算法分析与设计

Analysis and Design of Algorithm

第3次课

课堂练习（1）

练习：按照渐近阶从低到高的顺序排列以下表达式：

$$n!,\ 4n^2,\ \log n,\ 3^n,\ 20n,\ 2,\ n^{2/3}$$

答案： $2, \log n, n^{2/3}, 20n, 4n^2, 3^n, n!$

课堂练习（2）

- 练习：求下列函数的渐近上界
  - $3n^2 + 10n$ $O(n^2)$
  - $n^2/10 + 2^n$ $O(2^n)$
  - $21 + 1/n$ $O(1)$
  - $\log n^3$ $O(\log n)$
  - $10\log 3^n$ $O(n)$

课程回顾

- 算法的重要性
- 算法的概念和特点
- 算法、数据结构、程序之间的关系
- 评估算法的性能的两种方法
- 算法效率和算法复杂性的关系
- 算法复杂性的概念
- 算法复杂性的渐进符号
- 经典的NP完全问题

课程内容

课程内容				层次
NP完全性理论与近似算法				算法高级理论
随机化算法		线性规划与网络流		高级算法
递归分治	动态规划	贪心算法	回溯与分支限界	基础算法
算法分析与问题的计算复杂性				算法基础理论

第二章 递归与分治策略

学习要点

- 理解分治和递归的概念。
- 掌握设计有效算法的分治策略。
- 通过下面的范例学习分治策略设计技巧。
  - 二分搜索技术；
  - 大整数乘法；
  - Strassen矩阵乘法；
  - 棋盘覆盖；
  - 合并排序和快速排序；
  - 最接近点对问题；
  - 循环赛日程表。

