HIT
CS&E

第三章

分治算法

高宏
海量数据计算研究中心

HIT
CS&E

参考材料

《Introduction to Algorithm》

Chapter 6, 7, 8, 9

《Introduction to the Design and Analysis of Algorithm》

Chapter 4

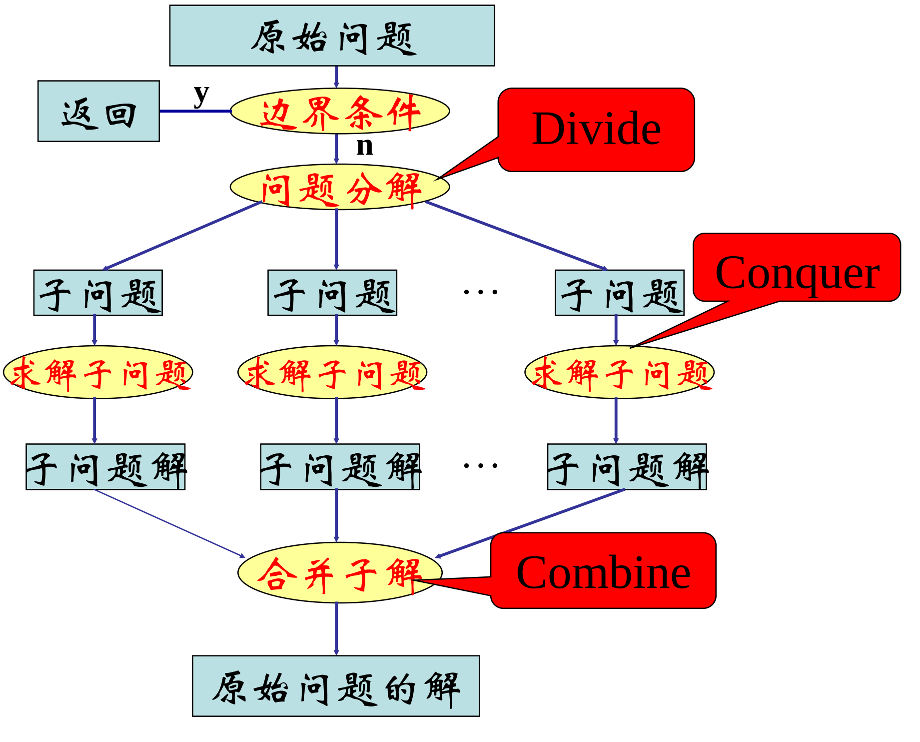
提要

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$p_1, p_2, \ldots p_m$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



HIT
CS&E

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$n<c$, $T(n)=\theta(1)$

HIT
CS&E

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

HIT
CS&E

举例：最大最小值问题

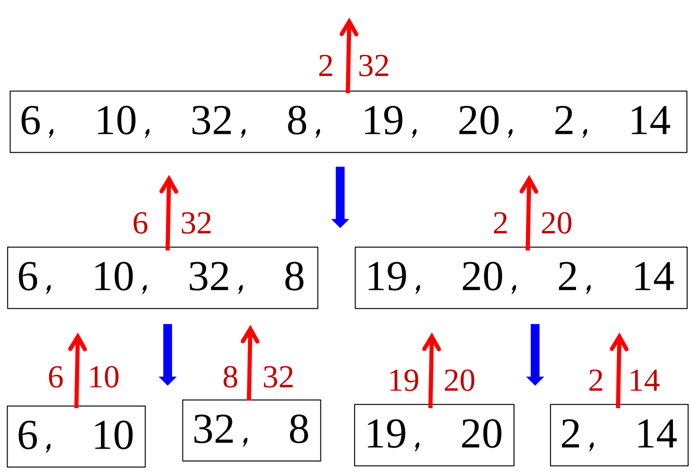
输入：数组$A[1,\ldots,n]$
输出：A中的max和min

通常，直接扫描需要2n-2次比较操作

我们给出一个仅需$\lceil 3n/2-2 \rceil$次比较操作的算法

HIT
CS&E

基本思想



HIT
CS&E

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 If $A[i]< A[j]$ Then 输出$A[i]$,$A[j]$;
3. else 输出$A[j]$,$A[i]$;
   算法结束
4. $k\leftarrow(j-i+1)/2$
5. $m_1,M_1 \leftarrow$MaxMin($A[i:k]$);
6. $m_2,M_2 \leftarrow$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

$T(1)=0$
$T(2)=1$
$T(n)=2T(n/2)+2$

HIT
CS&E

算法复杂性分析

$$
\begin{aligned}
T(1)&=0\\
T(2)&=1\\
T(n)&=2T(n/2)+2\\
&= 2(2T(n/2^2)+2)+2 = 2^2T(n/2^2)+2^2+2\\
&= \dots\\
&=2^{k-1}T(2)+2^{k-1}+2^{k-2}+\dots+2^2+2\\
&=2^{k-1}+ 2^k-2 \qquad n=2^k\\
&=n/2+ n -2\\
&=3n/2-2
\end{aligned}
$$

与Naïve算法相比，虽然同阶，但系数有所改进

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

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

如此计算需要
$T(n)=4T(n/2) +O(n)=O(n^2)$

时间复杂性

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

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

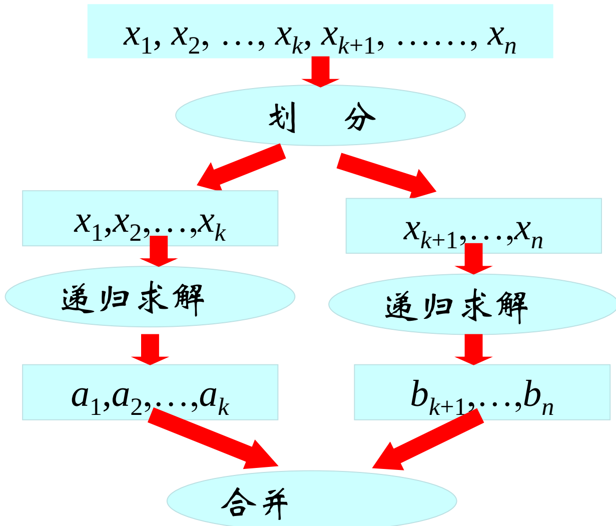
使用Master定理
$T(n)=O(n^{log3}) =O(n^{1.59})$

3.3 基于分治的排序算法

- *Quicksort* Algorithm
- 排序问题的下界

HIT
CS&E

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

HIT
CS&E

3.3.1 *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]*.

