算法分析与问题的计算复杂度

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$W'(n) = \Theta(F'(n))$.

8.1 平凡下界

算法的输入规模和输出规模是它的平凡下界.

例8.1 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8.2 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8.3 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8.4 问题：货郎问题
求解的时间复杂度下界是 $\Omega(n^2)$

8.2 直接计数最少运算数

例8.5 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.

决策树(Decision Tree)

- 决策树是一棵二叉树，对于给定问题（以比较运算作为基本运算）规定一个决策树的构造规则. 求解这个问题的不同算法所构造的决策树结构不一样.
- 给定一个算法的决策树. 对于任何输入实例，算法将从树根开始，沿一条路径向下，在每个结点做一次基本操作(比较). 然后根据比较结果(<, =, >)走到某个儿子结点或者在该处停机. 对于给定实例的计算恰好对应了一条从树根到树叶或者某个内部结点的路径.
- 给定一个算法，对于不同的输入，算法将在对应决策树的某个结点(树叶或者内部结点)停机. 将该结点标记为输入. 问题的输入规模对应于决策树的结点总数或者叶结点数.

决策树与问题复杂度

决策树的特点：

- 以比较作基本运算的算法模型
- 一个问题确定了一类决策树，具有相同的构造规则，该决策树类决定了求解该问题的一个算法类
- 结点数(或树叶数)等于输入规模
- 最坏情况下的时间复杂度对应于决策树的深度
- 平均情况下的时间复杂度对应于决策树的平均路径长度

用决策树模型界定确定问题难度

- 给定结点数(或树叶数)的决策树的深度至少是多少？
- 给定结点数(或树叶数)的决策树的平均路径长度至少是多少？

8.3 决策树（二叉树的性质）

命题8.1 在二叉树的 t 层至多 2^t 个结点（根为 0 层）

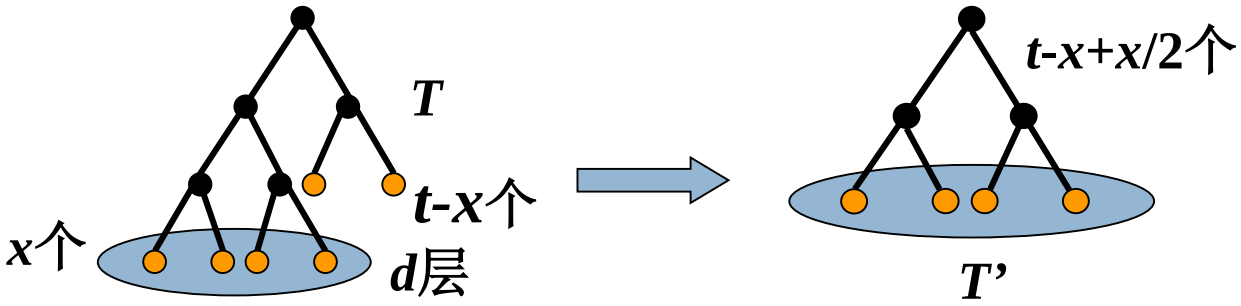
命题8.2 深度为 d 的二叉树至多 $2^{d+1}-1$ 个结点.

命题8.3 n 个结点的二叉树的深度至少为 $\lfloor \log n \rfloor$.

命题8.4 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'.　$t' = (t-x) + x/2 = t - x/2$，　$x \le 2^d$

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



8.4 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8.1 $W(n) = \lfloor \log n \rfloor + 1 \qquad n \geq 1$

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

$$W(n) = 1 + W\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

$$= 1 + \left\lfloor \log \left\lfloor \frac{n}{2} \right\rfloor \right\rfloor + 1$$

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

$$= \lfloor \log n \rfloor + 1$$

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

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

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

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

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

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

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

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

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

有没有更好的算法？

$$\sum_{t=1}^{k}t2^{t-1}=\sum_{t=1}^{k}t(2^t-2^{t-1})=\sum_{t=1}^{k}t2^t-\sum_{t=1}^{k}t2^{t-1}=\sum_{t=1}^{k}t2^t-\sum_{t=0}^{k-1}(t+1)2^t$$

$$=\sum_{t=1}^{k}t2^t-\sum_{t=0}^{k-1}t2^t-\sum_{t=0}^{k-1}2^t=k2^k-(2^k-1)=(k-1)2^k+1$$

捡索问题的决策树

设A是一个捡索算法, 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, ... , n, 且标记规则是:

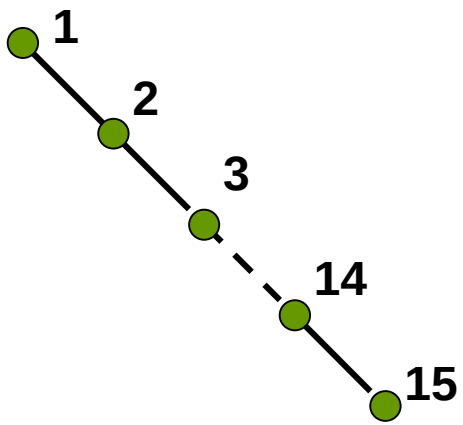
(1) 根据算法A，首先与 x 比较的 L 的项的下标标记为树根.

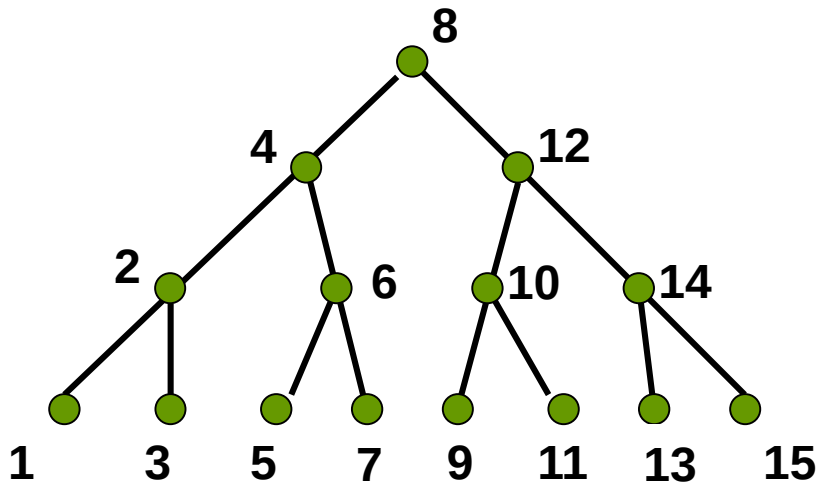
(2) 假设某结点被标记为 i,

- 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8.2 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.

8.5 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
- 排序算法的时间复杂度下界

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

原始输入	5	3	2	6	9	1	4	8	7
第一次巡回后	3	2	5	6	1	4	8	7	9
第二次巡回后	2	3	5	1	4	6	7	8	9
第三次巡回后	2	3	1	4	5	6	7	8	9
第四次巡回后	2	1	3	4	5	6	7	8	9
第五次巡回后	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 $L=\{1, 2, \dots, n\}$，排序的任何输入为 L上的置换. 在置换 $a_1\ a_2 \dots a_n$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, $i = 1, 2, \dots, n$. ($b_1, b_2, \dots, b_n$) 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

- $b_1 = 0$; $b_2 = 0,1$; … ; $b_n = 0,1, \ldots, n-1$
- 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：$W(n)=O(n^2)$，至多巡回$O(n)$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 $(b_1, b_2, \ldots, b_n)$, 置换α' 的逆序序列为 $(0-b_1, 1-b_2, \ldots, n-1-b_n)$，$\alpha$ 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时间复杂性均为$\Theta(n^2)$

