第11讲 算法分析与问题的计算复杂度

汪小林

北京大学 信息科学技术学院

主要内容

6.1 平凡下界

6.2 直接计数求解该问题所需要的最少运算

6.3 决策树

6.4 检索算法的时间复杂度分析

6.5 排序算法的时间复杂度分析

冒泡排序、堆排序、排序算法的决策树与时间复杂度下界

6.6 选择算法的时间复杂度分析

找最大和最小问题、找第二大问题、找中位数的问题

6.7 通过归约确认问题计算复杂度的下界

寻找最优算法的途径

(1) 设计算法A, 求$W(n)$，得到算法类最坏情况下时间复杂度的一个上界

(2) 寻找函数$F(n)$, 使得对任何算法都存在一个规模为 n 的输入并且该算法在这个输入下至少要做$F(n)$次基本运算，得到该算法类最坏情况下时间复杂度的一个下界

(3) 如果$W(n)=F(n)$或$W(n)=\Theta(F(n))$, 则A是最优的.

(4) 如果$W(n)>F(n)$，A不是最优的或者$F(n)$的下界过低.

改进A或设计新算法A'使得$W'(n)<W(n)$.

重新证明新下界$F'(n)$使得$F'(n)>F(n)$.

重复以上两步, 最终得到$W'(n) = F'(n)$ 或者$W'(n) = \Theta(F'(n))$

6.1 平凡下界

算法的输入规模和输出规模是它的平凡下界

例6.1 问题：写出所有的n阶置换

求解的时间复杂度下界为$\Omega(n!)$

例6.2 问题：求n次实系数多项式多项式在给定x的值

求解的时间复杂度下界为$\Omega(n)$

例6.3 问题：求两个$n\times n$ 矩阵的乘积

求解的时间复杂度下界是$\Omega(n^2)$

例6.4 问题：货郎问题

求解的时间复杂度下界是$\Omega(n^2)$

6.2 直接计数最少运算数

例6.5 找最大

算法 **Findmax**

输入 数组L, 项数 $n \geq 1$

输出 L中的最大项MAX

1. $MAX \leftarrow L(1)$; $i \leftarrow 2$;
2. while $i \leq n$ do
3. if $MAX < L(i)$ then $MAX \leftarrow L(i)$;
4. $i \leftarrow i+1$;

$W(n)=n-1$

以比较作为基本运算的算法类的上界：$n-1$

找最大问题的复杂度

下界：在 n 个数的数组中找最大的数, 以比较做基本运算的算法类中的任何算法在最坏情况下至少要做 $n-1$ 次比较.

证　因为MAX是唯一的, 其它的 $n-1$ 个数必须在比较后被淘汰. 一次比较至多淘汰一个数, 所以至少需要 $n-1$ 次比较.

结论: **Findmax** 算法是最优算法.

6.3 决策树（二叉树的性质）

命题6.1 在二叉树的 t 层至多 2^t 个结点（根为0层）

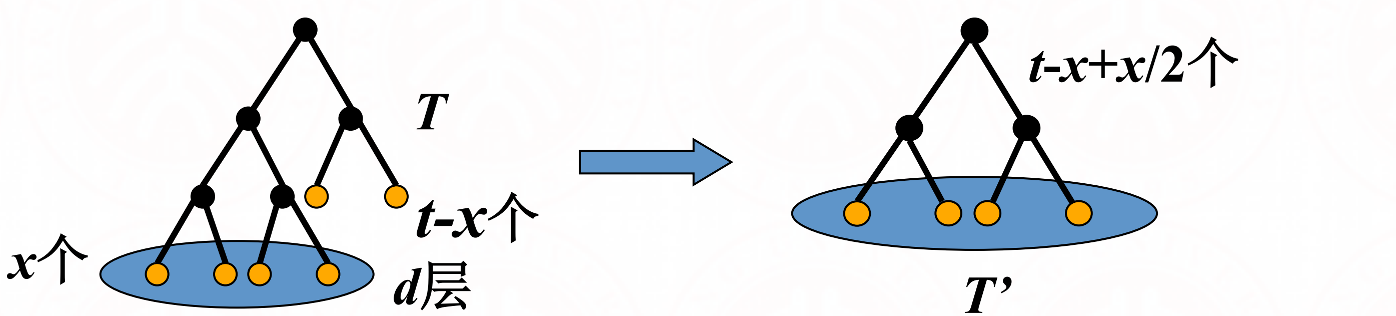
命题6.2 深度为 d 的二叉树至多 $2^{d+1}-1$ 个结点.

命题6.3 n个结点的二叉树的深度至少为 $\lfloor \log n \rfloor$.

命题6.4 设 t 为二叉树的树叶个数，d 为树深，如果树的每个内结点都有2个儿子，则 $t \le 2^d$.

归纳法：$d=0$, 命题为真. 假设对小于d 的深度为真，设 T 深度为 d ，树叶数 t. 取走 T 的 d 层 树叶，得到T'. T'的深度为$d-1$，树叶数 t'。 $t' = (t-x) + x/2 = t - x/2$， $x \le 2^d$

$$t = t' + x/2 \le 2^{d-1} + 2^{d-1} = 2^d$$



6.4 检索算法时间复杂度分析

检索问题：给定按递增顺序排列的数组 L (项数 $n \geq 1$)和数 x，如果 x 在 L 中, 输出 x 的下标; 否则输出0.

算法　顺序检索

输入：L, x

输出：j

1. $j \leftarrow 1$
2. while $j \leq n$ and $L(j) \neq x$ do $j \leftarrow j+1$
3. if $j > n$ then $j \leftarrow 0$

分析：设 x 在L中每个位置和空隙的概率都是$1/(2n+1)$

$W(n)=n$

$A(n)=[(1+2+...+n)+n(n+1)]/(2n+1) \approx 3n/4.$

二分检索最坏时间复杂度

定理6.1 $W(n)=\lfloor \log n \rfloor+1 \qquad n\geq 1$

证　对n归纳

n=1时, 左= $W(1)$=1, 右 $=\lfloor \log 1 \rfloor+1=1$.

假设对一切k, $1\leq k<n$, 命题为真, 则

$$
\begin{aligned}
W(n) &= 1+W(\lfloor \tfrac{n}{2} \rfloor) \\
&= 1+\lfloor \log \lfloor \tfrac{n}{2} \rfloor \rfloor+1 \\
&= \begin{cases} \lfloor \log n \rfloor+1 & n\text{为偶数} \\ \lfloor \log(n-1) \rfloor+1 & n\text{为奇数} \end{cases} \\
&= \lfloor \log n \rfloor+1
\end{aligned}
$$

二分检索的平均时间复杂度

令 $n=2^k-1$, S_t 是算法做 t 次比较的输入个数, $1\le t\le k$

则

$$S_1=1=2^0,\ S_2=2=2^1,\ S_3=2^2,\ S_4=2^3, \ldots ,$$

$$S_t= 2^{t-1},\ t<k$$

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

其中 2^{k-1} 为 x 在表中做 k 次比较的输入个数

$$A(n) = \frac{1}{2n+1}(1S_1 + 2S_2 + \ldots + kS_k)$$

求　和

$$A(n) = \frac{1}{2n+1}(1S_1 + 2S_2 + \ldots + kS_k)$$

$$= \frac{1}{2n+1}[\sum_{t=1}^{k} t2^{t-1} + k(n+1)]$$

$$= \frac{1}{2n+1}[(k-1)2^k + 1 + k(n+1)]$$

$$\approx \frac{k-1}{2} + \frac{k}{2} = k - \frac{1}{2} = \lfloor \log n \rfloor + \frac{1}{2}$$

检索问题的决策树

设A是一个检索算法, 对于给定输入规模 n, A的一棵决策树是一棵二叉树, 其结点被标记为$1, 2, \ldots, n$, 且标记规则是:

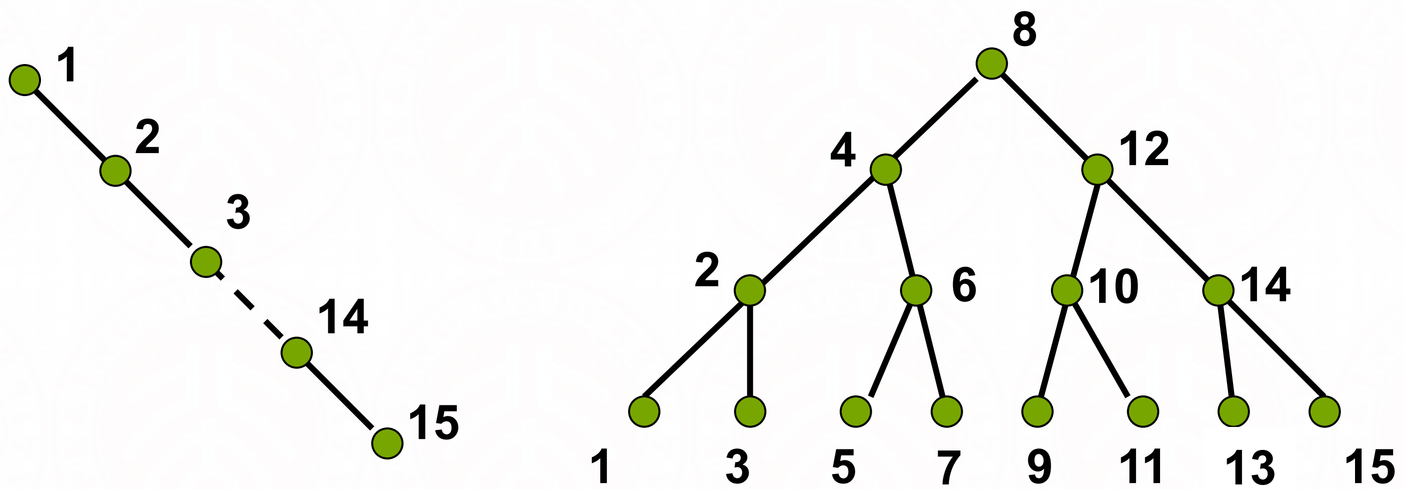
(1) 根据算法A，首先与 x 比较的 L 的项的下标标记为树根.

