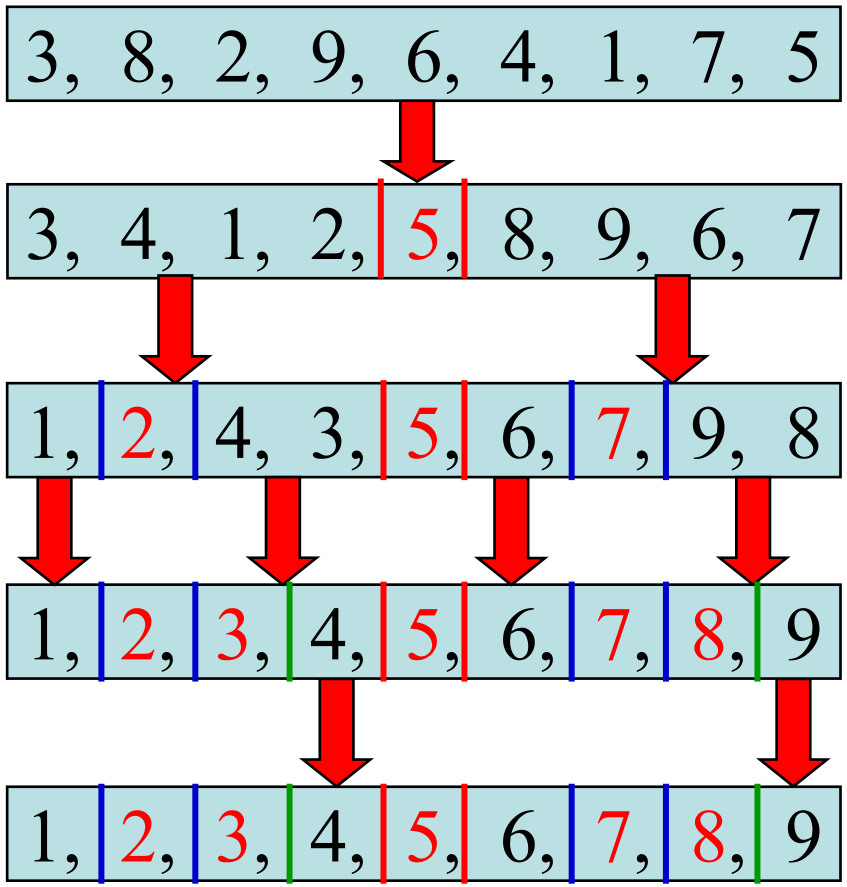
Li s?u

Whq t@Sduwlwlrq+D/s/u,>

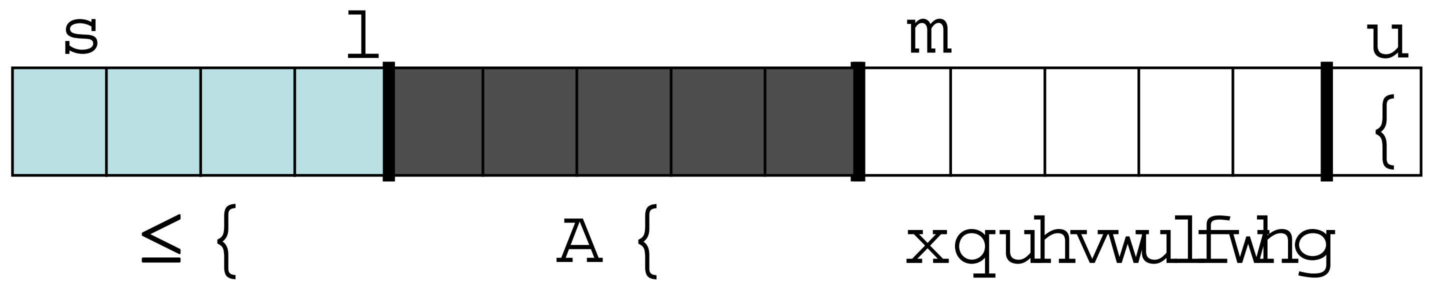
TXLFNVRUW+D/s/t04,>

TXLFNVRUW+D/t.4/u,>

- **Example**



- Sduwlwlrq#Dojrulwkp
  - Wdnh#{@D^u`#dv#d#slyrw
  - Fodvvli|#rwkhu#hohp hqwv#e|#frp sdulqj#z lwk#{1#
  - Gxulqj#wkh#uxqqlqj#ri#wkh#dojrulwkp /#D lv# sduwlwlrqhg#lqwr#7#uhjlrqv=