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

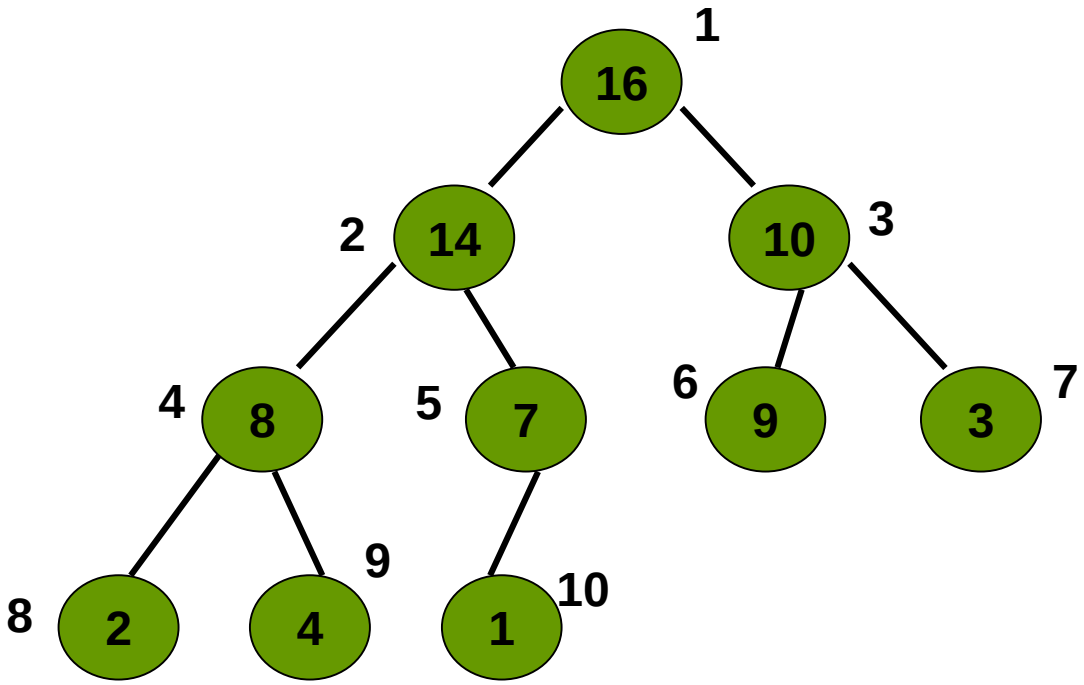
(3) d–1 层的所有树叶在内结点的右边

(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：

$left(i) = 2i$ ， $right(i) = 2i + 1$

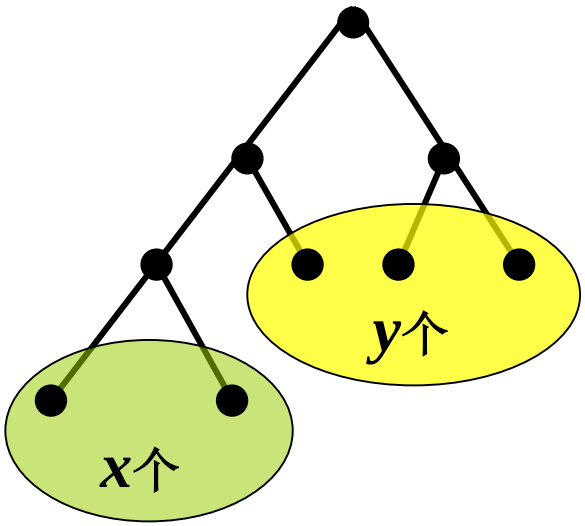
2. 引理8.1 n 个结点的堆恰好$\lceil n/2 \rceil$片树叶.

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8.1证明（续）

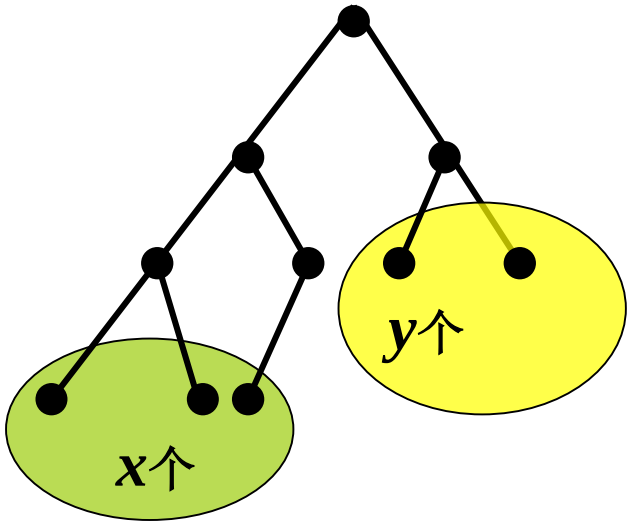
Case2 x为奇数 (x为奇数，n为偶数)

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

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

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

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



堆的运算：整理

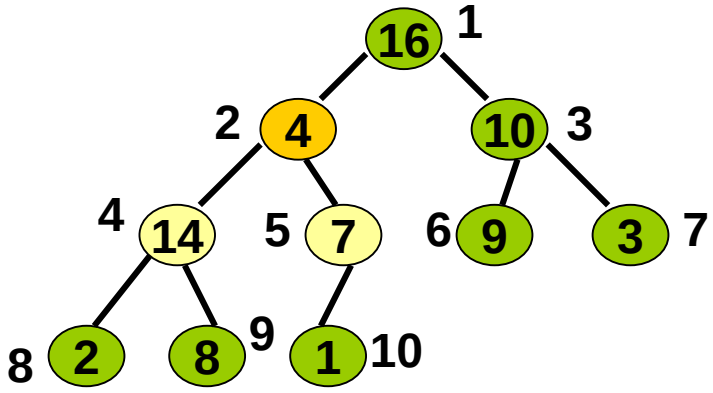
算法8.2 Heapify(*A*,*i*)

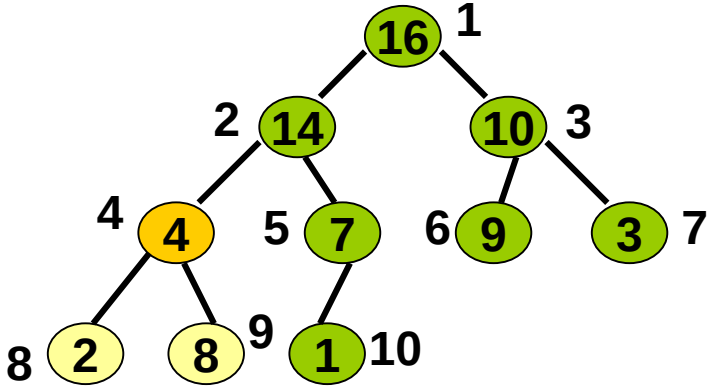
输入：堆结构 *A*, *A*的结点 *i*

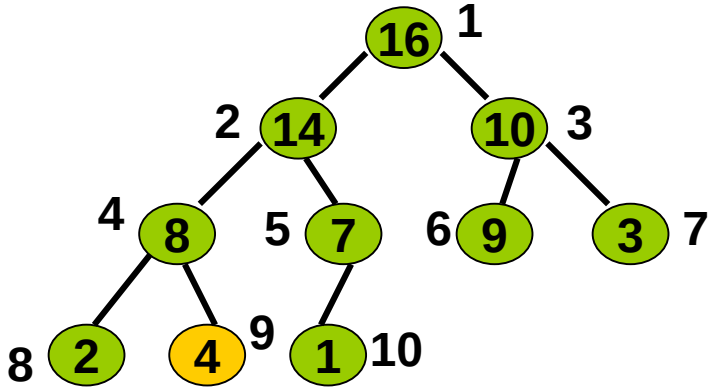
输出：从 *i* 向下满足堆存储要求的堆结构

1. $l \leftarrow left(i)$
2. $r \leftarrow right(i)$
3. if $l \le heap\text{-}size[A]$ and $A[l] > A[i]$
4. then $largest \leftarrow l$
5. else $largest \leftarrow i$
6. if $r \le heap\text{-}size[A]$ and $A[r] > A[largest]$
7. then $largest \leftarrow r$
8. if $largest \ne i$
9. then exchange $A[i] \leftrightarrow A[largest]$
10. Heapify(A, $largest$)

Heapify 实例







Heapify(*A*,2)

复杂度分析

每次调用为$O(1)$

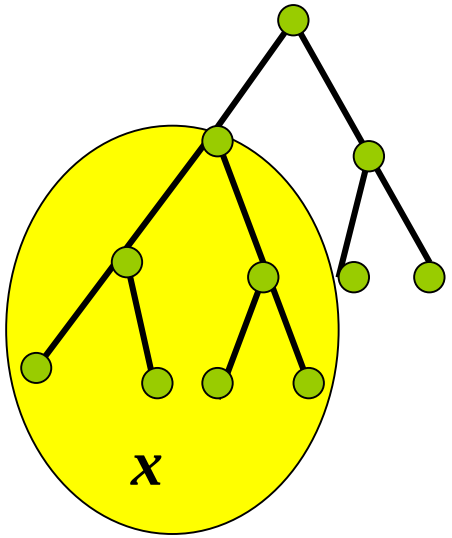
子堆大小至多为原来的 2/3

递推不等式

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

解得 $T(n) = \Theta(\log n)$

或者 $T(h) = \Theta(h)$



结点总数

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

说明：

h为 结点 i 的高度(距树叶最大距离)，n为以 i 为根的子树的结点数.