(2) 假设某结点被标记为i,

- i 的左儿子是：当 $x<L(i)$时，算法A下一步与x比较的项的下标
- i 的右儿子是：当 $x>L(i)$时，算法A下一步与x比较的项的下标
- 若 $x<L(i)$ 时算法 A 停止, 则 i 没有左儿子.
- 若 $x>L(i)$ 时算法 A 停止, 则 i 没有右儿子.

实例

改进顺序检索算法和二分检索算法的决策树，$n=15$



给定输入, 算法 A 将从根开始, 沿一条路径前进, 直到某个结点为止. 所执行的基本运算次数是这条路径的结点个数. 最坏情况下的基本运算次数是树的深度+1.

检索问题的复杂度分析

定理6.2 对于任何一个搜索算法存在某个规模为n 的输入使得该算法至少要做 $\lfloor \log n \rfloor + 1$ 次比较.

证　由命题3, n 个结点的决策树的深度 d 至少为 $\lfloor \log n \rfloor$, 故

$$W(n)=d+1 = \lfloor \log n \rfloor + 1.$$

结论: 对于有序表搜索问题, 在以比较作为基本运算的算法类中, 二分法在最坏情况下是最优的.

6.5 排序算法时间复杂度分析

- 冒泡排序
- 快速排序与二分归并排序
- 堆排序
- 排序算法的复杂度下界

冒泡排序

输入: L, $n \geq 1$.

输出: 按非递减顺序排序的L.

算法 bubbleSort

1. $FLAG \leftarrow n$ //标记被交换的最后元素位置
2. while $FLAG > 1$ do
3. $k \leftarrow FLAG - 1$
4. $FLAG \leftarrow 1$
5. for $j=1$ to k do
6. if $L(j) > L(j+1)$ then do
7. $L(j) \leftrightarrow L(j+1)$
8. $FLAG \leftarrow j$

实例

5	3	2	6	9	1	4	8	7
3	2	5	6	1	4	8	7	9
2	3	5	1	4	6	7	8	9
2	3	1	4	5	6	7	8	9
2	1	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

特点：交换发生在相邻元素之间

置换与逆序

- 逆序 令$L=\{1,2,...,n\}$, 排序的任何输入为L上的置换. 在置换$a_1\ a_2...a_n$中若$i<j$ 但$a_i>a_j$, 则称 (a_i,a_j) 为该置换的一个逆序.
- 逆序序列 在 i 右边，并且小于i 的元素个数记作b_i, $i=1, 2,…,n$. $(b_1, b_2, …, b_n)$ 称为置换的逆序序列
- 实例

置换 3 1 6 5 8 7 2 4

逆序序列为 (0, 0, 2, 0, 2, 3, 2, 3)

逆序序列的性质

- $b_1=0$; $b_2=0,1$; … ; $b_n= 0,1, \ldots , n\text{-}1$
- 总共$n!$个不同的逆序序列

  置换与它的逆序序列构成一一对应
- 逆序数：置换中的逆序总数

$$b_1 + b_2 + \ldots + b_n$$

- 实例

  置换　3 1 6 5 8 7 2 4

  逆序序列为　(0, 0, 2, 0, 2, 3, 2, 3)

  逆序数　12

冒泡排序算法复杂度分析

- 最坏情况分析：$W(n)=O(n^2)$, 至多循环$O(n)$次，每次$O(n)$.
- 对换只发生在相邻元素之间，每次相邻元素交换只消除1个逆序，比较次数不少于逆序数，最大逆序数 $n(n-1)/2$，于是$W(n)=\Omega(n^2)$.
- 平均情况：设各种输入是等可能的，置换 α 的逆序序列是 $(b_1, b_2, \dots , b_n)$, 置换 α' 的逆序序列为 $(0-b_1, 1-b_2, \dots , n-1-b_n)$， α 与 α' 的逆序数之和为 $n(n-1)/2$. $n!$个置换分成 $n!/2$个组，每组逆序之和为 $n(n-1)/2$.

  平均逆序数 $n(n-1)/4$，平均的交换次数$=\Omega(n(n-1)/4)$.
- 冒泡排序的最坏和平均复杂性均为$\Theta(n^2)$

快速排序与二分归并排序

- 快速排序

  最坏情况 $O(n^2)$

  平均情况 $O(n\log n)$

- 二分归并排序

  最坏情况 $O(n\log n)$

  平均情况 $O(n\log n)$

堆排序

- 堆的定义
- 堆的运算
  - 堆整理 **Heapify(*A*,*i*)**
  - 复杂度分析
  - 建堆 **Build-Heap(*A*)**
  - 复杂度分析
- 堆排序算法 **Heap-sort(*A*)**
  - 复杂度分析

堆的定义

设 T 是一棵深度为 d 的二叉树，结点为L中的元素.

若满足以下条件，称作**堆**.

(1) 所有内结点（可能一点除外）的度数为 2

(2) 所有树叶至多在相邻的两层

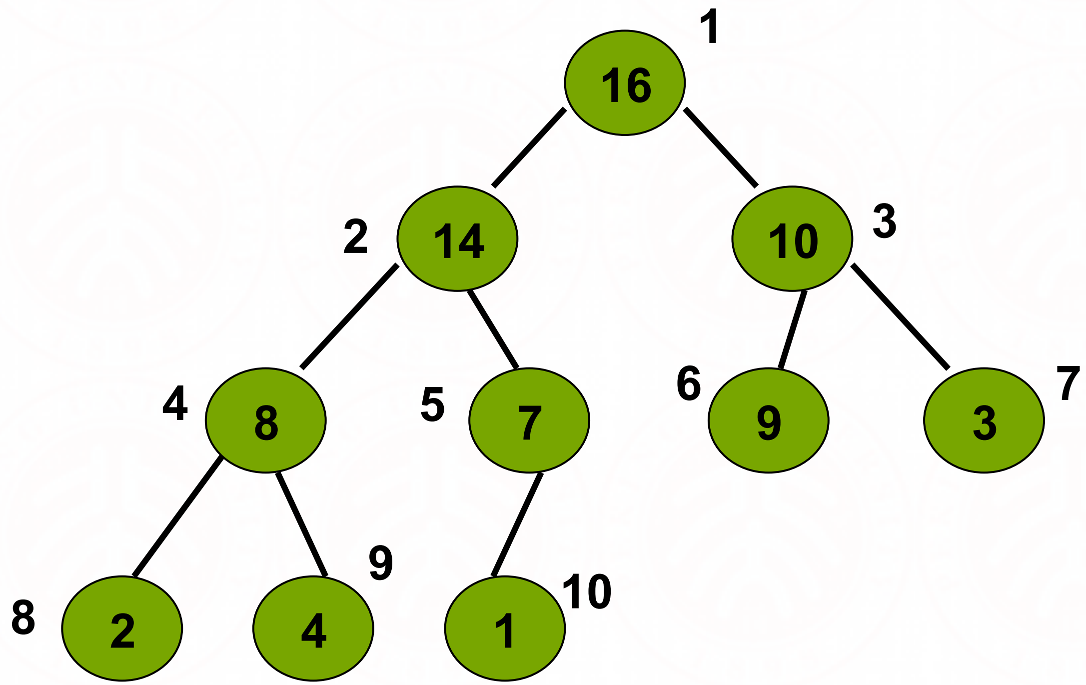
(3) d-1 层的所有树叶在内结点的右边

(4) d-1 层最右边的内结点可能度数为1（没有右儿子）

(5) 每个结点的元素不小于儿子的元素

若只满足前(4)条，不满足第(5)条，称作**堆结构（伪堆）**

实例



堆存储在数组*A*

A[*i*]：结点 *i* 的元素，例如*A*[2]=14.

left(*i*)，*right*(*i*) 分别表示 *i* 的左儿子和右儿子

堆的性质

1. 根据二叉树的性质和堆的定义得到：

$left(i)=2i$， $right(i)=2i+1$

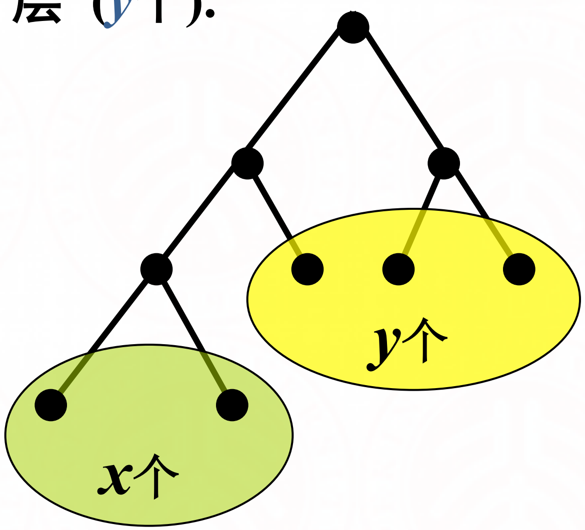
2. 引理6.1 n个结点的堆恰好$\lceil n/2 \rceil$片树叶.

证 树叶分布在 d 和 d-1 层，d 层（x个），d-1 层（y个).