分治法的初衷

- 任何一个问题的求解时间都与其规模有关。
- 例子：
  - n个元素排序：

    当n=1，不需计算；

    当n=2，只作一次即可；

    当n=3，三次or两次？ …

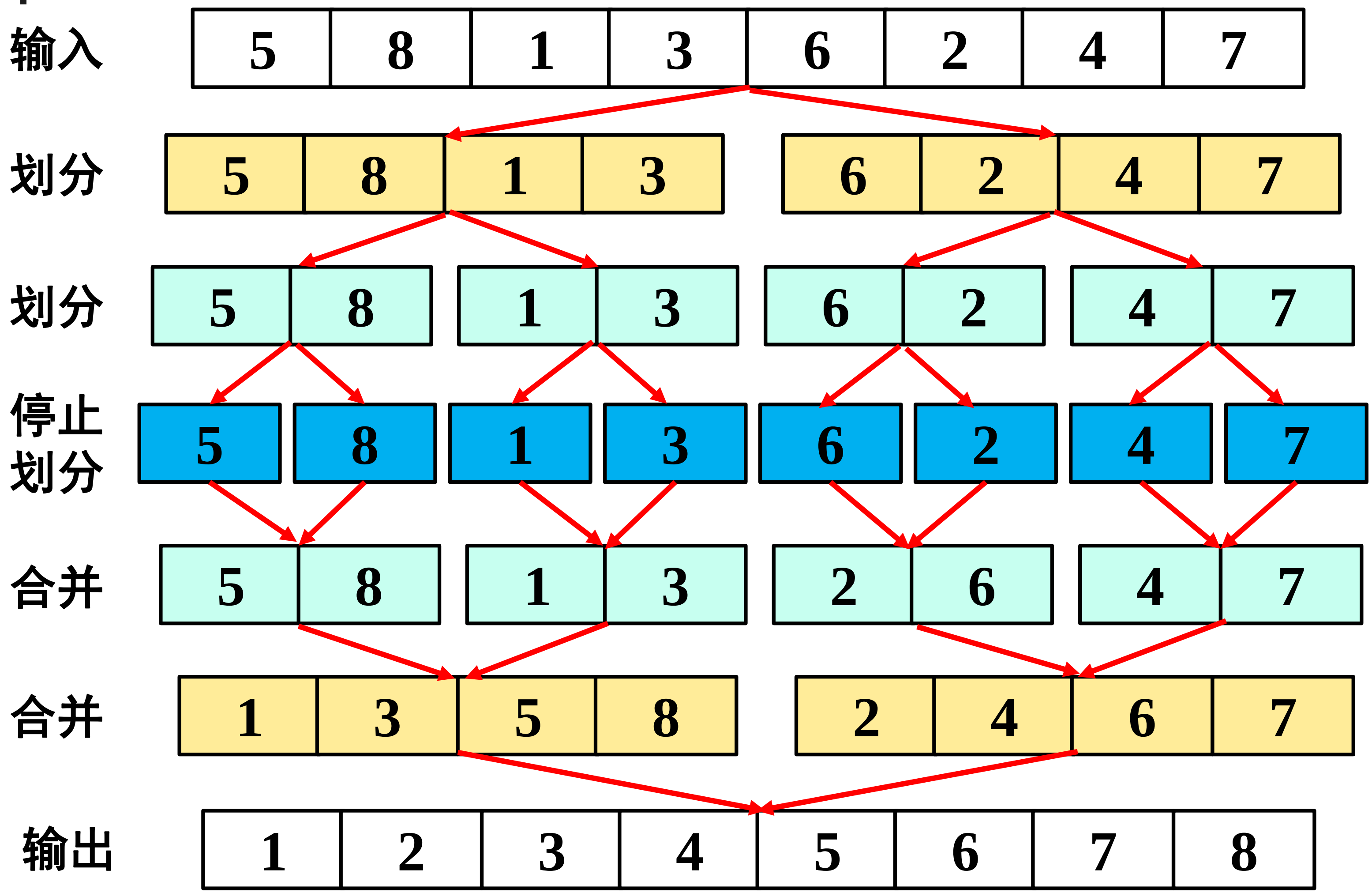
显然，随着n的增加，问题也越难处理。

分治法

- 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相同问题，以便各个击破，分而治之。

- 如果问题可分割成k个子问题，且这些子问题都可解，利用这些子问题可解出原问题的解，此分治法是可行的。

分治法的例子：归并排序



输入
5 8 1 3 6 2 4 7
划分
5 8 1 3
6 2 4 7
划分
5 8
1 3
6 2
4 7
停止划分
5
8
1
3
6
2
4
7
合并
5 8
1 3
2 6
4 7
合并
1 3 5 8
2 4 6 7
输出
1 2 3 4 5 6 7 8

分治与递归

- 分治和递归紧密联系

- 由分治法产生的子问题往往是原问题的较少模式，为递归提供了方便

递归策略

递归

- **定义：直接/间接调用自身的算法称为递归算法。**

```
void mergeSort(int[] A,int[] temp,
int start,int end){
    if (start<end){
        int mid = (start+end)/2;
        //把数组分解为两个子列
        mergeSort(A,temp,start,mid);
        mergeSort(A,temp,mid+1,end);
        //逐级合并两个子列
        Merge(A,temp,start,mid,end);
    }
}
```