堆的运算：建堆

算法8.3 Build-Heap (*A*)

输入：数组 *A*

输出：堆 *A*

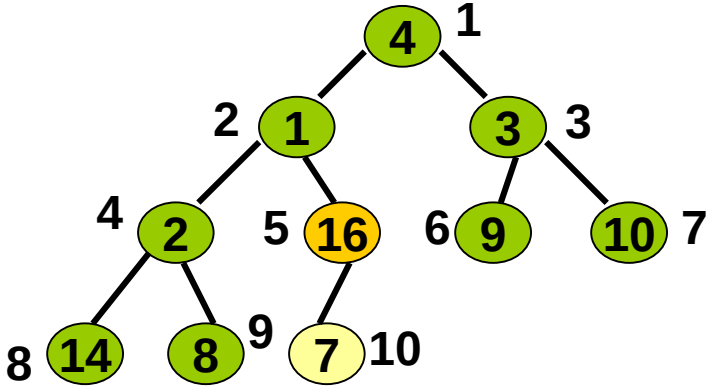
1．*heap-size*[*A*] ← *length*[*A*]

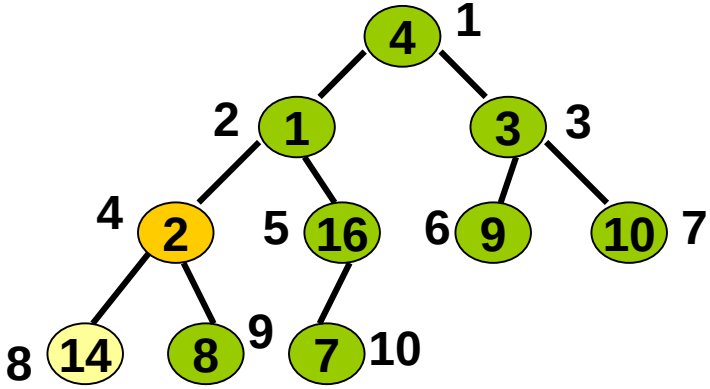
2．for *i* ←⌊*length*[*A*]/2⌋ downto 1

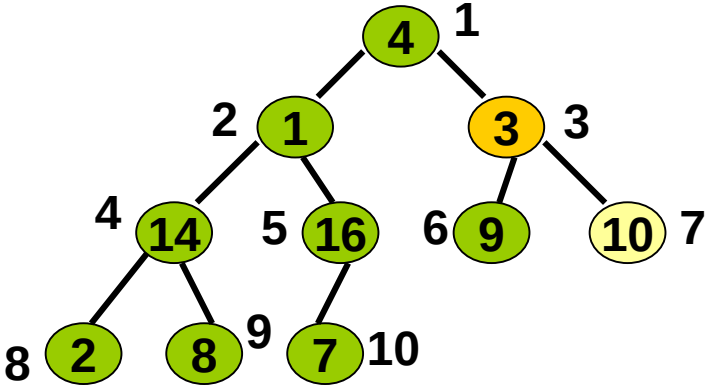
3． do Heapify (*A*, *i*)

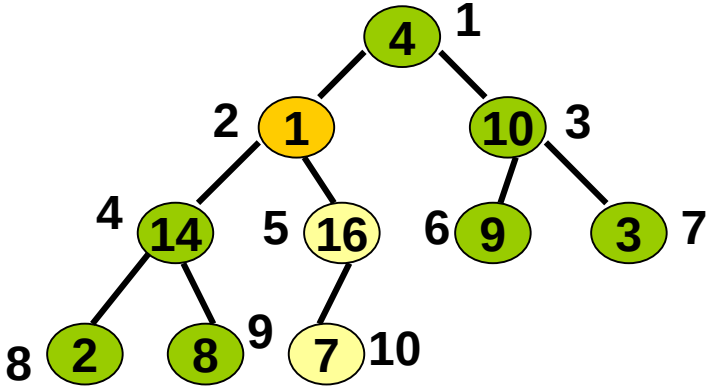
说明：堆有*n*个结点，由引理 8.1，树叶有⌈*n*/2⌉片，最大内结点标号为 *n*–⌈*n*/2⌉=⌊*n*/2⌋

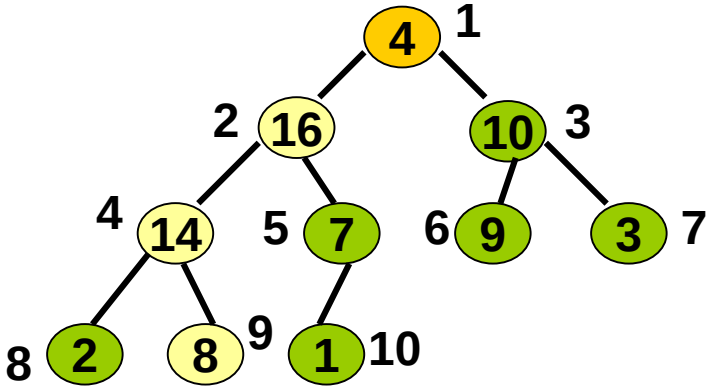
实例

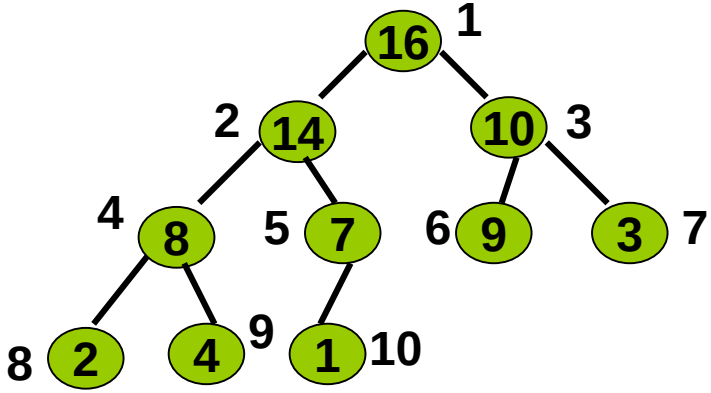

4 1
2 1 3 3
4 2 5 16 6 9 10 7
8 14 8 9 7 10


4 1
2 1 3 3
4 2 5 16 6 9 10 7
8 14 8 9 7 10


4 1
2 1 3 3
4 14 5 16 6 9 10 7
8 2 8 9 7 10


4 1
2 1 10 3
4 14 5 16 6 9 3 7
8 2 8 9 7 10


4 1
2 16 10 3
4 14 5 7 6 9 3 7
8 2 8 9 1 10


16 1
2 14 10 3
4 8 5 7 6 9 3 7
8 2 4 9 1 10

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8.2

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 8.1

归纳步骤；假设对 $h-1$为真，证明对 h 也为真.

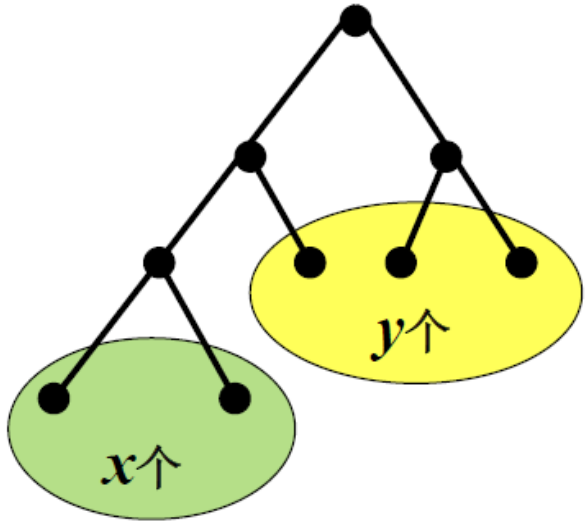
归纳基础

h=0, 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

(x为偶数，d-1 层以前各层结点总数 2^d–1)

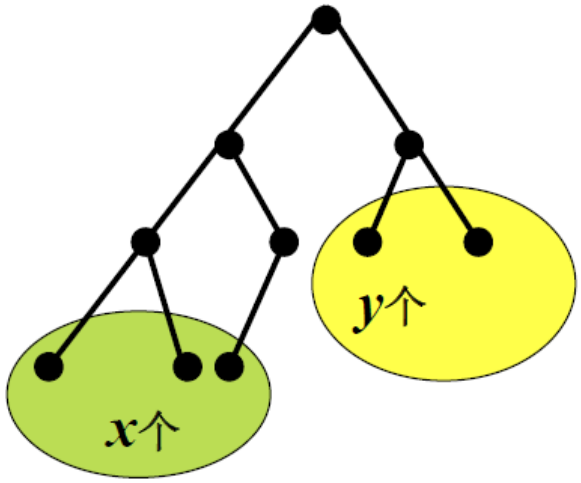
归纳基础（续）

Case2 x为奇数 (x为奇数，n为偶数)

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

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

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



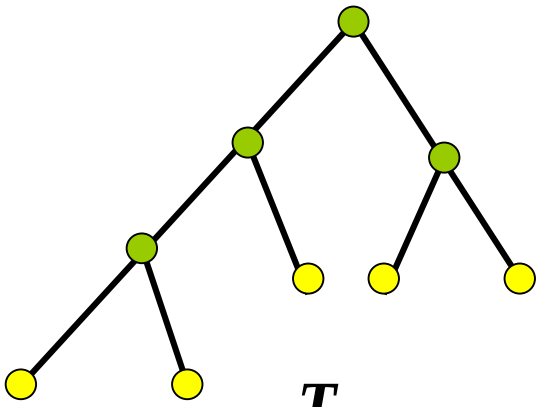
归纳步骤

假设命题对于高度 h-1为真，证明对于高度为 h 也为真.