Case1: x为偶数

$$x+y=x+2^{d-1}-\frac{x}{2}=2^{d-1}+\frac{x}{2}$$

$$=\frac{(2^d+x)}{2}=\left\lceil\frac{2^d+x-1}{2}\right\rceil=\left\lceil\frac{n}{2}\right\rceil$$



每个内结点有 2 个儿子，树叶数为

(x为偶数，d-1 层以前各层结点总数 2^d-1)

引理6.1证明（续）

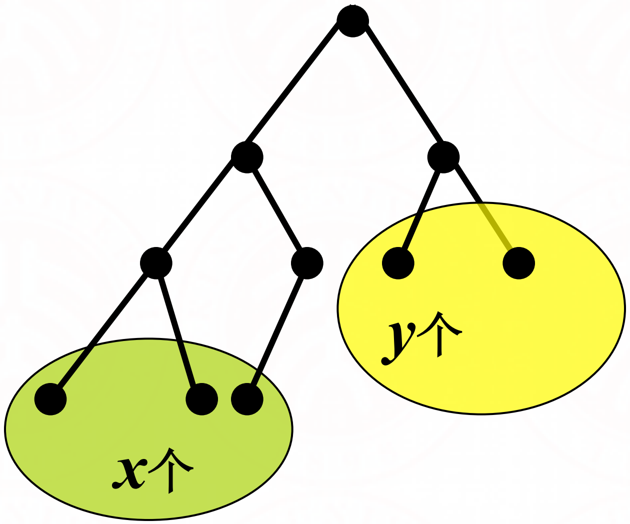
Case2 x为奇数 (x为奇数，n为偶数)

$$x + y = x + 2^{d-1} - \frac{x+1}{2}$$

$$= 2^{d-1} + \frac{x-1}{2}$$

$$= \frac{2^d + x - 1}{2}$$

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



堆的运算：整理

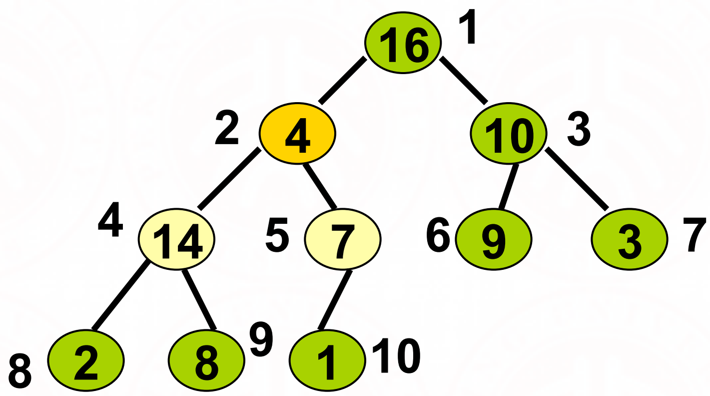
算法6.2 Heapify(A,i)

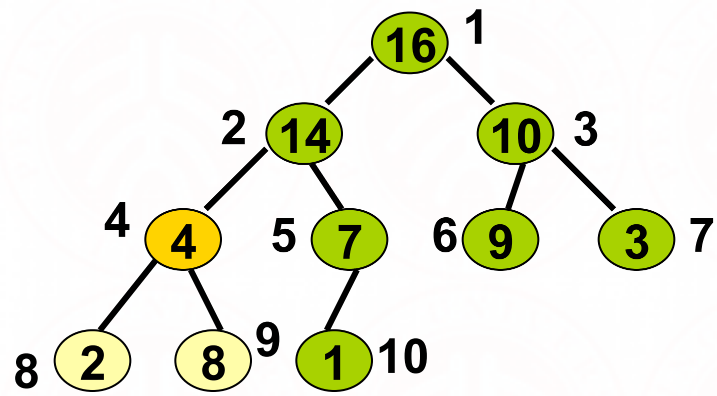
输入：堆结构A, A的结点i

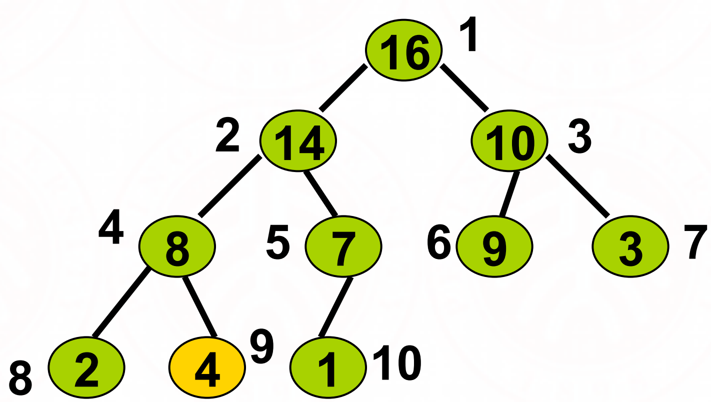
输出：从i向下满足堆存储要求的堆结构

```
l ← left(i)
r ← right(i)
if  l ≤ heap-size[A] and A[l] > A[i]
     then  largest ← l
     else  largest ← i
if  r ≤ heap-size[A]  and  A[r] > A[largest]
     then  largest ← r
if  largest ≠ i
     then exchange A[i] ↔ A[largest]
              Heapify(A, largest)
```

Heapify 实例







Heapify(A,2)

复杂度分析

每次调用为$O(1)$

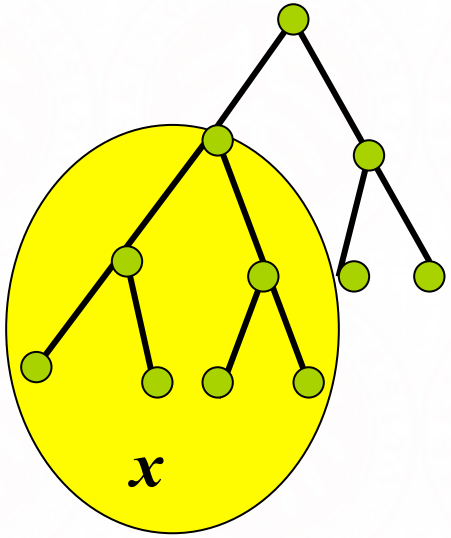
子堆大小至多为原来的 2/3

递推不等式

$$T(m) \le T(2m/3) + \Theta(1)$$

解得 $T(m) = \Theta(\log m)$

或者 $T(h) = \Theta(h)$



说明：

h为 结点 i 的高度(距树叶最大距离)，m为以i为根的子树的结点数

结点总数

$x+(x-1)/2+1=(3x+1)/2$

堆的运算：建堆

算法6.3 **Build-Heap(A)**

输入：数组A

输出：堆A

1．$heap\text{-}size[A] \leftarrow length[A]$

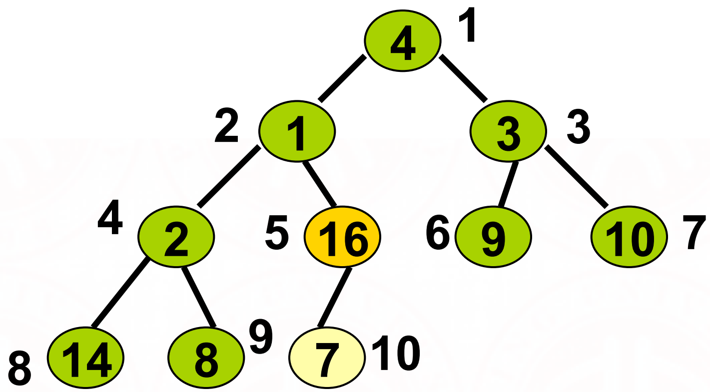
2．**for** $i \leftarrow \lfloor length[A]/2 \rfloor$ **downto** 1

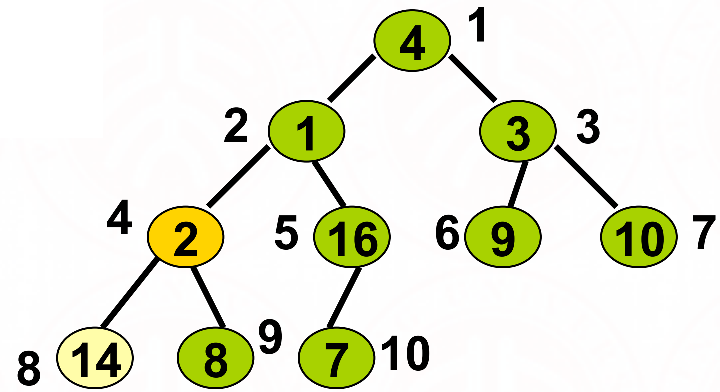
3．   **do** Heapify(A, i)

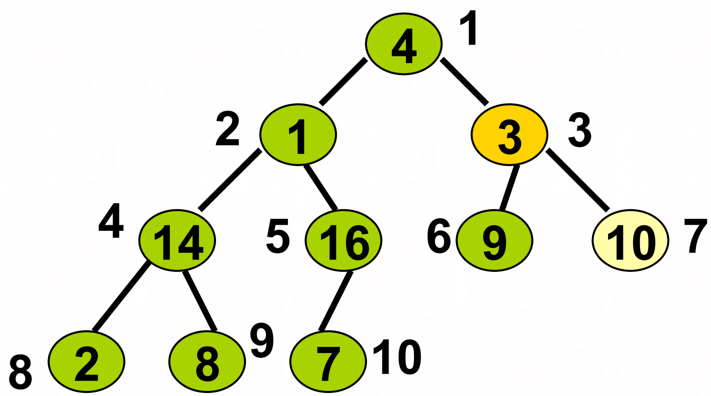
说明：堆有n个结点，由引理 6.1，树叶有$\lceil n/2 \rceil$片，

