algorithm

# 3.递归与主定理

## 3.1 递推方程

- 设序列 $a_0, a_1, \ldots, a_n, \ldots$, 简记为 $\{a_n\}$,
- 一个把 $a_n$ 与某些个 $a_i\,(i<n)$ 联系起来的等式叫做关于序列 $\{a_n\}$ 的递推方程

- 递推方程的求解:
- 给定关于序列$\{a_n\}$的递推方程和若干初值，计算$a_n$

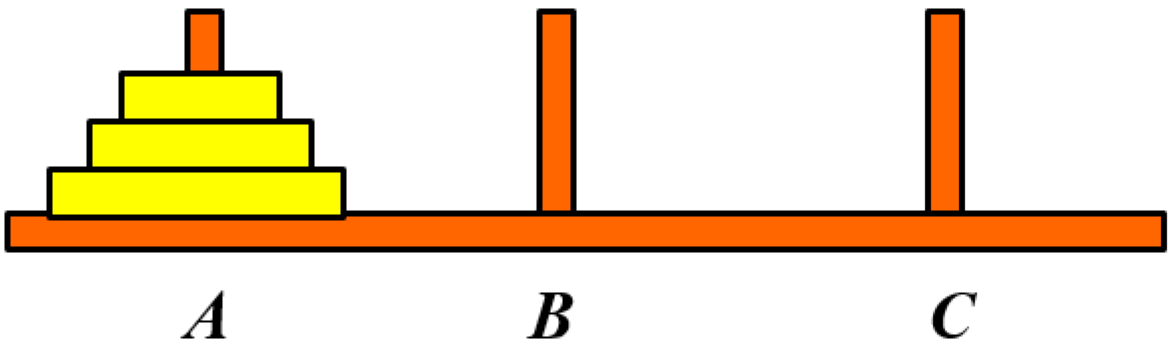
# 递推方程的例子

- Fibonacci数
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
- 递推方程：$f_n=f_{n-1}+f_{n-2}$
- 初值：$f_0=1, f_1=1$
- 解：

$$f_n=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}-\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}$$

# Hanoi塔问题

- $n$个盘子从大到小顺序放在 $A$ 柱上，要把它们从 $A$ 移到 $C$，每次移动1个盘子，移动时不允许大盘压在小盘上，设计一种移动方法



# 递归算法

- 算法 **Hanoi (*A*, *C*, *n*)** // *n*个盘子*A*到*C*

**1. if *n*=1 then move (*A*, *C*)** // 1个盘子*A*到*C*

**2. else Hanoi (*A*, *B*, *n*–1)**

**3. move (*A*, *C*)**

**4. Hanoi (*B*, *C*, *n*–1)**

- 设 $n$个盘子的移动次数为 $T(n)$
  - $T(n) = 2T(n-1) + 1$
  - $T(1) = 1$

# 分析算法

- $T(n) = 2T(n-1) + 1$，$T(1) = 1$
- 解：$T(n) = 2^n - 1$

- 假如1 秒移1个，64个盘子要多少时间?
- 5000亿年!

- 问：有没有更好的算法?

# 插入排序

- 算法 **Insert Sort ($A$, $n$)**

1. **for** $j \leftarrow 2$ **to** $n$
2. $\quad x \leftarrow A[j]$
3. $\quad i \leftarrow j-1$ // 把 $A[j]$ 插入
4. $\quad$ **while** $i > 0$ **and** $x < A[i]$ **do**
5. $\qquad A[i+1] \leftarrow A[i]$
6. $\qquad i \leftarrow i-1$
7. $\quad A[i+1] \leftarrow x$

# 最坏情况下时间复杂度

- 插入排序:
- 设基本运算是元素比较，对规模为 $n$
- 的输入最坏情况下的时间复杂度$W(n)$

$$W(n)=W(n-1)+n-1$$

$$W(1)=0$$

- 解为

$$W(n) = n(n-1)/2$$

# 小结

- 递推方程的定义及初值
- 递推方程与算法时间复杂度的关系
- Hanoi塔的递归算法
- 插入排序的迭代算法

# 3.2 迭代法求解递推方程

- 不断用递推方程的右部替换左部
- 每次替换，随着 n 的降低在和式中多出一项
- 直到出现初值停止迭代
- 将初值代入并对和式求和
- 可用数学归纳法验证解的正确性