递归第一式给出函数的初值，非递归定义。每个递归须有非递归初始值。

第二式是用较小自变量的函数值表示较大自变量

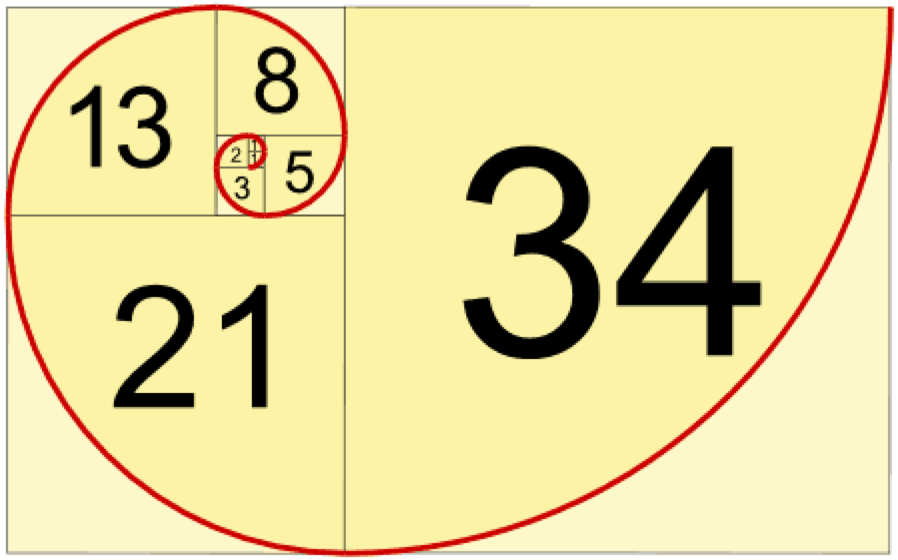
递归

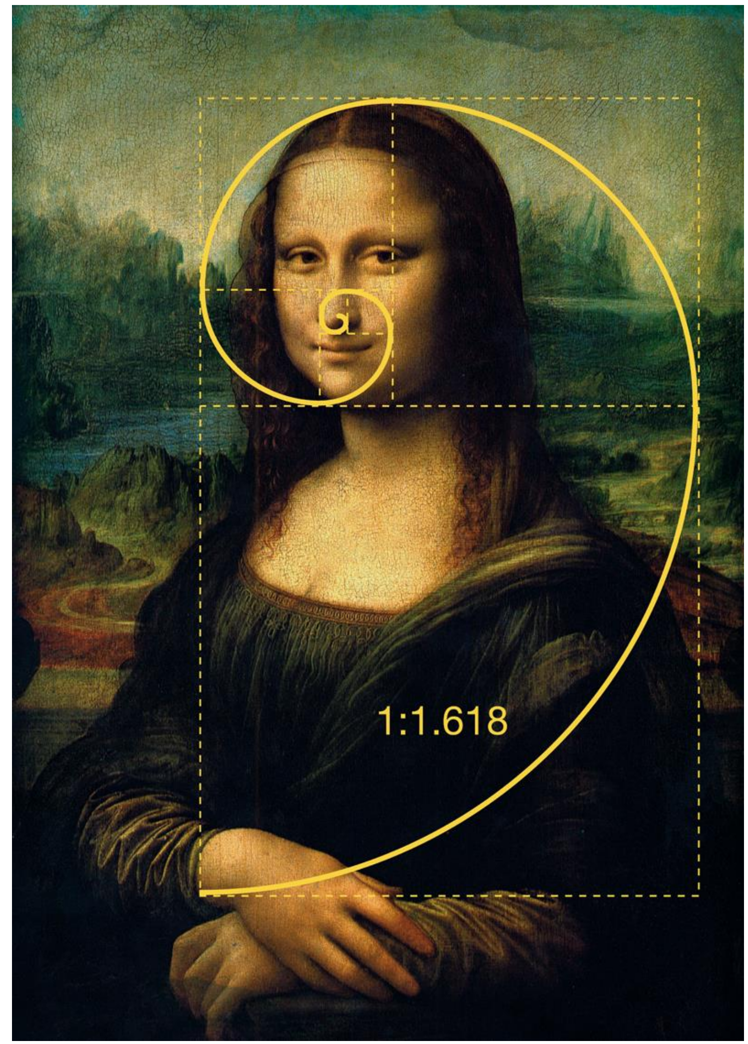
- 前面提到的 **Fibonacci数列**

$$F(n)=\begin{cases}1 & , \quad n=0\\ 1 & , \quad n=1\\ F(n-1)+F(n-2) & , \quad n>1\end{cases}$$

意大利数学家
Fibonacci
1170-1240

Fibonacci数列





Fibonacci数列

Fibonacci数列也可用非递归方式定义：

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

但是并不是所有的递归
表达式都有非递归的形式

双递归函数

- 一个函数与它的一个变量是由函数自身定义。

Ackerman函数

$$A(n,m):\begin{cases} A(1,0)=2 & \\ A(0,m)=1 & m\geq 0 \\ A(n,0)=n+2 & n\geq 2 \\ A(n,m)=A(A(n-1,m),m-1) & n,m\geq 1 \end{cases}$$

1. $m=0,\ A(n,0)=n+2$
2. $m=1,\ A(n,1)=A(A(n\text{-}1,1),0)=A(n\text{-}1,1)+2 \qquad \therefore A(n,1)=2n$
3. $\left.\begin{aligned} &m=2,\ A(n,2)=A(A(n-1,2),1)=2A(n-1,2) \\ &\because A(1,2)=A(A(0,2),1)=A(1,1)=2 \end{aligned}\right\} \Rightarrow A(n,2)=2^n$

双递归函数

4. $m=3$, $A(n,3)=2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ ，其中2的层数n
5. $m=4$, $A(n,4)$的增长速度非常快，以至于没有适当的数学式子来表示这一函数。

- **部分书上减少Ackerman函数的变量，如：**

$$A(n) \stackrel{\Delta}{=} A(n,n)$$

$$A(4)=2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$$ **（其中2的层数为65536）**

这个数非常大，无法用通常的方式来表达它

整数划分问题

- **前面的几个例子中，问题本身都具有比较明显的递归关系，因而容易用递归函数直接求解。**

- **实际情况下，许多问题的递归函数并不能快速找出**
  - 如整数划分问题

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。
  - 问题：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。