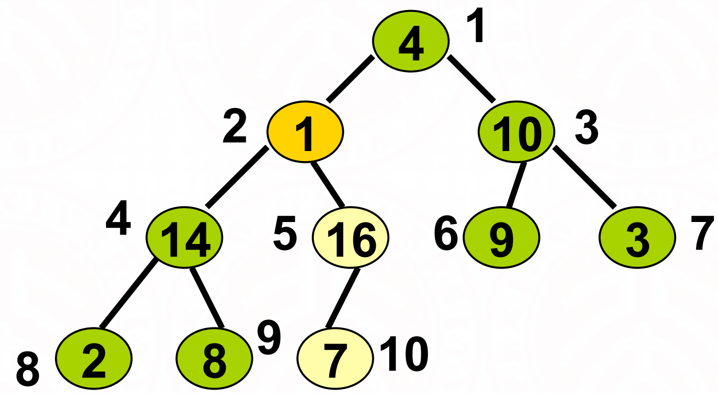
最大内结点标号为 $n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$

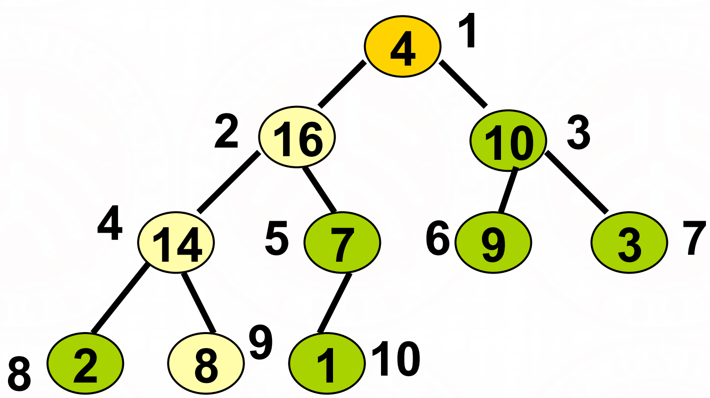
实例

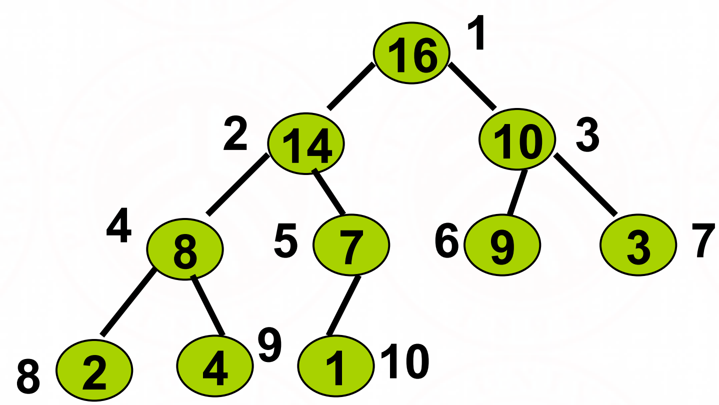












时间复杂度分析

- 结点的高度：该结点距树叶的距离
- 结点的深度：该结点距树根的距离
- 同一深度结点分布在树的同一层
  同一高度结点可以分布在树的不同的层
- 思路：
  按照高度计数结点数，乘以$O(h)$, 再求和
  Heapify(i) 的复杂度依赖于 i 的高度

$$T(n)=\sum_{h=0}^{\lfloor \log n \rfloor} \text{高为}h\text{的结点数} \times O(h)$$

计数高度为h的结点数

引理6.2

n个元素的堆高度 h 的层至多存在 $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ 个结点.

证明思路：对 h 进行归纳.

归纳基础：$h=0$，引理6.1

归纳步骤：假设对 $h-1$为真，证明对 h 也为真.

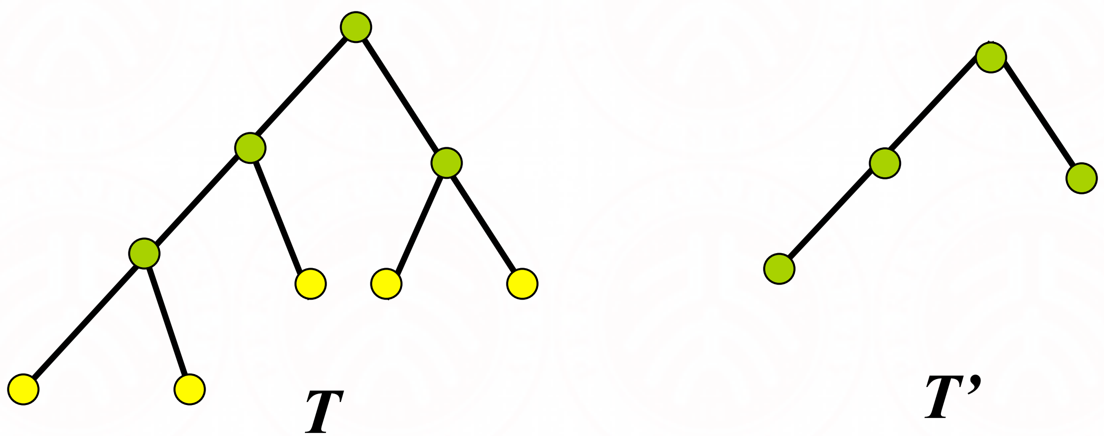
归纳步骤

假设命题对于高度 h-1为真，证明对于高度为 h 也为真.

设 T 表示 n 个结点的树，从 T 中移走所有的树叶得到树 T',
T 与T'的关系：

T 为 h 的层恰好是 T' 的高为 h-1 的层.

$n=n'+n_0$ （T' 的结点数为 n', n_0为T的树叶数）



归纳步骤（续）

令 n_h 表示 T 中高为 h 的层的结点数

根据归纳基础，$n_0 = \left\lceil \frac{n}{2} \right\rceil$

$$n' = n - n_0 = \left\lfloor \frac{n}{2} \right\rfloor$$

$$n_h = n'_{h-1}$$

$$n_h = n'_{h-1} \le \left\lceil \frac{n'}{2^h} \right\rceil = \left\lceil \frac{\left\lfloor \frac{n}{2} \right\rfloor}{2^h} \right\rceil \le \left\lceil \frac{\frac{n}{2}}{2^h} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

时间复杂度分析

定理6.3 n 个结点的堆算法 **Build-Heap** 的时间复杂性是$O(n)$.

证明：对高为 h 的结点调用Heapify算法时间是$O(h)$,

根据引理，高为 h 的结点数至多为$\left\lceil \frac{n}{2^{h+1}} \right\rceil$, 因此时间复杂度

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$$

$$= O(n\sum_{h=0}^{\infty} \frac{h}{2^{h+1}}) = O(n)$$

求　　和

$$\sum_{h=0}^{\infty}\frac{h}{2^h}=[0+\frac{1}{2}+\frac{2}{2^2}+\frac{3}{2^3}+\ldots]$$

$$=[\frac{1}{2}+\frac{1}{2^2}+\ldots]+[\frac{1}{2^2}+\frac{1}{2^3}+\ldots]+[\frac{1}{2^3}+\frac{1}{2^4}+\ldots]+\ldots$$

$$=[1+\frac{1}{2}+\frac{1}{2^2}+\ldots][\frac{1}{2}+\frac{1}{2^2}+\ldots]$$

$$=\frac{1}{2}\frac{1}{(1-\frac{1}{2})^2}=2$$

堆排序算法

算法 **Heap-sort(A)**

1. **Build-Heap(A)**
2. **for** $i \leftarrow length[A]$ **downto 2**
3. **do** *exchange* $A[1] \leftrightarrow A[i]$
4. $heap\text{-}size[A] \leftarrow heap\text{-}size[A]-1$
5. **Heapify(A,1)**

