



排序算法时间复杂度分析

- 冒泡排序
- 堆排序
- 排序算法的复杂度下界



北京大學

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 $L=\{1,2,\dots,n\}$, 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
- 性质 $b_1=0$; $b_2=0,1$; \dots ; $b_n=0,1,\dots,n-1$
- 总共 $n!$ 个不同的逆序序列, 置换与逆序序列一一对应
- 逆序数: 置换中的逆序总数

$$b_1 + b_2 + \dots + b_n$$

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



北京大学



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





堆的定义

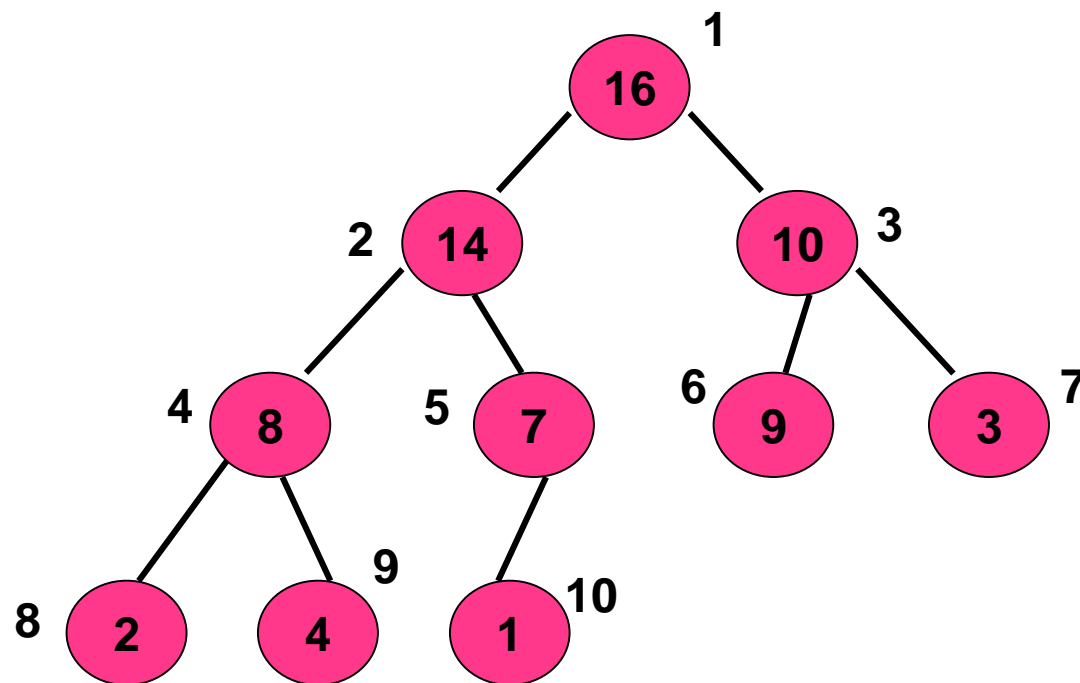
设 T 是一棵深度为 d 的二叉树，结点为 L 中的元素。
若满足以下条件，称作**堆**。

- (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**



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



北京大学



堆的运算：整理Heapify

算法 Heapify(A, i)

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



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

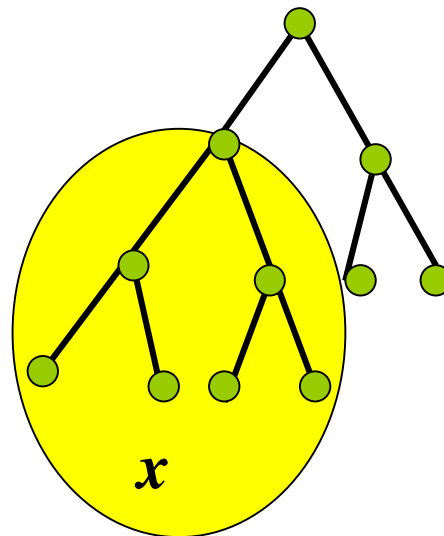
$$T(n) \leq T(2n/3) + \Theta(1)$$

解得 $T(n) = \Theta(\log n)$

或者 $T(h) = \Theta(h)$

h 为堆的根的高度

(距树叶最大距离)



