算法设计与分析

第2章 递归与分治策略

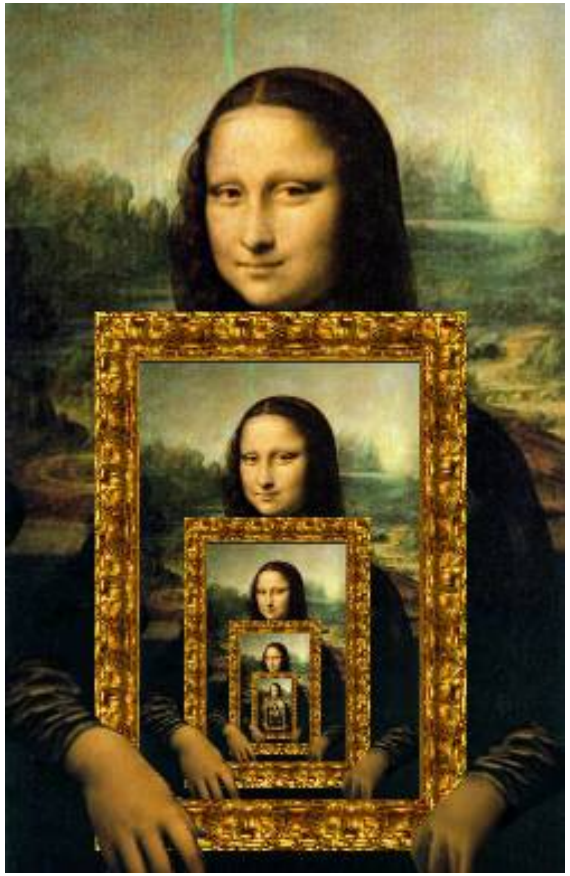
(1)之

基本概念，基本思想，大整数乘法

引子

- “你站在桥上看风景，看风景的人在楼上看你，明月装饰了你的窗子，你装饰了别人的梦。”

  —卞之琳《断章》



- 分形
  - 自相似的递归结构

引子

- 称球游戏：给定n个球，其中1个球为次品。次品从外表上看与正常球一样，但重量有区别。它可能比正常球重，也可能比正常球轻。现在给你一个天平，我们的问题是，需要称几次才能将次品甄别出来？

学习要点

- 掌握设计有效算法的分治策略
- 理解递归的概念
- 递归表达式的求解方法
- 通过范例学习分治策略设计技巧
  - 大整数乘法
  - Strassen矩阵乘法
  - 二分搜索技术
  - 合并排序和快速排序
  - 线性时间选择
  - 最接近点对问题

引言

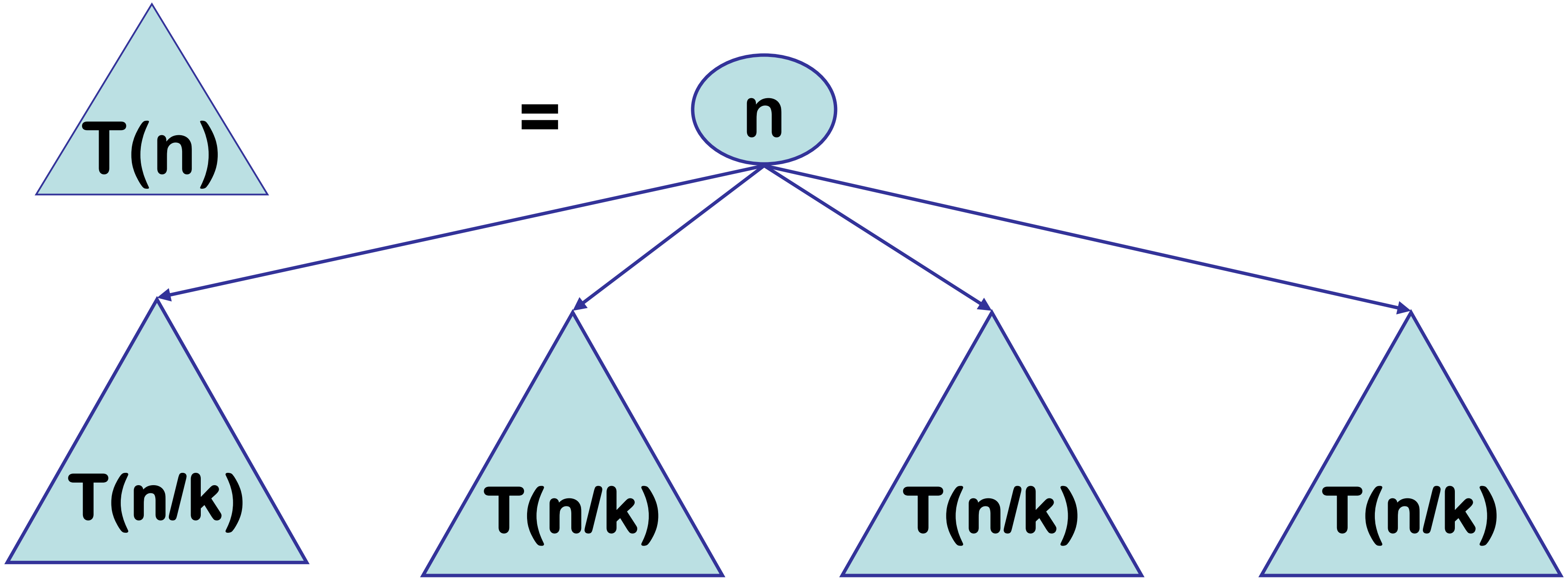
- 设计算法有许多方法
- 排序问题
  - 冒泡排序Bubble sort: bubbling
  - 插入排序Insertion sort: incremental approach（增量靠近)
  - 合并排序Merge sort: divide-and conquer（分而治之)
  - 快速排序Quick sort: location（元素定位)
  - ……
  - 分治算法的最坏运行时间远比插入排序还少

引言

- 分而治之
  - 清·俞樾《群经平议·周官二》“巫马下士二人医四人”：“凡邦之有疾病者，疕疡者造焉，则使医分而治之，是亦不自医也。”
- 各个击破
  - 集中红军相机应付当前之敌，反对分兵，避免被敌人各个击破。（毛泽东《中国的红色政权为什么能够存在》）
- 老子《道德经》
  - 天下大事,必做于细
  - 天下难事,必做于易