整数划分问题

- 仅仅考虑一个自变量：
  - 设$p(n)$为正整数n的划分数，但难以找到递归关系

- 考虑两个自变量：
  - 将最大加数n_1不大于m的划分个数记作$q(n, m)$。

整数划分问题

n=2

- 2
- 1+1

n=3

- 3
- 2+1
- 1+1+1

n=4

- 4
- 3+1
- 2+2
- 2+1+1
- 1+1+1+1

n=5

- 5
- 4+1
- 3+2
- 3+1+1
- 2+2+1
- 2+1+1+1
- 1+1+1+1+1

整数划分问题

- 可以建立$q(n, m)$的如下递归关系。

(1) $q(n,1)=1, n\geqslant 1$;

当最大加数n_1不大于1时，任何正整数n只有一种划分形式

(2) $q(n, m)=q(n, n), m\geqslant n$;

最大加数n_1实际上不能大于n。因此，$q(1, m)=1$。

(3) $q(n, n)=1+q(n, n-1)$;

正整数n的划分由$n_1=n$的划分和$n_1\leq n-1$的划分组成。

(4) $q(n, m)=\underline{q(n, m-1)}+\underline{q(n-m, m)}, n>m>1$;

正整数n的最大加数n_1不大于m的划分由$\boxed{n_1\leqslant m-1 \text{ 的划分}}$和$\boxed{n_1=m\text{的划分}}$组成

整数划分问题

n=2
- 2
- 1+1

n=3
- 3
- 2+1
- 1+1+1

n=4
- 4
- 3+1
- 2+2
- 2+1+1
- 1+1+1+1

n=5
- 5
- 4+1
- 3+2
- 3+1+1
- 2+2+1
- 2+1+1+1
- 1+1+1+1+1

(1) $q(n,1)=1, n\geqslant 1$;

(2) $q(n, m)=q(n, n), m\geqslant n$;

(3) $q(n, n)=1+q(n, n-1)$;

(4) $q(n, m)=q(n, m-1)+q(n-m, m), n>m>1$;