# Hanoi 塔算法

$$
\begin{aligned}
T(n) &= 2\,T(n-1) + 1 \\
&= 2\,[\,2T(n-2) + 1] + 1 \\
&= 2^2\,T(n-2) + 2 + 1 \\
&= \ldots \\
&= 2^{n-1}T(1) + 2^{n-2} + 2^{n-3} + \ldots + 2 + 1 \\
&= 2^{n-1} + 2^{n-1} - 1 \\
&= 2^n - 1
\end{aligned}
$$

# 插入排序算法

$W(n) = W(n-1) + n-1$

$= [W(n-2) + n-2] + n-1$

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

$= \ldots$

$= W(1) + 1 + 2 + \ldots + (n-2) + (n-1)$

$= 1 + 2 + \ldots + (n-2) + (n-1)$

$= n(n-1)/2$

# 换元迭代

- 将对 $n$ 的递推式换成对其他变元 $k$ 的递推式
- 对 $k$ 直接迭代
- 将解 (关于 $k$ 的函数) 转换成关于 $n$ 的函数

# 二分归并排序

**MergeSort** $(A, p, r)$

输入：数组 $A[p..r]$

输出：按递增顺序排序的数组 $A$

**1.** **if** $p < r$

**2.** **then** $q \leftarrow \lfloor (p+r)/2 \rfloor$

**3.** **MergeSort** $(A, p, q)$

**4.** **MergeSort** $(A, q+1, r)$

**5.** **Merge** $(A, p, q, r)$

# 换元

假设 $n=2^k$，递推方程如下：

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

$$W(1) = 0$$

换元：

$$W(2^k) = 2W(2^{k-1}) + 2^k-1$$

$$W(0) = 0$$

# 迭代求解

$$\boxed{W(2^k) = 2W(2^{k-1}) + 2^k - 1}$$

$$W(n) = 2W(2^{k-1}) + 2^k - 1$$

$$= 2[2W(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1$$

$$= 2^2 W(2^{k-2}) + 2^k - 2 + 2^k - 1$$

$$= 2^2[2W(2^{k-3}) + 2^{k-2} - 1] + 2^k - 2 + 2^k - 1$$

$$= \ldots$$

$$= 2^k W(1) + k2^k - (2^{k-1} + 2^{k-2} + \ldots + 2 + 1)$$

$$= k\,2^k - 2^k + 1$$

$$= n\log n - n + 1$$

# 解的正确性-归纳验证

- 证明:下述递推方程的解是

$$W(n)=n(n-1)/2$$

$$W(n)=W(n-1)+n-1$$

$$W(1)=0$$

- 方法：数学归纳法

证 $n=1$，$W(1) = 1\times(1-1)/2 = 0$

假设对于$n$，解满足方程，则

$$W(n+1)$$

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

$$= n[(n-1)/2+1] = n(n+1)/2$$

# 小结

- 迭代法求解递推方程
  - 直接迭代，代入初值，然后求和
  - 对递推方程和初值进行换元，然后求和，求和后进行相反换元，得到原始递推方程的解
  - 验证方法——数学归纳法

## 3.3 差消法化简 高阶递推方程

- 假设 $A[p..r]$ 的元素彼此不等
- 以首元素 $A[1]$ 对数组 $A[p..r]$ 划分,使得:
  - 小于 $x$ 的元素放在 $A[p..q-1]$
  - 大于 $x$ 的元素放在 $A[q+1..r]$
- 递归对 $A[p..q-1]$ 和 $A[q+1..r]$ 排序
- 工作量：子问题工作量 + 划分工作量

# 输入情况

- 有 n 种可能的输入

| $x$ 排好序位置 | 子问题 1规模 | 子问题 2规模 |
|---|---|---|
| 1 | 0 | $n-1$ |
| 2 | 1 | $n-2$ |
| 3 | 2 | $n-3$ |
| … | … | … |
| $n-1$ | $n-2$ | 1 |
| $n$ | $n-1$ | 0 |