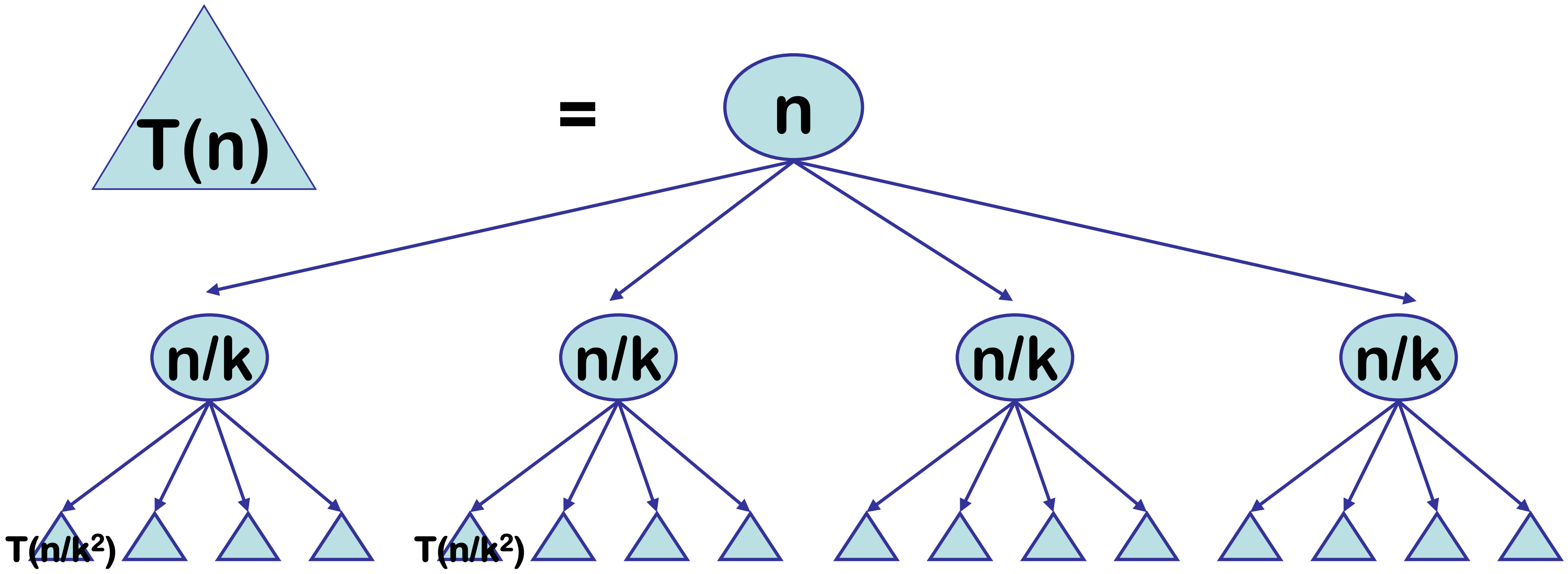
分治法总体思想

- 将一个难以解决的大问题分割为k个子问题



分治法基本思想

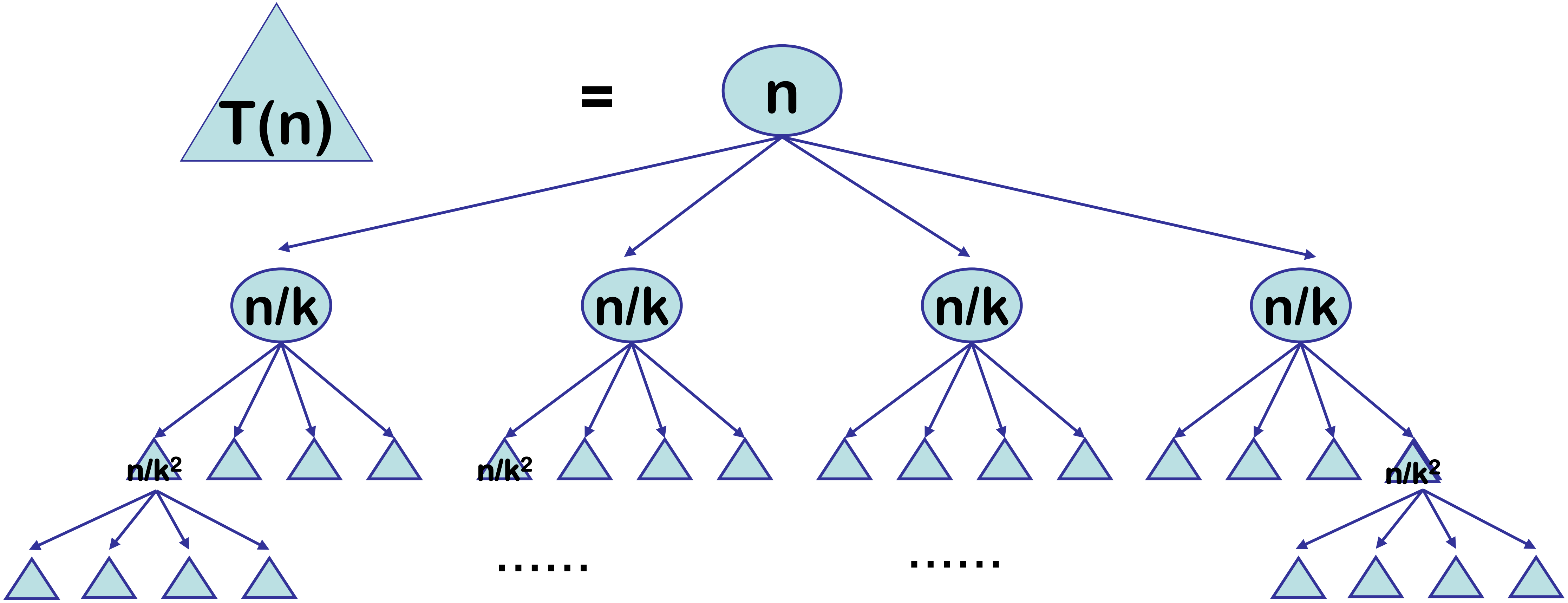
- 对这k个子问题分别求解。如果子问题的规模仍然不够小，则再划分为k个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止。