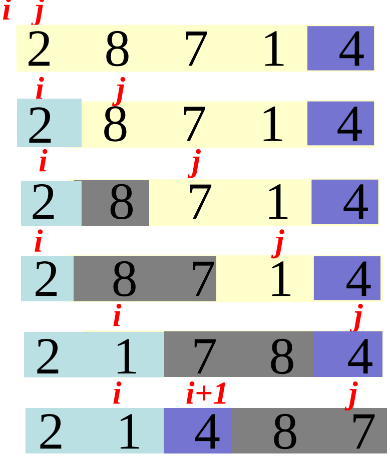
p		q-1	q	q+1		r

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

HIT
CS&E

- 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



HIT
CS&E

Partition(A, p, r)

$x \leftarrow A[r]$;

$i \leftarrow p-1$;

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

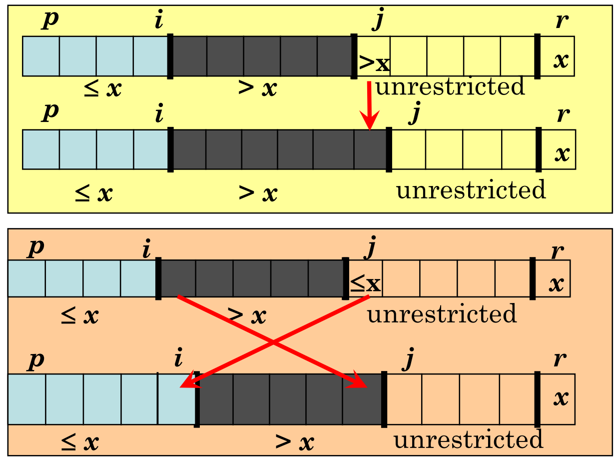
do if $A[j] \leq x$

$i \leftarrow i+1$;

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

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

return $i+1$;



Running time: $\Theta(n)$

HIT
CS&E

Quicksort Algorithm

Quicksort(A,p,r)

If $p<r$

Then q=Partition(A, p, r);

Quicksort (A,p,q-1);

Quicksort (A,q+1,r);

HIT
CS&E

Correctness Proof

- 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

HIT
CS&E

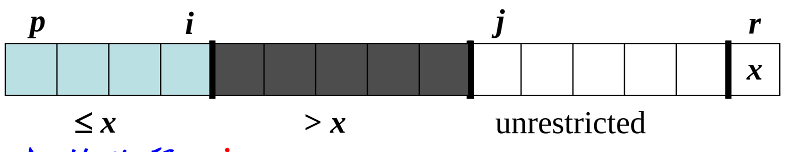
- Correctness Proof

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

定义循环不变量：

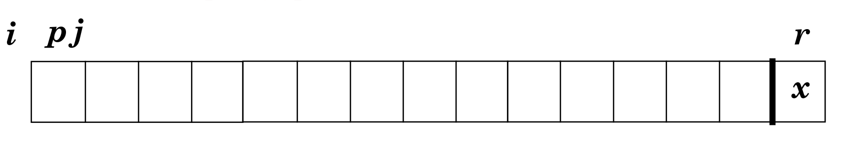
At the start of the loop of lines 3-6, for any k

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 $A[k]=x$.



– 初始阶段：$j=p$

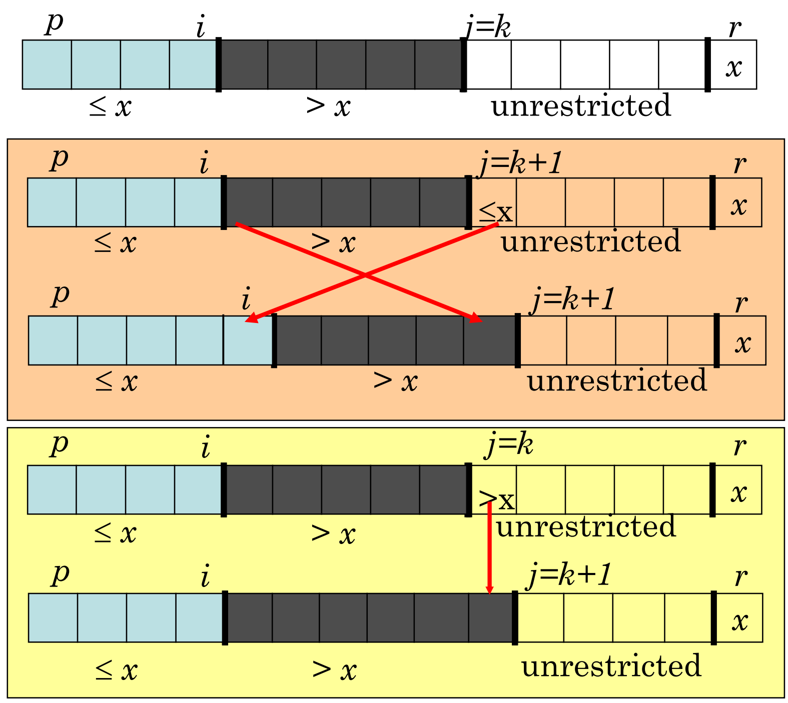
算法迭代前：$i=p-1$, $j=p$, 条件 1 和 2 为真. 算法第1行使得条件 3 为真.



– 保持阶段

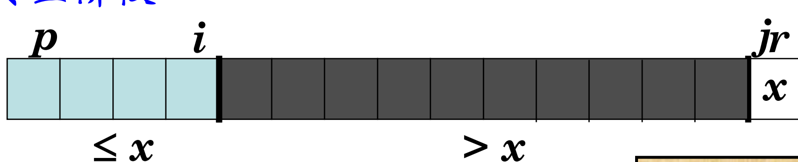
设$j=k$时循环不变量成立.

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



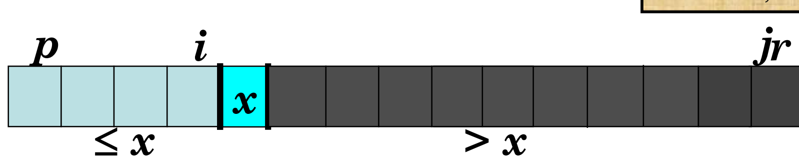
算法结束时, $j=r$, 产生三个集合:

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.

```
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;
```

算法结束时

最后一个步骤将A[r]与A[$i+1$]互换.



HIT
CS&E

Performance Analysis

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

HIT
CS&E

Performance Analysis

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

A[r]										

p=q *q+1* *r*

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

HIT
CS&E

Performance Analysis

What is the average time complexity?

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

Why?

HIT
CS&E

Performance Analysis

- 假如第一次划分后产生两个子序列，第一个子序列包含*s*个元素，第二个子序列包含*n-s*个元素
- 一共有*n*种可能的划分，即$1\le s\le n$，每种可能划分产生的概率为$1/n$
- 平均复杂性$T(n)=\frac{1}{n}\sum_{s=1}^{n}(T(s)+T(n-s))+cn$

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

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

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

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

Performance Analysis

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

用*n=n-1*代入上式，有：

$$(n-2)T(n\text{-}1) = 2T(1) + 2T(2) + \ldots\ldots + 2T(n-2) + c(n-1)^2$$

两式相减： $(n-1)T(n) - (n-2)T(n\text{-}1) = 2T(n-1) + c(2n-1)$

$$(n-1)T(n) - nT(n\text{-}1) = c(2n-1)$$

$$(n-1)T(n) = nT(n\text{-}1) + c(2n-1)$$

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

递归地： $\dfrac{T(n-1)}{n-1} = \dfrac{T(n-2)}{n-2} + c(\dfrac{1}{n\text{-}1} + \dfrac{1}{n-2})$

$$\cdots$$

$$\frac{T(2)}{2} = \frac{T(1)}{1} + c(\frac{1}{2} + \frac{1}{1})$$

Performance Analysis

我们得到： $\dfrac{T(n)}{n} = \dfrac{T(n-1)}{n-1} + c(\dfrac{1}{n} + \dfrac{1}{n-1})$

$$\frac{T(n-1)}{n-1} = \frac{T(n-2)}{n-2} + c(\frac{1}{n\text{-}1} + \frac{1}{n-2})$$