整数划分问题

- 可以建立$q(n, m)$的如下递归关系。

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

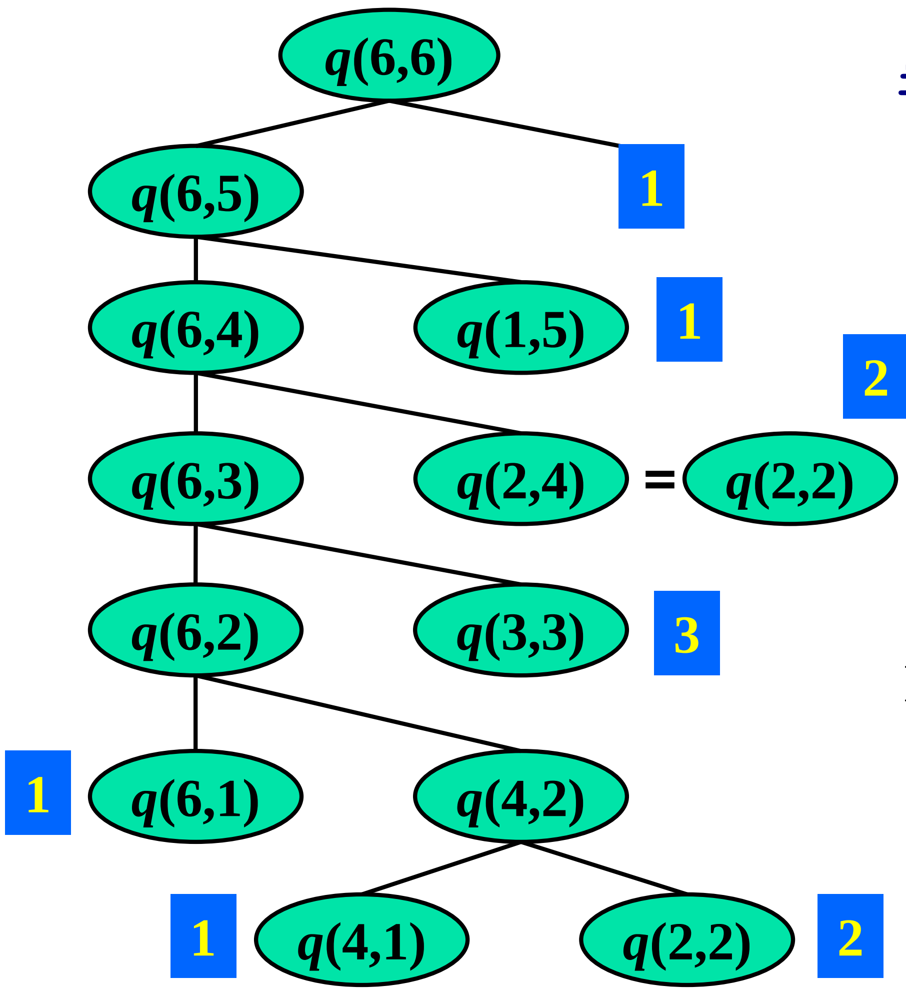
正整数n的划分数$p(n)=q(n, n)$

整数划分问题

- **可得算法伪代码为:**

```
int q(int n, int m){
    if ((n-1)||(m<1))  return 0;
    if ((n==1)||(m==1)) return 1;
    if (n<m) return q(n, n);
    if (n==m) return q(n, m-1)+1;
    return q(n, m-1)+q(n-m, m);
}
```

整数划分问题的递归调用



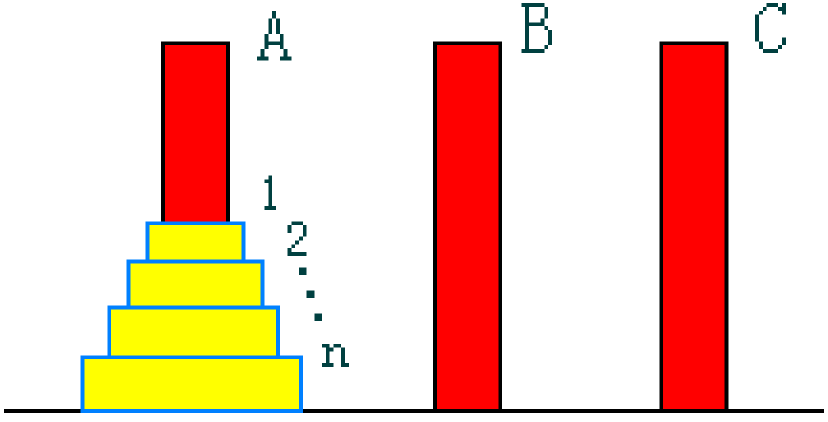

```
int q(int n, int m){
    if ((n-1)||(m<1))
       return 0;
    if ((n==1)||(m==1))
       return 1;
    if (n<m)
       return q(n, n);
    if (n==m)
       return q(n, m-1)+1;
    return q(n, m-1)+q(n-m, m);
}
```

$$q(6,6)=1+1+2+3+1+1+2=11$$

Hanoi塔传说

Hanoi塔问题

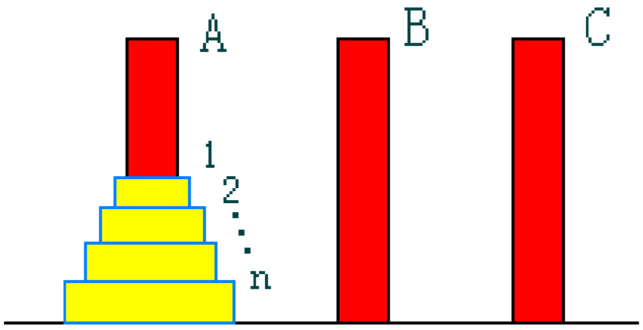
- 设A，B，C是3个塔座。开始时，在塔座A上有一叠共*n*个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1,2,…,*n*,现要求将塔座A上的这一叠圆盘移到塔座B上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：
  - **规则1：**每次只能移动1个圆盘；
  - **规则2：**任何时刻都不允许将较大的圆盘压在较小的圆盘之上；
  - **规则3：**满足规则1和2的前提下，可将圆盘移至A,B,C任一塔座上。



递归算法

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

```
if n = 1 then move (A, C)
else Hanoi(n-1, A, C, B)
        move (A, C)
        Hanoi(n-1, B, A, C)
```