- 对每个输入，划分的比较次数都是 $n-1$

# 工作量总和

$$
\begin{array}{rl}
& T(0) + T(n-1) + n-1 \\
& T(1) + T(n-2) + n-1 \\
& T(2) + T(n-3) + n-1 \\
& \quad \ldots \\
+ & T(n-1) + T(0) + n-1 \\
\hline
\end{array}
$$

$$2[T(1)+\ldots+T(n-1)]+n(n-1)$$

# 快速排序平均工作量

- 假设首元素排好序在每个位置是等概率的

$$T(n) = \frac{2}{n}\sum_{i=1}^{n-1} T(i) + O(n),\ \ n \geq 2$$

$$T(1) = 0$$

- 全部历史递推方程
- 对于高阶方程应该先化简，然后迭代

## 差消化简

- 利用两个方程相减，将右边的项尽可能消去，以达到降阶的目的

$$T(n) = \frac{2}{n}\sum_{i=1}^{n-1} T(i) + cn$$

$$nT(n) = 2\sum_{i=1}^{n-1} T(i) + cn^2$$

$$(n-1)T(n-1) = 2\sum_{i=1}^{n-2} T(i) + c(n-1)^2$$

# 差消化简

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

化简

$$nT(n) = (n+1)T(n-1) + c_1 n$$

变形

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c_1}{n+1}$$

# 迭代求解

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c_1}{n+1} = \cdots$$

$$= c_1[\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}] + \frac{T(1)}{2}$$

$$= c_1\left[\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right]$$

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

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

# 3.4 递归树

- 递归树是迭代计算的模型
- 递归树的生成过程与迭代过程一致
- 递归树上所有项恰好是迭代之后产生和式中的项
- 对递归树上的项求和就是迭代后方程的解

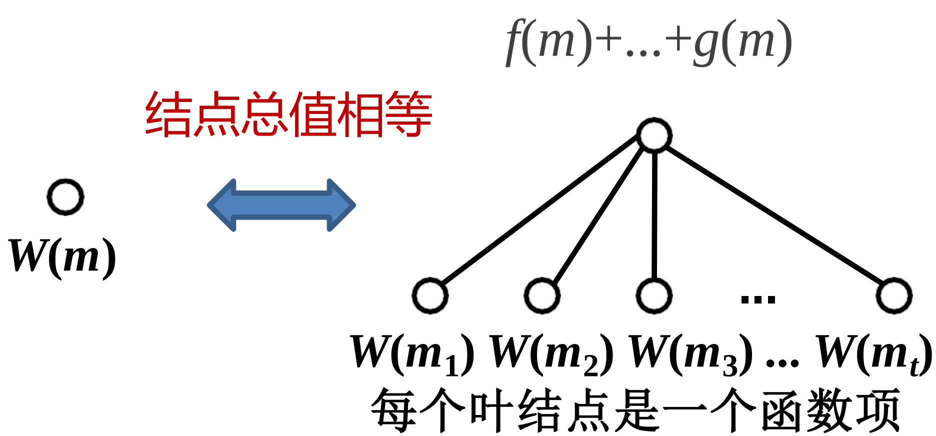
# 迭代在递归树中的表示

如果递归树上某结点标记为$W(m)$

$$W(m) = W(m_1)+\ldots+W(m_t)$$

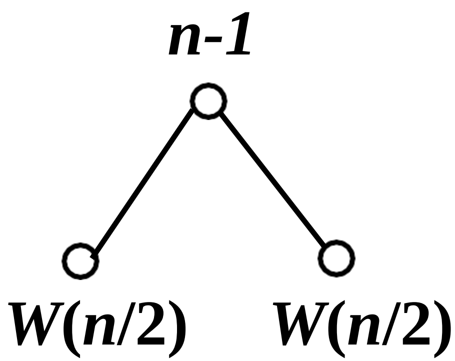
$$+ f(m)+\ldots+g(m),\quad m_1, \ldots, m_t < m$$

其中$W(m_1),\ldots,W(m_t)$称为函数项.