$$\cdots$$

$$\frac{T(2)}{2} = \frac{T(1)}{1} + c(\frac{1}{2} + \frac{1}{1})$$

$$\frac{T(n)}{n} = c(\frac{1}{n} + \frac{1}{n-1}) + c(\frac{1}{n\text{-}1} + \frac{1}{n-2}) + \ldots\ldots + c(\frac{1}{2} + \frac{1}{1})$$

$$\frac{T(n)}{n} = c(\frac{1}{n} + \frac{1}{n-1} + \ldots\ldots + \frac{1}{2}) + c(\frac{1}{n\text{-}1} + \frac{1}{n-2} + \ldots\ldots + \frac{1}{1})$$

$$\frac{T(n)}{n} = c(H_n - 1) + cH_{n-1} = c(H_n + H_{n-1} - 1) = c(2H_n - \frac{1}{n} - 1) = c(2H_n - \frac{n+1}{n})$$

$$T(n) = 2cnH_n - c(n+1) = 2cn\ln n - c(n+1) = O(n\log n)$$

HIT
CS&E

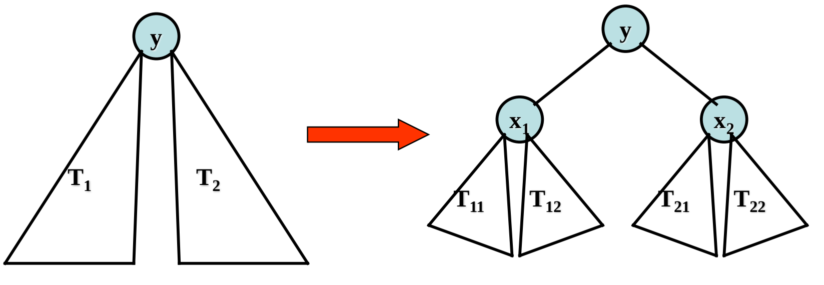
Randomized Quicksort Algorithms

- **Randomized-Partition(*A*, *p*, *r*)**
  1. i := 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 := Randomized-Partition(A, p, r);
  3. Randomized-Quicksort(A, p, q-1);
  4. Randomized-Quicksort(A, q+1, r).

HIT
CS&E

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

HIT
CS&E

随机快速排序复杂性分析

- 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)= $E[\sum_{i=1}^{n}\sum_{j>i}X_{ij}]=\sum_{i=1}^{n}\sum_{j>i}E[X_{ij}]$

HIT
CS&E

随机快速排序复杂性分析

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

- 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

    $E[X_{ij}]=p_{ij}\times1+(1-p_{ij})\times0=p_{ij}$

关键问题成为求解p_{ij}

HIT
CS&E

随机快速排序复杂性分析

- 求解p_{ij}
  - $Z_{ij}=\{x_{(i)}, x_{(i+1)}, \ldots, x_{(j)}\}$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：
    - Z_{ij}在同一棵子树，且
    - 当$x_{(i)}$或$x_{(j)}$被选为划分点
  - 设Z_{ij}在同一棵子树T中的概率为p
  - 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 = $2/|T|=2/(j-i+1)$.
  - $x_{(i)}$和$x_{(j)}$被进行比较的概率：

$$p_{ij} = p\times(2/(j-i+1)) \leq 2/(j-i+1)$$

HIT
CS&E

随机快速排序复杂性分析

- 现在我们有

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

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

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

定理. 随机排序算法的期望时间复杂性为$O(n\log n)$

HIT
CS&E

3.3.2 排序问题的下界

HIT
CS&E

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

HIT
CS&E

问题的下界的意义

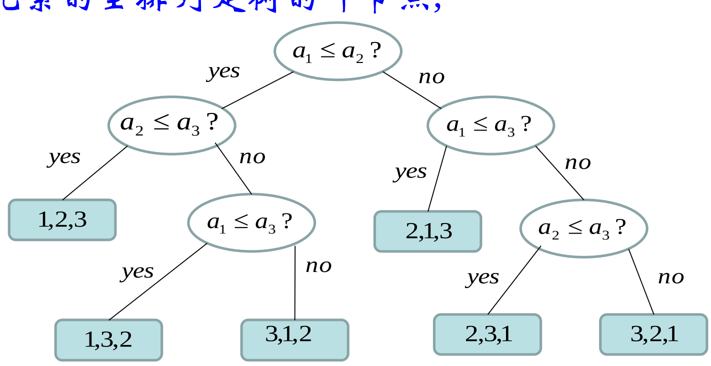
- 如果一个问题的最高下界是 $\Omega(n\log n)$ 而当前最好算法的时间复杂性是 $O(n^2)$.
  - 我们可以寻找一个更高的下界.
  - 我们可以设计更好的算法.
  - 下界和算法都是可以改进的.
- 如果一个问题的下界是 $\Omega(n\log n)$ 且算法的时间复杂性是 $O(n\log n)$, 那么这个算法是**最优的**

HIT
CS&E

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



HIT
CS&E

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: $\Omega(n\log n)$

HIT
CS&E

3.4 Medians and Order Statistics

- Decrease and Conquer原理
- Selection Problem

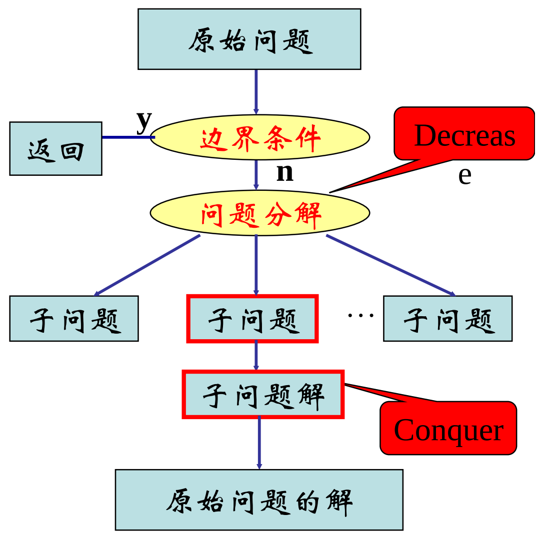
HIT
CS&E

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



HIT
CS&E

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

HIT
CS&E

Medians and Order Statistics

- **The i^{th} order statistic problem**
  - Input: set S of n (distinct) elements, and a number i.
  - Output: element x in S that is greater than exactly $i-1$ 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

HIT
CS&E

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: $O(n\log n)$

HIT
CS&E

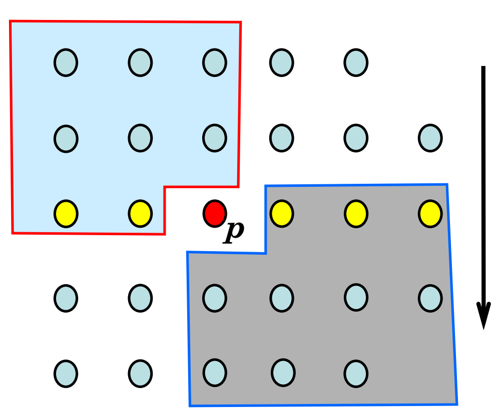
Algorithm of Selection Problem

- **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.

HIT
CS&E

Algorithm of 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.)
  - Sort each subset
  - Find the element ***p*** which is the median of the medians of the $\lceil n/5 \rceil$ subsets



HIT CS&E

Algorithm of Selection Problem

算法步骤:

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.

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

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.

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;

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.