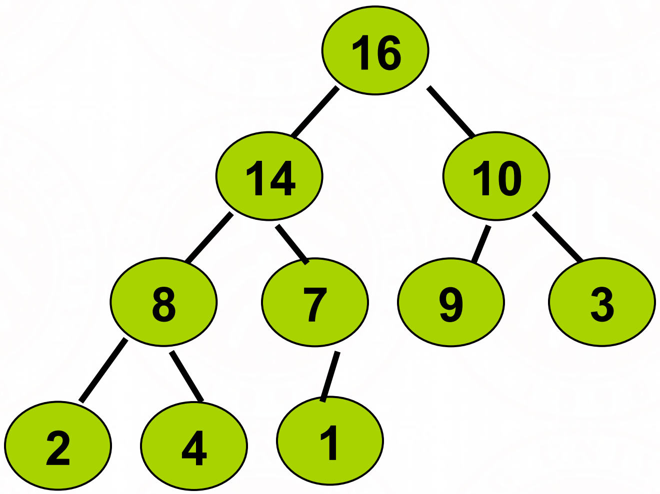
复杂性：$O(n\log n)$

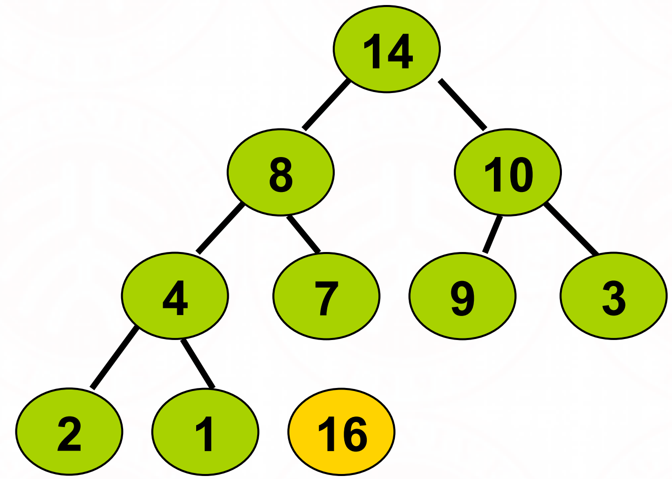
Build-Heap 时间为 $O(n)$,

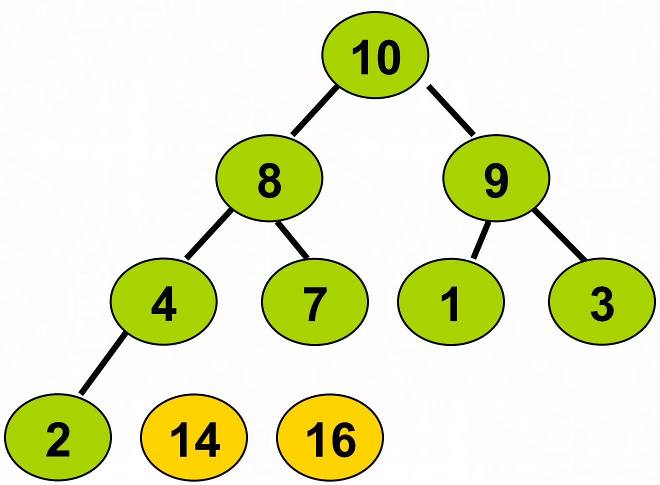
从行 2-5 调用 **Heapify** n-1次，每次$O(\log n)$，

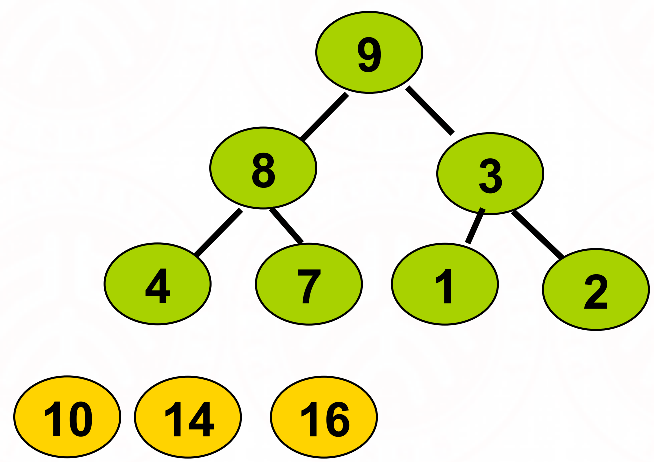
时间为$O(n\log n)$

实　例

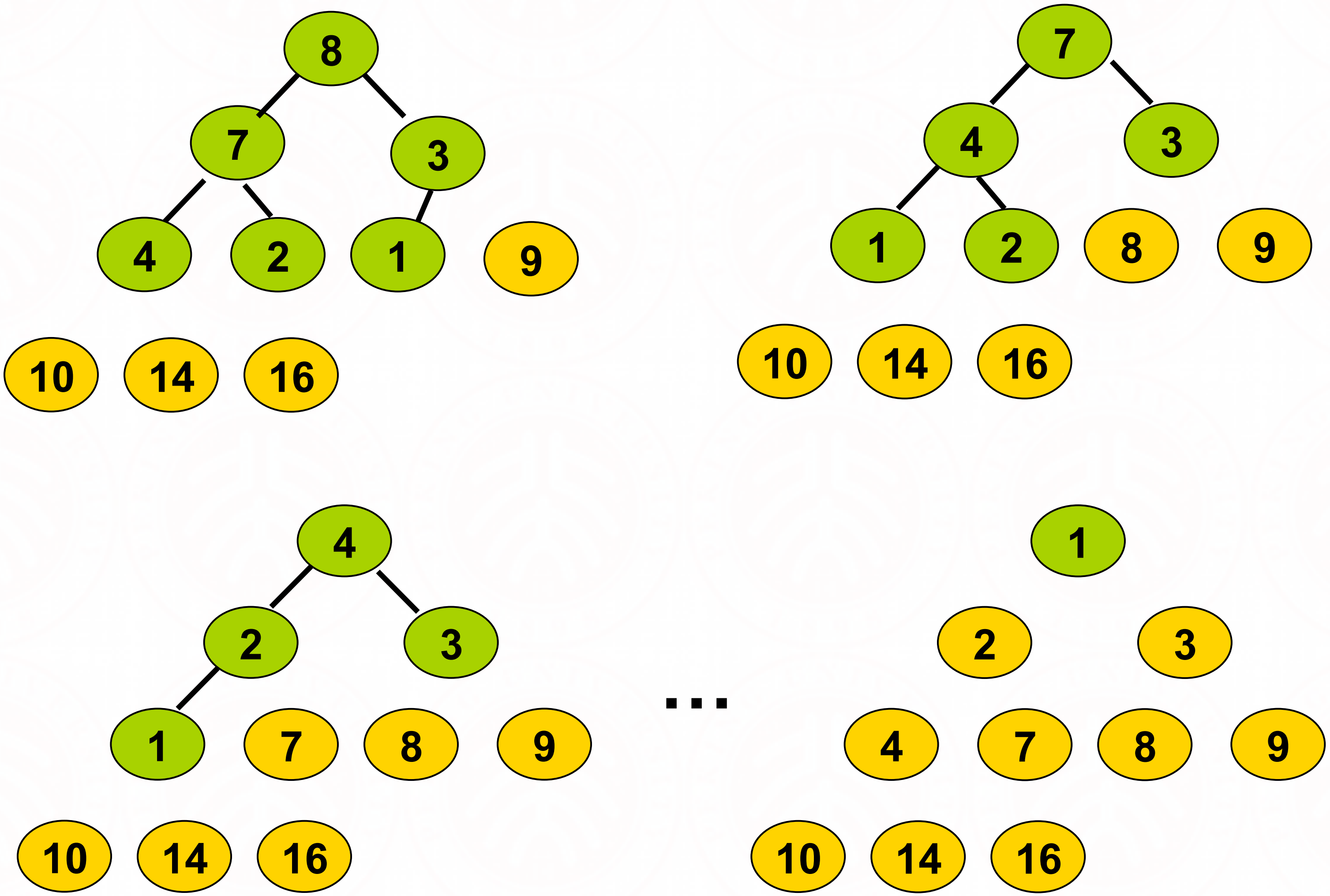

16
14
10
8
7
9
3
2
4
1


14
8
10
4
7
9
3
2
1
16


10
8
9
4
7
1
3
2
14
16


9
8
3
4
7
1
2
10
14
16

实 例


8
7 3
4 2 1 9
10 14 16
7
4 3
1 2 8 9
10 14 16
4
2 3
1 7 8 9
10 14 16
…
1
2 3
4 7 8 9
10 14 16

排序问题的决策树

考虑以比较运算作为基本运算的排序算法类，

任取算法 A, 输入 $L=\{x_1, x_2, \ldots, x_n\}$, 如下定义决策树：

1．A第一次比较的元素为 x_i, x_j，那么树根标记为 i, j

2．假设结点 k 已经标记为 i, j,

(1) $x_i < x_j$

若算法结束，k 的左儿子标记为输入；

若下一步比较元素 x_p, x_q，那么 k 的左儿子标记为 p,q

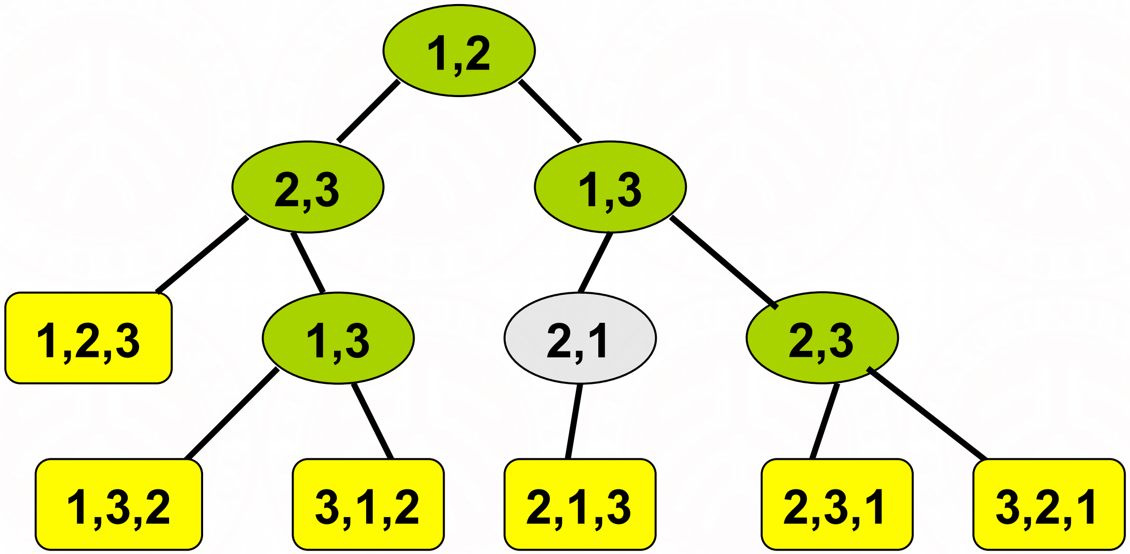
(2) $x_i > x_j$

若算法结束，k 的右儿子标记为输入；

若下一步比较元素 x_p, x_q，那么 k 的右儿子标记为 p,q

一棵冒泡排序的决策树

设输入为 x_1, x_2, x_3，冒泡排序的决策树如下



任意输入：对应了决策树树中从树根到树叶的一条路经，

算法最坏情况下的比较次数：树深

删除非二叉的内结点（灰色结点），得到二叉树叫做 **B-树**

B-树深度不超过决策树深度，**B-**树有$n!$片树叶.

