Analysis of Algorithms(2)

Pseudocode (伪代码)
Solving Recurrences(解递归)

"pseudocode"

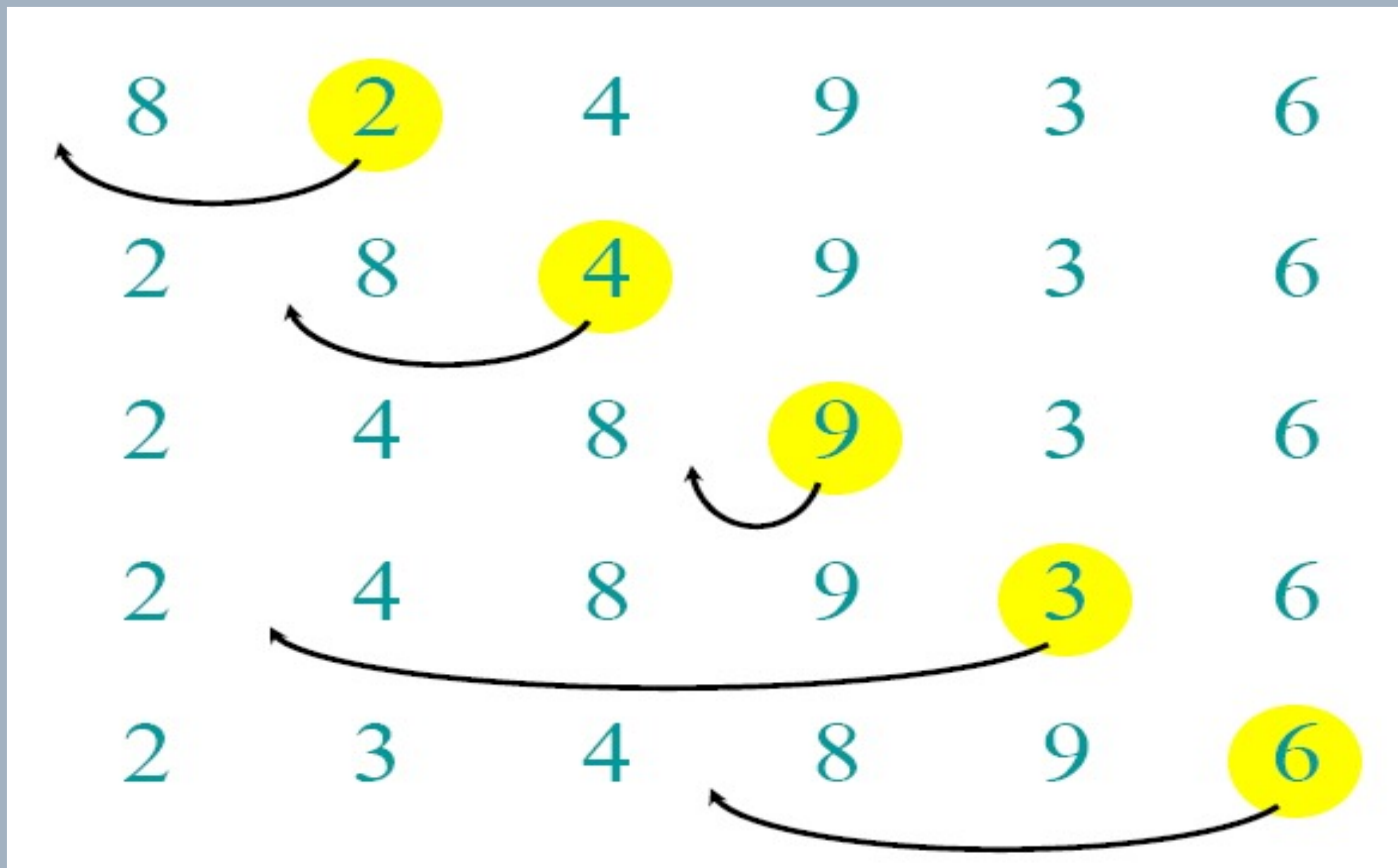
```
INSERTION-SORT (A, n)      ◁ A[1 . . n]
   for j ← 2 to n
        do key ← A[j]
             i ← j – 1
             while i > 0 and A[i] > key
                  do A[i+1] ← A[i]
                       i ← i – 1
             A[i+1] = key
```

- While-循环最坏情形Θ(j).
- While-循环平均情形Θ(j/2),当插入位置有相同概率时.

Pseudocode-Insertion Sort

- “←”表示” 赋值 “(assignment).
- 忽略数据类型、变量的说明等与算法无关的部分.
- 允许使用自然语言表示的一些 “macros”.
- 伪代码突出了程序使用的算法.

Example of insertion sort



插入排序(Insertion sort)分析

Worst case: Input reverse sorted.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \qquad \text{[arithmetic series]}$$

Average case: All permutations equally likely.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n.
- Not at all, for large n.

最优二叉树(optimized binary tree)

for m $\leftarrow$2 to n $\Theta(1)$

do

{for i $\leftarrow$0 to n-m $\Theta(1)$

do{

j $\leftarrow$ i + m $\Theta(1)$

w(i, j) $\leftarrow$w(i, j-1) + P(i) + Q(j) $\Theta(1)$

c(i, j)$\leftarrow$$\min_{i<l\leq j}$ { c(i, l-1) + c (l, j) } + w(i, j)

}

}

W(n,n),P(n),Q(n),c(n,n) 是算法使用的数组,假定已初始化.