# 二层子树的例子

- 二分归并排序
- $W(n) = 2W(n/2) + n\text{-}1$

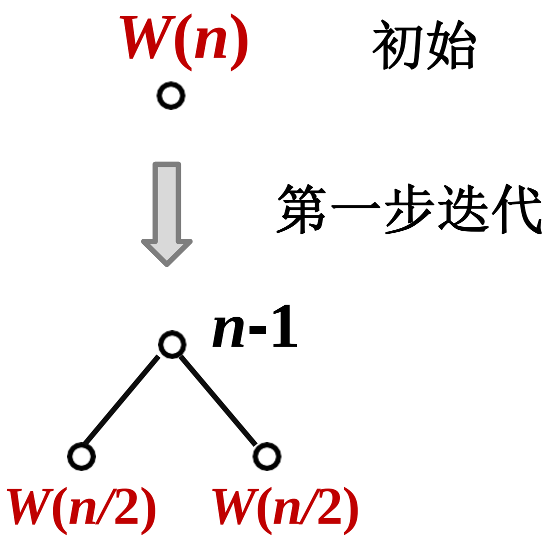


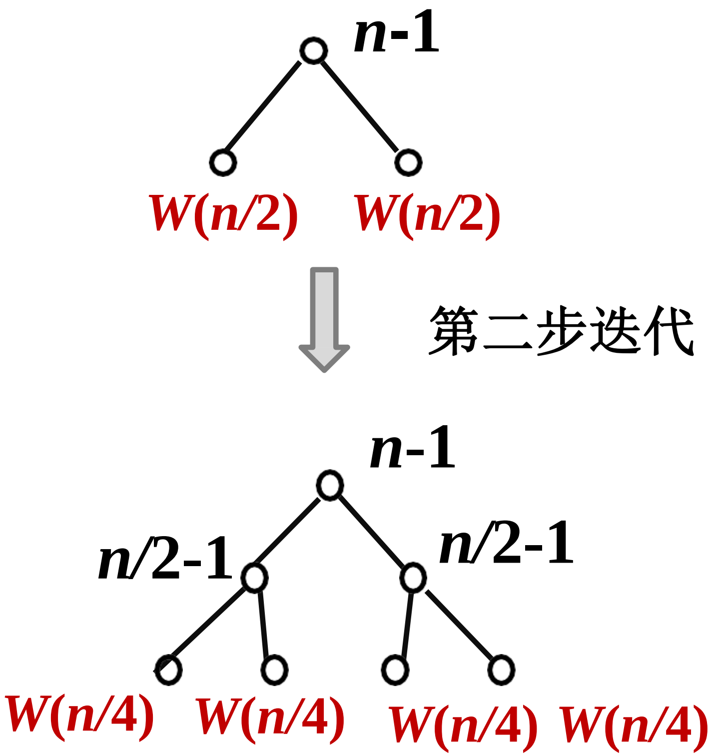
## 递归树的生成规则

- 初始，递归树只有根结点, 其值为$W(n)$
- 不断继续下述过程:
  - 将函数项叶结点的迭代式$W(m)$表示成二层子树
  - 用该子树替换该叶结点
- 继续递归树的生成，直到树中无函数项(只有初值)为止