引　　理

引理1　设 t 为B-树中的树叶数，d 为树深，则 $t \le 2^d$.

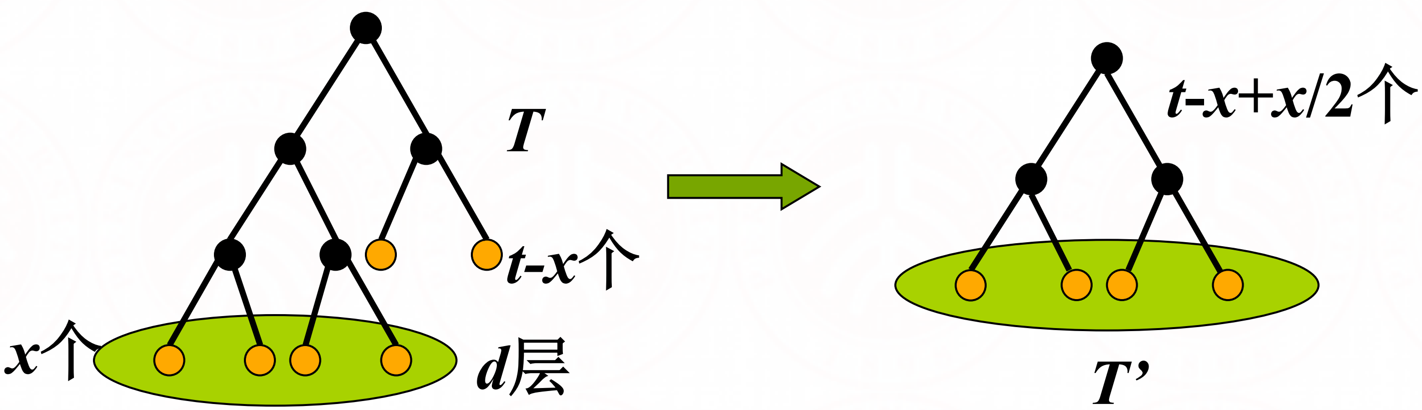
证明　归纳法.

$d=0$, 树只有1片树叶，深度为0，命题为真.

假设对一切小于d 的深度为真，设 T 是一棵深度为 d 的树，树叶数为 t. 取走 T 的 d 层的 x片树叶，得到树T'. 则 T'的深度为d-1，树叶数 t'。那么

$$t' = (t-x) + x/2 = t - x/2,\qquad x \le 2^d$$

$$t = t' + x/2 \le 2^{d-1} + 2^{d-1} = 2^d$$



最坏情况复杂度的下界

引理2 对于给定的n，任何通过比较对 n 个元素排序的算法的决策树的深度至少为 $\lceil \log n! \rceil$.

证明　判定树的树叶有$n!$个，由引理1得证.

定理4 任何通过比较对 n 个元素排序的算法在最坏情况下的时间复杂性是 $\lceil \log n! \rceil$, 近似为 $n \log n - 1.5n$.

证明　最坏情况的比较次数为树深，由引理2树深至少为

$$\log n! = \sum_{j=1}^{n} \log j \geq \int_1^n \log x dx = \log e \int_1^n \ln x dx$$

$$= \log e(n \ln n - n + 1)$$

$$= n \log n - n \log e + \log e$$

$$\approx n \log n - 1.5n$$

结论：堆排序算法在最坏情况阶达到最优.

平均情况分析

epl(T)：假设所有的输入等概分布，令 epl(T) 表示 B-树中从根到树叶的所有路径长度之和, epl(T)/n! 的最小值对应平均情况复杂性的下界.

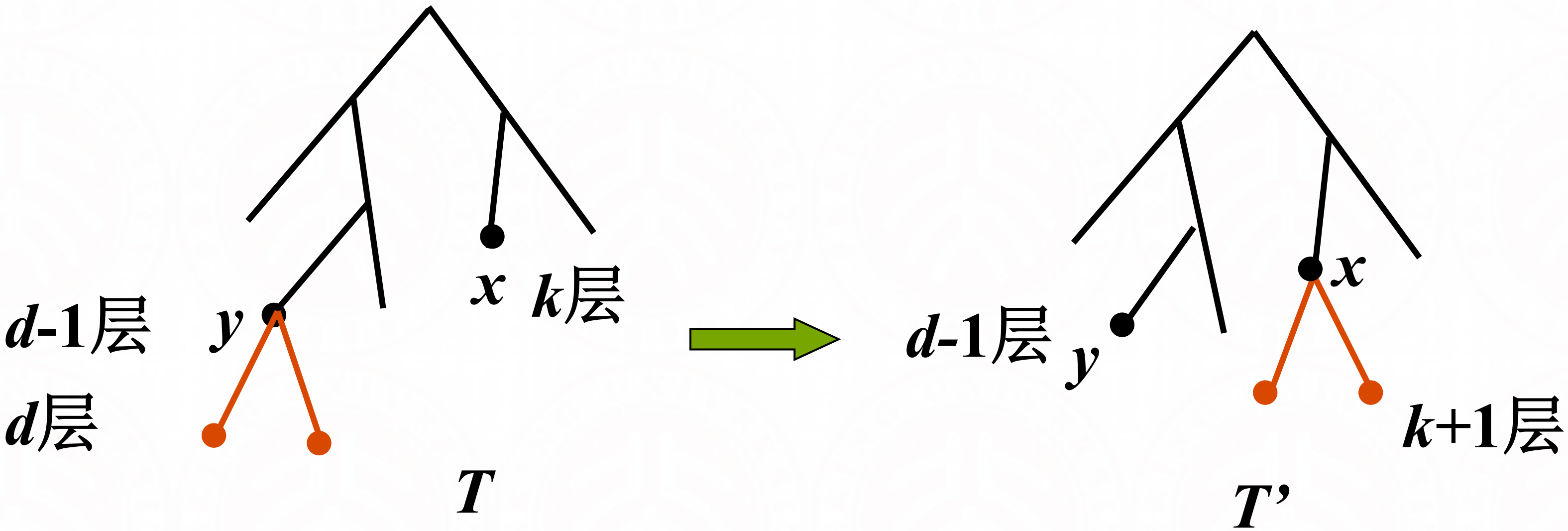
思路：分析具有最小 epl(T) 值的树的结构求得这个最小值.

引理3 在具有 t 片树叶的所有 B-树中,树叶分布在两个相邻层上的树的 epl 值最小

证明：反证法.

设树 T 的深度为 d，假设树叶 x 在第 k 层，$k<d-1$. 取 d-1 层的某个结点 y，y 有两个儿子是第 d 层的树叶.将 y 的两个儿子作为 x 的儿子得到树 T'.

具有最小epl 值的树结构



$$\text{epl}(T) - \text{epl}(T') = (2d+k) - [(d-1)+2(k+1)]$$
$$= 2d+k - d+1-2k-2 = d-k-1 > 0 \qquad (d > k+1)$$

T’的树叶相距层数小于 *T* 的树叶相距的层数，

而 *T*’的 epl 值小于 *T* 的 epl 值

epl值的下界

引理4 具有t片树叶且 epl 值最小的 B-树 T 满足

$$\mathrm{epl}(T) = t\lfloor \log t \rfloor + 2(t - 2^{\lfloor \log t \rfloor})$$

证明：由引理1 树 T 的深度 $d \geq \lceil \log t \rceil$，

由引理3 树 T 只有 d 和 d-1层有树叶.

Case1 $t = 2^k$. 必有$d = k$,

$$\mathrm{epl}(T) = t\,d = t\,k = t\lfloor \log t \rfloor$$

epl值的下界（续）

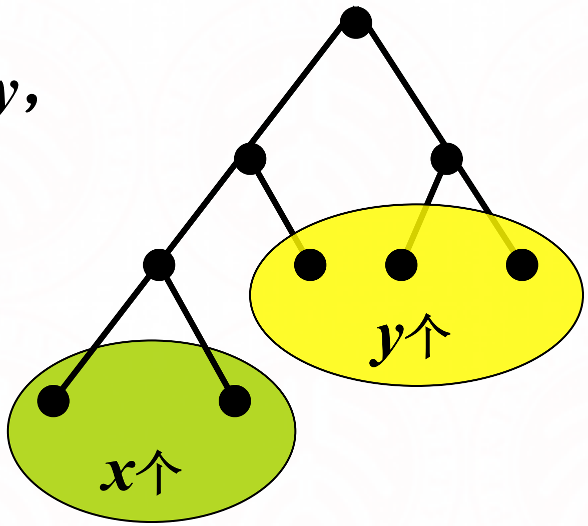
Case2 $t \neq 2^k$.

设 d 层和 $d-1$ 层树叶数分别为 x, y，

$$x + y = t$$

$$x/2 + y = 2^{d-1}$$

解得 $x = 2t - 2^d,\ y = 2^d - t$.



$$\begin{aligned}\mathrm{epl}(T) &= x\,d + y\,(d-1) \\ &= (2t - 2^d)d + (2^d - t)(d-1) \\ &= td - 2^d + t = t(d-1) + 2(t - 2^{d-1}) \\ &= t\lfloor \log t \rfloor + 2(t - 2^{\lfloor \log t \rfloor}) \quad (\lfloor \log t \rfloor = d-1)\end{aligned}$$

平均复杂度的下界

定理4 在输入等概分布下任何通过比较对n个项排序的算法平均比较次数至少为 $\lfloor \log n! \rfloor$, 近似为 $n\log n - 1.5\, n$.