8

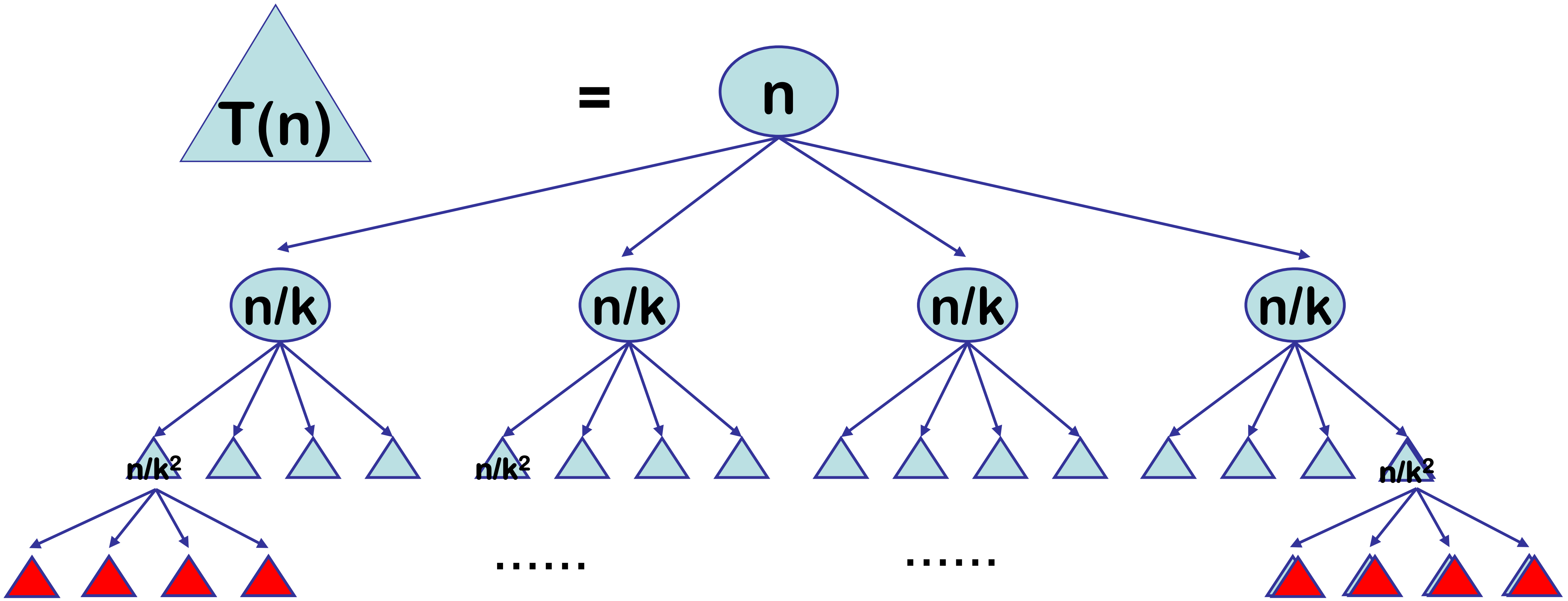
分治法基本思想

- 对这k个子问题分别求解。如果子问题的规模仍然不够小，则再划分为k个子问题，如此递归的进行下去，直到问题规模**足够小**，很容易求出其解为止。



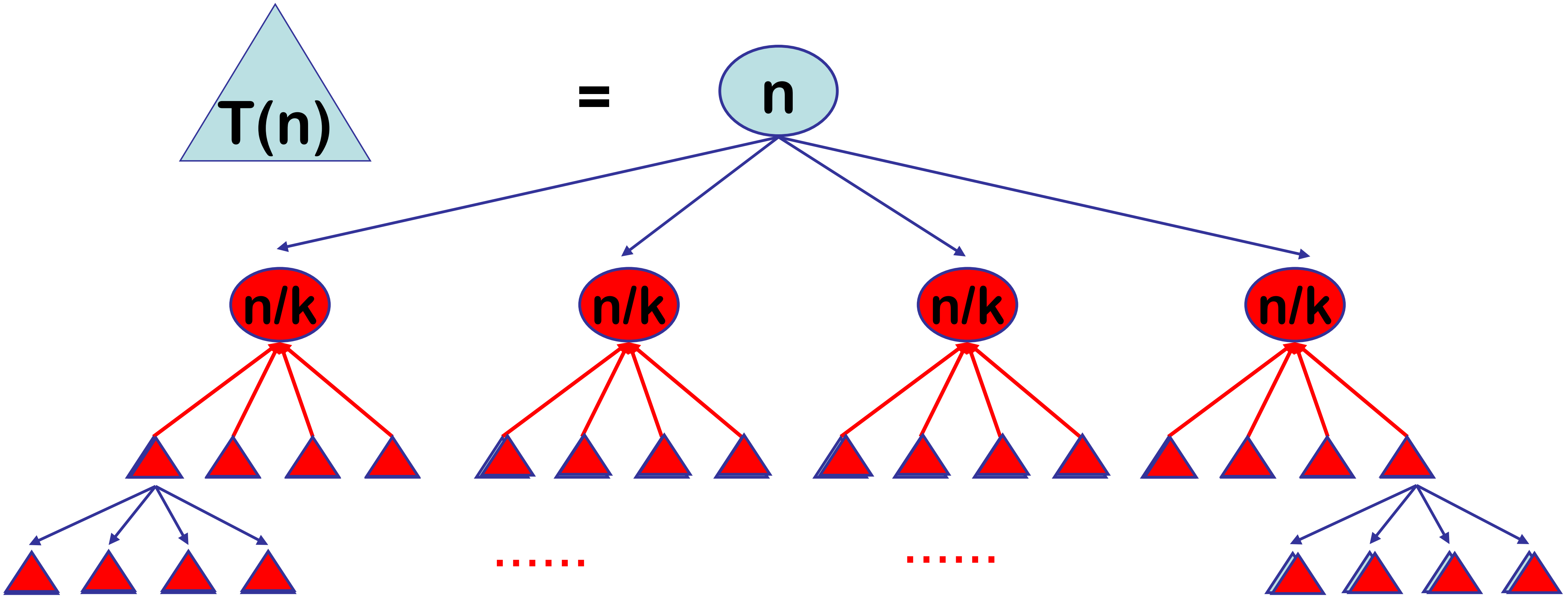
分治法基本思想

- 将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解



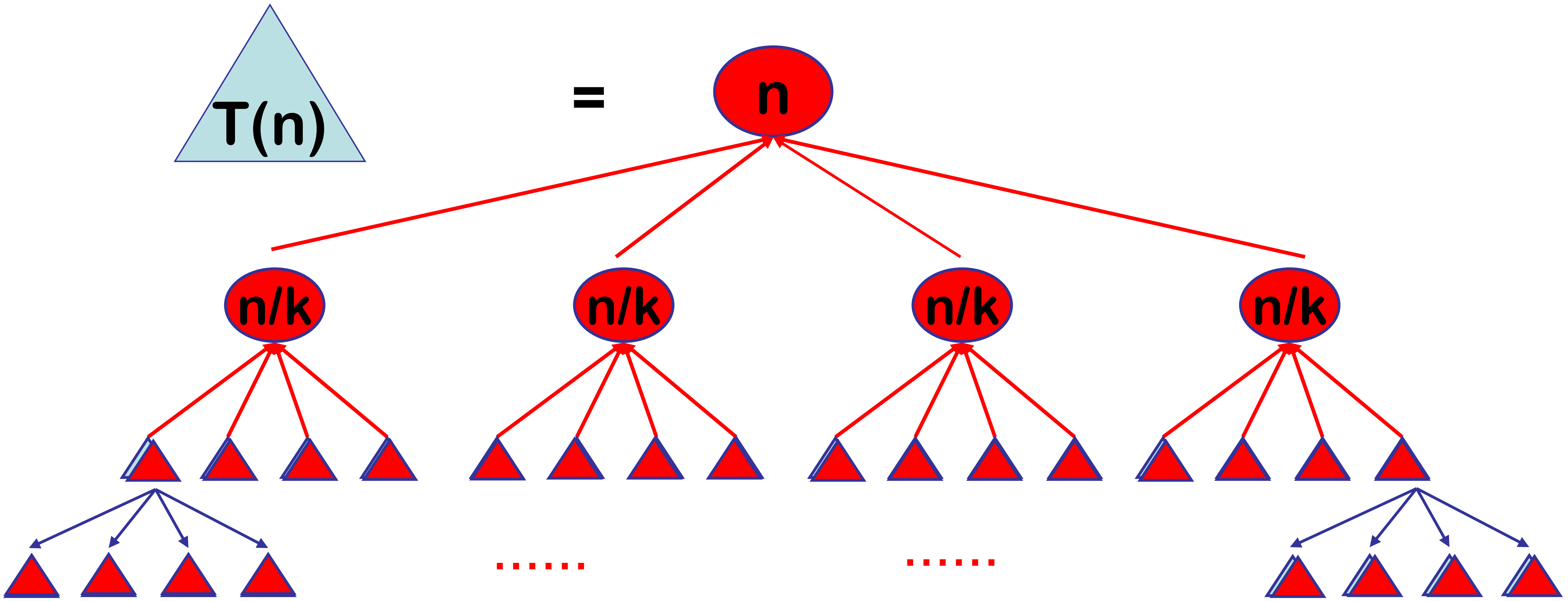
分治法基本思想

- 将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。



分治法基本思想

- 将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。

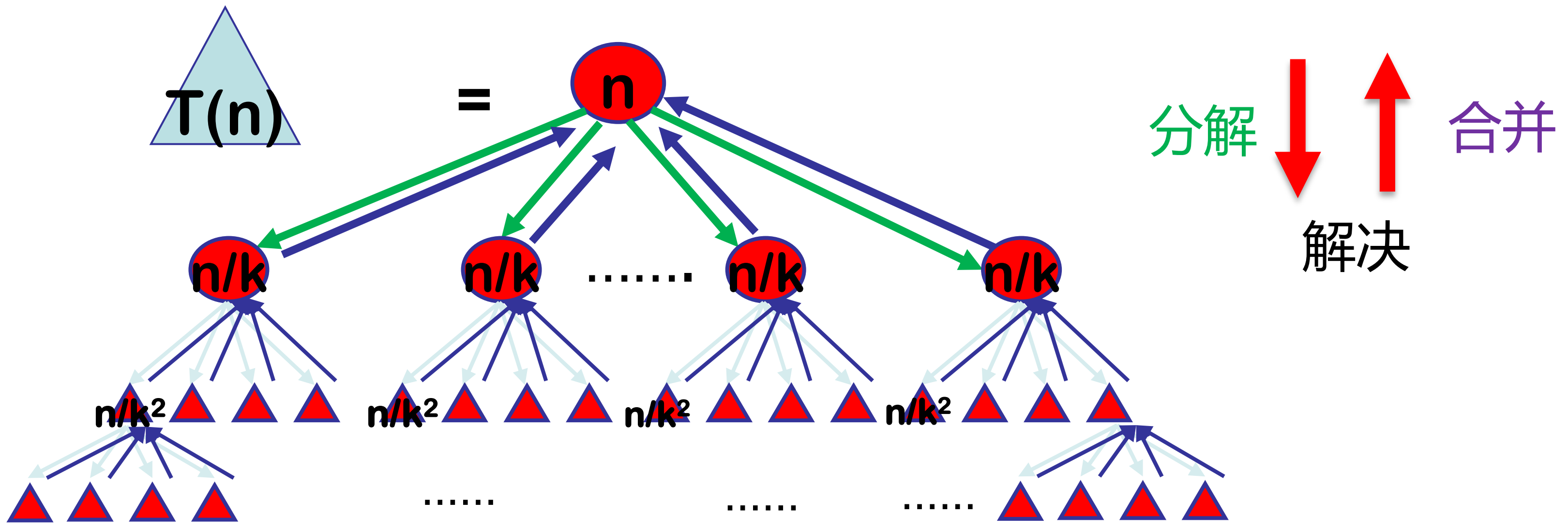


分治法基本思想

- 分治法的设计思想是:
  - 将一个难以直接解决的大问题，**分**割成一些规模较小的子问题; 这些子问题互相独立且与原问题性质相同
  - 若子问题规模足够小，则直接求**解**；否则**递归求解**各个子问题
  - 将各个子问题的解**合**并为一个更大规模的问题的解.

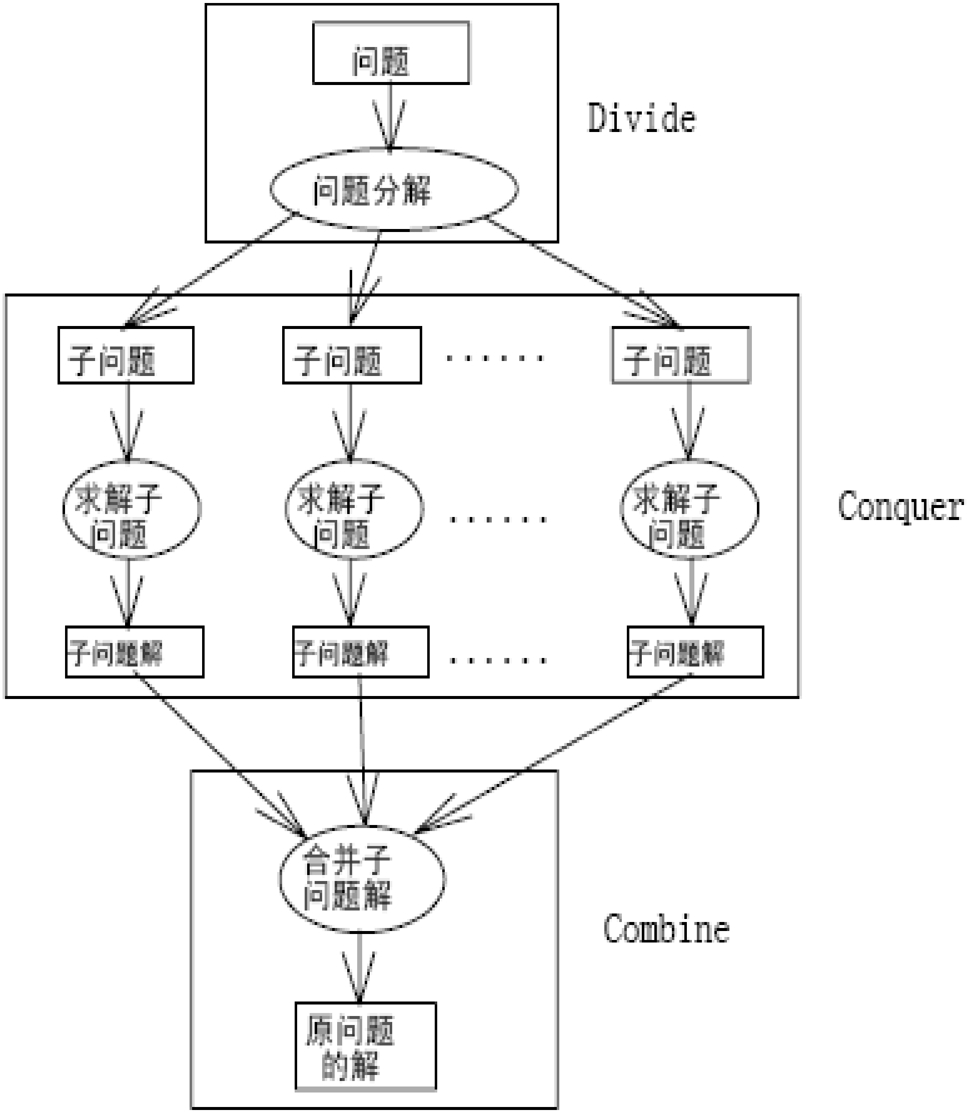
分治法基本思想

- 分治模式在每一层递归上都有分解、解决和合并三个步骤。



分治法基本思想

- Divide=整个问题划分为多个子问题