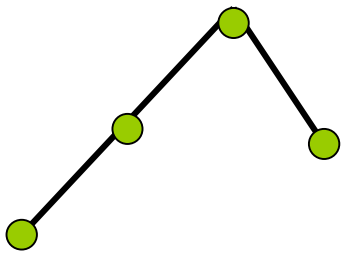
设 T 表示 n 个结点的树，从 T 中移走所有的树叶得到树 T’，T 与T ’的关系：

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）



T



T’

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8.3 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

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

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

推导

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

令$2^{k-1}<n\le 2^k$，于是$k-1\le \lfloor \log n \rfloor \le k$. 若$n=2^k$，则

$$T(n)=\sum_{h=0}^{k}\left\lceil \frac{2^k}{2^{h+1}} \right\rceil ch=\sum_{h=0}^{k-1}2^{k-h-1}ch+\left\lceil \frac{1}{2} \right\rceil ch$$

$$=2^k\sum_{h=0}^{k-1}\frac{ch}{2^{h+1}}+ch=O(n)\sum_{h=0}^{k-1}\frac{h}{2^{h+1}}+O(h)=O(n).$$

若$2^{k-1}<n<2^k$，则

$$T(n)<\sum_{h=0}^{k-1}\left\lceil \frac{2^k}{2^{h+1}} \right\rceil ch=\sum_{h=0}^{k-1}2^{k-h-1}ch=2^k\sum_{h=0}^{k-1}\frac{ch}{2^{h+1}}=O(n)$$

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8.4 Heap-sort(*A*)

输入：数组 *A*

输出：排好序的数组 *A*

1. Build-Heap(*A*)

2. for *i* ← *length*[*A*] downto 2 do

3. *exchange* *A*[1]↔*A*[*i*]

4. *heap-size*[*A*] ←*heap-size*[*A*]–1

5. Heapify(*A*,1)

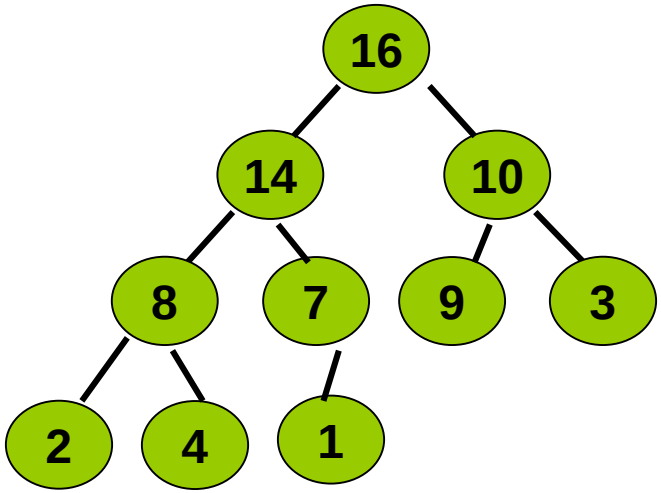
复杂性：*O*(*n*log*n*)

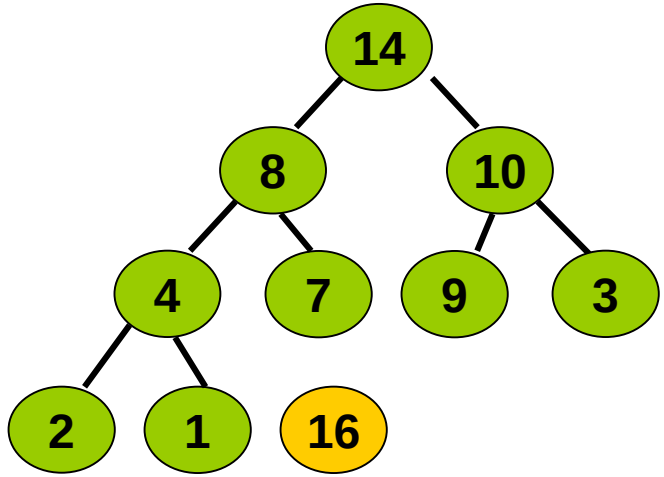
Build-Heap 时间为 *O*(*n*),

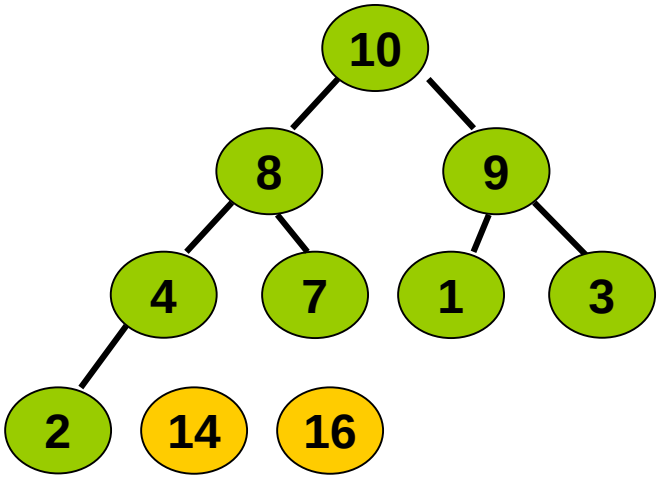
从行2-5 调用 Heapify *n*-1次，每次*O*(log*n*)，

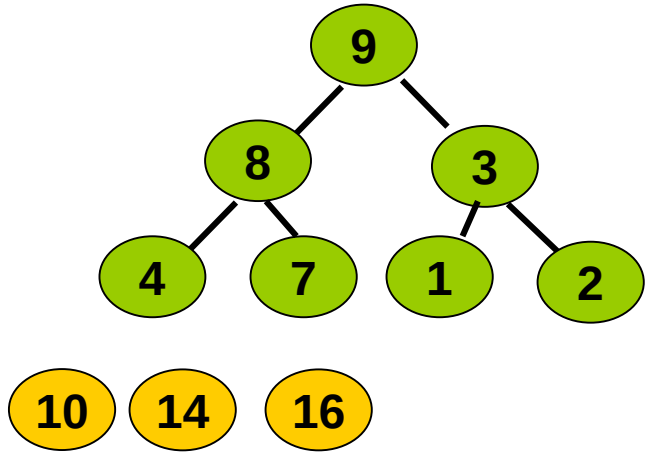
时间为*O*(*n*log*n*)