证明：算法类中任何算法的平均比较次数是该算法决策树T 的 epl(T)/n!, 根据引理4

$$
\begin{aligned}
A(n) &\geq \frac{1}{n!} epl(T) \\
&= \frac{1}{n!}\left(n!\lfloor \log n! \rfloor + 2(n! - 2^{\lfloor \log n! \rfloor}\right) \\
&= \lfloor \log n! \rfloor + \varepsilon, \qquad 0 \leq \varepsilon < 1 \\
&\approx n \log n - 1.5n
\end{aligned}
$$

$$
0 \leq n! - 2^{\lfloor \log n! \rfloor} < n! - 2^{\log n! - 1} = n! - \frac{n!}{2} = \frac{n!}{2}
$$

结论：堆排序在平均情况下阶达到最优.

几种排序算法的比较

算法	最坏情况	平均情况	占用空间	最优性
冒泡排序	$O(n^2)$	$O(n^2)$	原地	
快速排序 归并排序	$O(n^2)$ $O(n\log n)$	$O(n\log n)$ $O(n\log n)$	$O(\log n)$ $O(n)$	平均最优 最优
堆排序	$O(n\log n)$	$O(n\log n)$	原地	最优

6.6 选择算法的时间复杂度分析

下界证明方法：构造最坏输入

- 任意给定一个算法 A，A对于任意输入 x 都存在一个确定的操作序列 τ
- τ中的操作分成两类：
  - 决定性的：能够对确定输出结果提供有效信息
  - 非决定性的：对确定结果没有帮助的冗余操作
- 根据算法A构造某个输入实例 x，使得A对x 的操作序列 τ 包含尽量多的非决定性操作.
- 给出冗余操作+必要的操作的计数公式

选择算法的有关结果

	算法	最坏情况	空间
选最大	顺序比较	n-1	O(1)
选最大和最小	顺序比较	2n-3	O(1)
	算法 FindMaxMin	$\lceil 3n/2 \rceil$-2	O(1)
选第二大	顺序比较	2n-3	O(1)
	锦标赛方法	$n+\lceil \log n \rceil$-2	$O(n)$
选中位数	排序后选择	$O(n\log n)$	$O(\log n)$
	算法Select	$O(n)$~2.95n	$O(\log n)$

选最大算法 Findmax是最优的算法

6.6.1 选最大与最小算法

定理6 任何通过比较找最大和最小的算法至少需要 $\lceil 3n/2 \rceil - 2$ 次比较.

证明思路：任给算法A，根据算法 A 的比较结果构造输入T，使得 A 对 T 至少做 $\lceil 3n/2 \rceil - 2$ 次比较.

证：不妨设n个数彼此不等，A为任意找最大和最小的算法. max是最大，A必须确定有n-1个数比max小，通过与max的比较被淘汰. min是最小，A也必须确定有n-1个数比min大，通过与min的比较而淘汰.总共需要$2n$-2个信息单位.

基本运算与信息单位

数的状态标记及其含义：

N：没有参加过比较　　　　W：赢

L：输　　　　　　　　　　WL：赢过且至少输1次

如果比较后数的状态改变，则提供信息单位，状态不变不提供信息单位，每增加 1 个W 提供 1个信息单位

每增加 1 个L 提供 1 个信息单位.

两个变量通过一次比较增加的信息单位个数不同：0,1,2

case1： N,N → W,L：增加2个信息单位

case2： W,N → W,L：增加1个信息单位

case3： W,L → W,L：增加0个信息单位

算法输出与信息单位

算法输出的条件：

n-2 个数带有 **W** 和 **L** 标记，最大数只带 **W** 标记，最小数只带 **L** 标记，总计 2n-2个信息单位

对于任意给定的算法，构造输入的原则是：

根据算法的比较次序，针对每一步参与比较的两个变量的状态，调整对参与比较的两个变量的赋值，使得每次比较后得到的信息单位数达到最小. 从而使得为得到 输出所需要的 2n-2个信息单位，该算法对所构造的输入至少要做 $\lceil 3n/2 \rceil$-2 次比较.

对输入变量的赋值原则

x 与 y 的状态	赋值策略	新状态	信息单位
N,N	$x>y$	**W,L**	**2**
W,N；WL,N	$x>y$	**W,L；WL,L**	**1**
L,N	$x<y$	**L,W**	**1**
W,W	$x>y$	**W,WL**	**1**
L,L	$x>y$	**WL,L**	**1**
W,L；WL,L；W,WL	$x>y$	不变	**0**
WL,WL	保持原值	不变	**0**

一个赋值的实例

x_1,x_2---$x_1>x_2$; x_1,x_5---$x_1>x_5$; x_3,x_4---$x_3>x_4$; x_3,x_6---$x_3>x_6$
x_3,x_1---$x_3>x_1$; x_2,x_4---$x_2>x_4$; x_5,x_6---$x_5>x_6$; x_6,x_4---$x_6>x_4$ …

	x_1	x_2	x_3	x_4	x_5	x_6
	状态值	状态值	状态值	状态值	状态值	状态值
	N *	N *	N *	N *	N *	N *
$x_1>x_2$	W 20	L 10				
$x_1>x_5$	W 20				L 5	
$x_3>x_4$			W 15	L 8		
$x_3>x_6$			W 15			L 12
$x_3>x_1$	WL $\underline{20}$		W $\underline{25}$			
$x_2>x_4$		WL $\underline{10}$		L 8		
$x_5>x_6$					WL $\underline{5}$	L 3
$x_6>x_4$				L $\underline{2}$		WL $\underline{3}$

构造的输入为（20，10，25，2，5，3）

问题复杂度的下界

为得到$2n$-2个信息单位，对上述输入A至少做 $\lceil 3n/2 \rceil$-2 次比较.
一次比较得到2个信息单位只有case1．A至多有 $\lfloor n/2 \rfloor$ 个case1，至多得到 $2\lfloor n/2 \rfloor \le n$个信息单位．其它case, 1次比较至多获得1个信息单位，至少还需要 n-2次比较.

当 n 为偶数，A做的比较次数至少为

$$\lfloor n/2 \rfloor + n\text{-}2 = 3n/2 - 2 = \lceil 3n/2 \rceil - 2$$

当 n 为奇数，A做的比较次数至少为

$$\lfloor n/2 \rfloor + n\text{-}2 + 1 = (n\text{-}1)/2 + 1 + n\text{-}2 = \lceil 3n/2 \rceil\text{-}2$$

结论：FindMaxMin是最优算法

6.6.2 找第二大问题

元素x的权：$w(x)$, 表示以x为根的子树中的结点数

初始，$w(x_i)=1$，$i=1, 2, \ldots, n$;

赋值原则： 在比较的时候进行赋值或者调整赋值. 只对没有失败过的元素（权大于0的元素）进行赋值. 权大者胜，原来胜的次数多的仍旧胜，输入值也大.

1. $w(x), w(y)>0$:

   若 $w(x)>w(y)$, 那么x的值大于y 的值；//权大者胜

   若 $w(x)=w(y)$, 那么x的值大于y 的值；//权等，任意分配

2. $w(x)=w(y)=0$, 那么x, y值不变；// x与 y 比较对于确定第二大无意义

实　　例

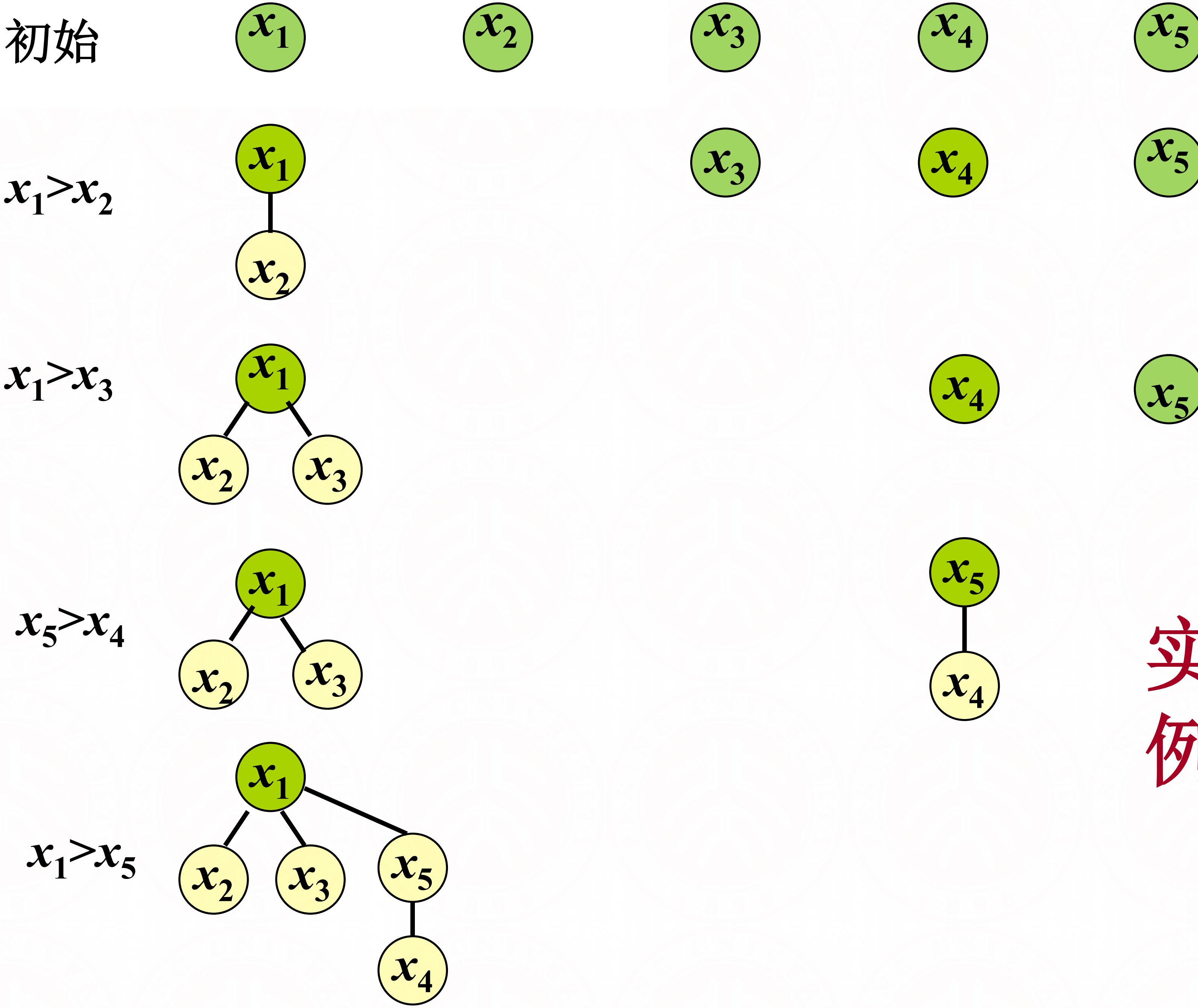
	$w(x_1)$	$w(x_2)$	$w(x_3)$	$w(x_4)$	$w(x_5)$	值
初始	1	1	1	1	1	*, *, *, *, *
第1步 $x_1>x_2$	2	0	1	1	1	20, 10, *, *, *
第2步 $x_1>x_3$	3	0	0	1	1	20, 10, 15, *, *
第3步 $x_5>x_4$	3	0	0	0	2	20, 10, 15, 30, 40
第4步 $x_1>x_5$	5	0	0	0	0	41, 10, 15, 30, 40