Optimized binary tree

- $\min_{i<l\leq j} \{ c(i, l-1) + c(l, j) \}$: $\Theta(j-i)=\Theta(m)$
- Inner for-loop: $\Theta(m(n-m))$;
- Total : $\Theta(\sum_{2\leq m\leq n} m(n-m))=\Theta(n^3)$

解递归

Solving Recurrences-

(1)Recursion tree

(2)Substitution method

(3)Master method

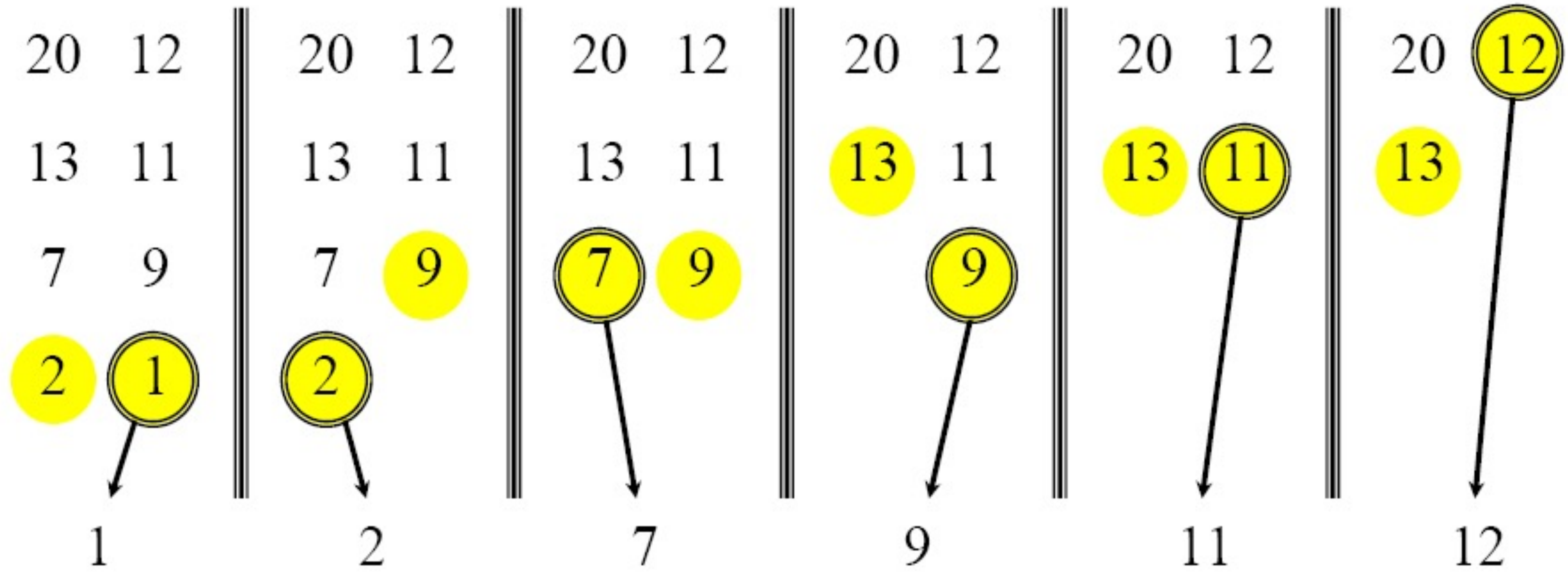
Merge-Sort

MERGE-SORT $A[1 \,..\, n]$

1. If $n = 1$, done.
2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$.
3. "***Merge***" the 2 sorted lists.

Key subroutine: **MERGE**

Merging two sorted arrays



Time = $\Theta(n)$ to merge a total of n elements (linear time).

Merge-Sort Analysis

$T(n)$	**MERGE-SORT** $A[1 \,..\, n]$
$\Theta(1)$	1. If $n = 1$, done.
$2T(n/2)$	2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$.
$\Theta(n)$	3. ***"Merge"*** the 2 sorted lists

- Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$
- 假定$n=2^h$,$h \geq 0$,上式变为$2T(n/2)$

Recurrence for merge sort

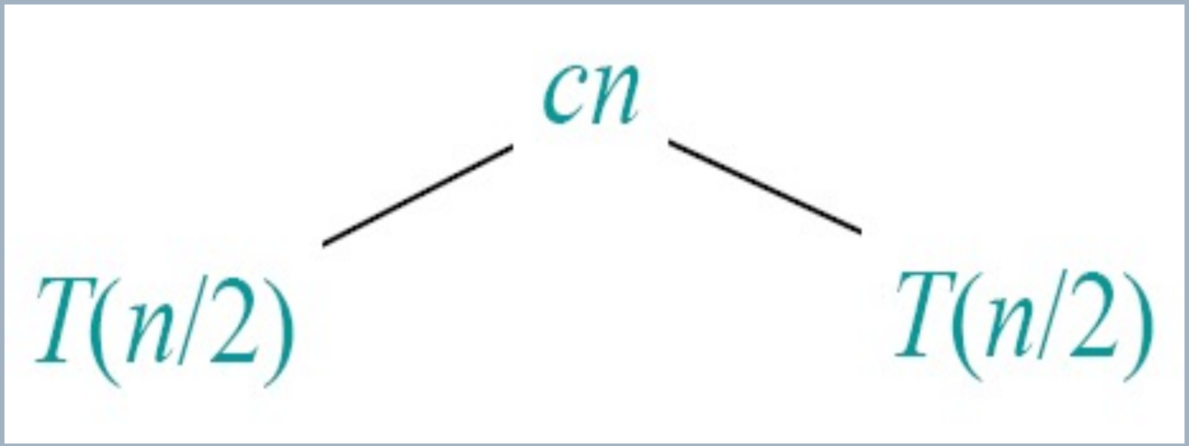
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- 隐含假定$n=2^h$.
- 以cn代替**Θ**(n),不影响渐近分析的结果.
- “If n=1”,更一般的是 “if $n<n_0$, $T(n)=\Theta(1)$”：指:可找到足够大常数c_1,使得