$\text{Partition}(A, p, r)$

$x \leftarrow A[r];$

$i \leftarrow p-1;$

$\text{for } j \leftarrow p \text{ to } r-1$

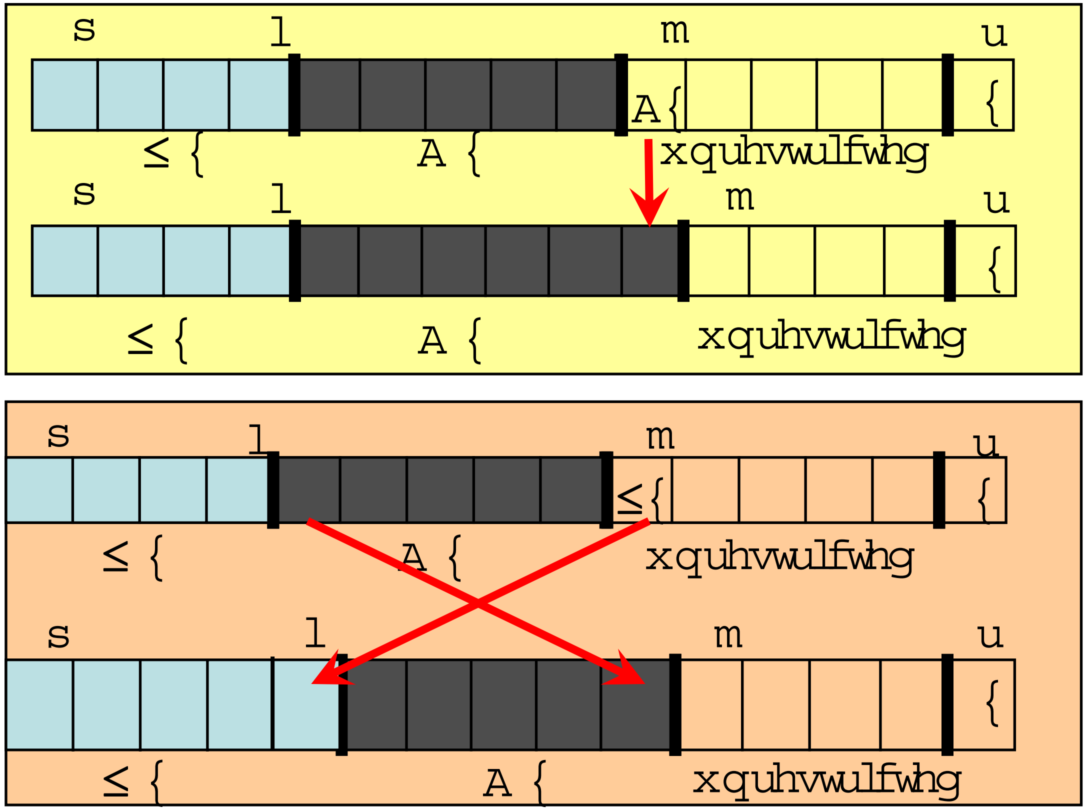
$\quad \text{do if } A[j] \leq x$

$\quad\quad i \leftarrow i+1;$

$\quad\quad \text{exchange } A[i] \leftrightarrow A[j];$

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

$\text{return } i+1;$



Uxqqlqj#wlph#=#Θ+q,

HIT
CS&E

$\text{Partition}(A, p, r)$