HIT CS&E

Performance Analysis

算法步骤:

O(n)

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. **O(n)**

Step 3: Find the element p which is the median of the medians of the $\lceil n/5 \rceil$ subsets. **T(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. **O(n)**

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; **?**

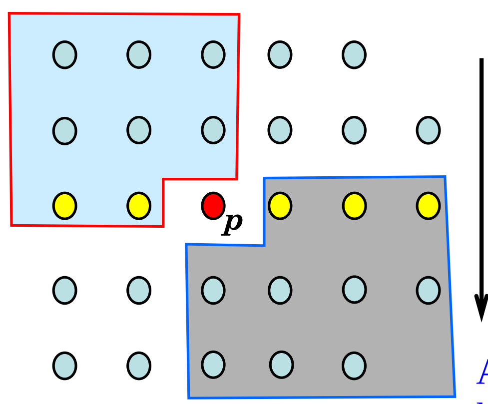
otherwise, let $k' = k - |S_1| - |S_2|$, solve the problem that selects the k'^{th} smallest element from S_3 during the next iteration.

HIT
CS&E

Performance Analysis

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

At least 1/4 of S known to be less than or equal to p



At least 1/4 of S known to be greater than or equal to p.

HIT
CS&E

Performance Analysis

算法步骤:

$\mathbf{O(n)}$

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. $\mathbf{O(n)}$

Step 3: Find the element p which is the median of the medians of the $\lceil n/5 \rceil$ subsets. $\mathbf{T(n/5)}$

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. $\mathbf{O(n)}$

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; $\mathbf{T(3n/4)}$

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.

HIT CS&E

Performance Analysis

- 算法复杂性分析

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

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

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

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

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

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

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

HIT CS&E

Performance Analysis

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

$\leq cn + (19/20)cn + T((19/20)^2 n)$

$\vdots$

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

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

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

$\leq 20\ cn + b$

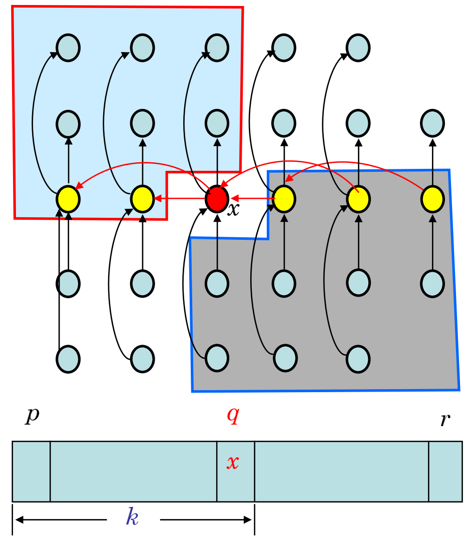
$= O(n)$

HIT
CS&E

- **Time complexity analysis**
  - The number of elements that greater than the partition element *x* is at least

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

  - Thus, in the worst case, the number of elements that great than the *x* is at most
    n-((3n/10)-6) =7n/10+6.
  - Similarly, the number of elements that less than the *x* is also at most
    7n/10+6



HIT
CS&E

1. Divide n elements in A into $\lceil n/5 \rceil$ groups of 5 elements each, at most one group has (n mod 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^{th} smallest element on the low side if $i \leq k$, or the $(i-k)^{th}$ smallest element on the high side if $i > k$.	$T(7n/10 +6)$

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

HIT
CS&E

- **Now we have**

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

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

HIT
CS&E

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

3.5 Finding the closest pair of points

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,***

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

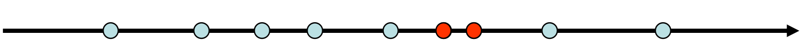
Dis(P_i, P_j) **是Euclidean距离：**

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

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

一维空间算法

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

HIT
CS&E

一维空间算法(续)

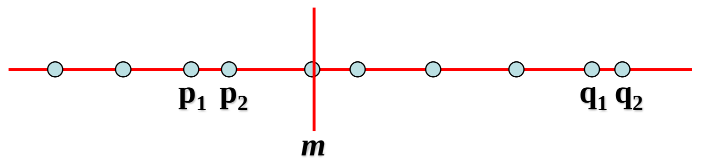
- Divide-and-conquer算法

边界条件:

1. 如果Q中仅包含2个点，则返回这个点对；

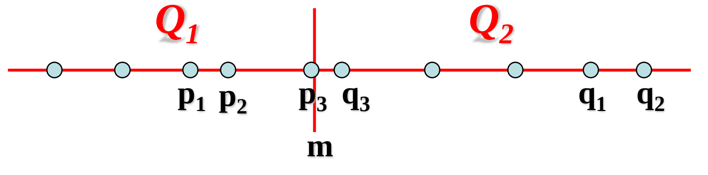
Divide:

2. 求Q中点的中位数m;



HIT
CS&E

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



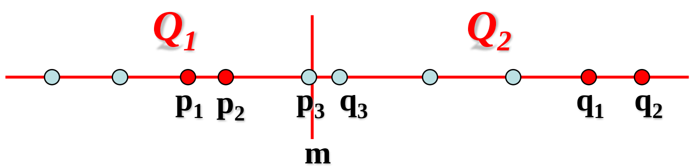
Conquer:

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)

HIT
CS&E

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

HIT
CS&E

- 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$T(n)$

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

HIT
CS&E

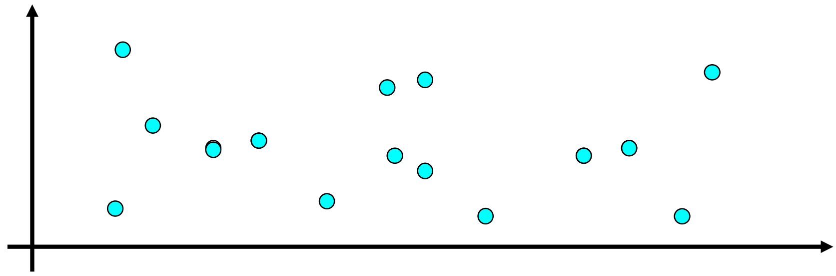
二维空间算法

- 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:

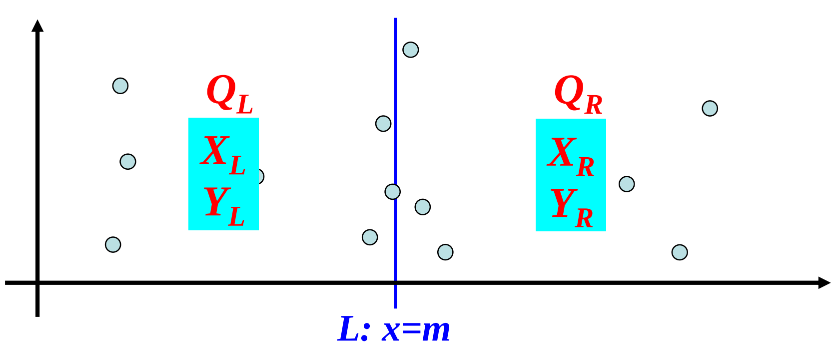
1. 如果Q中仅包含3个点,则返回最近点对,结束;