$$T(n)<c_1 \text{ if } n<n_0.$$

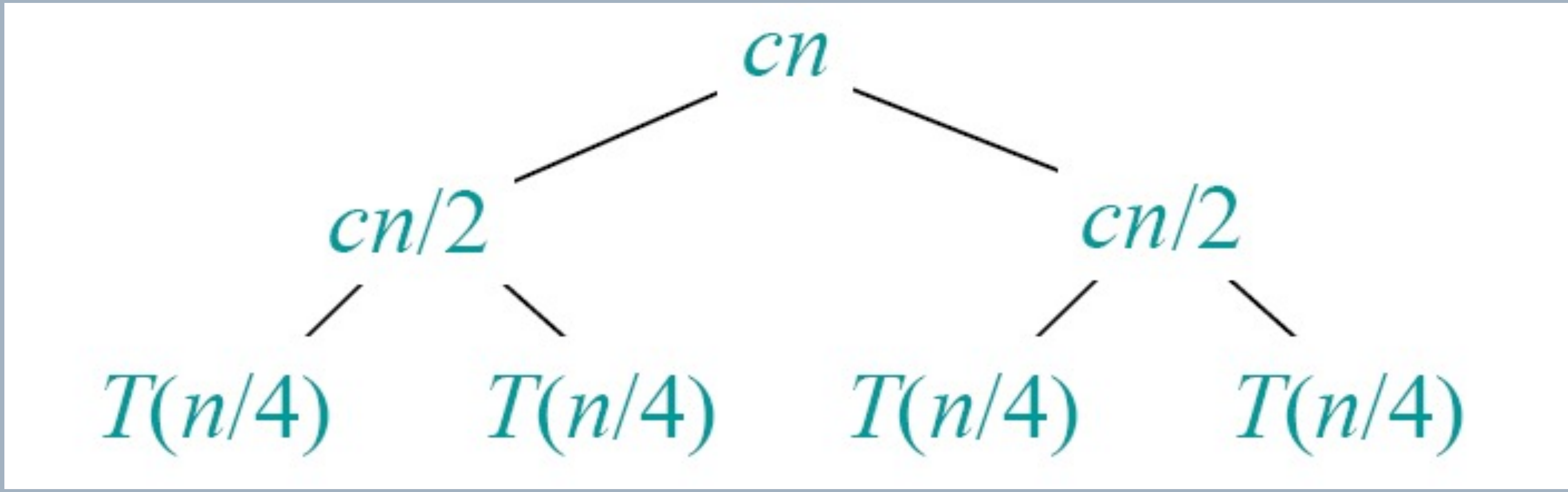
Recursion tree

- 解$T(n) = 2T(n/2) + cn$, 其中 $c > 0$ 为常数.
- 递归展开到$T(n_0)$,会导致推导的麻烦.所以展开到T(1).然后再从前n_0个T(n)的值确定渐近分析的常数.