- Conquer=求解每个子问题(递归调用正在设计的算法)
- Combine=合并子问题的解,形成原始问题的解。



大整数的乘法

- 问题:　X和Y是两个n位的二进制整数，分别表示为$X=x_{n-1}x_{n-2}...x_0$, $Y=y_{n-1}y_{n-2}...y_0$，其中$0 \le x_i, y_j \le 1$ $(i, j=0,1,...n-1)$，设计一个算法求$X\times Y$，并分析其计算复杂度
  - 输入：X，Y
  - 输出：$X\times Y$
- 说明：
  - “**基本操作**”　约定为两个个位整数相乘$x_i\times y_j$，以及两个整数相加。

大整数的乘法

- 问题： n位数 X 和Y, 求(X × Y) = ?
  - Naive (原始的) pencil-and-paper algorithm

$$
\begin{array}{r}
12 \\
\times 23 \\
\hline
6 \\
3 \\
\hdashline
4 \\
2 \\
\hline
276
\end{array}
\qquad\qquad
\begin{array}{r}
31415962 \\
\times 27182818 \\
\hline
251327696 \\
31415962 \\
251327696 \\
62831924 \\
251327696 \\
31415962 \\
219911734 \\
62831924 \\
\hline
853974377340916
\end{array}
$$

  - 复杂度分析: n^2 乘法 and 最多n^2-1 加法. So, $T(n)=O(n^2)$.