实　例

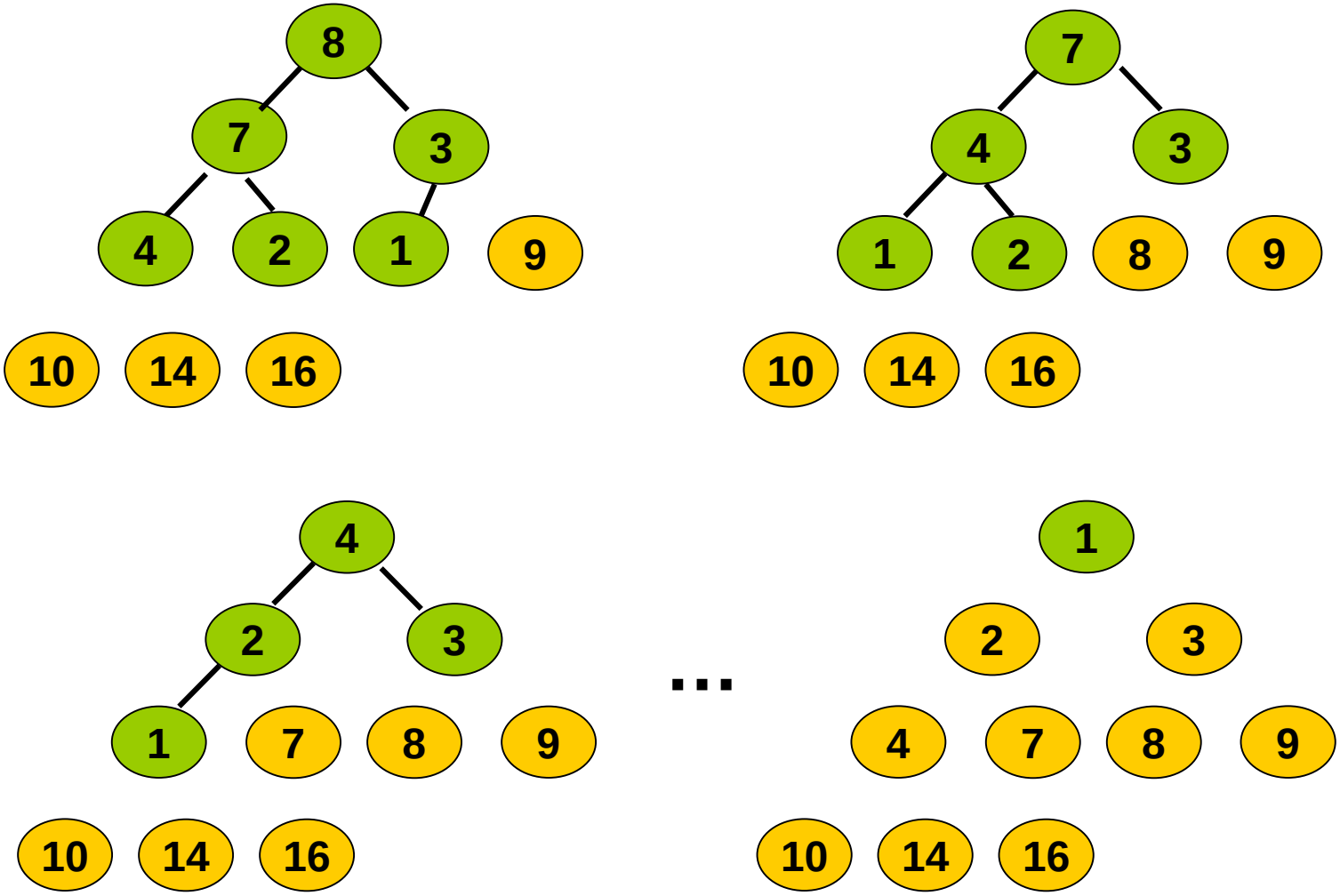

16
14
10
8
7
9
3
2
4
1


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 的左儿子标记为输出；

若下一步比较元素 x_p, x_q，那么 k 的左儿子标记为 (p,q)

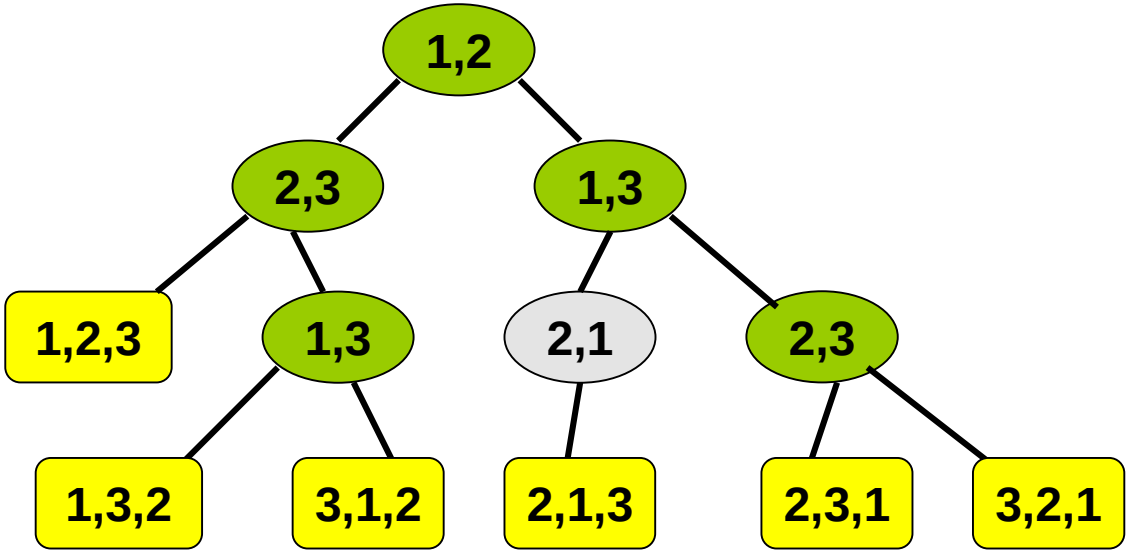
(2) $x_i > x_j$

若算法结束，k 的右儿子标记为输出；

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 $t \le 2^d$.

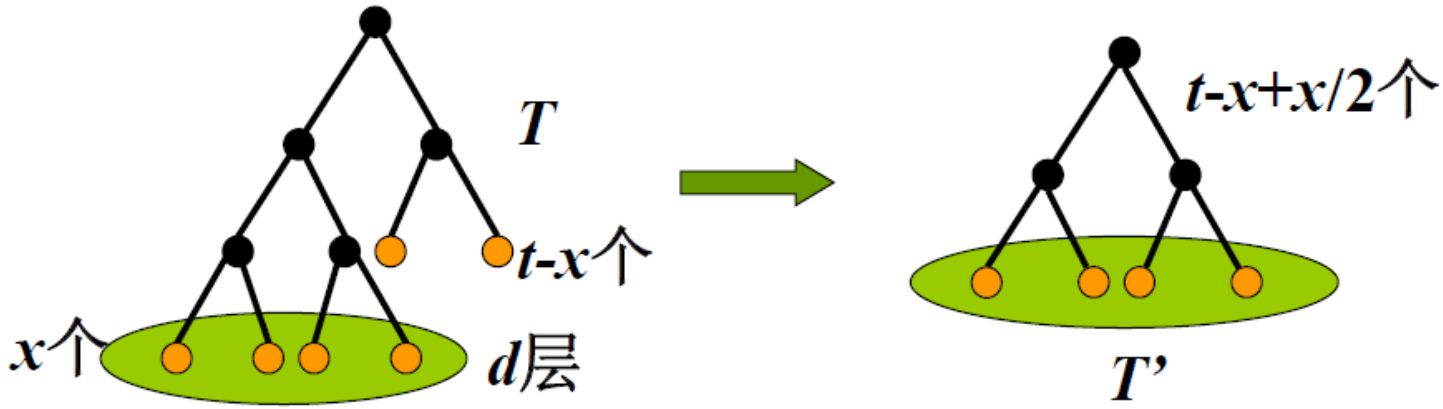
证明 归纳法.

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

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

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



最坏情况复杂度的下界

引理8.3 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8.4 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

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

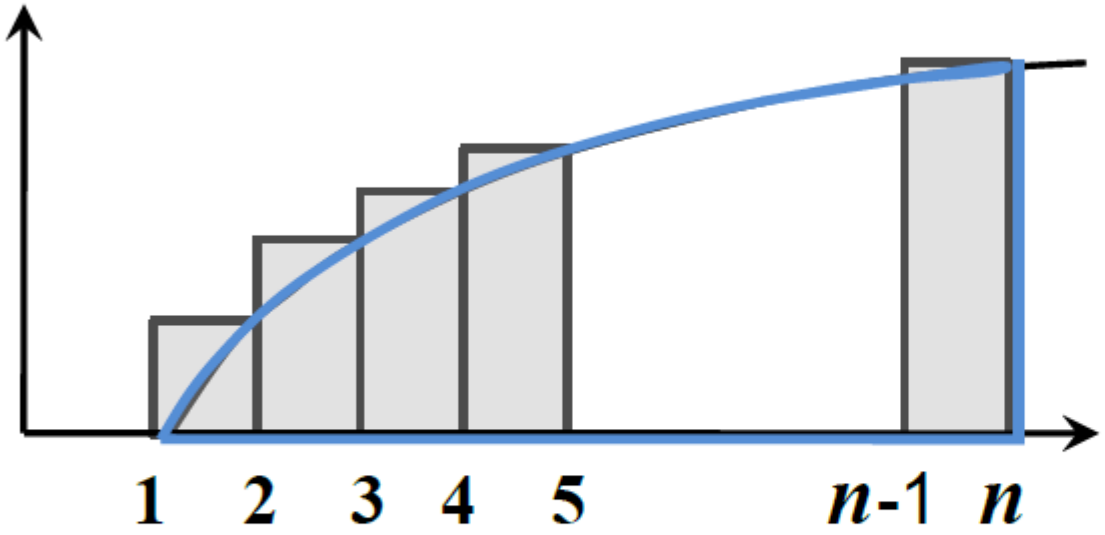
$$= \log e(n \ln n - n + 1)$$

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

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

结论：堆排序算法在最坏情况阶达到最优.