$x \leftarrow A[r];$

$i \leftarrow p-1;$

$\text{for } j \leftarrow p \text{ to } r-1$

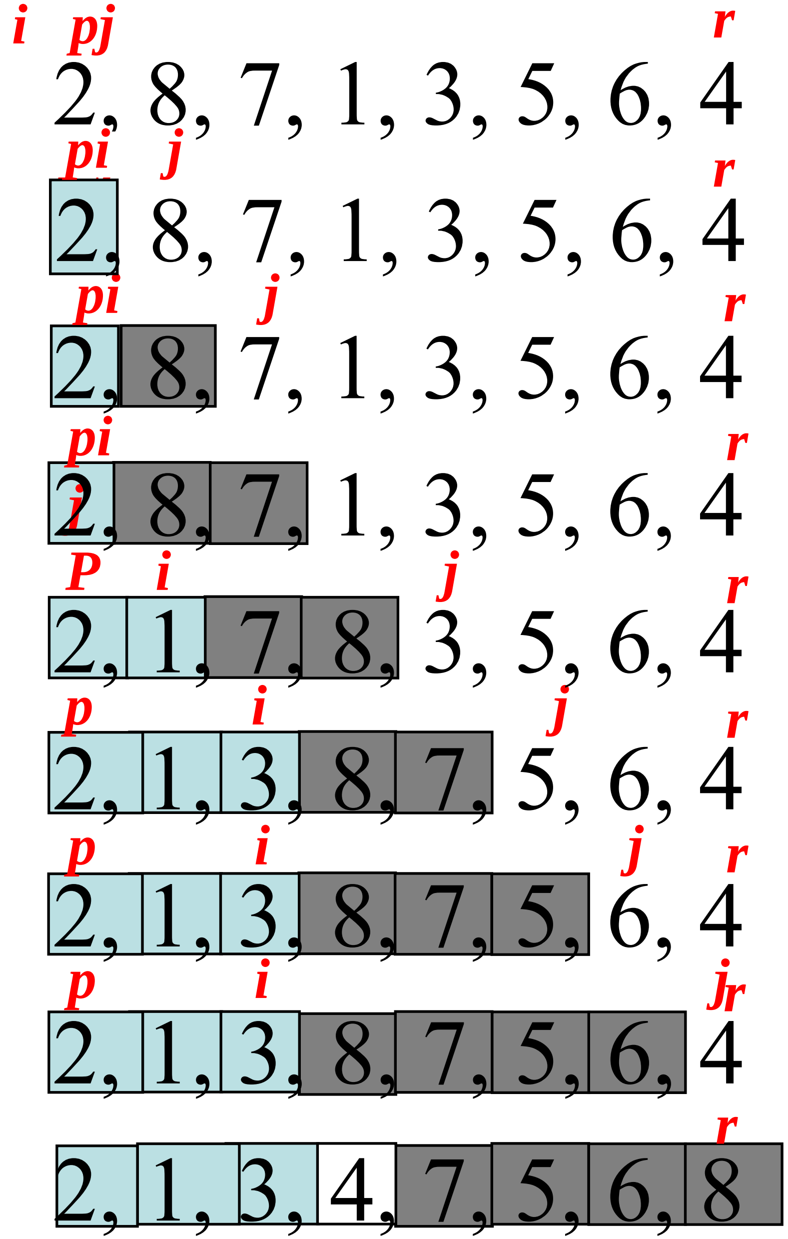
$\quad \text{do if } A[j] \leq x$

$\qquad i \leftarrow i+1;$

$\qquad \text{exchange } A[i] \leftrightarrow A[j];$

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

$\text{return } i+1;$

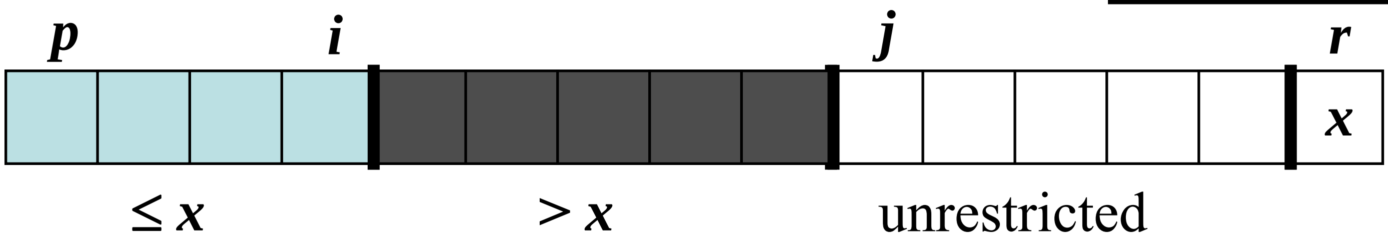


- **算法正确性证明**
  - Loop Invariant(循环不变量)