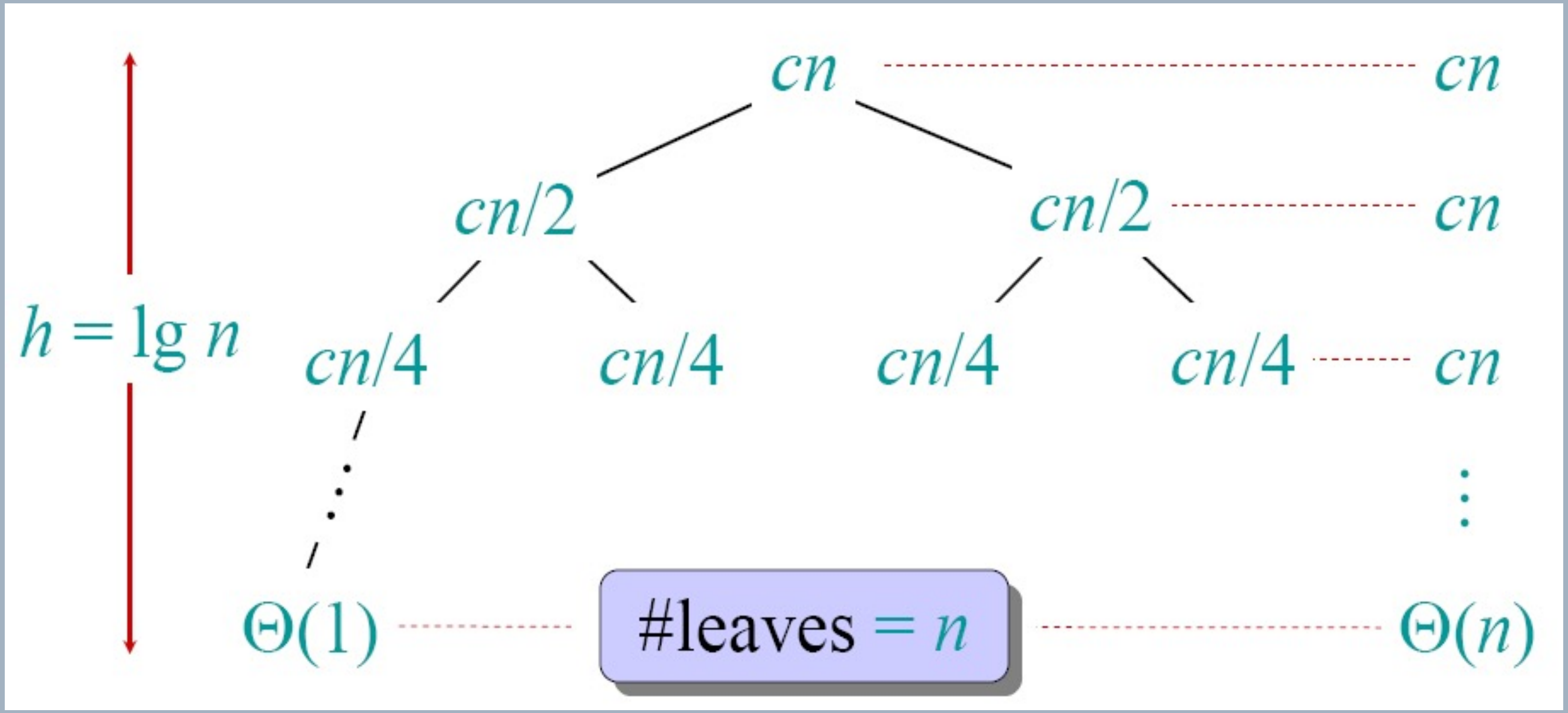
Recursion tree

- Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

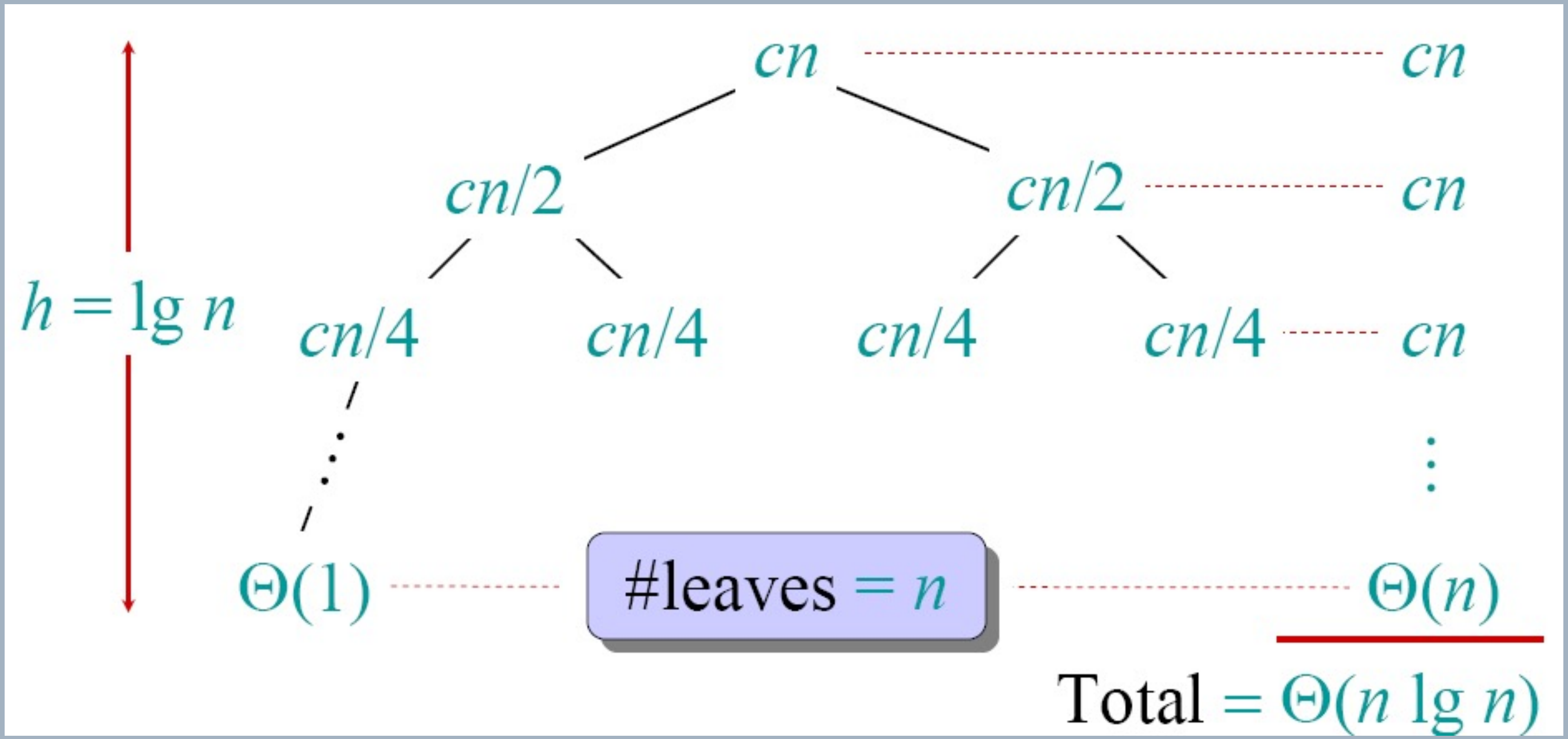


Recursion tree

- Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree



递归树等价于迭代展开

- $T(n)=4T(n/2)+n$
  $=4(4T(n/2^2)+n/2)+n$
  $=4^2T(n/2^2)+n+2n$
  $=4^3T(n/2^3)+n+2n+2^2n$
  $=4^hT(n/2^h)+n(1+2+\cdots+2^{h-1})$
  $=n^2T(1)+n(2^h-1)$
  $=\Theta(n^2)$
- 很多递归式用递归树解不出来,但递归树能提供直觉,帮助我们用归纳法求解(Guess 归纳假设).

Conclusions

- **$\Theta(n \lg n)$比$\Theta(n^2)$增长的慢.**
- 所以,merge sort 渐近(asymptotically)优于插入排序.
- 实际上, merge sort 优于 insertion sort 仅当 $n > 30$ or so.
- 1000*nlogn算法当n比较小时未必比n^2算法要快. n足够大时前者才能看出优势.

当$n \neq 2^h$

- $2^h \leq n < 2^{h+1} =>$

  $n = \Theta(2^h)$, $h = \Theta(\log n)$
- $T(2^h) \leq T(n) \leq T(2^{h+1})$.
- 所以,$\Theta(h2^h) \leq T(n) \leq \Theta((h+1)2^{h+1})$
- $\Theta((h+1)2^{h+1}) = \Theta(h2^{h+1} + 2^{h+1})$

  $= \Theta(h2^{h+1}) = \Theta(h2^h)$
- 所以$T(n) = \Theta(h2^h) = \Theta(n\log n)$

较一般的递归式

- 较一般的递归:$T(n)=aT(n/b)+cn$, a,b是大于1的整数,递归树方法仍可使用.
- 首先考虑$n=b^h$情形:

$$T(n)=a^hT(1)+cn(1+(a/b)+\cdots+(a/b)^{h-1})$$

$$= a^hT(1)+cb^h(1+(a/b)+\cdots+(a/b)^{h-1})$$

- 当$b^h \le n < b^{h+1}$,仍有:

$$h=\Theta(\log_b n)$$

换底公式: $\log_b n=\log_2 n/\log_2 b \Rightarrow$

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

Substitution methods

- *The most general method:*

1. Guess the form of the solution.

2. Verify by induction.

3. Solve for constants.

Example1

- $T(n) = 4T(n/2) + n$ （n=2^{h}）
- [Assume that $T(1)=\Theta(1)$.]
- Guess O(n^3) :归纳假定为$T(n) \leq cn^3$. c是待定常数.
- 应用假定有:$T(k) \leq ck^3$ for $k < n$.
- 归纳证明: $T(n) \leq cn^3$ 并确定常数c.