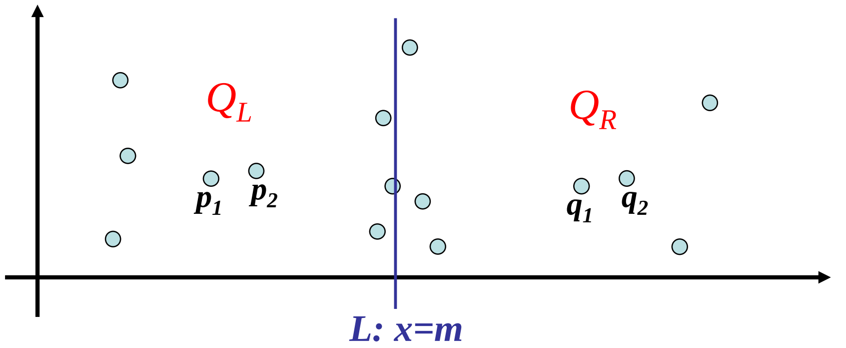


HIT
CS&E

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 ;

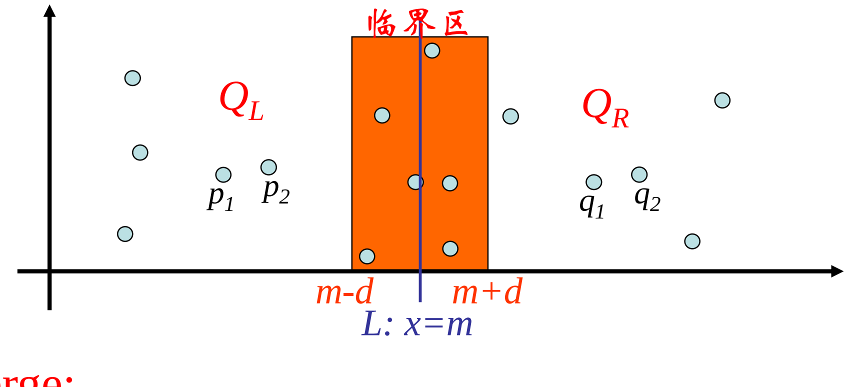




Conquer:

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{$Dis(p_1, p_2)$, $Dis(q_1, q_2)$};



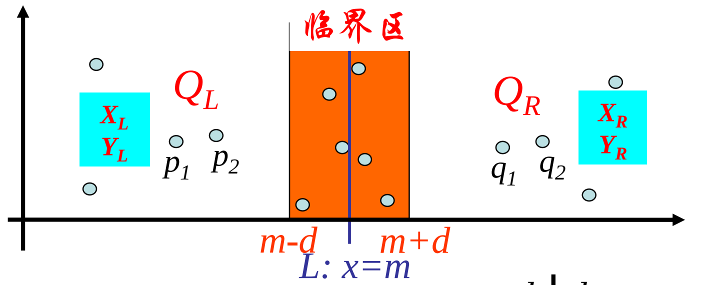
Merge:

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

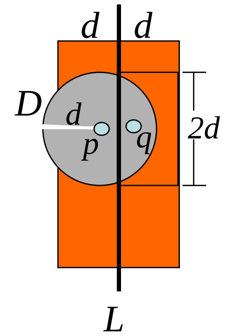
HIT
CS&E

- 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).



HIT
CS&E

- 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

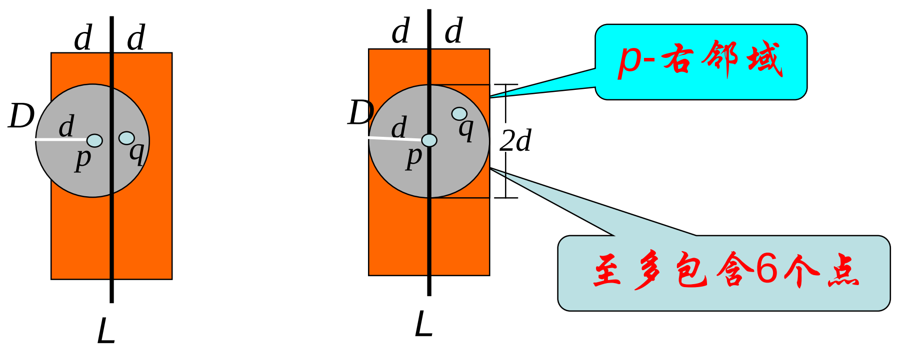
$$T(n)= O(1) \qquad n \leqq 3$$
$$T(n) = 2T(n/2) + O(n) \qquad n > 3$$

  - 用Master定理求解$T(n)$

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

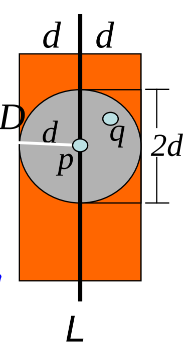
(p_l, q_r)的搜索时间：

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

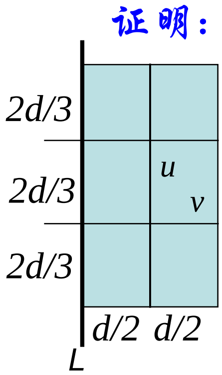


HIT
CS&E

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。

HIT
CS&E

- ~~**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?