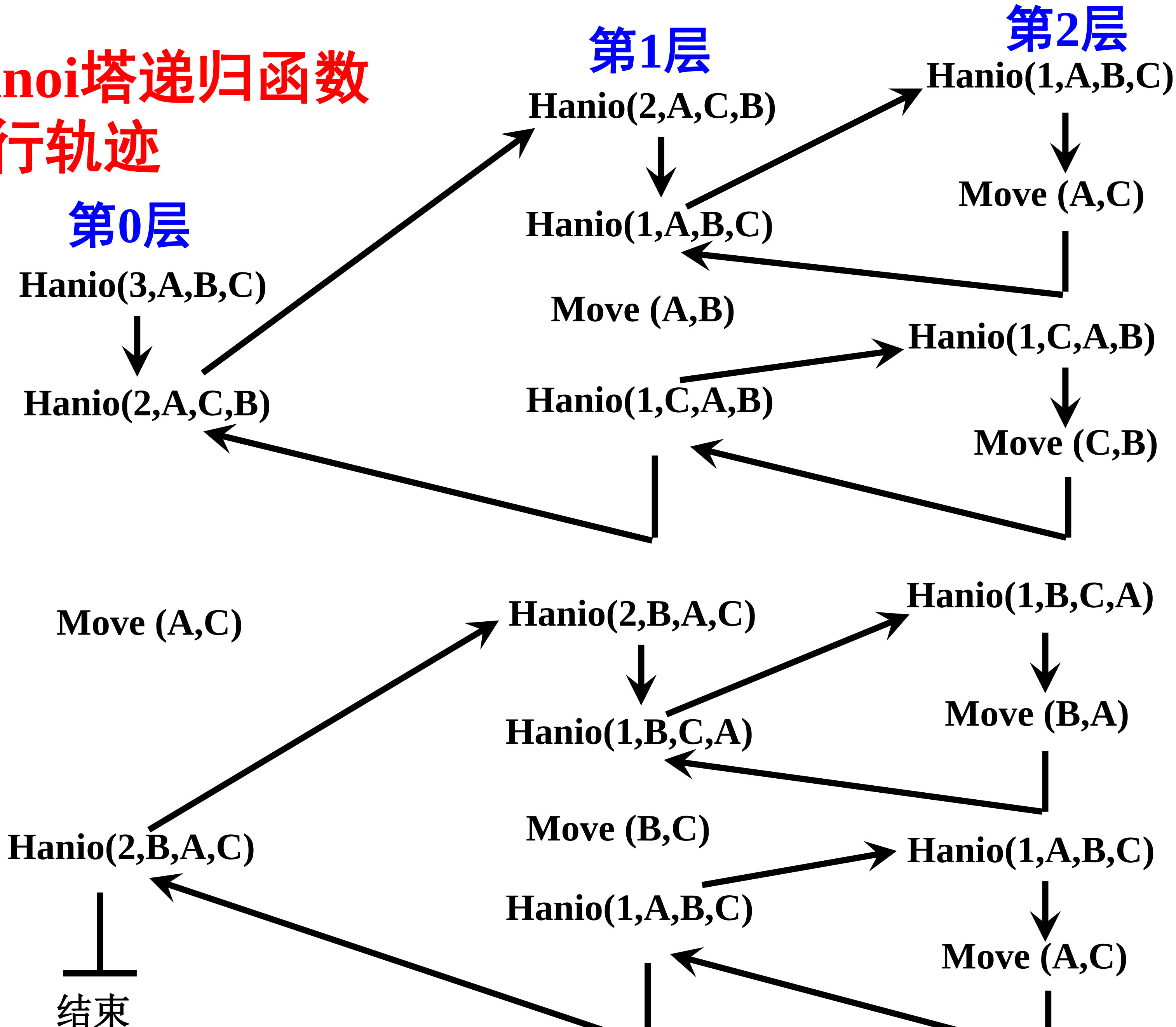


算法Hanoi以递归形式给出，每个圆盘的具体移动方式并不清楚，因此很难用手工移动来模拟这个算法。

递归函数的运行轨迹

- 在递归函数中，调用函数和被调用函数是同一个函数
- 注意递归函数的调用层次，如果把调用递归函数的主函数称为第0层，进入函数后，首次递归调用自身称为第1层调用；从第i层递归调用自身称为第i+1层。反之，退出第i+1层调用应该返回第i层。
- 采用图示方法描述递归函数的运行轨迹，从中可较直观地了解到各调用层次及其执行情况。