Example(continued)

- $T(n)=4T(n/2)+n$

  $\leq 4c(n/2)^3+n$

  $=(c/2)n^3+n$

  $=cn^3-((c/2)n^3-n)$

  $\leqslant cn^3$

  取 $c\geq 2$ and $n\geq 1$,不等式$(c/2)n^3–n\geq 0$成立,

Example(continued)

- 还要检验初始条件是否满足归纳假设.
- 初始条件为:

  $T(n)=\Theta(1)$,当 $n<n_0$,n_0为常数.
- 因为有常数(n_0-1)个T的值:T(1),---,$T(n_0-1)$
- 所以可取c足够大,使得$T(n)\le cn^3$, 对$n<n_0$成立.
- ***This bound is not tight!***

Example(continued)

- We shall prove that $T(n)=O(n^2)$.
- Assume that $T(k) \leq ck^2$ for $k<n$:
- $T(n)=4T(n/2)+n$

  $\leq 4c(n/2)^2+n$

  $=cn^2+n$
- 归纳不能往下进行！

Continued

- **IDEA:** Strengthen the inductive hypothesis.
- ***Subtract*** a low-order term.
- *Inductive hypothesis*: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.
- $T(n) = 4T(n/2) + n$

$$\le 4[c_1(n/2)^2 - c_2(n/2)] + n$$

$$= c_1 n^2 - c_2 n - (c_2 n - n)$$

$$\le c_1 n^2 - c_2 n \qquad \text{if } c_2 > 1$$

Continued

- For $1 \le n < n_0$,
  we have "$\Theta(1)$" $\le c_1 n^2 - c_2 n$,
  if we pick c_1 big enough.

The Master Method

- The master method用来解下述递归

$$T(n)=aT(n/b)+f(n) ,$$

式中$a\geq1$, $b>1$, 为整数, $f(n)>0$.

- 按$f(n)$相对于$n^{\log a}$的渐近性质, 分三种情形进行分析.
- 这里loga 指以b为底的a的对数$\log_b a$.

The Master Method:情形1

- 情形1.

  $f(n)=O(n^{\log a-\varepsilon})$, $\varepsilon>0$,为某一常数

  $f(n)$的增长渐近地慢于$n^{\log a}$ (慢 n^{ε} 倍).
- ***Solution:*** $T(n)=\Theta(n^{\log a})$.

The Master Method:情形2

- 情形2:
  $f(n)=\Theta(n^{\log a} \lg^k n)$ $k\geq 0$ 为某一常数.
- $f(n)$ 和 $n^{\log a}$ 几乎有相同的渐近增长率.
- ***Solution:*** $T(n)=\Theta(n^{\log a} \lg^{k+1} n)$.

The Master Method:情形3

- 情形3
- $f(n)=\Omega(n^{\log a+\varepsilon})$ $\varepsilon>0$为一常数.
  $f(n)$多项式地快于$n^{\log a}$ (by an n^{ε} factor),
- $f(n)$ 满足以下规则性条件:
  $af(n/b)\le cf(n)$, $0<c<1$ 为常数.
- ***Solution:*** $T(n)=\Theta(f(n))$.

Examples 1

- $T(n)=4T(n/2)+n$
- $a=4,\ b=2 \Rightarrow n^{\log a}=n^2;\ f(n)=n.$
- 情形1: $f(n)=O(n^{2-\varepsilon})$ for $\varepsilon=1$.
- $\therefore\ T(n)=\Theta(n^2)$.