大整数的乘法

- 小学的方法：$O(n^2)$ ✘效率太低
- 分治法:

X和Y是两个n位的二进制整数，都**分为2段**，每段为n/2位

X =	A	B
Y =	C	D

X = 10110110

A = 1011 B = 0110

$$X = A\,2^{n/2} + B \qquad Y = C\,2^{n/2} + D$$

$$XY = AC\,2^{n} + (AD+BC)\,2^{n/2} + BD$$

大整数的乘法

- 设计：　　　$XY = \mathbf{AC}\ 2^n + (\mathbf{AD}+\mathbf{BC})\ 2^{n/2} + \mathbf{BD}$
  - 分：
    - 原问题X*Y两个n位数相乘
    - 分解为 AC, AD, BC, BD 这样4个(n/2)位整数相乘的子问题
  - 解：
    - 递归求解AC, AD, BC, BD
    - 若为1位数时直接相乘
  - 合：
    - O(n)次（加法和移位)

大整数的乘法

- 分治法:

$$X = A\,2^{n/2} + B \qquad Y = C\,2^{n/2} + D$$

$$XY = AC\,2^{n} + (AD+BC)\,2^{n/2} + BD$$

复杂度分析

$$T(n) = \begin{cases} O(1) & n = 1 \\ 4T(n/2) + O(n) & n > 1 \end{cases}$$

$T(n)=O(n^2)$ ✘**没有改进**

分治法没有效果吗？

大整数的乘法

- 问题：请设计一个**有效的**算法，可以进行两个n位大整数的乘法运算
  - ◆小学的方法：$O(n^2)$　　✘效率太低
  - ◆分治法:

$$XY = AC\ 2^n + (AD+BC)\ 2^{n/2} + BD$$

$4*(n/2*n/2)=n^2$

$3*(n/2*n/2)=3/4*n^2$

为了降低时间复杂度，必须减少乘法的次数。

1. $XY = AC\ 2^n + ((A-B)(D-C)+AC+BD)\ 2^{n/2} + BD$
2. $XY = AC\ 2^n + ((A+B)(D+C)-AC-BD)\ 2^{n/2} + BD$

大整数的乘法

- 为了降低时间复杂度，必须减少乘法的次数。

$$XY = \mathbf{AC}\ 2^n + (\mathbf{AD+BC})\ 2^{n/2} + \mathbf{BD}$$

$$\rightarrow XY = \mathbf{AC}\ 2^n + ((\mathbf{A-B})(\mathbf{D-C})+\mathbf{AC}+\mathbf{BD})\ 2^{n/2} + \mathbf{BD}$$

$$4*(n/2*n/2)=n^2$$

$$\downarrow$$

$$3*(n/2*n/2)=\frac{3}{4}n^2$$

- 改进的设计：
  - 分：n位整数乘法分解为AC, (A-B)(D-C),BD 3个n/2位整数乘法的子问题
  - 解：递归求解AC, (A-B)(D-C) , BD
    若为1位数时直接相乘
  - 合：O(n)次 (加法, 移位)

大整数的乘法

- 如果一件事的成功率是1%，那么反复尝试100次，至少成功1次的概率大约是多少？

  A 10%　B 23%　C 38%　D 63%　E 1%

- 奇迹在坚持中
  - 一件事若反复尝试，它的成功率竟然由1%奇迹般地上升到63%

大整数的乘法

- 减少乘法的次数后： $n^2 \rightarrow \frac{3}{4}n^2$
- 子问题个数： 4个→ 3个

$1000^2 \quad =1,000,000$
$1000^{1.59} = \quad 58,884$

复杂度分析

$$T(n)=\begin{cases} O(1) & n=1 \\ 3T(n/2)+O(n) & n>1 \end{cases}$$

$T(n)=O(n^{\log 3}) =O(n^{1.59})$✔ **较大的改进**

1. $XY = AC\ 2^n + ((A-B)(D-C)+AC+BD)\ 2^{n/2} + BD$ ✓
2. $XY = AC\ 2^n + ((A+B)(D+C)-AC-BD)\ 2^{n/2} + BD$

细节问题：两个XY的复杂度都是$O(n^{\log 3})$，但考虑到a+b,d+c可能得到m+1位的结果，使问题的规模变大，故不选择第2种方案。

大整数的乘法

- 算法： MULT(X, Y, n)
- 输入： 2个n位的二进制整数X,Y
- 输出：X和Y的乘积XY

if n=l then
　　return(X*Y);
else { A ← X的左边n/2位;
　　B ← X的右边n/2位;
　　C ← Y的左边n/2位;
　　D ← Y的右边n/2位;
　　m1 ← MULT(A， C， n/2);
　　m2 ← MULT(A-B,D-C， n/2);
　　m3 ← MULT(B， D， n/2);
　　S ← $m1*2^{n}+(m1+m2+m3)*2^{n/2}+m3$;
　　return (S)
}

分解

求解三个子问题

合并

大整数的乘法

- 二进制大整数乘法算法同样可应用于十进制大整数的乘法

$$1364\times5261=?$$

- 十进制分治算法
  - Let $X = a\ b$, $Y = c\ d$

    where a, b, c and d are n/2 digit numbers,

    e.g. $1364=13\times10^2+64$.
  - Let $m= n/2$. Then

    $XY = (10^m a+b)(10^m c+d)$
  - $=10^{2m}ac+10^m(bc+ad)+bd$

大整数的乘法

- 分治法
  - Let X = a b , Y = c d
  - then XY = $(10^m a+b)(10^m c+d) = 10^{2m}ac+10^m(bc+ad)+bd$

```
Multiply(X; Y; n){
   if n = 1
      return X×Y;
   else
      m = ⌈n/2⌉;
      a = ⌊X/10^m⌋; b=X mod 10^m;
      c = ⌊Y/10^m⌋; d=Y mod 10^m;
      e = Multiply(a; c; m);
      f = Multiply(b; d; m) ;
      g = Multiply(b; c; m) ;
      h = Multiply(a; d; m) ;
      return 10^(2m)e + 10^m(g + h) + f;
}
```

复杂度分析:
基本操作：乘法
$T(1)=1$,
$T(n)=4T(\lceil n/2 \rceil)+O(n)$.
应用主定理法有：$T(n)= O(n^2)$.

大整数的乘法

- 分治法
  - Let X = a b , Y = c d
  - then XY = $(10^m a+b)(10^m c+d) = 10^{2m}ac+10^m(bc+ad)+bd$
  - Note that bc + ad = ac + bd – (a – b)(c – d). So, we have

```
FastMultiply(X; Y; n){
    if n = 1
        return X×Y;
    Else{
        m = ⌈n/2⌉;
        a = ⌊X/10^m⌋; b=X mod 10^m;
        c = ⌊Y/10^m⌋; d=Y mod 10^m;
        e = FastMultiply(a; c; m);
        f = FastMultiply(b; d; m);
        g = FastMultiply(a–b; c–d; m);
        return 10^{2m}e + 10^m(e + f – g ) + f;
    }
}
```

复杂度分析:

$T(1)=1$,

$T(n)=3T(\lceil n/2 \rceil)+O(n)$.

Applying Master Theorem, we have

$$T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

大整数的乘法

- 小学的方法：$O(n^2)$ ✗效率太低
- 分治法: $O(n^{1.59})$ ✓较大的改进
- 更快的方法??