HIT
CS&E

3.6 Finding the convex hull

HIT
CS&E

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

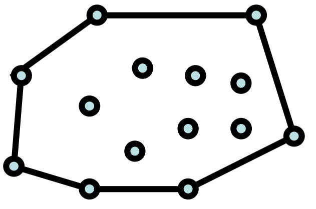
Q的convex hull是一个最小凸多边形P，Q的点或者在P上或者在P内

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

HIT
CS&E

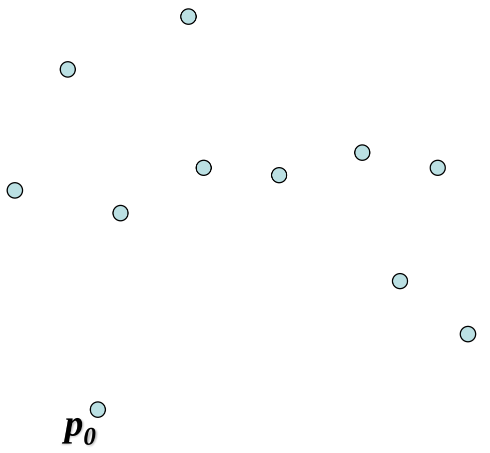
Graham-Scan算法

- 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)。



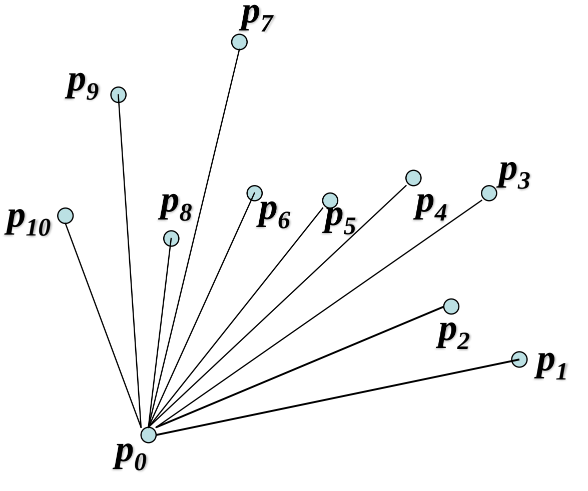
HIT
CS&E

第1步



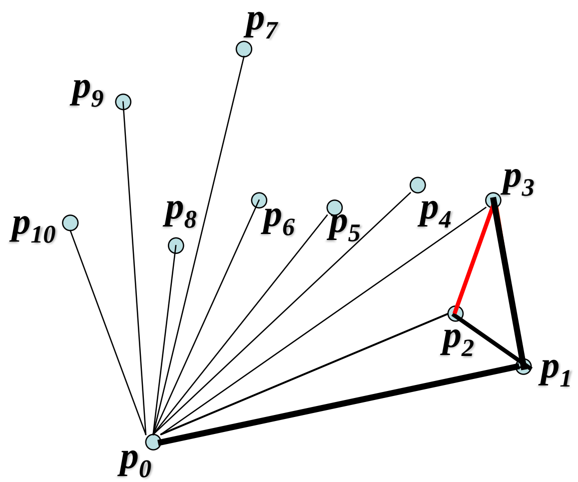
HIT
CS&E

第2步

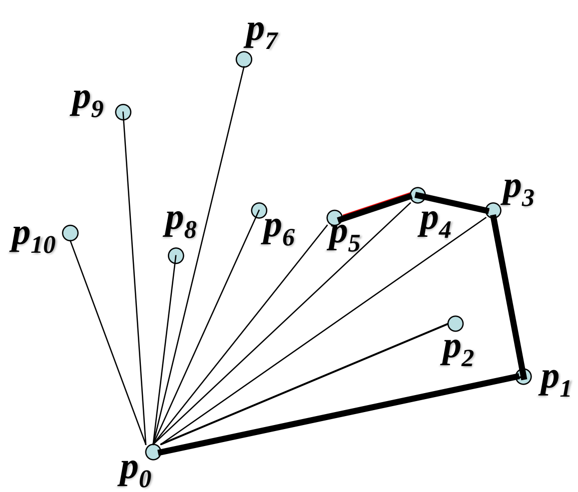


HIT
CS&E

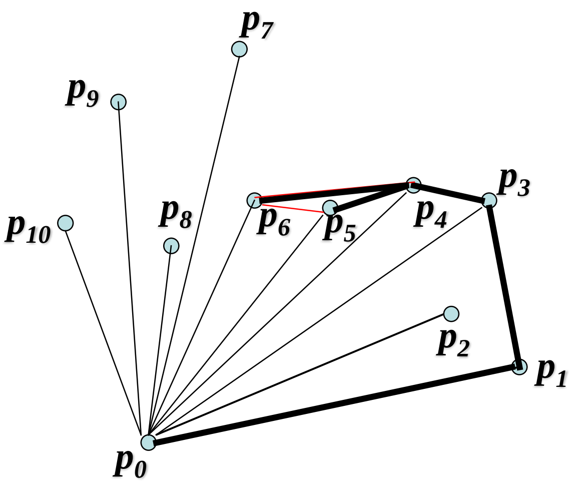
第3步



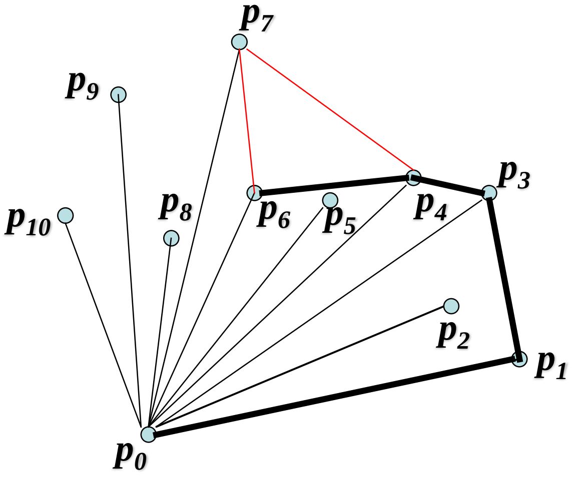
HIT
CS&E

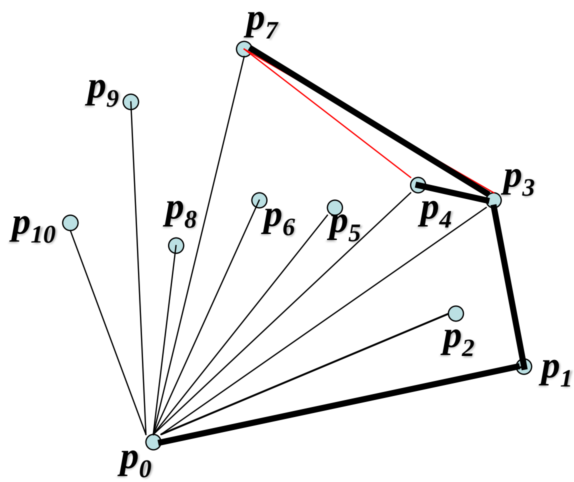


HIT
CS&E

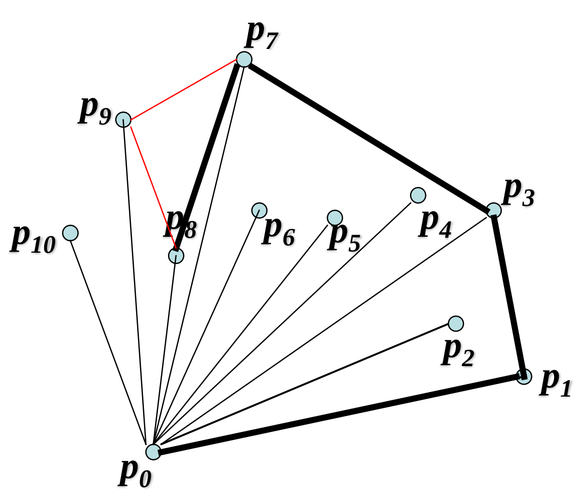


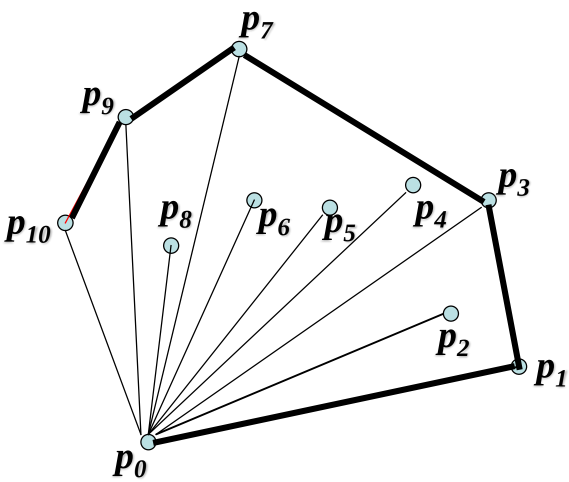
HIT
CS&E





HIT
CS&E

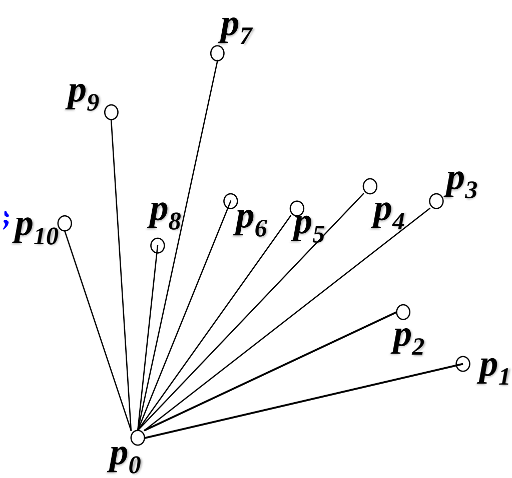




算法Graham-Scan(Q)

/* 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$\langle p_1, p_2, \ldots, p_n \rangle$;
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.