Examples 2

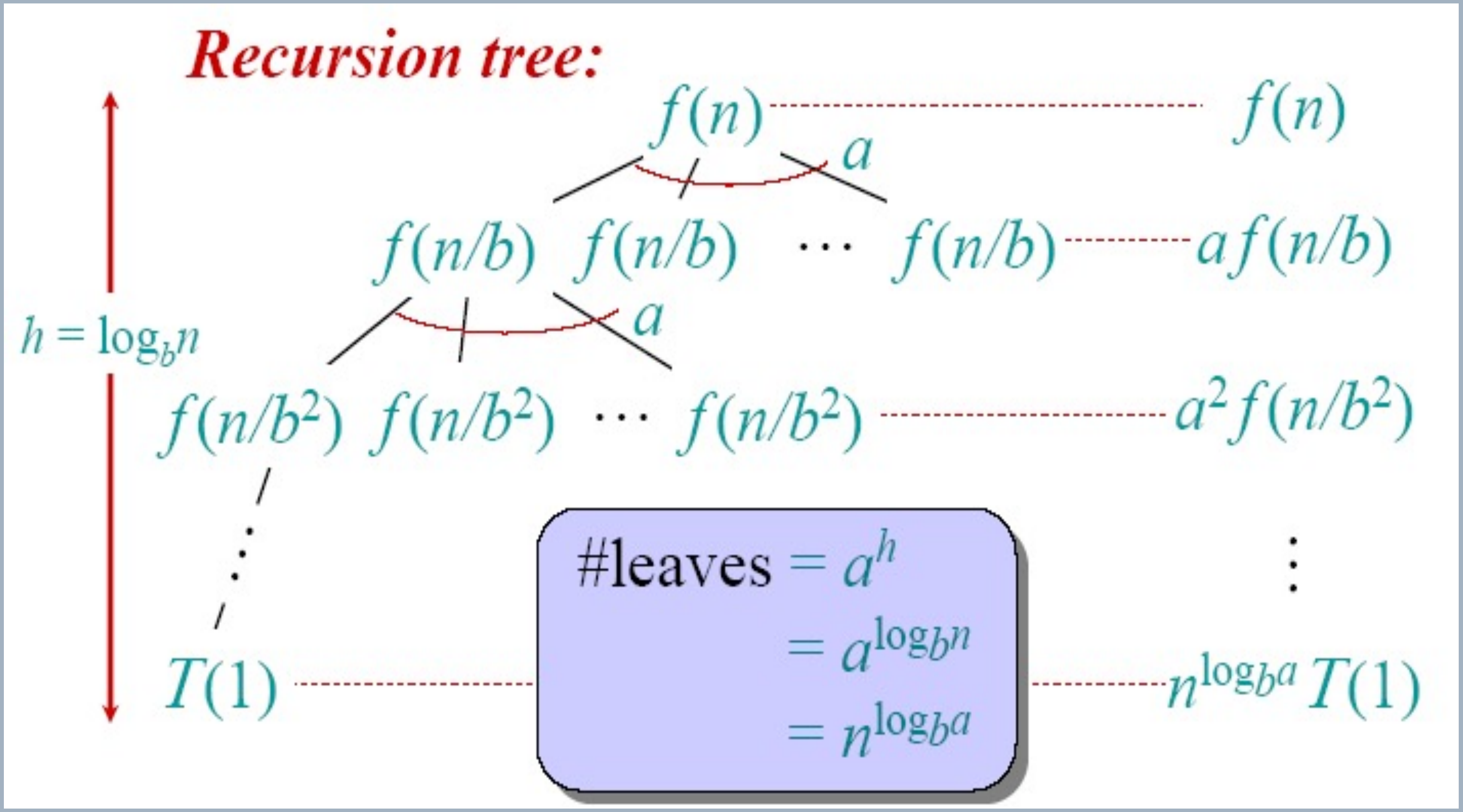
- $T(n)=4T(n/2)+n^2$

  $a=4,\ b=2 \Rightarrow n^{\log a}=n^2;\ f\,(n)=n^2.$
- 情形 2: $f(n)=\Theta(n^2\lg^0 n),\ k = 0.$
- $T(n)=\Theta(n^2\lg n).$

Examples 3

- $T(n)=4T(n/2)+n^3$
- $a=4$, $b=2 \Rightarrow n^{\log a}=n^2$; $f(n)=n^3$.
- **CASE 3**: $f\,(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon=1$
- ***and*** $4(n/2)^3 \leq cn^3$ (reg. cond.) for $c=1/2$.
- $\therefore$ $T(n)=\Theta(n^3)$.

Idea of the Master method



Idea of the Master method

- 从递归树可知:

$T(n)= a^hT(1)+\sum a^kf(n/b^k)$

$=a^hT(1)+\sum a^kf(b^{h-k})$

$=n^{loga}T(1)+\sum a^kf(b^{h-k})$

- T(n)的渐近性质由n^{loga}和$\sum a^kf(b^{h-k})$的渐近性质决定:两者中阶较高的决定！
- 为什么要比较f(n)和n^{loga} 的原因！

Master 方法分析

- $n=b^h$, $h=\log_b n$, $n^{\log a}=a^h$
- loga 指 $\log_b a$.
- $T(n)= a^hT(1)+\sum_k a^k f(n/b^k)$

  $=a^hT(1)+\sum_k a^k f(b^{h-k})$

  $\sum_k$求和范围:$k=0,\ldots,h-1$
- $\sum_k a^k f(b^{h-k})= \sum_k a^{h-k} f(b^k)$, (后者求和从1到h)

Proof (Case 1)

- $n=b^h$, $n^{\log a}=a^h$, $n^{\varepsilon}=b^{\varepsilon h}$.因$f(n)=O(n^{\log a-\varepsilon})$,所以 $f(n)\le cn^{\log a-\varepsilon}=ca^h/b^{\varepsilon h}$ $=>f(b^h)\le c(a^h/b^{\varepsilon h})$
- 取充分大的c,使$f(b^i)\le c(a^i/b^{\varepsilon i})$ **对**所有i成立.
- $T(n)=a^hT(1)+\sum a^kf(b^{h-k})$,($\sum$: $k=0,\ldots,h-1$)
- $T(n)\le a^hT(1)+c\sum a^k(a^{h-k}/b^{\varepsilon(h-k)})$

  $=a^hT(1)+ca^h\sum(1/b^{\varepsilon(h-k)})$

  $\le a^hT(1)+ca^h\sum_{1\le k<\infty}(1/b^{\varepsilon k})$

  $\le c'a^h$ (因$b>1,\varepsilon>0$,无穷和收敛)
- 所以$T(n)=O(n^{\log a})$; 显然, $T(n)=\Omega(n^{\log a})$

Proof of Case2

- $f(n)=\Theta(n^{\log a} \lg^k n)$, $n=b^h$, $n^{\log a}=a^h$
- $\Rightarrow f(b^h) \le ca^h(h\lg b)^k = ca^h h^k(\lg b)^k$
- 类似,$f(b^i) \le ca^i i^k(\lg b)^k$.　(取充分大的c)
- $T(n) \le a^h T(1) + ca^h \sum_i i^k (\lg b)^k \le$
  $a^h T(1) + c'a^h h^{k+1} (\lg b)^k = a^h T(1) +$
  $c'a^h h^{k+1} (\lg b)^{k+1} / \lg b = c''(n^{\log a} \lg^{k+1} n)$
- 上面推导中用到:
  $1^k + \ldots + h^k = \Theta(h^{k+1})$

Case 3

- 反复应用规则性条件有:

  $$a^k f(n/b^k) \le c^k f(n)$$

- 所以: $\sum a^k f(n/b^k) \le \sum c^k f(n)$
- $T(n)=a^h T(1)+\sum a^k f(n/b^k) \le a^h T(1)+f(n)\sum c^k \le a^h T(1)+f(n)(1-c)^{-1}$
- 因为, $f(n)=\Omega(n^{\log a+\varepsilon})$,量级高于$a^h$, 所以 $T(n)=O(f(n))$.
- 又,$T(n)>f(n)$.所以,$T(n)=\Theta(f(n))$