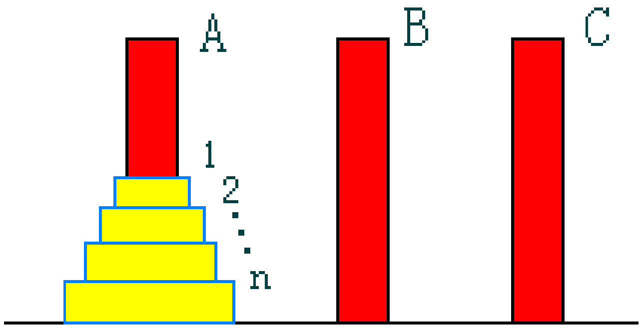
Hanoi塔递归函数运行轨迹



递归算法

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

1. **if *n* = 1 then move (A, C)**
2. **else Hanoi(*n*-1, A, C, B)**
3. **move (A, C)**
4. **Hanoi(*n*-1, B, A, C)**

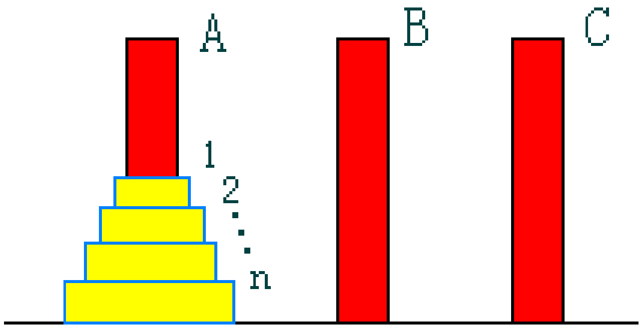


如果用了递归，就找不到while/for循环语句。
此时，该如何算时间复杂度呢？

递归算法

- 算法Hanoi(n, A, B, C) //n个盘子A到C

1. **if** n = 1 **then** move (A, C)
2. **else** Hanoi(n-1, A, C, B)
3. move (A, C)
4. Hanoi(n-1, B, A, C)



设n个盘子的移动次数为T(n)