HIT
CS&E

- 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. 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(nlogn)***

HIT
CS&E

- 正确性分析

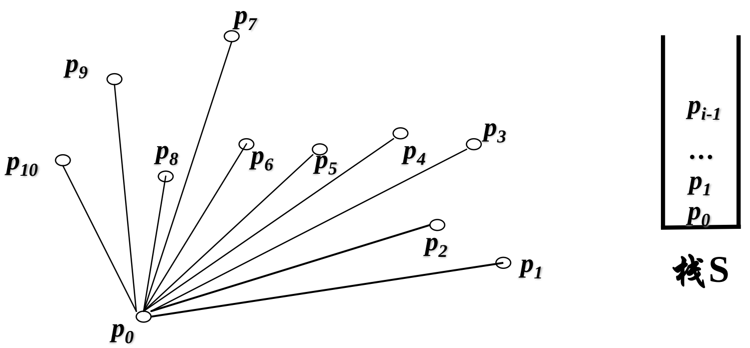
定理. 设n个二维点的集合Q是Graham-Scan算法的输入，$|Q|\geq 3$，算法结束时，栈S中自底到顶存储CH(Q)的顶点（按照逆时针顺序）.

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.



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);

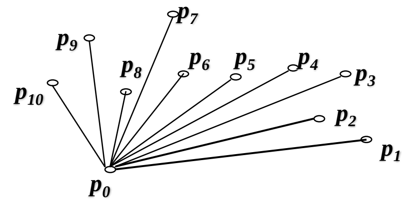
循环不变量

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：

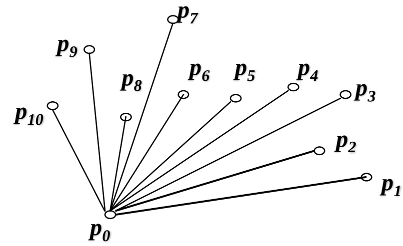
      算法执行5~7步之后， 栈S中自底到顶存储CH(Q_i)的顶点.

```
3.  Push 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);
```



- 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.即： CH($Q_j \cup \{p_i\}$)= CH(Q_i)
  - 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

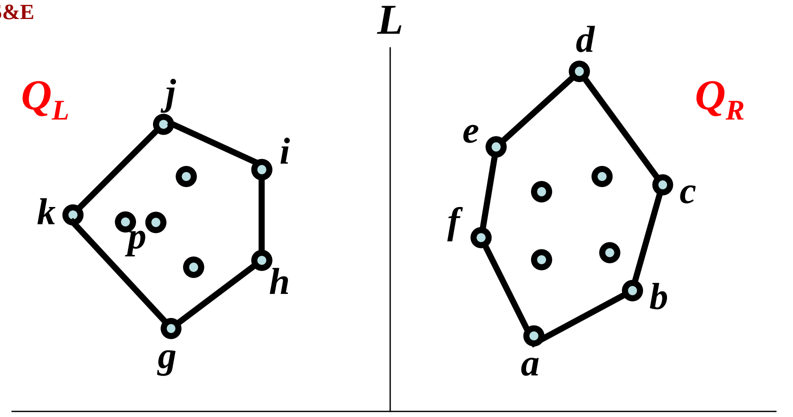
边界条件：(时间复杂性为$O(1)$)

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;

HIT
CS&E



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);

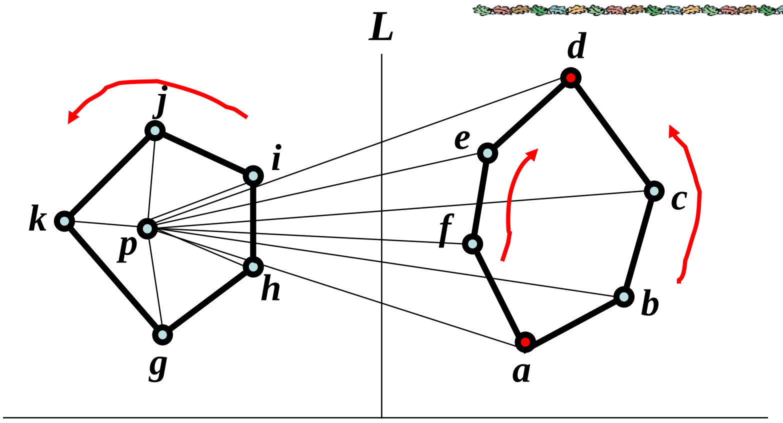
HIT
CS&E

Merge:

我们先通过一个例子来看Merge的思想

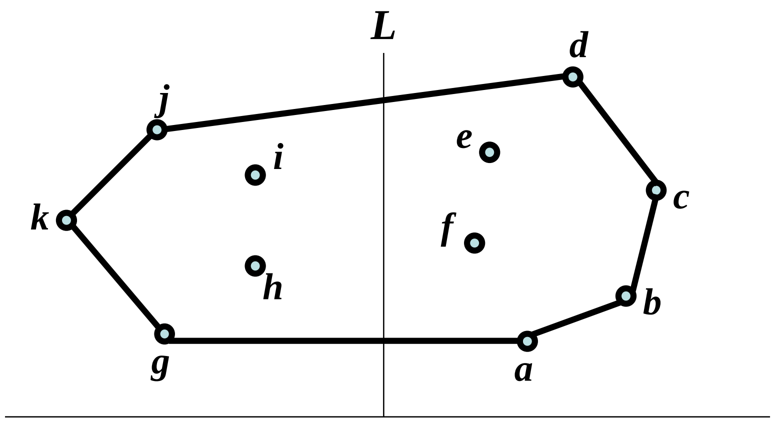
HIT
CS&E

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*>

HIT
CS&E



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

HIT
CS&E

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
  - $O(n)$

HIT
CS&E

时间复杂性

- 总的时间复杂性

  $T(n)=2T(n/2)+O(n)$
- 使用Master定理

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

HIT CS&E

3.7 快速傅里叶变换

HIT CS&E

问题定义

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

输出：$A_0, A_1, \ldots, A_{n-1}$，使得,

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

其中：

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

(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$\Theta(n^2)$

算法的数学基础

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

$$\begin{aligned}A_j&=a_0+a_1w_n^j+a_2w_n^{2j}+a_3w_n^{3j}+a_4w_n^{4j}+\ldots\ldots+a_{n-2}w_n^{(n-2)j}+a_{n-1}w_n^{(n-1)j}\\&=(a_0+a_2w_n^{2j}+a_4w_n^{4j}+\ldots\ldots+a_{n-2}w_n^{(n-2)j})\\&\quad+(a_1w_n^j+a_3w_n^{3j}+a_5w_n^{5j}+\ldots\ldots+a_{n-1}w_n^{(n-1)j})\\&=(a_0+a_2w_n^{2j}+a_4w_n^{4j}+\ldots\ldots+a_{n-2}w_n^{(n-2)j})\\&\quad+w_n^j(a_1+a_3w_n^{2j}+a_5w_n^{4j}+\ldots\ldots+a_{n-1}w_n^{(n-2)j})\end{aligned}$$

HIT
CS&E

算法的数学基础

$$\begin{aligned}A_j=&\underline{(a_0+a_2w_n^{2j}+a_4w_n^{4j}+\ldots\ldots+a_{n-2}w_n^{(n-2)j})}\\&+\underline{w_n^j(a_1+a_3w_n^{2j}+a_5w_n^{4j}+\ldots\ldots+a_{n-1}w_n^{(n-2)j})}\end{aligned}$$

奇数输入项的FFT

偶数输入项的FFT

由于$w_n^2=w_{n/2}$,以及$w_n^{n+k}=w_n^k$,

$$\begin{aligned}A_j=&\underbrace{(a_0+a_2w_{n/2}^{j}+a_4w_{n/2}^{2j}+\ldots\ldots+a_{n-2}w_{n/2}^{\frac{(n-2)j}{2}})}_{B_j}\\&+w_n^j\underbrace{(a_1+a_3w_{n/2}^{j}+a_5w_{n/2}^{2j}+\ldots\ldots+a_{n-1}w_{n/2}^{\frac{(n-2)j}{2}})}_{C_j}\end{aligned}$$

将问题划分为两个子问题

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

HIT CS&E

分治算法过程

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

递归求解： 递归计算$a_0,a_2,\ldots,a_{n-2}$的变换$B_0,B_1,\ldots,B_{n/2-1}$
递归计算$a_1,a_3\ldots,a_{n-1}$的变换$C_0,C_1,\ldots,C_{n/2-1}$

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

HIT CS&E

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.