Summary

- Recursion：$T(b^h)=aT(b^{h-1})+f(b^h)$
- h为非负整数,但a,b为正数(可不是整数).

$$T(b^h) = a^h T(1) + \sum_{i=0}^{h-1} a^i f(b^{h-i})$$

- Transform into

$$\sum_{i=0}^{h-1} a^i f(b^{h-i}) = \sum_{i=1}^{h} a^{h-i} f(b^i)$$

Master (主项)法

- Case 1

$$f(b^i) \le c d^i / b^{i\varepsilon}$$

$$\sum_{i=1}^{h} a^{h-i} f(b^i) \le c d^h \sum_{i=1}^{h} (1/b^{i\varepsilon})$$

- Case 2

$$f(b^i) \le c d^i (i \lg b)^k$$

$$\sum_{i=1}^{h} a^{h-i} f(b^i) \le c a^h (\lg b)^k \sum_{i=1}^{h} i^k$$

Master (主项)法

- Case 3

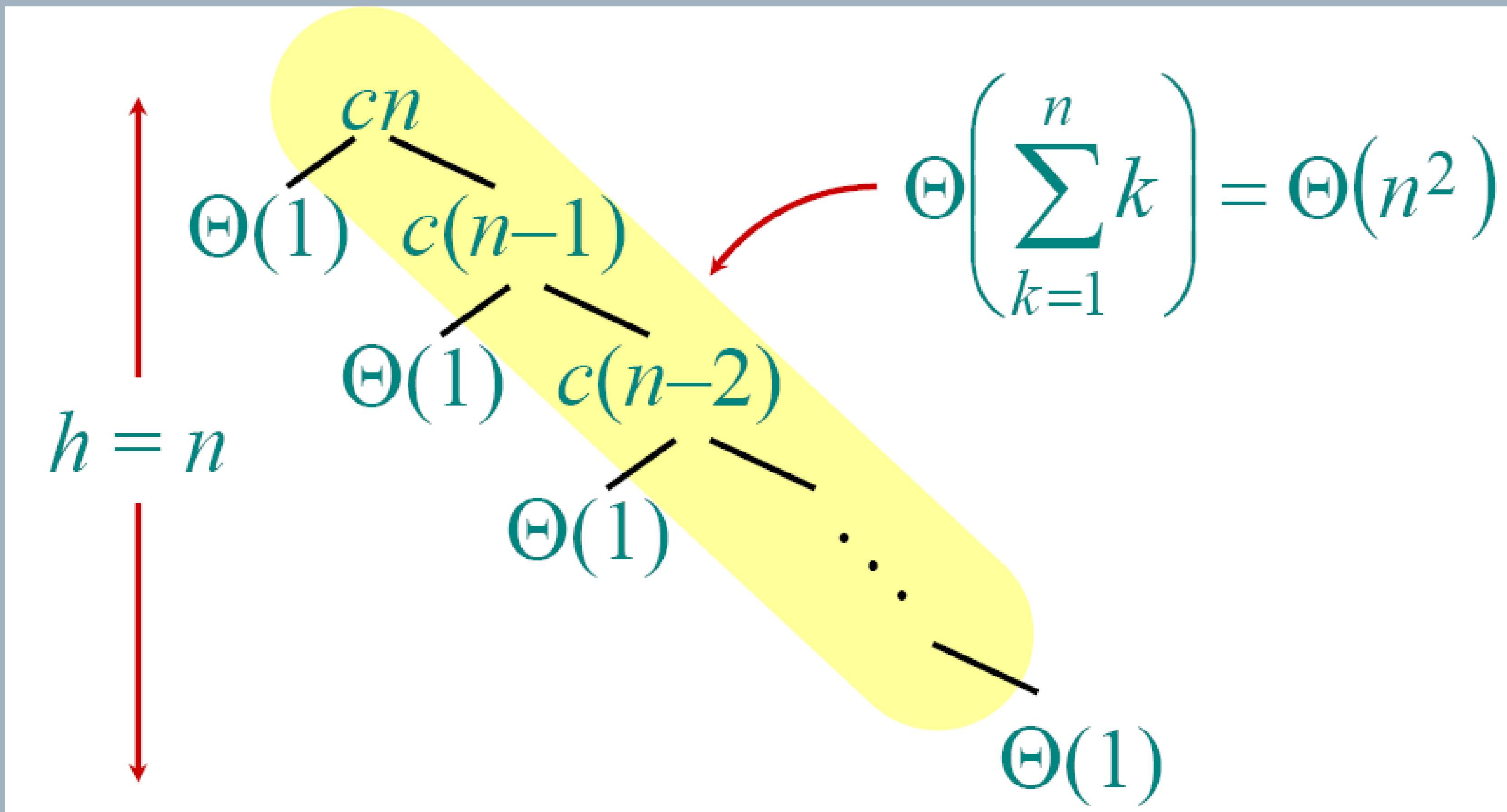
$$af(n/b) \le cf(n) => f(n/b^i) \le (c/a)^i f(n)$$