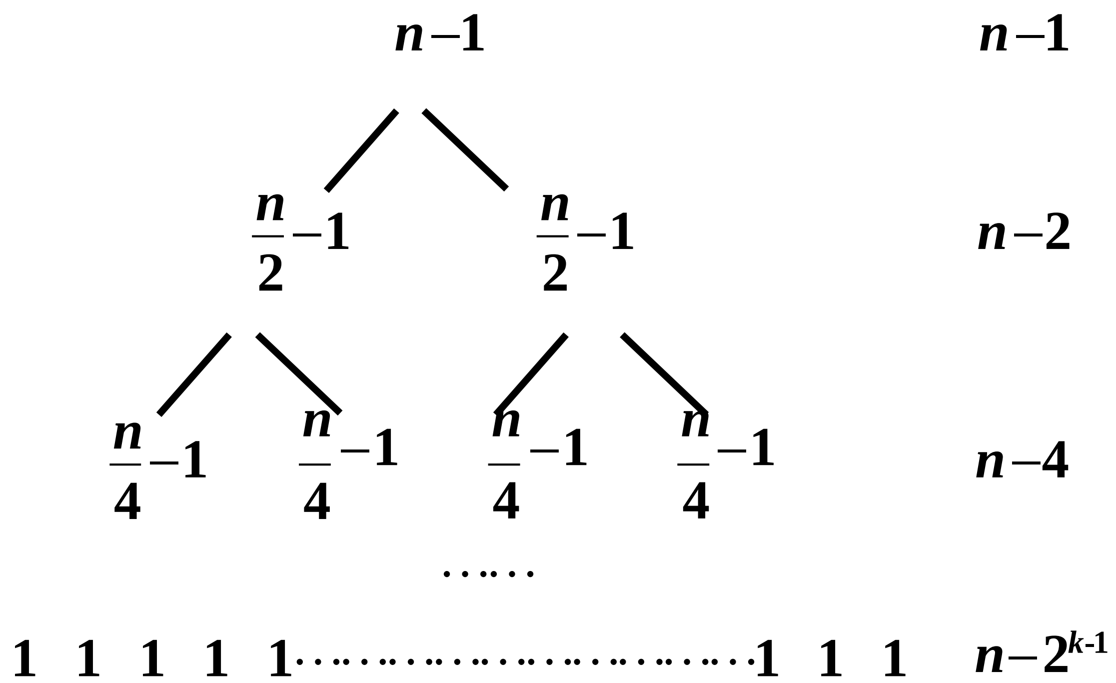
# 递归树生成实例

$$W(n) = 2W(n/2) + n\text{-}1$$



# 递归树


$n-1$
$\frac{n}{2}-1$ $\frac{n}{2}-1$
$\frac{n}{4}-1$ $\frac{n}{4}-1$ $\frac{n}{4}-1$ $\frac{n}{4}-1$
……
1 1 1 1 1……………………………1 1 1
$n-1$
$n-2$
$n-4$
$n-2^{k-1}$

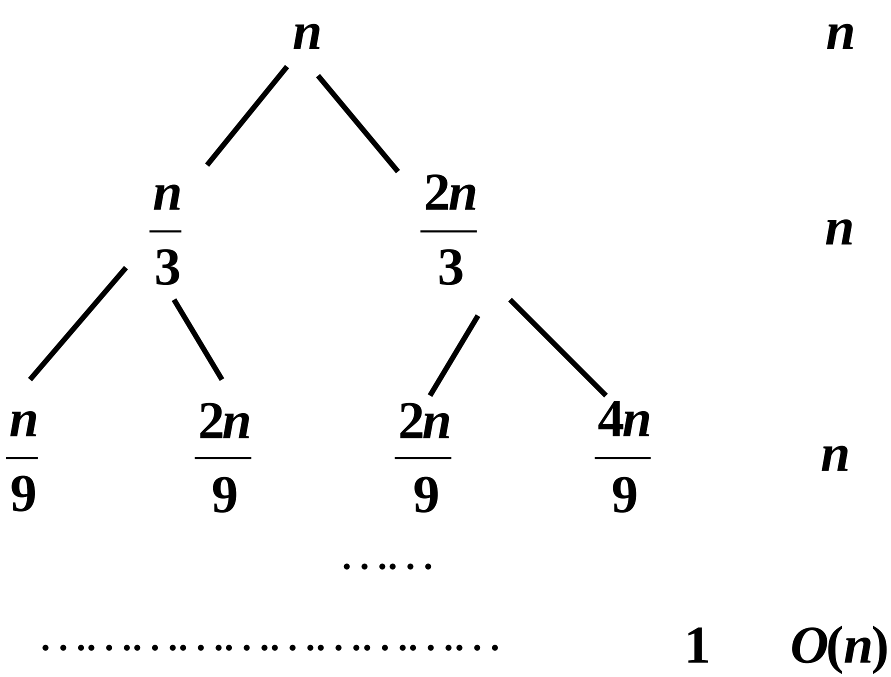
# 对递归树上的量求和

$$W(n) = 2W(n/2) + n - 1,\ n = 2^k,$$
$$W(1) = 0$$

$$W(n) = n - 1 + n - 2 + \ldots + n - 2^{k\text{-}1}$$
$$= kn - (2^k - 1)$$
$$= n\log n - n + 1$$