构造树

根据算法A 的比较次序, 在比最大的过程中如下构造树：

1. 初始是森林，含有n 个结点；
2. 如果 x, y 是子树的树根，则算法比较 x, y；
3. 若 x, y 以前没有参加过比较，任意赋值给 x, y，比如 $x>y$；那么将 y 作为 x 的儿子；
4. 若 x, y已经在前面的比较中赋过值，且 $x>y$，那么把 y 作为 x 的儿子，以 y 为根的子树作为 x 的子树；

实例



找第二大问题复杂度下界

针对这个输入，估计与max比较而淘汰的元素数

根的权为n，其它的结点权为0，根为max

w_k 表示 max 在它第 k 次比较后形成以max为根子树的结点总数，则 $w_k \le 2w_{k-1}$，设 K 为max最终与权不为0的结点的比较次数，则

$$n = w_K \le 2^K w_0 \le 2^K \Rightarrow K \ge \log n \Rightarrow K \ge \lceil \log n \rceil$$

这 K 个元素彼此不同，因为同一个元素不可能被 计数 2 次. 其中为确定第二大，要淘汰K-1个元素, 至少用$\lceil \log n \rceil$ -1 次比较.

结论：锦标赛方法是找第二大的最优算法.

6.6.3 找中位数问题

定理8 设n为奇数，任何通过比较运算找n个数的中位数(median）的算法在最坏情况下至少做 $3n/2-3/2$ 次比较

证 首先定义决定性的比较与非决定性的比较.

决定性的比较: 建立了x与 median 的关系的比较.

$\exists y$ ($x>y$ 且 $y \geq$ median), x满足上述条件的第一次比较

$\exists y$ ($x<y$ 且 $y \leq$ median), x满足上述条件的第一次比较

（比较时 y 与median的关系可以不知道）

非决定性的比较：当$x>$median, $y<$median, 这时$x>y$的比较不是决定性的.

为找到中位数，必须要做 $n-1$ 次决定性的比较.

针对算法构造输入，使得非决定性比较达到$(n-1)/2$次.

输入构造方法

1. 分配一个值给中位数 **median**；
2. 如果A比较x与y，且x 与 y 没有被赋值，那么赋值x,y 使得
   $x>\text{median}$, $y<\text{median}$;
3. 如果A比较x与y, 且$x>\text{median}$, y没被赋值，则赋值 y 使得
   $y<\text{median}$;
4. 如果A比较x与y, 且$x<\text{median}$, y没被赋值，则赋值y 使得
   $y>\text{median}$;
5. 如果存在 $(n-1)/2$个元素已得到小于**median**的值，则对未赋值的全部分配大于**median**的值；
6. 如果存在 $(n-1)/2$个元素已得到大于**median**的值，则对未赋值的全部分配小于**median**的值.
7. 如果剩下1个元素则分配**median**给它.

构造实例

x_1,x_2---$x_1>x_2$； x_3,x_4---$x_3>x_4$； x_5,x_6---$x_5>x_6$；
x_1,x_3---$x_1>x_3$； x_3,x_7---$x_3>x_7$； x_7,x_4---$x_7>x_4$； …

1. 初始　　median=4
2. $x_1>x_2$　　$x_1=7$, $x_2=1$
3. $x_3>x_4$　　$x_3=5$, $x_4=2$
4. $x_5>x_6$　　$x_5=6$, $x_6=3$
5. $x_7=4$
6. $x_1>x_3$
7. $x_3>x_7$
8. …

非决定性比较

决定性比较

复杂性分析

元素状态　　N：未分配值；S：得到小于median值；L：得到大于median值

比较前的状态	分配策略
N, N	一个大于median，一个小于median
L, N 或 N, L	分配给状态N的元素的值小于median
S, N 或 N, S	分配给状态N的元素的值大于median

这样赋值的输入使得A在这个输入下所进行的上述比较都是非决定性的. 这样的比较至少有$(n-1)/2$个. 因此总比较次数至少为

$$(n-1)+(n-1)/2 = 3n/2-3/2$$

结论：Select算法在阶上达到最优.

几种选择算法的总结

问题	算法	最坏情况	问题下界	最优性
找最大	**Findmax**	$n-1$	$n-1$	最优
找最大最小	**FindMaxMin**	$\lceil 3n/2\rceil-2$	$\lceil 3n/2\rceil-2$	最优
找第二大	锦标赛	$n+\lceil \log n\rceil-2$	$n+\lceil \log n\rceil-2$	最优
找中位数	**Select**	$O(n)$	$3n/2-3/2$	阶最优
找第k小	**Select**	$O(n)$	$n+\min\{k,n-k+1\}-2$	阶最优

6.7 通过归约确认问题计算复杂度的下界

问题P，问题Q

问题Q的复杂度已知（至少线性） $\Omega\ (g(n))$

存在变换 f 将Q的任何实例转换成 P 的实例，f 的时间为线性时间 $f(n)=O(n)$，解的反变换$s(n)$也是线性时间

解Q的算法：$T_Q(n)= f(n)+T_p(n)+s(n)$

1. 将Q的实例 I 变成 $f(I)$， $f(n)$
2. 用解 P 的算法作为子程序解 $f(I)$，时间与解P的时间为同样的阶 $T_p(n)$
3. 将解变换成原问题的解 $s(n)$

解P的算法可以解Q. 且时间的阶一样，因此 P至少与Q一样难. $Q \le^l P$

$$f(n)+T_p(n)+s(n)= T_Q(n) =\Omega(g(n))$$

因子分解与素数测试

- 问题：因子分解问题 factor， Factor为任意求解算法，输出是全部素因子(含重复)，规定Factor(1)={1}.
  素数测试问题 testp
- 假设testp问题的难度是$W(n)$
- 素数测试算法 $A(n)$

  1. if $n=1$ then return “No”
  2. else $p \leftarrow \text{Factor}(n)$
  2. if $|p| \geq 2$ then return “No”
  3. else return “Yes”

- 结论： $\Omega(W(n))=T_{\text{testp}}(n) \leq T_{\text{factor}}(n)$

最邻近点对与唯一性问题

- **P问题与Q问题：**
  P: 平面直角坐标系中n个点的最邻近点对问题**Close**
  Q: 元素的唯一性问题 **Uniqueness**：给定 n 个数的集合S，判断 S 中的元素是否存在相同元素. 时间 $\Omega(n\log n)$.
- **变换** f：
  Q的实例：$x_1, x_2, \ldots, x_n$，变成点$(x_1,0),(x_2,0),\ldots,(x_n,0)$
- **解Q算法：**
  1. 利用求最邻近点对算法 **P** 计算最短距离 d.
  2. **if** d=0 **then return "No"**
  3. **else return "Yes"**
- 结论：计算平面直角坐标系中n个点的最邻近点对问题的时间是 $\Omega(n\log n)$，其中算法以比较为基本运算

最小生成树与唯一性问题

- **P问题与Q问题：**

  P: 平面直角坐标系中n个点的最小生成树问题；

  Q: 元素的唯一性问题**Uniqueness** $\Omega(n\log n)$

- 变换 f：

  Q的实例：$x_1, x_2, \ldots, x_n$，变成**X** 轴上的n个点,

- 解**Q**算法：

  1. 利用求最小生成树算法**P**构造树T，确定T的最短边e.
  2. 检测e的长度是否为**0**
  3. **if** $|e|=0$ **then** 不唯一，**else** 是唯一的.

- 结论：计算平面直角坐标系 n点最小生成树时间是$\Omega(n\log n)$，其中算法以比较为基本运算