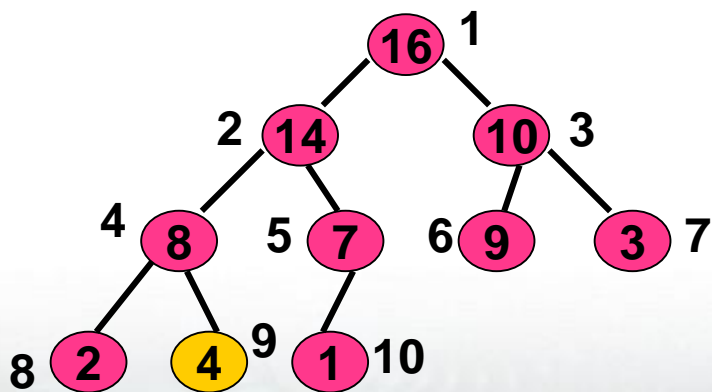
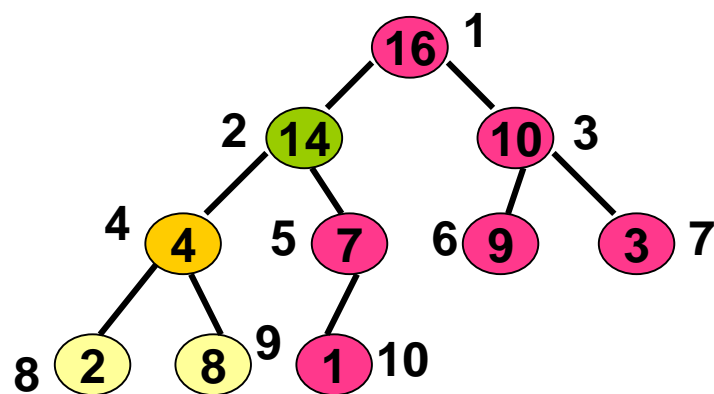
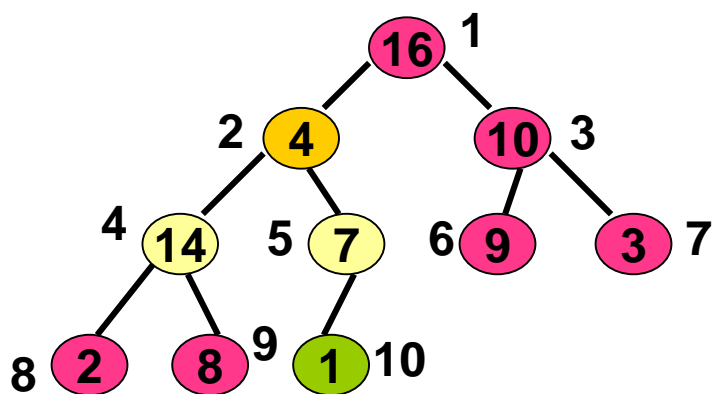
结点总数

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



北京大学

Heapify 实例



Heapify(A,2)



建堆时间复杂度分析

算法 Build-Heap(A)

1. $heap-size[A] \leftarrow length[A]$
2. for $i \leftarrow \lfloor length[A]/2 \rfloor$ downto 1
3. do Heapify(A, i)

- 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
- 思路：Heapify(i) 的复杂度依赖于 i 的高度，按照高度计数结点数，乘以 $O(h)$ ，再求和

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



计数高度为 h 的结点数

引理

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

$h=0$, 命题为真, 即证堆中树叶数为 $\left\lceil \frac{n}{2} \right\rceil$

归纳步骤

假设对 $h-1$ 为真, 证明对 h 也为真.

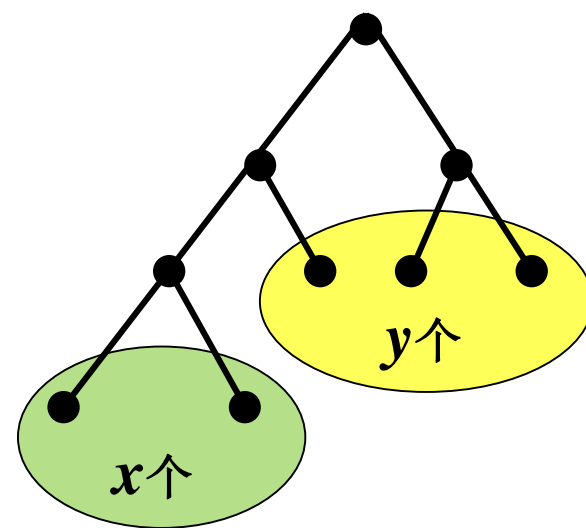


归纳基础

$h=0$, 树叶分布在 d 和 $d-1$ 层, d 层 (x 个), $d-1$ 层 (y 个).