$$\sum_{i=0}^{h-1} a^i f(n/b^i) \le f(n) \sum_{i=0}^{h-1} c^i$$

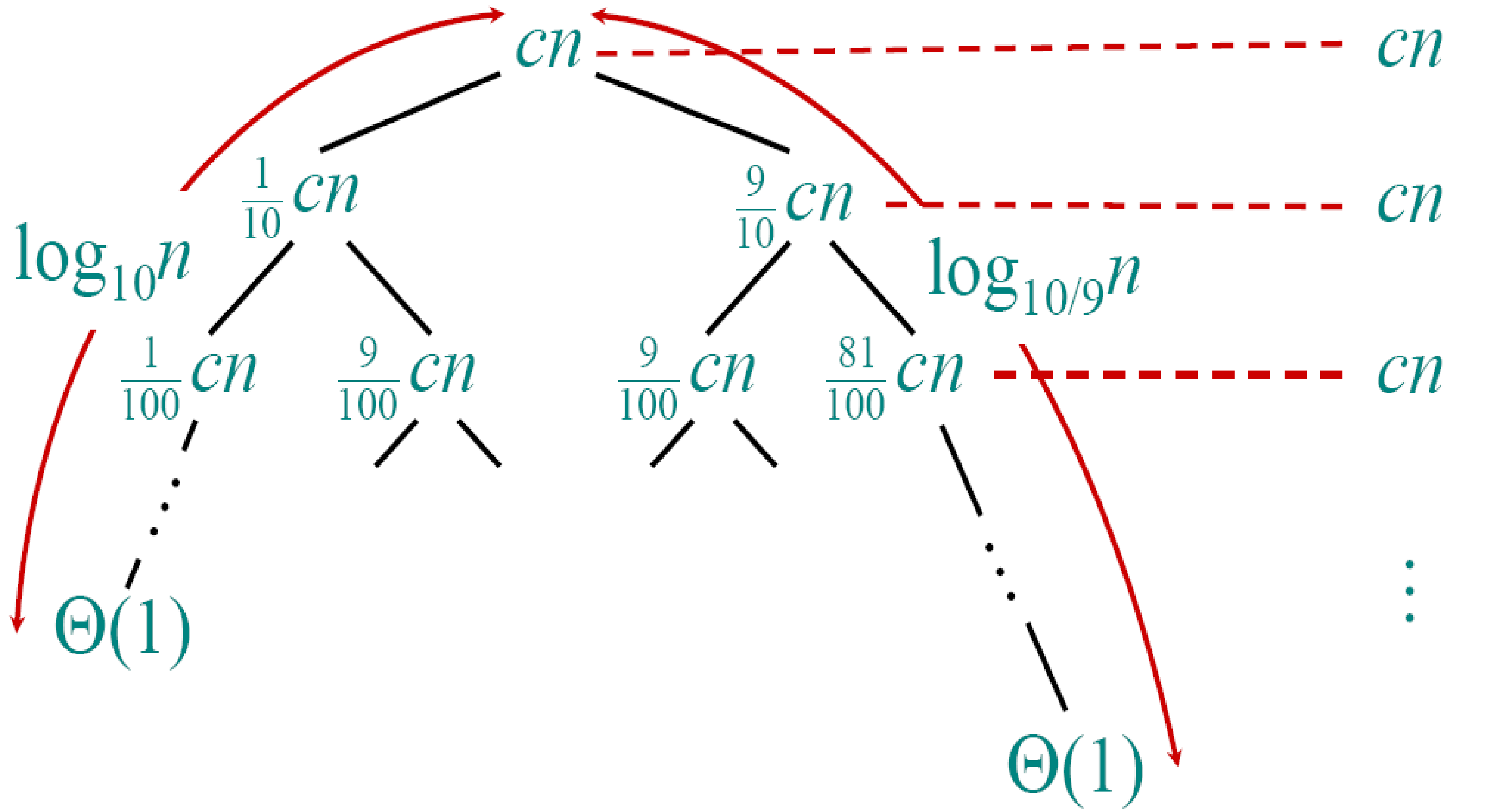
- 自n_0起满足规则条件,则结论亦真.

补充例题

例题1：展开递归树:$T(n)=T(1)+T(n-1)+cn$, 并做渐近分析


cn
$\Theta(1)$ $c(n-1)$
$\Theta(1)$ $c(n-2)$
$\Theta(1)$
$\Theta(1)$
$h = n$
$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$

例题2：展开T(n)=T(0.1n)+T(0.9n)+**Θ**(n)的递归树并计算递归树的深度和T(n)的渐近值.



$$cn\log_{10}n \le T(n) \le cn\log_{10/9}n + O(n)$$

裴波那契(Fibonacci)序列

- 序列$F_0=F1=1, F_i=F_{i-1}+F_{i-2}$ (i>1)称为Fibonacci序列. 以下算法计算第n个Fibonacci数:

  ```
  proc F(n)
  if n≤1 return(1)
    else return( F(n-1)+F(n-2));
  end proc
  ```

- 令t(n)为算法执行的加法次数,有:

  $t(n)=0$ for $n=0,1$

  $t(n)=t(n-1)+t(n-2)+1$

  特别$t(2)=1, t(3)=2$

续

- 因为t(n-1)>t(n-2),有
  $t(n)<2t(n-1)+1$,for $n>2$. 用归纳法易证:
  $$t(n) \le 2^n$$
- 又有$t(n)>2t(n-2)>\cdots>2^{k-1}t(2)=2^{k-1}\quad n=2k$
  $$>2^{k-1}t(3)=2^k \quad n=2k+1$$
- $t(n)=O(2^n)$
- 算法有指数的时间复杂度.
- 实际上这是因递归引起的大量的重复计算而非问题本身的难度所致.可设计一非常简单的线性时间复杂度的迭代算法.

Homework(2)

- 1.用归纳法证明

$$T(N) \le \begin{cases} 0 & \text{if } N=1 \\ \underbrace{T(\lceil N/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor N/2 \rfloor)}_{\text{solve right half}} + \underbrace{cN}_{\text{combine}} & \text{otherwise} \end{cases}$$

$$\Rightarrow T(N) \le cN \lceil \log_2 N \rceil$$

- 2.应用master方法求解$T(n)=2T(n/2)+\Theta(n^{1/2})$
- 3.展开递归树:$T(n)=T(2)+T(n-2)+cn$,并做渐近分析
- 展开$T(n)=T(0.2n)+T(0.8n)+\Theta(n)$的递归树并计算递归树的深度和$T(n)$的渐近值.
- 14章练习33-(a),(b),(c),(d),(e),(f),(g),(h),(i),(j)