算法FFT($a_0,a_2,\ldots,a_{n-1}$, n)

输入：$a_0,a_1,\ldots,a_{n-1}$， $n=2^k$

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

1. $W \leftarrow \exp(2\pi i/n)$;
2. If (n=2) Then
3、 $A_0 \leftarrow a_0+a_1$;
4. $A_1 \leftarrow a_0-a_1$;
5. 输出A_0,A_1,算法结束；
6. $B_0,B_1,\ldots,B_{n/2-1} \leftarrow$FFT($a_0,a_2,\ldots,a_{n-2}$, $n/2$)；
7. $C_0,C_1,\ldots,C_{n/2-1} \leftarrow$FFT($a_1,a_3,\ldots,a_{n-1}$, $n/2$)；
8. For j=0 To $n/2-1$
9. $A_j \leftarrow B_j+C_j \cdot W^{j}$;
10. $A_{j+n/2} \leftarrow B_j+C_j \cdot W^{j+n/2}$;
11. 输出$A_0,A_1,\ldots,A_{n-1}$,算法结束；

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

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

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

3.8 Sorting in Linear Time

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

HIT
CS&E

Counting Sort

- Input: *A[1..n]* , 0≤*A[i]*≤*k* for 1≤*i*≤*n*
- Output: *B[1..n]*=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]]]*

HIT
CS&E

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

	0	1	2	3	4	5
C	2	0	2	3	0	1

	1	2	3	4	5	6	7	8
B								

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

	1	2	3	4	5	6	7	8
B					3			

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

	1	2	3	4	5	6	7	8
B		0					3	

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

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

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

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

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

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

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

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

HIT
CS&E

- Algorithm

for $i \leftarrow 0$ to k
  do $C[i] \leftarrow 0;$
for $j \leftarrow 1$ to $length[A]$
  do $C[A[j]] \leftarrow C[A[j]]+1;$
for $i \leftarrow 1$ to k
  do $C[i] \leftarrow C[i]+C[i-1];$
for $j \leftarrow length[A]$ downto 1
  do begin
    $B[C[A[j]]] \leftarrow A[j];$
    $C[A[j]] \leftarrow C[A[j]]-1;$

$A:\ 3,6,4,1,3,4,1,4$

$C:\ 0,0,0,0,0,0$

$C:\ 2,0,2,3,0,1$

$C:\ 2,2,4,7,7,8$

$A:\ 3,6,4,1,3,4,1,\hat{4}$

$B:\ \ ,\ ,\ ,\ ,\ ,\ ,4,$

$C:\ 2,2,4,6,7,8$

HIT
CS&E

- Time complexity

for $i \leftarrow 0$ to k — $O(k)$
  do $C[i] \leftarrow 0;$
for $j \leftarrow 1$ to $length[A]$ — $O(n)$
  do $C[A[j]] \leftarrow C[A[j]]+1;$
for $i \leftarrow 1$ to k — $O(k)$
  do $C[i] \leftarrow C[i]+C[i-1];$
for $j \leftarrow length[A]$ downto 1 — $O(n)$
  do begin
    $B[C[A[j]]] \leftarrow A[j];$
    $C[A[j]] \leftarrow C[A[j]]-1;$

Time Complexity=O(n+k)

- Property of Counting Sort
  - 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

HIT
CS&E

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

329		720		720		329
457		355		329		355
657		436		436		436
839	⇒	457	⇒	839	⇒	457
436		657		355		657
720		329		457		720
355		839		657		839

HIT
CS&E

- Radix sort algorithm

Ininput: 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;

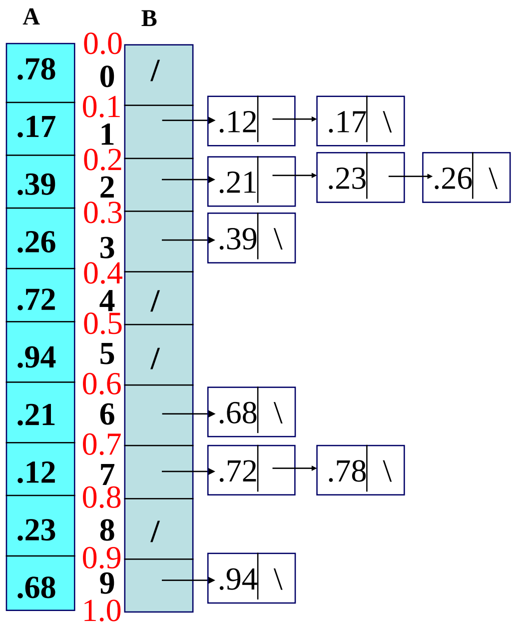
- Time complexity of Radix sort algorithm
  - Using Counting sort algorithm, $0 \leq A[i] \leq k$
  - The time complexity is $O(d(n+k))$
- Problems

HIT
CS&E

- 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?

HIT
CS&E

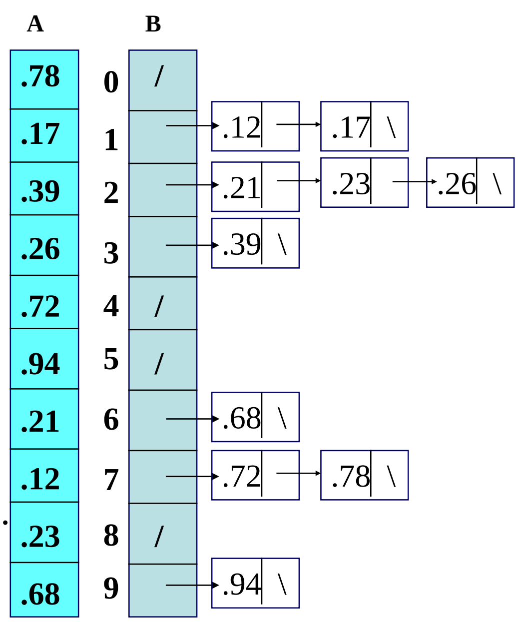
Bucket Sort



- **Assumption of Bucket Sort**
  - Input is elements uniformly distributed in [0, 1)
- **Idea of Bucket Soot**
  - 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

HIT
CS&E

- **Bucket Sort Algorithm**



Bucket-Sort(*A*)

1. $n=length[A]$;
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 $B[0]$, …, $B[n-1]$.

HIT
CS&E

- **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-$1/n$ (Homewok)

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