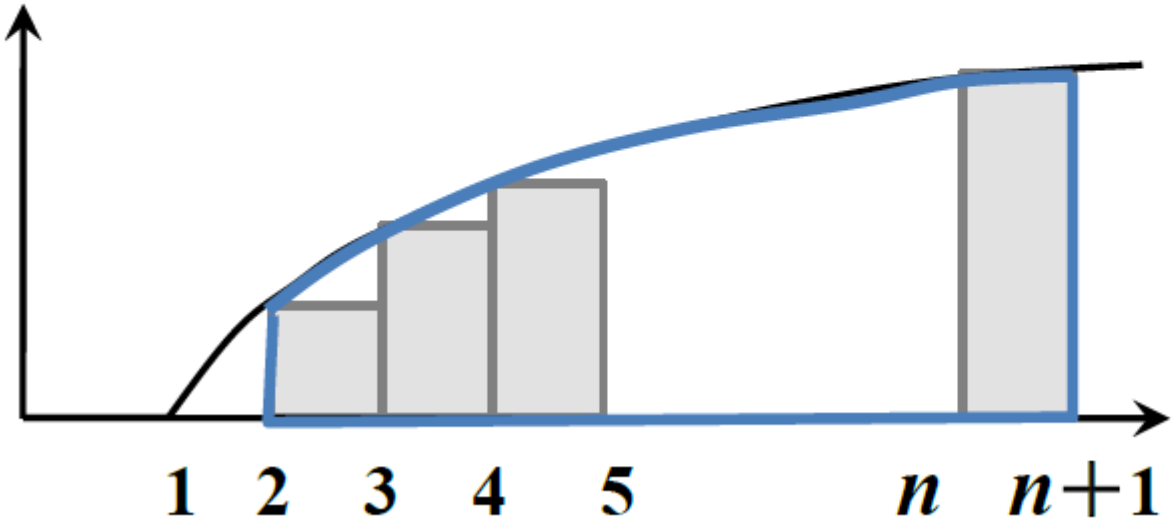
$\log(n!) = \Omega(n\log n)$的证明



$$\log(n!) = \sum_{k=1}^{n} \log k \geq \int_{1}^{n} \log\, x dx$$

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

$\log(n!) = O(n\log n)$的证明



$$\log(n!) = \sum_{k=1}^{n} \log k \le \int_{2}^{n+1} \log x dx = O(n \log n)$$

平均情况分析

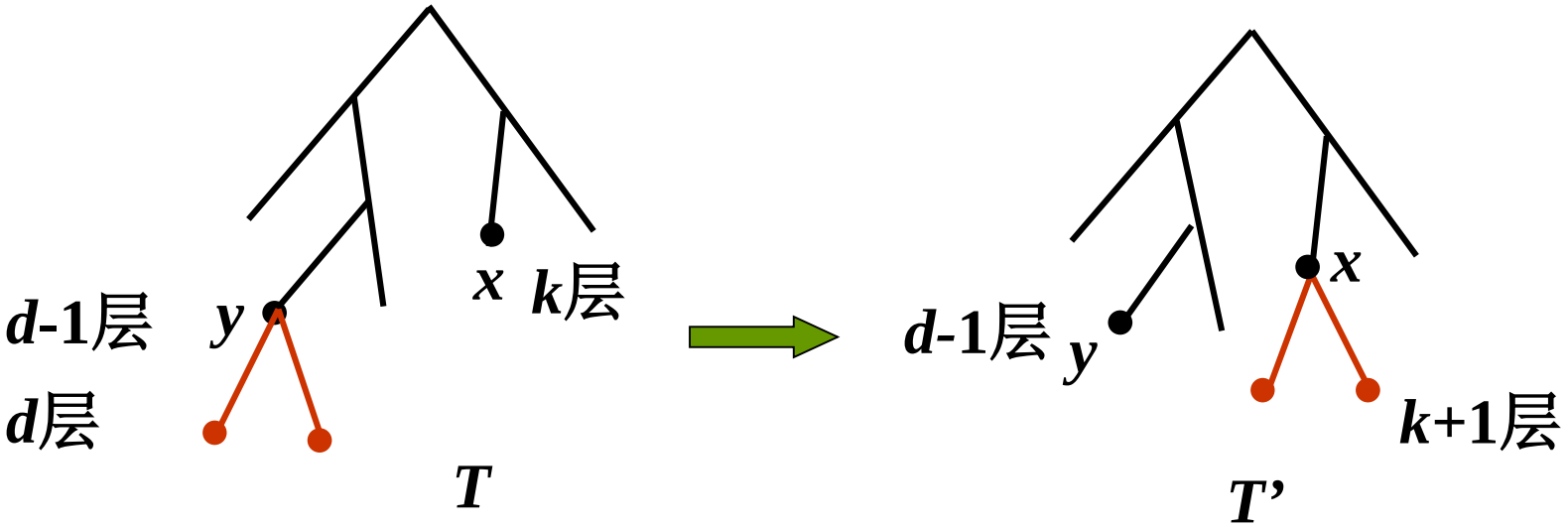
epl(*T*)：假设所有的输入等概分布，令 **epl(*T*)** 表示 **B** 树中从根到树叶的所有路径长度之和, **epl(*T*)/*n*!** 的最小值对应平均情况时间复杂性的下界.

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8.4 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



$$epl(T) - epl(T') = (2d+k) - [(d-1)+2(k+1)]$$

$$= 2d+k-d+1-2k-2 = d-k-1 > 0 \qquad (d > k+1)$$

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

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：由命题8.4 树 T 的深度 $d \geq \lceil \log t \rceil$,

由引理8.4 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）

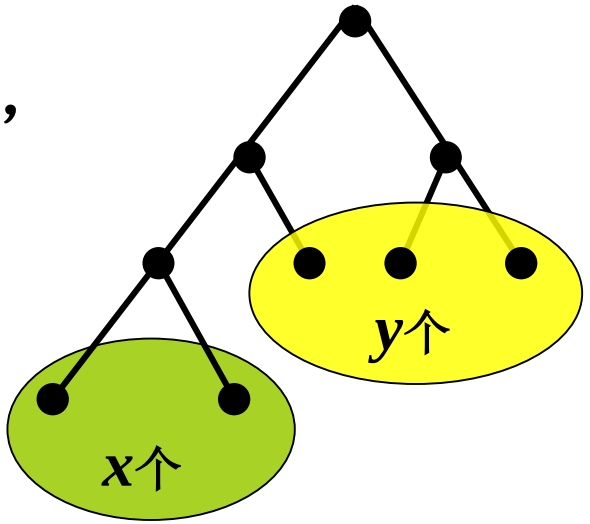
Case2 $t \neq 2^k$.

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 $x = 2t - 2^d$, $y = 2^d - t$.



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

平均时间复杂度的下界

定理8.5 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epl值

$$A(n) \geq \frac{1}{n!} epl(T)$$

$$= \frac{1}{n!}(n!\lfloor \log n! \rfloor + 2(n! - 2^{\lfloor \log n! \rfloor}))$$

$$= \lfloor \log n! \rfloor + \varepsilon, \qquad 0 \leq \varepsilon < 1$$

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

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

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	时间最优性
冒泡排序	$O(n^2)$	$O(n^2)$	原地	
快速排序	$O(n^2)$	$O(n\log n)$	$O(\log n)$	平均最优
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n)$	最优
堆排序	$O(n\log n)$	$O(n\log n)$	原地	最优

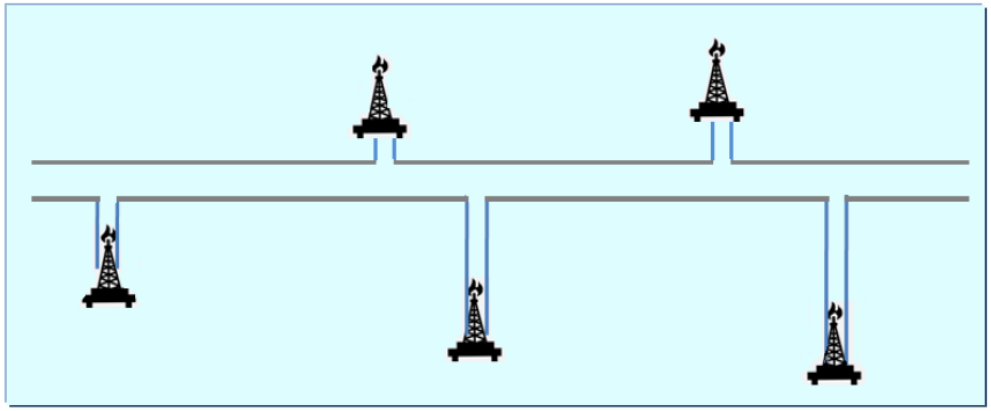
8.6 选择算法的时间复杂度分析

	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	$O(n)\sim 2.95n$	$O(\log n)$