    At the start of the loop of lines 3-6 for any k

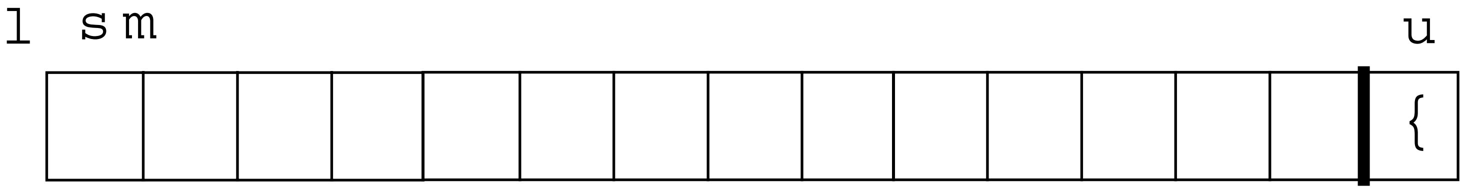
    1. *if $p \le k \le i$, then $A[k] \le x$.*
    2. *if $i+1 \le k \le j-1$, then $A[k] > x$.*
    3. *if $k=r$, then $A[k]=x$.*

```
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;
```



  - Initialization: $j=p$

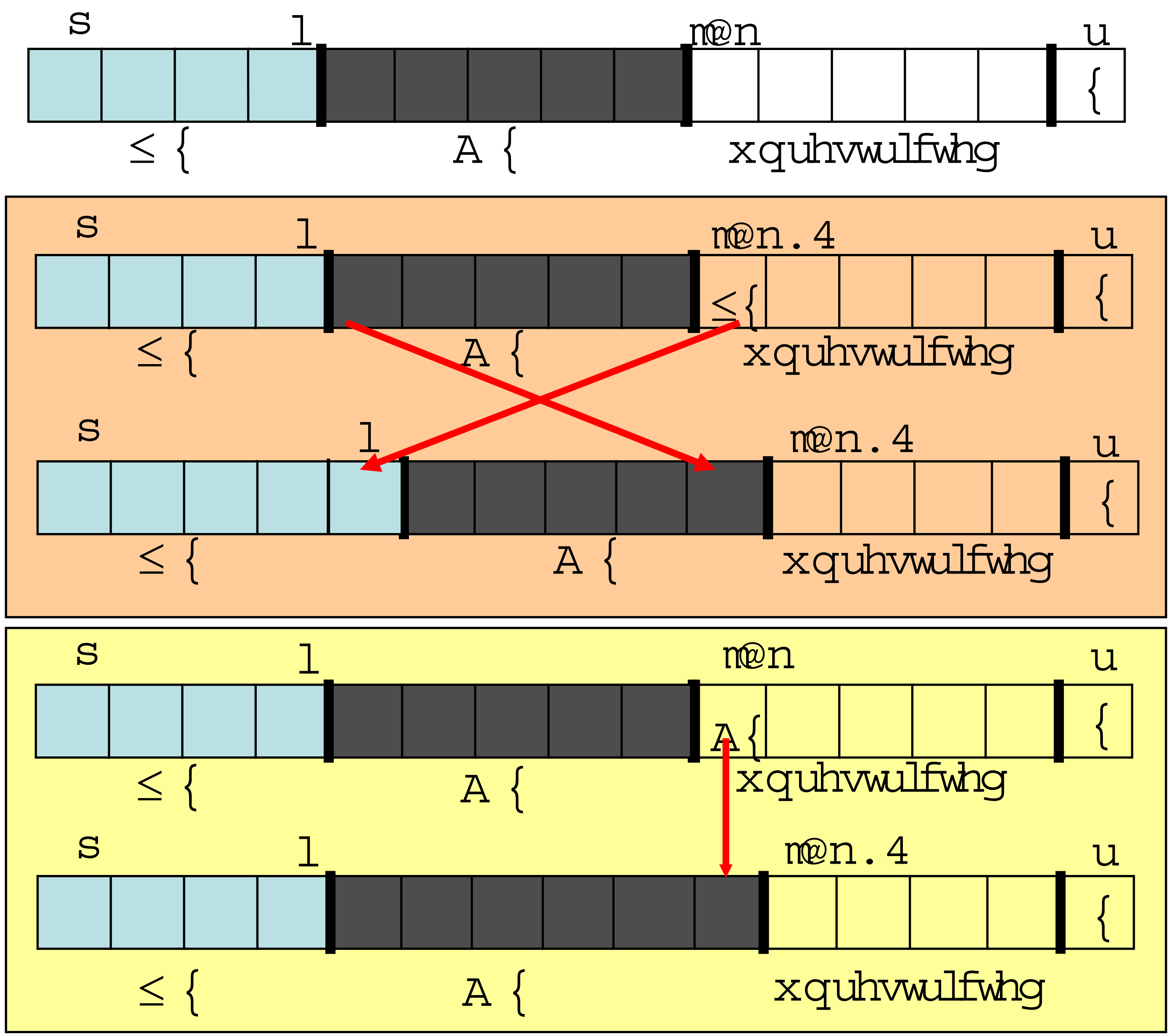
    Prior the first iteration: $i=p$-1, $j=p$, condition 1 and 2 are trivially satisfied. Line 1 make condition 3 true.