➢如果将大整数分成更多段，用更复杂的方式把它们组合起来，将有可能得到更优的算法。

➢最终，这个思想导致了**快速傅利叶变换**(Fast Fourier Transform)的产生。该方法也可以看作是一个复杂的分治算法。

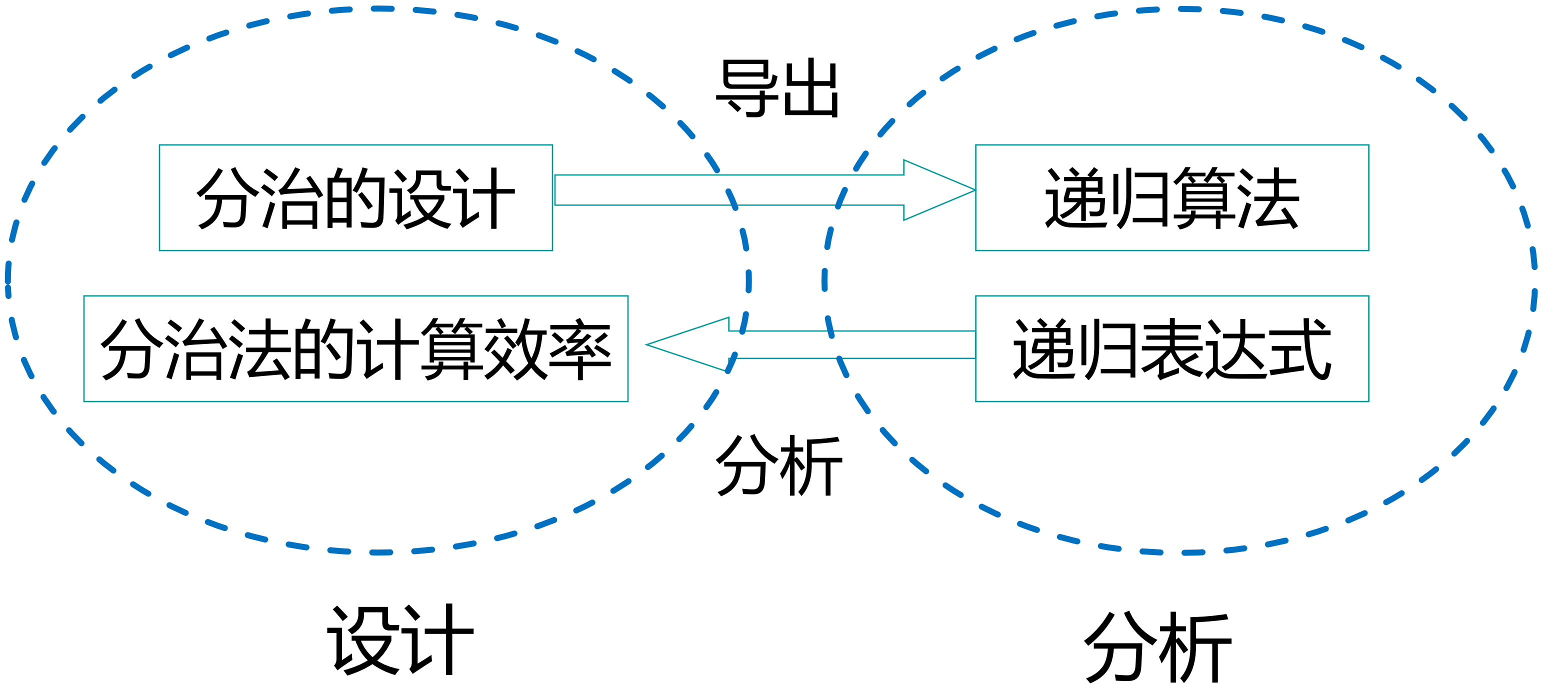
任何连续测量的时序或信号，都可以表示为不同频率的正弦波信号的无限叠加

大整数的乘法小结

- 大整数乘法问题
  - 问题描述
  - 蛮力算法： $O(n^2)$
  - 分治算法：通过减少子问题个数， $O(n^{1.59})$
  - 分治算法的设计、伪码、分析
  - 推演：二进制 → 十进制→ FFT

递归的概念

- 直接或间接地调用自身的算法称为**递归算法**。
- 用函数自身给出定义的函数称为**递归函数**。
- 分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。



递归的概念

- 递归是分析算法的一个基本方法
- 算法研究的难点
  - Specify 描述
  - Model 建模
  - Correctness 正确性
    - Verify 验证
    - Proof 证明
  - Complex 复杂度（Efficiency 有效性）
    - Actual computing 实际可计算性

设计

正确性分析

可计算性分析

递归的概念

- 递归表达式是一个满足下列特征的等式或不等式
  - 边界条件
  - 递归函数：
    - **更小**输入下的该函数
- **边界条件与递归方程**是递归式的两个要素，递归式只有具备了这两个要素，才能在有限次计算后得出结果。

(1) $T(n)=\begin{cases}1 & \text{, if } n\text{=}1,\\ T(n-1)+1 & \text{, if } n>1.\end{cases}$

Solution: $T(n)=n.$

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

Solution: $T(n)=n\lg n+n.$

(3) $T(n)=\begin{cases}0 & \text{, if } n\text{=}2,\\ T(\sqrt{n})+1 & \text{, if } n>2.\end{cases}$

Solution: $T(n)=\lg\lg n.$

(4) $T(n)=\begin{cases}1 & \text{, if } n\text{=}1,\\ T(n/3)+T(2n/3)+n, & \text{if } n>1.\end{cases}$

Solution: $T(n)=\Theta(n\lg n)$

递归的概念

- 例1 阶乘函数

阶乘函数可递归地定义为 $n! = \begin{cases} 1 & , n = 0 \\ n(n-1)! & , n > 0 \end{cases}$

边界条件

递归方程

n的阶乘可递归地计算如下：

```
int FACTORIAL(int n){                    T(n)
      if (n == 0) return 1;
      return n*FACTORIAL(n-1);      T(n-1)+1
}
```

- 分析：
  - $T(n) = \begin{cases} 0 & if\ n = 1 \\ T(n-1)+1 & if\ n > 1 \end{cases}$
  - 解为：T(n)=n

递归的概念

- 例2 合并排序

	cost
MERGE-SORT(*A*, *p*, *r*)	$T(n)$
if $p < r$ **Then{**	
$q \leftarrow \lfloor (p + r)/2 \rfloor$	
MERGE-SORT(*A*, *p*, *q*)	$T(n/2)$
MERGE-SORT(*A*, *q*+1, *r*)	$T(n/2)$
MERGE(*A*, *p*, *q*, *r*)}	n

当n=1时
终止排序

- 算法分析：如何从递归算法写出递归表达式并求解？
  - $T(n) = \begin{cases} 1 & ,if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + n & ,if\ n > 1 \end{cases}$
  - 解得：$T(n) = nlogn + n$

递归的概念

- 例3 整数划分问题
  - 将正整数n表示成一系列正整数之和：$n=n_1+n_2+\ldots+n_k$，其中$n_1 \geq n_2 \geq \ldots \geq n_k \geq 1$，$k \geq 1$。正整数n的这种表示称为正整数n的划分。求正整数n的不同划分个数。
- 例如正整数6有如下11种不同的划分：
  - 6；
  - 5+1；
  - 4+2，4+1+1；
  - 3+3，3+2+1，3+1+1+1；
  - 2+2+2，2+2+1+1，2+1+1+1+1；
  - 1+1+1+1+1+1。

递归的概念

- 例3 整数划分问题

- 思路：
  - （1）分治法：怎么分？
  - （2）是否递归？
    - 如果设p(n)为正整数n的划分数，则难以找到递归关系
  - （3）如何转化为递归？
    - 增加一个自变量：将**最大加数**不大于m的划分个数记作q(n,m)。可以建立q(n,m)的递归关系

递归的概念

- 例3 整数划分问题
  - 增加一个自变量：将最大加数不大于m的划分个数记作q(n,m)
- 分析：
- (1) m=1时： $q(n,m)=1$
  - 当最大加数不大于1时，任何正整数n只有一种划分形式，即 $n=\overbrace{1+1+\cdots+1}^{n}$
- (2) n=1时：不论m为多少, 只有一种划分{1} $q(n,m)=1$
- (3) m=n时：根据划分是否包含n分为2种情况 $q(n,m)=1+q(n,n-1)$
  - 包含n的划分: 只有一个{n} $q(n,m)=q(n,n)=1$
  - 不包含n的划分: 划分的最大加数一定比n小，即所有n-1划分 $q(n,m)=q(n,n-1)$
- (4) m>n时： $q(n,m)=q(n,n)$
  - 最大加数实际上不能大于n。例如，q(1,m)=q(1,1)