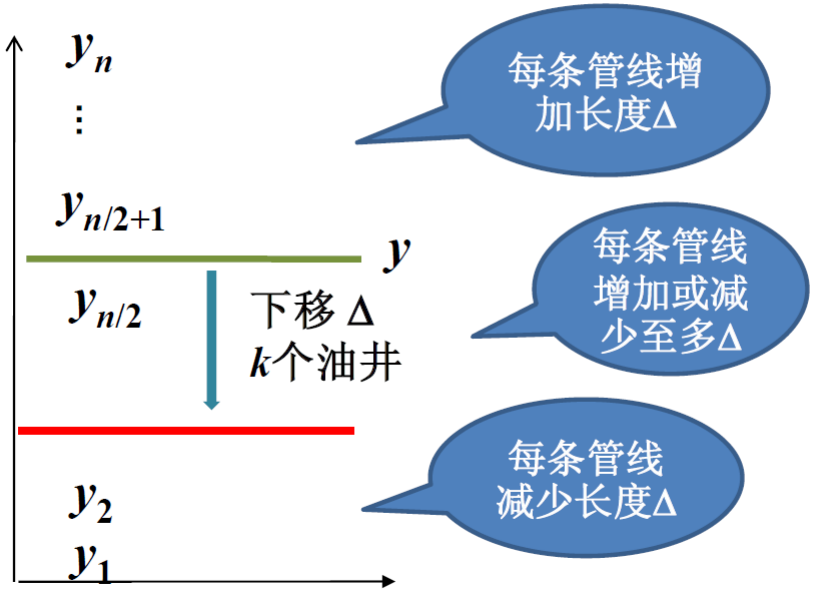
选最大算法 Findmax是最优的算法

一个应用：管道位置

问题：某区域有n口油井，需要修建输油管道. 根据设计要求，水平方向有一条主管道，每口油井修一条垂直方向的支管道通向主管道. 如何选择主管道的位置，以使得支管道长度的总和最小？



最优解：Y坐标的中位数



下移后支管线总长度增加

简单的算法

算法一：

调用 k 次选最小算法

时间复杂度为 $O(kn)$

算法二：

先排序，然后输出第 k 小的数

时间复杂度为 $O(n \log n)$

分治算法

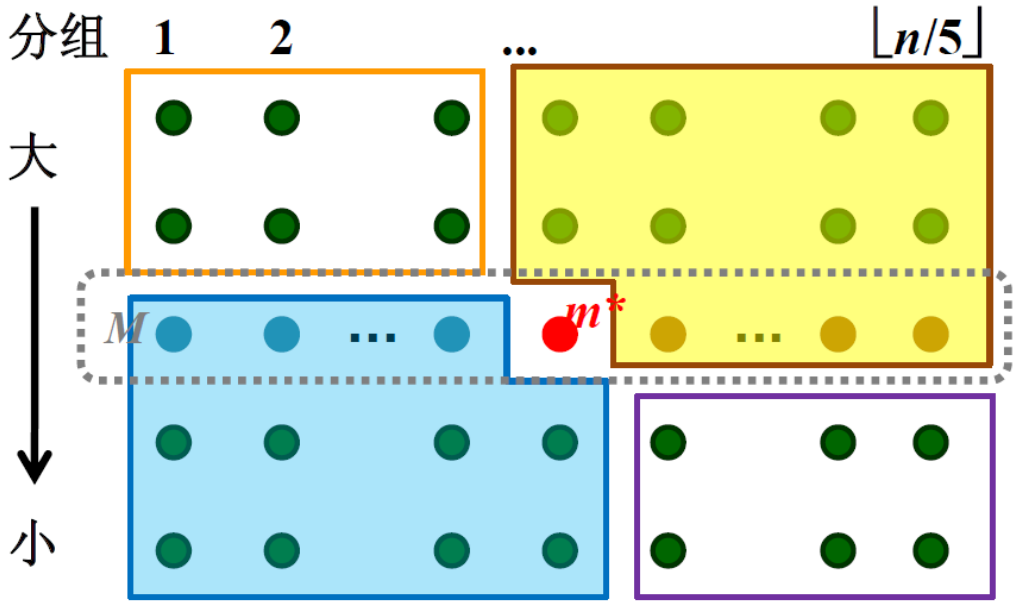
假设元素彼此不等，设计思想：

1. 用某个元素 m^*作为标准将 S 划分成 S_1 与 S_2，其中 S_1的元素小于 m^*，S_2 的元素大于等于 m^*.

2. 如果 $k \le |S_1|$，则在 S_1中找第 k 小.
   如果 $k = |S_1|+1$，则 m^*是第 k 小
   如果 $k > |S_1|+1$，则在 S_2中找第 $k-|S_1|-1$小

算法效率取决于子问题规模，
如何通过m^*控制子问题规模？

m^*的选择与划分过程



A: 数需要与m^*比大小，*B*：数大于m^*

C: 数小于m^*，　*D*:数需要与m^*比大小

(1) 寻找 m^* 的时间代价不能太高，如果时间已经达到 $O(n\log n)$，那就不如直接使用排序算法了. 因此，如果直接寻找 m^*，时间应该是 $O(n)$. 设选择算法的时间复杂度为 $T(n)$，递归调用这个算法在 S 的一个真子集 M 上寻找 m^*，应该使用 $T(cn)$时间，这里的 c 是一个小于 1 的常数，它反映了 M 的规模与 S 相比缩小了多少.

(2) 通过 m^* 划分的两个子问题的大小分别记作 $|S_1|$ 和 $|S_2|$，考虑算法在最坏情况下的性能，不妨假设每次递归调用时算法都进入规模较大的一个，即子问题规模为 $\max\{|S_1|, |S_2|\}$. 每次递归调用时，子问题规模与原问题规模 n 的比都不超过一个常数 d，那么 $d<1$，调用时间为 $T(dn)$. 特别地，在采用递归算法寻找 m^* 时，还应该保证 $c+d<1$. 否则方程

$$T(n) = T(cn) + T(dn) + O(n)$$

的解不会达到 $O(n)$.