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

$$T(1)=1$$

→ 递推方程

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

中国MOOC 2.2 序列求和的方法
2.3 递推方程与算法分析

换元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

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

$$\boxed{\begin{aligned} &T(n)=2T(n-1)+1 \\ &T(1)=1 \end{aligned}}$$

解的正确性-归纳验证

- **证明：** 下述递推方程的解是 $T(n)=2^n-1$
  - $T(n)=2T(n-1)+1$
  - $T(1)=1$

- **方法：** 数学归纳法
- **证** $n=1$, $T(1)=2^1-1=1$

  假设对于 n，解满足方程，则

  $T(n+1)$

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

Hanoi塔算法

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

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

问题： 如果1秒移动1个，64个金蝶要多少时间？

5000亿年！！！


每秒能算60万亿次
要计算三天左右

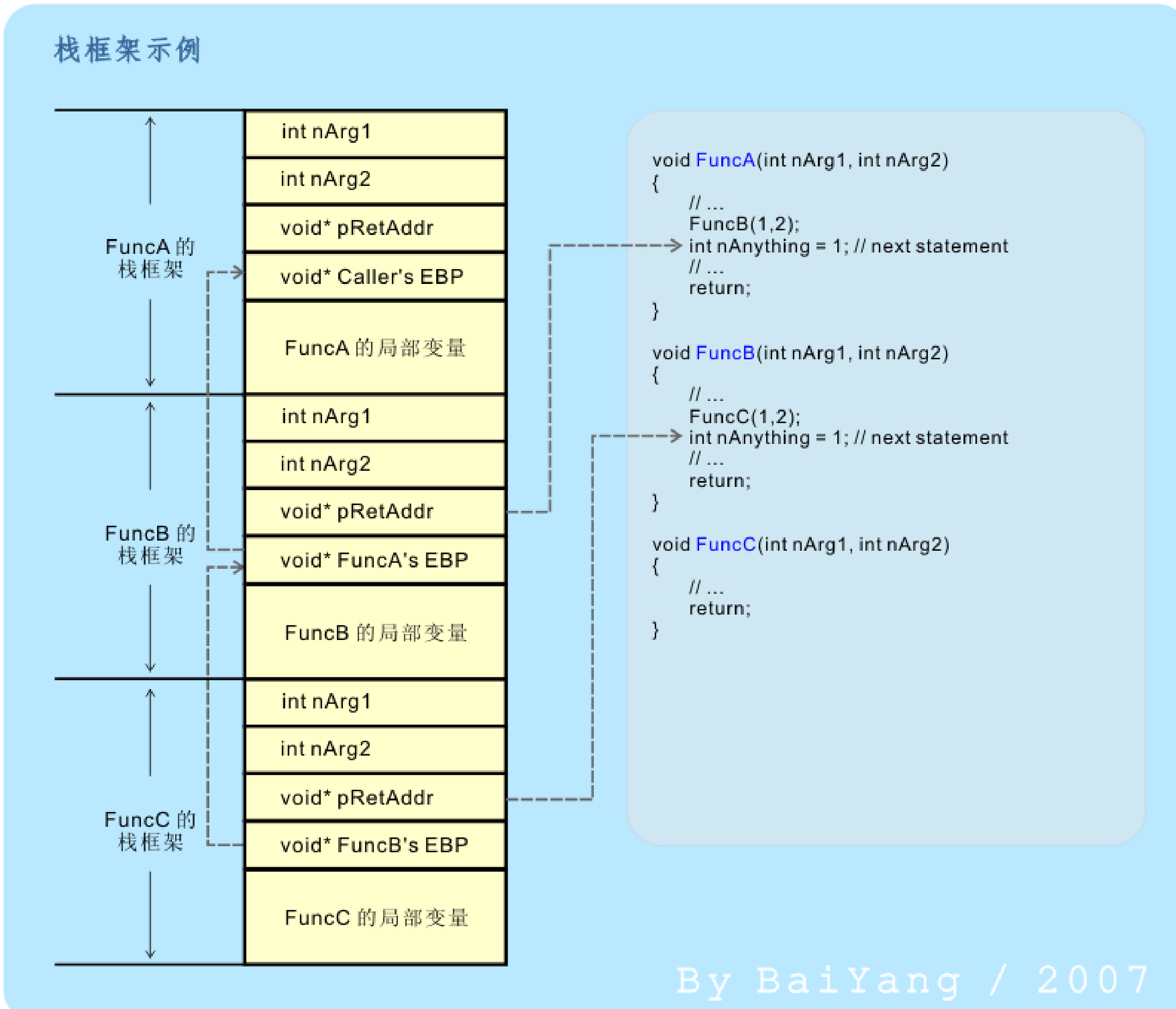
Hanoi塔算法

有没有更好的算法???

没有！！！

这是一个难解的问题，不存在多项式时间的算法！

递归的函数调用（回顾C语言）



递归的函数调用

- 实现递归调用的关键是建立递归调用工作栈。在调用算法之前：
  - 将所有实参指针，返回地址等信息传递给被调用算法；
  - 为被调算法的局部变量分配存储区；
  - 将控制移到被调算法的入口。
- 返回调用算法时，系统要完成：
  - 保存被调算法的结果；
  - 释放分配给被调用算法的数据区；
  - 依保存的返回地址将控制转移到调用算法；

递归小结

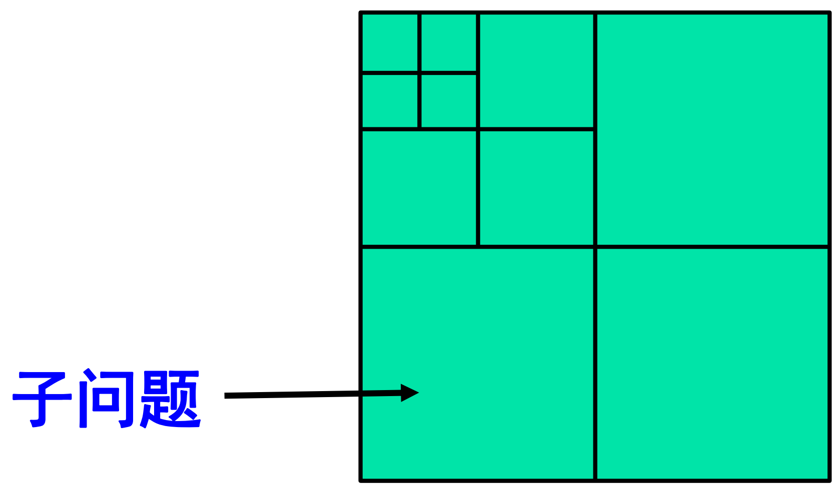
优点：结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。

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。

分治策略

分治法

基本思想：将一个规模为*n*的问题分解为*k*个规模较小的子问题，这些子问题互相独立且与原问题相同。递归解这些子问题，再将子问题合并得到原问题的解。



中国MOOC：3.2 分治策略的设计思想
3.3 分治策略的一般描述和分析方法

小结

- 分治和递归的概念
  - Ackerman
  - 整数划分
  - 汉诺塔