Case1: x 为偶数

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



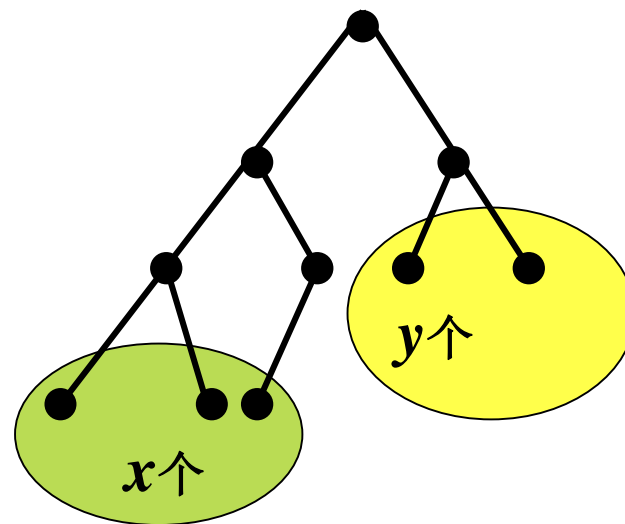
每个内结点有 2 个儿子, 树叶数为 x
(x 为偶数, $d-1$ 层以前各层结点总数 $2^d - 1$)



归纳基础（续）

Case2 x 为奇数 (x 为奇数, n 为偶数)

$$\begin{aligned}x + y &= x + 2^{d-1} - \frac{x+1}{2} \\&= 2^{d-1} + \frac{x-1}{2} \\&= \frac{2^d + x - 1}{2} = \frac{n}{2} = \left\lceil \frac{n}{2} \right\rceil\end{aligned}$$



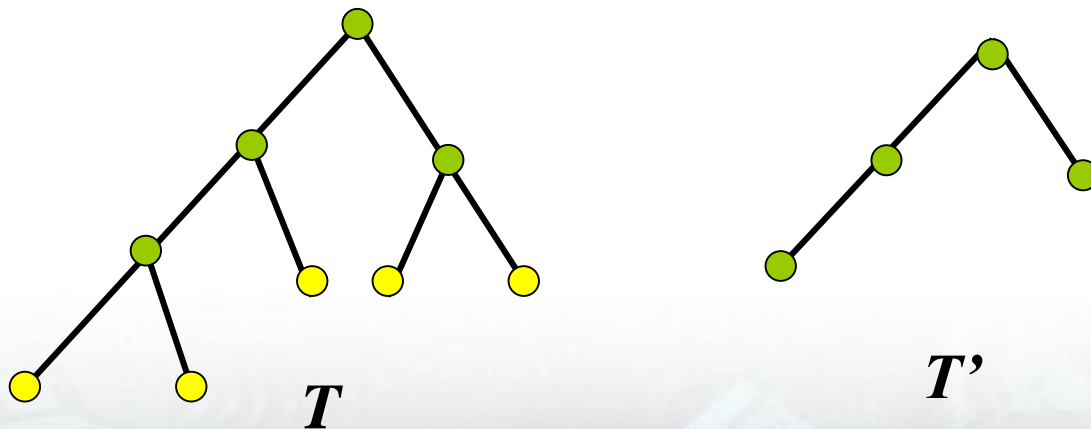
归纳步骤

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, $n_0 = \left\lceil \frac{n}{2} \right\rceil$

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

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

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



时间复杂度分析

定理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

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



推导

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

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

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

若 $2^{k-1} < n < 2^k$, 则

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



北京大學

求和

$$\begin{aligned}\sum_{h=0}^{\infty} \frac{h}{2^h} &= [0 + \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \dots] \\&= [\frac{1}{2} + \frac{1}{2^2} + \dots] + [\frac{1}{2^2} + \frac{1}{2^3} + \dots] + [\frac{1}{2^3} + \frac{1}{2^4} + \dots] + \dots \\&= [1 + \frac{1}{2} + \frac{1}{2^2} + \dots] [\frac{1}{2} + \frac{1}{2^2} + \dots] \\&= \frac{1}{2} \frac{1}{(1 - \frac{1}{2})^2} = 2\end{aligned}$$





堆排序算法

算法 **Heap-sort(A)**

1. **Build-Heap(A)**

2. **for** $i \leftarrow \text{length}[A]$ **downto** 2

3. **do** *exchange* $A[1] \leftrightarrow A[i]$

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

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 $O(n \log n)$





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 $L=\{x_1, x_2, \dots, x_n\}$, 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) $x_i > x_j$