实例：n=15, k=6

8	2	3	5	7	6	11	14	1	9	13	10	4	12	15

M

8	14	15
7	11	13
5	9	12
3	6	10
2	1	4

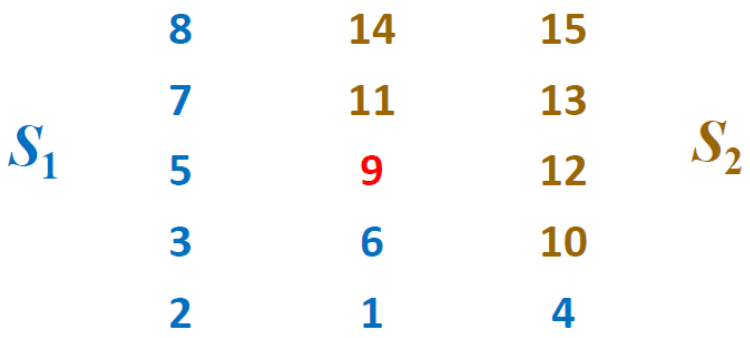
$m^* = 9$

A		B
8	14	15
7	11	13
5	9	12
3	6	10
2	1	4

C D

8, 7, 10, 4 需要与9比较

归约为子问题



子问题

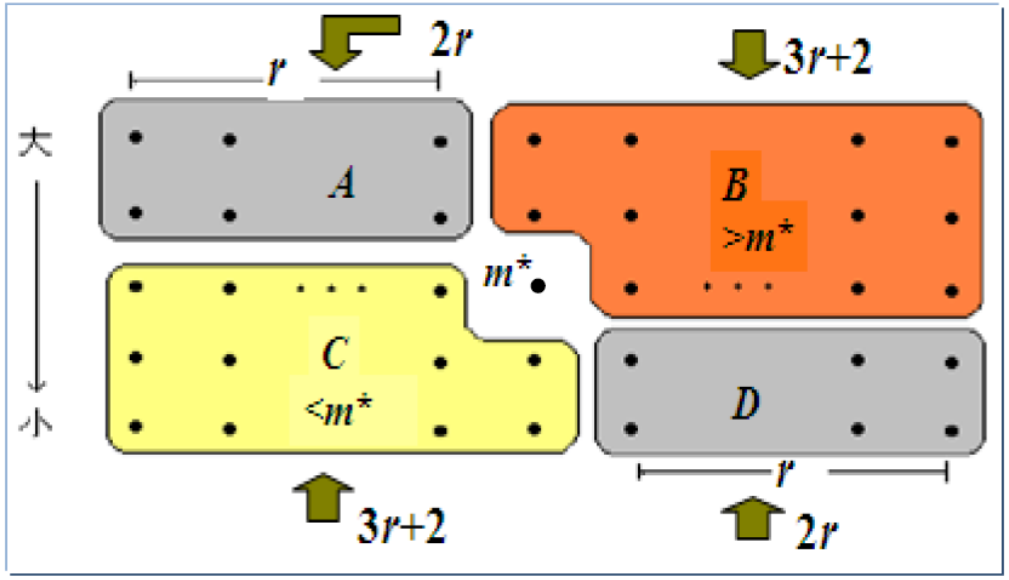
8 7 5 3 2 6 1 4

子问题规模 = 8，k=6

伪码

算法 Select (S, k)
输入：数组 S，正整数 k，
输出：S 中的第 k 小元素
1. 将S分5个一组, 共 $n_M=\lceil n/5\rceil$组
2. 每组排序,中位数放到集合 M
3. $m^*\leftarrow$Select(M,$\lceil |M|/2 \rceil$) //S分A,B,C,D
4. A,D元素小于m^*放 S_1,大于m^*放S_2
5. $S_1 \leftarrow S_1 \cup C$; $S_2 \leftarrow S_2 \cup B$ 划分
6. if $k = |S_1| + 1$ then 输出 m^*
7. else if $k \le |S_1|$ ⇐ 递归计算子问题
8. then Select (S_1, k)
9. else Select (S_2, $k - |S_1| - 1$)

用m^*划分



$n = 5\ (2r+1)$，$|A| = |D| = 2r$

子问题规模至多：$2r+2r+3r+2 = 7r+2$

子问题规模估计

不妨设 $n = 5(2r+1)$，$|A|=|D|=2r$,

$$r = \frac{n/5-1}{2} = \frac{n}{10} - \frac{1}{2}$$

划分后子问题规模至多为

$$7r+2 = 7\left(\frac{n}{10} - \frac{1}{2}\right) + 2$$

$$= \frac{7n}{10} - \frac{3}{2} < \frac{7n}{10}$$

时间复杂度递推方程

算法工作量 $W(n)$

行2: $O(n)$ //每5个数找中位数,构成M

行3: $W(n/5)$ // M 中找中位数 m^*

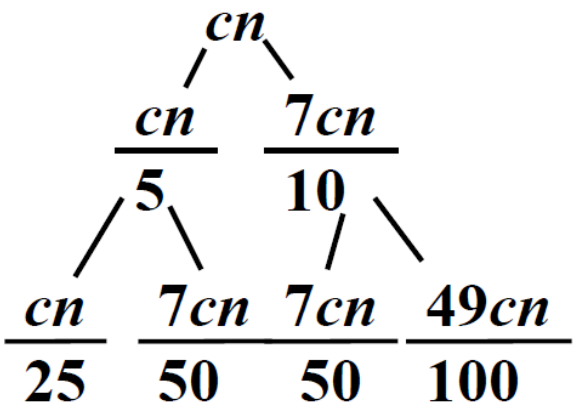
行4: $O(n)$ // 用m^*划分集合 S

行8-9： $W(7n/10)$ //递归

$$W(n) \le W(n/5) + W(7n/10) + O(n)$$

递归树

$W(n)=W(n/5)+W(7n/10)+cn$



cn

$0.9cn$

$0.81cn$

……

$W(n) \le cn\ (1+0.9+0.9^2+...)=O(n)$

讨论

分组时为什么5个元素一组？
3个一组或 7个一组行不行？

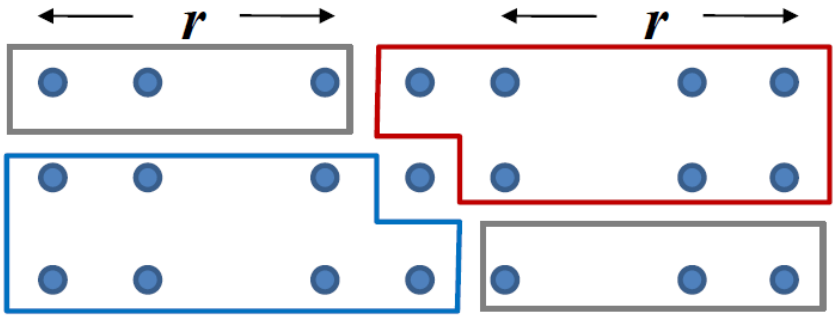
分析：递归调用

1. 求 m^*的工作量与 $|M| = n/t$ 相关，t 为每组元素数. t大, $|M|$小

2. 归约后子问题大小与分组元素数t 有关 . t 大，子问题规模大

3分组时的子问题规模

假设 $t=3$，3个一组：



$n = 3(2r+1)$

$r = (n/3-1)/2 = n/6-1/2$

子问题规模最多为 $4r+1 = 4n/6-1$

算法的时间复杂度

算法的时间复杂度满足方程

$$W(n) = W(n/3) + W(4n/6) + cn$$

由递归树得 $W(n)=\Theta(n\log n)$

关键：

$|M|$与归约后子问题规模之和小于 n，递归树每行的工作量构成公比小于 1 的等比级数， 算法复杂度才是$O(n)$.

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

8.6.1 选最大与最小算法

定理8.6 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	20			W	25						
$x_2>x_4$			WL	10			L	8				
$x_5>x_6$									WL	5	L	3
$x_6>x_4$							L	2			WL	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

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

当 n 为奇数，A做的比较次数至少为

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

结论：FindMaxMin是最优算法

8.6.2 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

   $w(x) \leftarrow w(x)+w(y), w(y) \leftarrow 0$

   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；

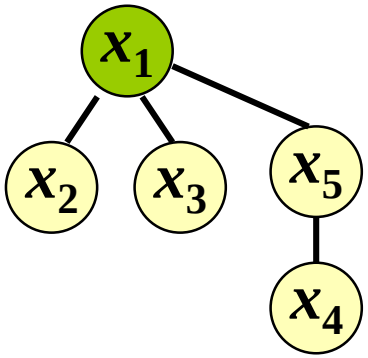
初始

$x_1>x_2$

$x_1>x_3$

$x_5>x_4$

$x_1>x_5$



实例

找第二大问题复杂度下界

引理8.5 任何找第二大算法，针对上述赋值规则产生的输入，通过与最大元素的比较直接淘汰的元素数至少是$\lceil \log n \rceil$.

证 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

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

定理8.7 任何通过比较找第二大的算法至少需要 $n+\lceil \log n \rceil -2$次比较.

证 确定最大需要 $n-1$比较，为确定第二大还要淘汰$K-1$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.

8.6.3 找中位数问题

定理8.8 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

*A*是求中位数的任意算法，如下构造 *A*的输入：

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. $x_5 > x_6$　　$x_5=6,\ x_6=3$

非决定性比较

5. $x_7 = 4$
6. $x_1 > x_3$
7. $x_3 > x_7$

决定性比较

8. …

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

8.7 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） $\Omega(f(n))$

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

算法8.5 解 Q 的算法：

1. 将Q的实例 I 变成 $g(I)$　　时间T_1
2. 用解 P 的算法作为子程序解 $g(I)$，时间与解 P 的时间为同样的阶　　时间 T_2
3. 将解变换成原问题的解　　时间T_3

$T_Q(n)= T_1+T_2+T_3$　T_1与T_3不超过T_2的阶

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.　$Q \le^l P$，$T_P(n) \ge T_Q(n) = \Omega(f(n))$

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 $f(n)$
- 算法8.6 素数测试算法 $A(n)$

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

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

元素唯一性问题

- 问题：给定 n 个数的集合S，判断 S 中的元素是否存在相同元素.
- 元素唯一性问题的复杂度为 $\Theta(n\log n)$

  输入：多重集 $S=\{n_1\cdot a_1, n_2\cdot a_2, \ldots, n_k\cdot a_k\}$

  构造决策树，树叶为 S 的全排列数

  最坏情况下树深为 $\dfrac{n!}{n_1!\,n_2!\ldots n_k!}$

$$\Theta(\log n!)=\Theta(n\log n)$$

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 Ω(*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 Ω(*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, 时间复杂度 Ω(*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 Ω(*n*log*n*)，其中算法以比较为基本运算.**