P dlqwhqdqfh

设m@n时循环不变量成立1

往证m@n.4时不变量成立1



Partition(A, p, r)

$x \leftarrow A[r]$;

$i \leftarrow p-1$;

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

do if $A[j] \le x$

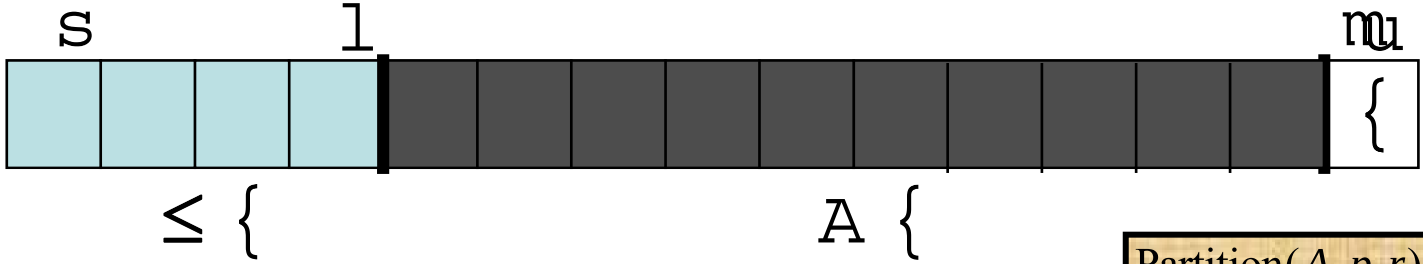
$i \leftarrow i+1$;

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

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

return $i+1$;

– Termination



At termination, $j=r$. We have three sets:

1. those less than or equal to x.
2. those greater than x.
3. a singleton set containing x.

Partition(A, p, r)

$x \leftarrow A[r]$;

$i \leftarrow p-1$;

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

do if $A[j] \le x$

$i \leftarrow i+1$;

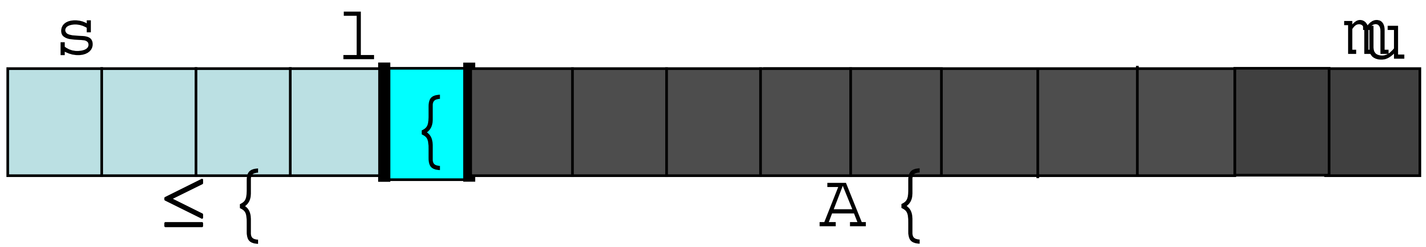
exchange $A[i] \leftrightarrow A[j]$;

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

return $i+1$;

– After finishing the algorithm

最后一个步骤将A[r]与A[i+1]互换.



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

- Worst case time complexity of Quicksort
  - Worst Case
    - $|A[p..q\text{-}1]|=0$， $|A[q+1..r]|=n\text{-}1$

    - The worst case happens in call to Partition Algorithm
  - Worst case time complexity
    - $T(n)= T(0)+T(n\text{-}1)+\theta(n)=T(n\text{-}1)+\theta(n)=\theta(n^2)$

What is the average time complexity?

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

Why?