递归的概念

- 例3 整数划分问题
  - 增加一个自变量：将最大加数不大于m的划分个数记作q(n,m)
- 分析：
- (5) n>m>1 时：分为包含m和不包含m两种情况
  - 划分中包含m： {m, $\{x_1,x_2,...,x_i\}$}, 且$\{x_1,x_2,...,x_i\}$的和为n-m, 也可能包含m，因此是n-m的m划分 → 划分个数为q(n-m, m)
  - 划分中不包含m: 则划分中所有值都比m小，即n的m-1划分→ 划分个数为q(n, m-1)
  - 因此， $q(n,m) = q(n-m,m) + q(n,m-1)$
    - 例如，假设m=4, q(6, 4)=q(2,4)+q(6,3)
    - q(2,4): 整数2的划分数
    - q(6,3): 最大加数小于4的划分数

递归的概念

- 例3 整数划分问题
  - q(n,m)的递归关系为

$$q(n,m)=\begin{cases} 1 & n=1,m=1 \\ q(n,n) & n<m \\ 1+q(n,n-1) & n=m \\ q(n,m-1)+q(n-m,m) & n>m>1 \end{cases}$$

正整数n的划分数p(n)=q(n,n)

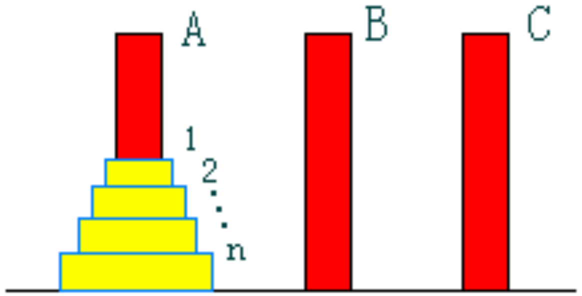
递归的概念

- 例3 整数划分问题
  - 输入：整数n
  - 输出：划分个数p(n)
- 思路：引入新变量转化为一个递归式, 输出变为q(n,m)
- 设计：
  - 分： 引入最大加数值m， p(n)→q(n,n)
  - q(n,n) →q(n,m),m<n，问题规模缩小
  - →q(n,m-1),q(n-m,m)
  - 解： 递归求解q(n,m)； 足够小(n=1,m=1)时直接解
  - 合： 加法合并

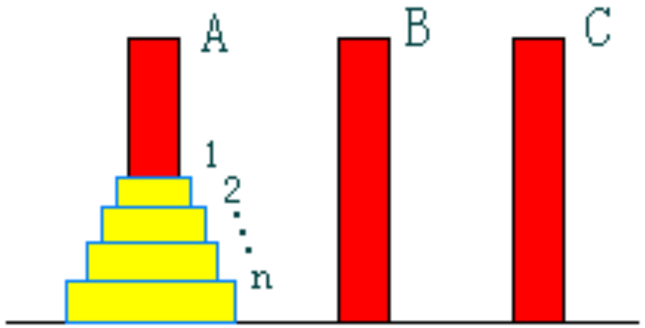
$$q(n,m)=\begin{cases}1 & n=1,m=1\\ q(n,n) & n<m\\ 1+q(n,n-1) & n=m\\ q(n,m-1)+q(n-m,m) & n>m>1\end{cases}$$

递归的概念

- 例4 Hanoi塔问题
  - 设A,B,C是3个塔座。开始时，在塔座A上有一叠共n个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1,2,…,n,
  - 现要求将塔座A上的这一叠圆盘移到塔座B上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：
    - 规则1：每次只能移动1个圆盘；
    - 规则2：任何时刻都不允许将较大的圆盘压在较小的圆盘之上；
    - 规则3：在满足移动规则1和2的前提下，可将圆盘移至A,B,C中任一塔座上。



递归的概念



- 例4 Hanoi塔问题
- 思路

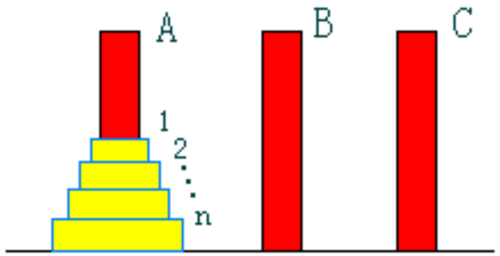
在问题规模较大时，较难找到一般的方法，因此采用递归技术来解决问题

- 当n=1时，只要将编号为1的圆盘从塔座A直接移至塔座B上
- 当n＞1时，需要利用塔座C作为辅助塔座。此时若能设法将n-1个较小的圆盘依照移动规则从塔座A移至塔座c，然后，将剩下的最大圆盘从塔座A移至塔座B，最后，再设法将n-1个较小的圆盘依照移动规则从塔座C移至塔座B

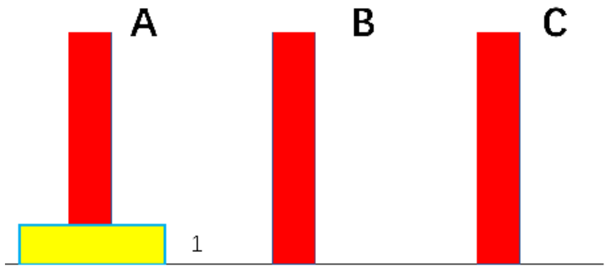
- 由此可见，n个圆盘的移动问题可分为2次n-1个圆盘的移动问题，这又可以递归地用上述方法来做。

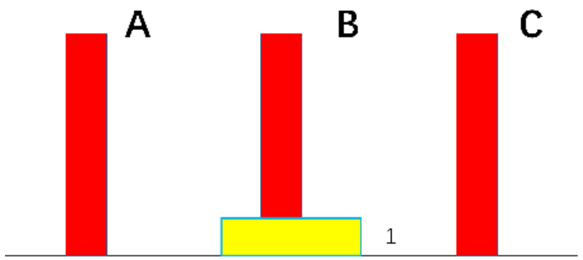
递归的概念

- 例4 Hanoi塔问题



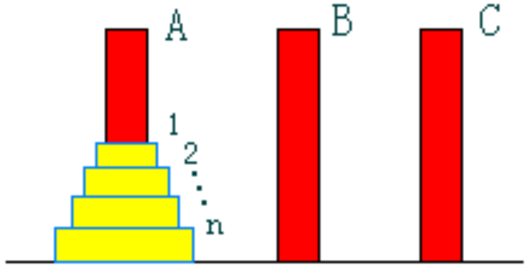
- 思路：
  - 在问题规模较大时，较难找到一般的方法，因此尝试用递归技术来解决
  - 当n=1时，只要将编号为1的圆盘从塔座A直接移至塔座B上



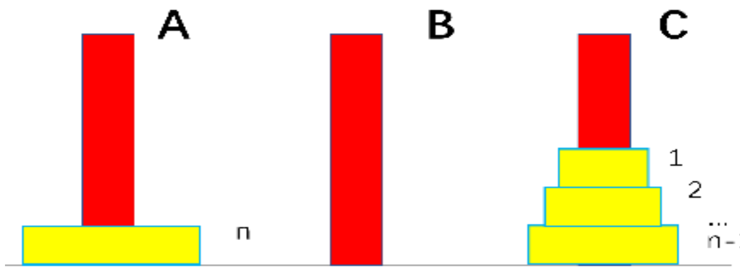


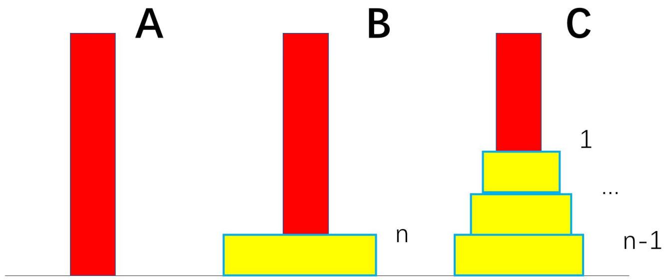
递归的概念

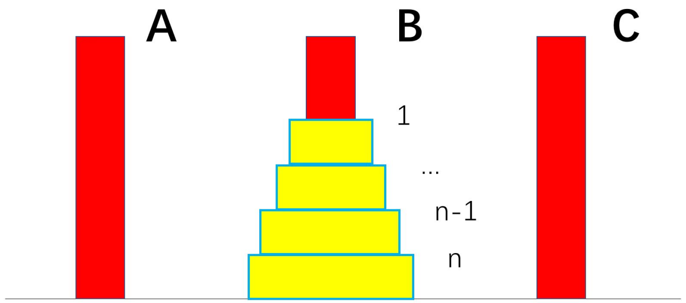
- 例4 Hanoi塔问题



- 思路：
  - 当n＞1时，需要利用塔座C作为辅助塔座。此时若能设法将n-1个较小的圆盘依照移动规则从塔座A移至塔座C，然后，将剩下的最大圆盘从塔座A移至塔座B，最后，再设法将n-1个较小的圆盘依照移动规则从塔座C移至塔座B。