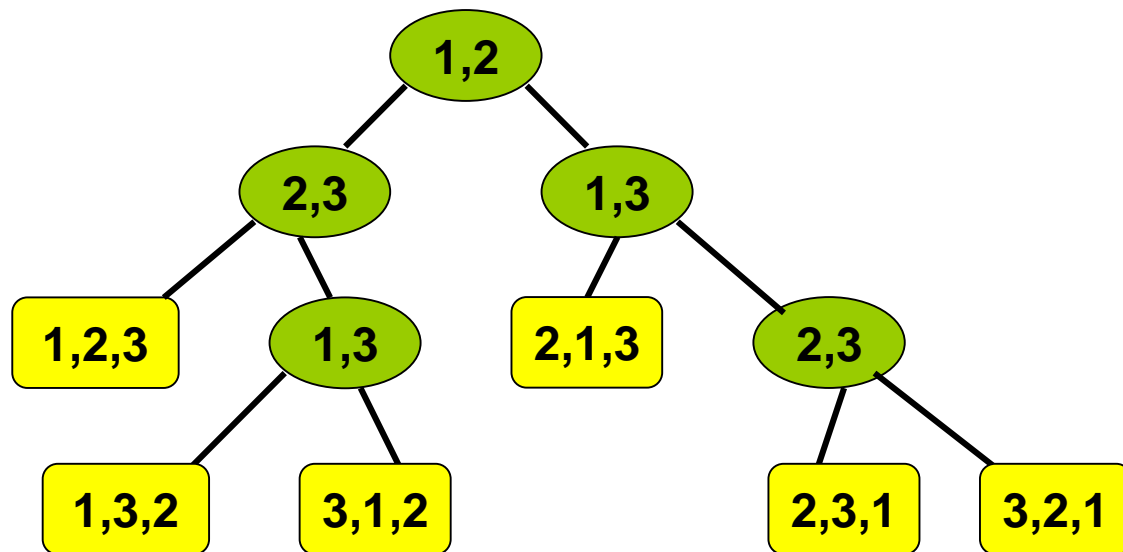
若算法结束, k 的右儿子标记为输入;

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径，
算法最坏情况下的比较次数：树深

删除非二叉的内结点，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 $t \leq 2^d$.

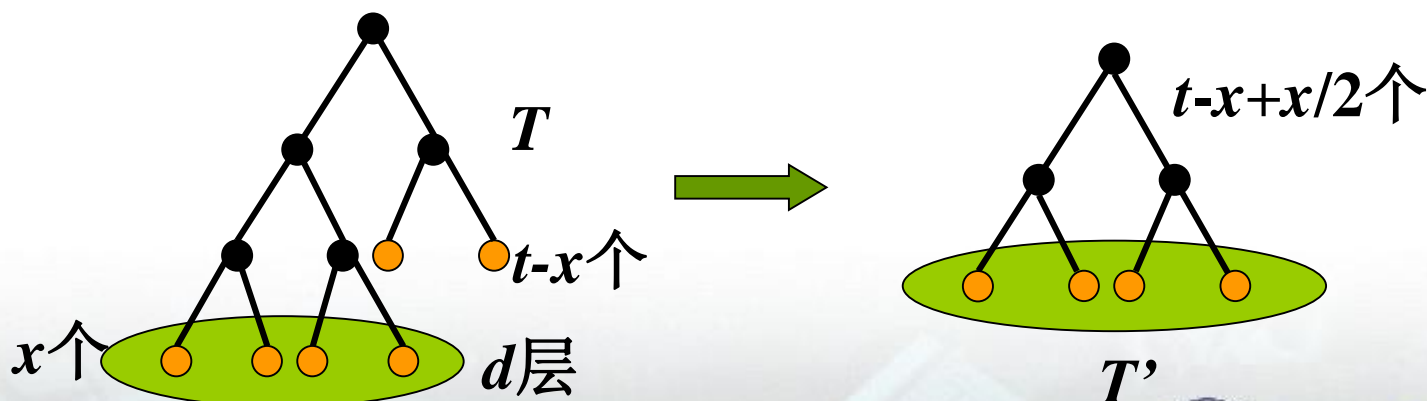
证明 归纳法.

$d=0$, 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

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

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



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 决策树的树叶有 $n!$ 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

$$\begin{aligned}\log n! &= \sum_{j=1}^n \log j \geq \int_1^n \log x dx = \log e \int_1^n \ln x dx \\ &= \log e (n \ln n - n + 1) \\ &= n \log n - n \log e + \log e \\ &\approx n \log n - 1.5n\end{aligned}$$

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.