# 递归树应用实例

求解方程：$T(n)=T(n/3)+T(2n/3)+n$




递归树层数 $k$，每层$O(n)$

$$n(2/3)^k = 1$$

$$\Rightarrow (3/2)^k = n$$

$$\Rightarrow k = O(\log_{3/2} n)$$

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

## 3.5 主定理及其应用

- 求解递推方程
    - $T(n) = a\,T(n/b) + f(n)$
  - $a$： 归约后的子问题个数
  - $n/b$： 归约后子问题的规模
  - $f(n)$： 归约过程及组合子问题的解的工作量
- 二分检索： $T(n) = T(n/2)+1$
- 二分归并排序： $T(n) =2T(n/2)+n-1$

# 主定理

- 定理：设$a \geq 1$, $b>1$为常数, $f(n)$为函数, $T(n)$为非负整数，且 $T(n)=aT(n/b)+f(n)$， 则

  *Case1*. 若 $f(n)=O(n^{\log_b a-\varepsilon})$, $\varepsilon>0$, 那么

  $$\boldsymbol{T(n)=\Theta(n^{\log_b a})}$$

  *Case2*. 若 $f(n)=\Theta(n^{\log_b a})$, 那么

  $$\boldsymbol{T(n)=\Theta(n^{\log_b a}\log n)}$$

  *Case3*. 若 $f(n)=\Omega(n^{\log_b a+\varepsilon})$, $\varepsilon>0$, 且对于某个常数 $c<1$和充分大的 $n$ 有$af(n/b)\leq cf(n)$, 那么

  $$\boldsymbol{T(n)=\Theta(f(n))}$$

# 求解递推方程（1）

例**1** 求解递推方程

$$T(n) = 9T(n/3) + n$$

解 上述递推方程中的

$a = 9$， $b = 3$， $f(n) = n$

$n^{\log_3 9} = n^2$， $f(n) = O(n^{\log_3 9-1})$

相当于主定理的**case1**，其中**$\varepsilon$=1.**

根据定理得到 $T(n) = \Theta(n^2)$

# 求解递推方程（2）

例2 求解递推方程

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

解 上述递推方程中的

$$a = 1,\ b = 3/2,\quad f(n) = 1,$$

$$n^{\log_{3/2} 1} = n^0 = 1$$

相当于主定理的Case2 .

根据定理得到 $T(n) = \Theta(\log n)$

# 求解递推方程（3）

例3 求解递推方程

$$T(n) = 3T(n/4) + n\log n$$

解 上述递推方程中的

$$a = 3, \quad b = 4, \quad f(n) = n\log n$$

$$n\log n = \Omega(n^{\log_4 3+\varepsilon}) \approx \Omega(n^{0.793+\varepsilon})$$

取 $\varepsilon = 0.2$ 即可.

# 不能使用主定理的例子

例4 求解 $T(n)=2T(n/2)+n\log n$

解 $a=b=2$，$n^{\log_b a}=n$，$f(n)=n\log n$

不存在 $\varepsilon>0$ 使下式成立

$$n\log n=\Omega(n^{1+\varepsilon})$$

不存在 $c<1$ 使 $af(n/b)\leq cf(n)$ 对所有 充分大的 $n$ 成立

$2(n/2)\log(n/2)=n(\log n-1)\leq cn\log n$

# 递归树求解

$$n\log n$$

$$\frac{n(\log n-1)}{2} \qquad \frac{n(\log n-1)}{2}$$

$$\frac{n(\log n-2)}{4} \qquad \frac{n(\log n-2)}{4} \qquad \frac{n(\log n-2)}{4} \qquad \frac{n(\log n-2)}{4}$$

$$\cdots\cdots\cdots$$

$n\log n$

$n(\log n-1)$

$n(\log n-2)$

:

$n(\log n-k+1)$

$$\begin{aligned}
&T(n)\\
&= n\log n + n(\log n-1) + n(\log n-2)\\
&\quad + \ldots + n(\log n-k+1)\\
&= (n\log n)\log n - n(1+2+\ldots+k-1)\\
&= n\log^2 n - nk(k-1)/2 = O(n\log^2 n)
\end{aligned}$$

- 3.1 递推方程与算法分析
- 3.2 迭代法求解递推方程
- 3.3 差消法化简递推方程
- 3.4 递归树
- 3.5 主定理及其应用