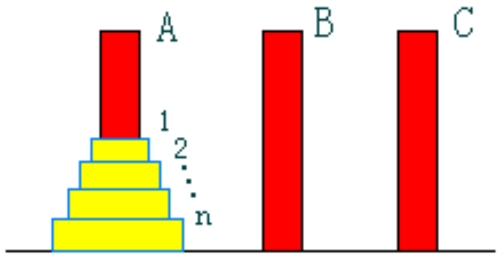






递归的概念

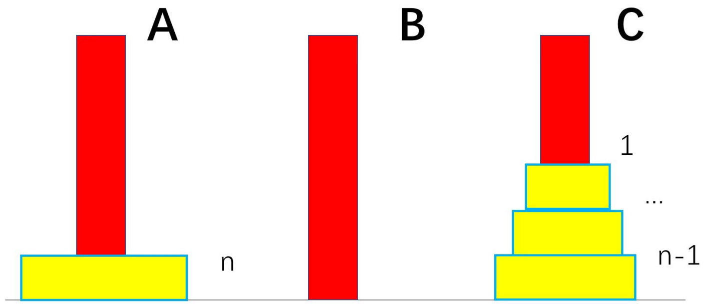
- 例4 Hanoi塔问题



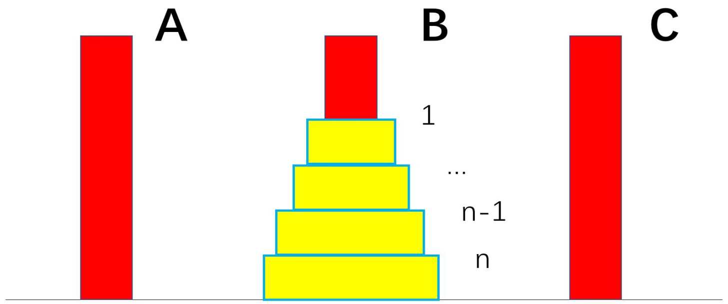
  - 由此可见：

    n个圆盘的移动问题可分为2次n-1个圆盘的移动问题，这又可以递归地用上述方法来实现

    - 从A移动到C

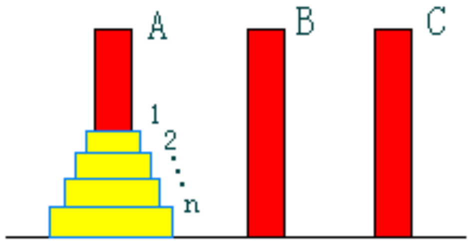


    - 从C移动到B

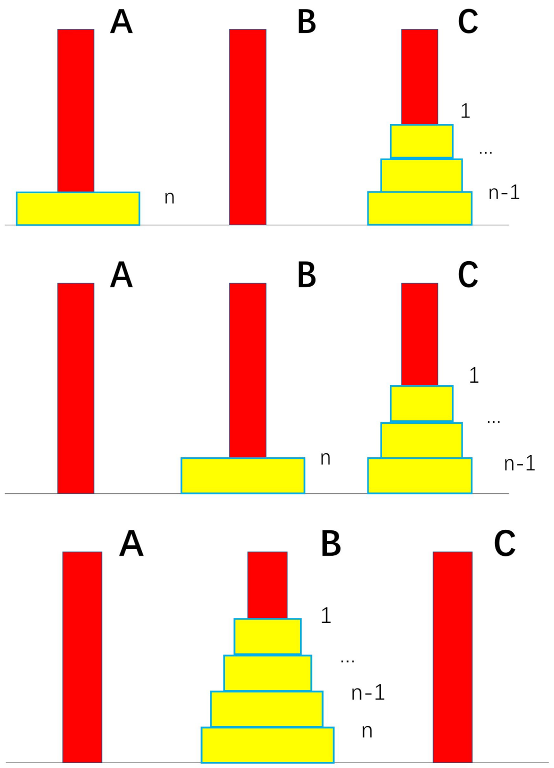


递归的概念

- 例4 Hanoi塔问题



- 设计：
  - 分： 问题规模缩小1， 从n规模变成n-1规模子问题
  - 解： 递归求解**2次** 规模为n-1的子问题；
    若只有一个圆盘则直接移动；
  - 合： 无。



递归的概念

- 例4 Hanoi塔问题

```
void hanoi(int n, int a, int b, int c){
    if (n > 0){
        hanoi(n-1, a, c, b);
        move(a,b);
        hanoi(n-1, c, b, a);
    }
}
```

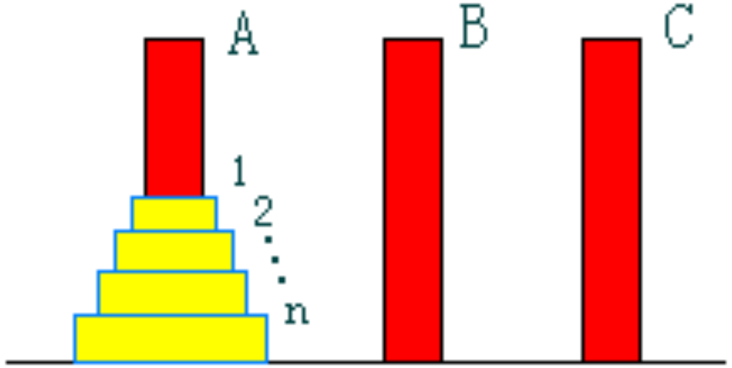
T(n)
T(n-1)
1
T(n-1)

分析

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

$T(1)=1$

解：$T(n)=O(2^n)$



递归

- 优点：
  - 结构清晰，可读性强，
  - 容易用数学归纳法来证明算法的正确性
  - 设计算法、调试程序带来很大方便。
- 缺点：
  - 递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。

递归

- 解决方法：
  - 在递归算法中消除递归调用，使其转化为非递归算法。
  - 采用一个用户定义的栈来模拟系统的递归调用工作栈。该方法通用性强，但本质上还是递归，只不过人工做了本来由编译器做的事情，优化效果不明显。
  - 用递推来实现递归函数。
  - 通过变换能将一些递归转化为尾递归，从而迭代求出结果。
- 后两种方法在时空复杂度上均有较大改善，但其适用范围有限。

小结

- 递归概念
  - 两个基本要素：边界条件，递归函数
  - 递归与分治的关系
- 案例分析
  - 大整数乘法
  - 阶乘、合并排序
    - 递归算法的分析关键得到递归表达式
  - 整数划分问题
    - 转化为递归的问题进行求解
  - 汉诺塔问题

小结

- 重点和难点：
  - 分治法的基本思想
  - 递归概念、两个要素
    - 尤其递归特征不明显的问题怎么进行转换思维

作业

- 1. 请求解递归表达式
  - $T(n)=\begin{cases}0 & 当n=2\\ T(\sqrt{n})+1 & 当n>2\end{cases}.$