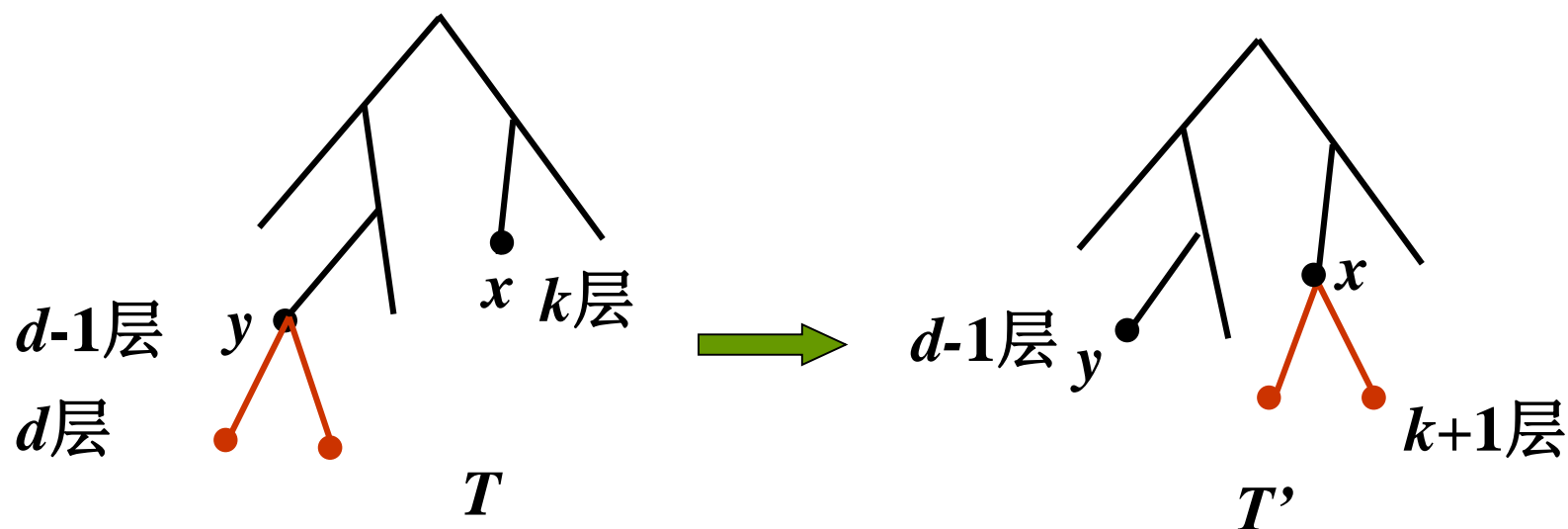
引理3 在具有 t 片树叶的所有 **B**-树中, 树叶分布在两个相邻层上的树的 **epl** 值最小

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



$$\begin{aligned}\text{epl}(T) - \text{epl}(T') &= (2d+k) - [(d-1) + 2(k+1)] \\ &= 2d+k - d+1 - 2k - 2 = d-k-1 > 0 \quad (d > k+1)\end{aligned}$$

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

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

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 $d = k$,

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



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 ,

$$x + y = t$$

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

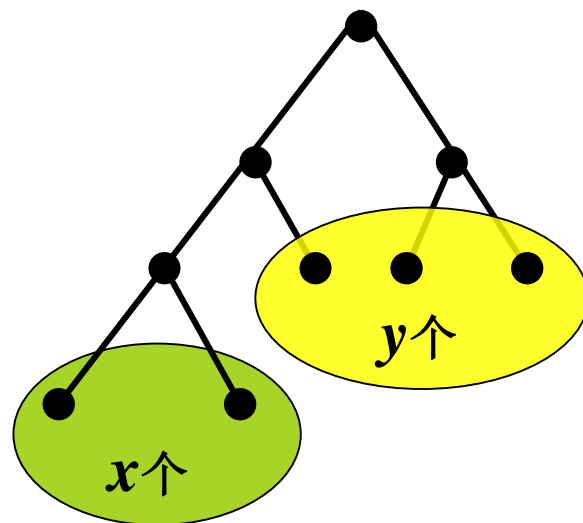
解得 $x = 2t - 2^d$, $y = 2^d - t$.

$$\text{epl}(T) = x d + y (d-1)$$

$$= (2t - 2^d)d + (2^d - t)(d-1)$$

$$= td - 2^d + t = t(d-1) + 2(t - 2^{d-1})$$

$$= t \lfloor \log t \rfloor + 2(t - 2^{\lfloor \log t \rfloor}) \quad (\lfloor \log t \rfloor = d-1)$$



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 $\text{epl}(T)/n!$, 根据引理4

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

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

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	$O(n^2)$	$O(n\log n)$	$O(\log n)$	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

