第3章 递归和分治策略

主讲人：甘文生 **PhD**

Email: wsgan001@gmail.com

暨南大学网络空间安全学院

Fall 2021

Jinan University, China

第三章 递归和分治策略

- 递归的定义、总体思想、特点；
- 通过具体例子理解递归策略与设计；
  - N的阶乘、Fibonacci数列、全排列、整数划分问题、Hanoi塔问题等
- 分治法的概念、步骤、复杂度分析；
- 通过几个范例学习分治策略的设计技巧；
  - 二分搜索、归并排序、乘法问题、找最大最小值问题、循环赛日程表等
- 掌握基于递归与分治策略的算法设计；

递归的定义

- Wiki：Recursion is the process of repeating items in a self-similar way.

- 递归(Recursion)，又译为递回，在数学与计算机科学中，是指在函数的定义中使用函数自身的方法。

- Recursion从词源上分析只是"re- (again)" + "curs- (come, happen)" 也就是重复发生，再次重现的意思，中文翻译"递归"表达了两个意思：递＋归。

- 递归是静中有动，有去(递去)有回(归来)。 循环是动静如一，有去无回。

递归的定义

- Recursive: In order to compute an, first compute an−1an−1 and let then an = 2an−1an = 2an−1. Terminate when you reach a0 = 1a0 =1.

- Inductive: Start with a0 = 1a0 = 1. Now if you know an, you can compute an+1an+1 by an+1= 2an+1= 2an.

递归、递推、迭代

递归形式的斐波那契数列:

```
int f[maxn] = 0;
f[0]=f[1]=1;
int fib(int n){
    if(f[n]) return f[n];
    return f[n] = fib(n-1) + fib(n-2);
}
```

递推形式的斐波那契数列:

```
int f[maxn] = 0;
f[0]=[1]=1;
for(int i=2; i<=n; ++i){
    f[i] = f[i-1] + f[i-2]
}
```

递推Inductive：一步步往后，从左往右。即有来无回。

递归Recursive：从最后面一步步往前嵌套，再从最前面一步步往后套。递归=递推+回归。即有来有回。

迭代Iteration：循环执行，每次把前面计算出的值套下一步。迭代是逐渐逼近，用新值覆盖旧值。

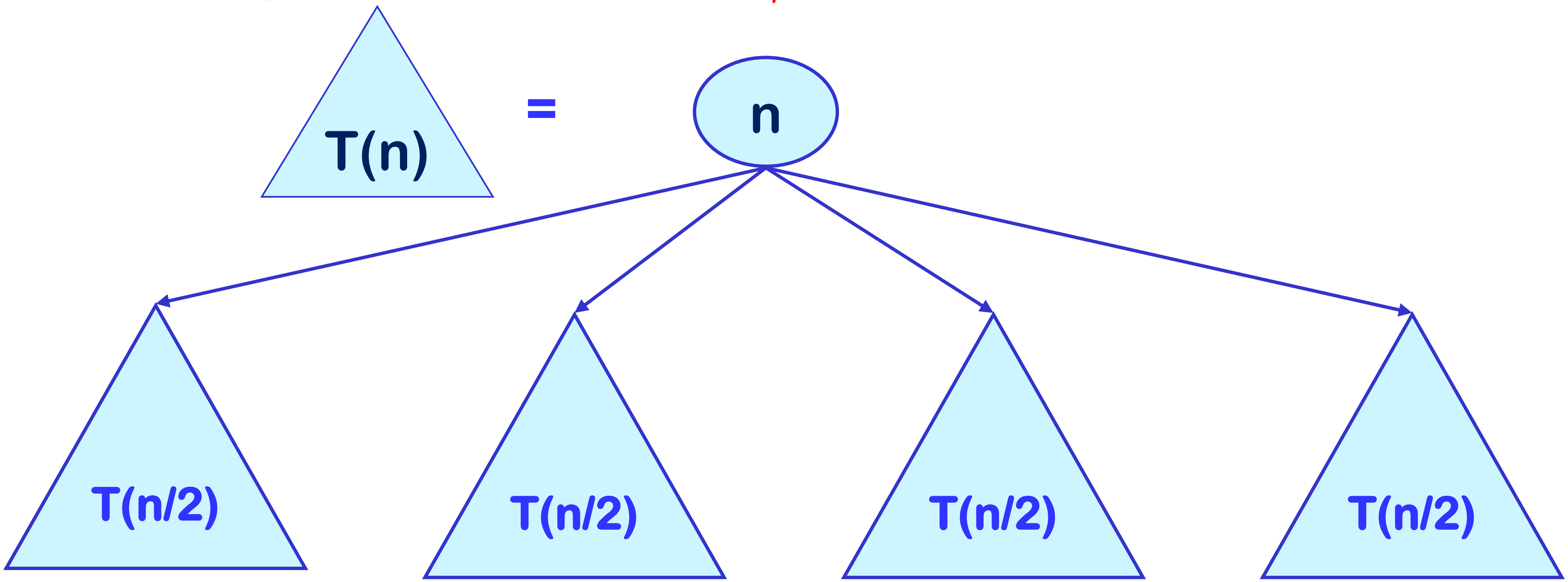
注意：递归次数太多可能会爆栈。

递归的定义

- 递归(Recursion)基本思想：把规模大的问题转化为规模小的相似的子问题来解决。在函数实现时，因为解决大问题的方法和解决小问题的方法往往是同一个方法，所以就产生了函数直接或间接调用它自身的情况。这个解决问题的函数必须有明显的结束条件，这样就不会产生无限递归的情况。
- 应用场景：树、阶乘、Fibonacci数列、Hanoi塔问题
- 递归的性能问题：栈的分配和函数调用代价需要在具体工程实践中考虑。

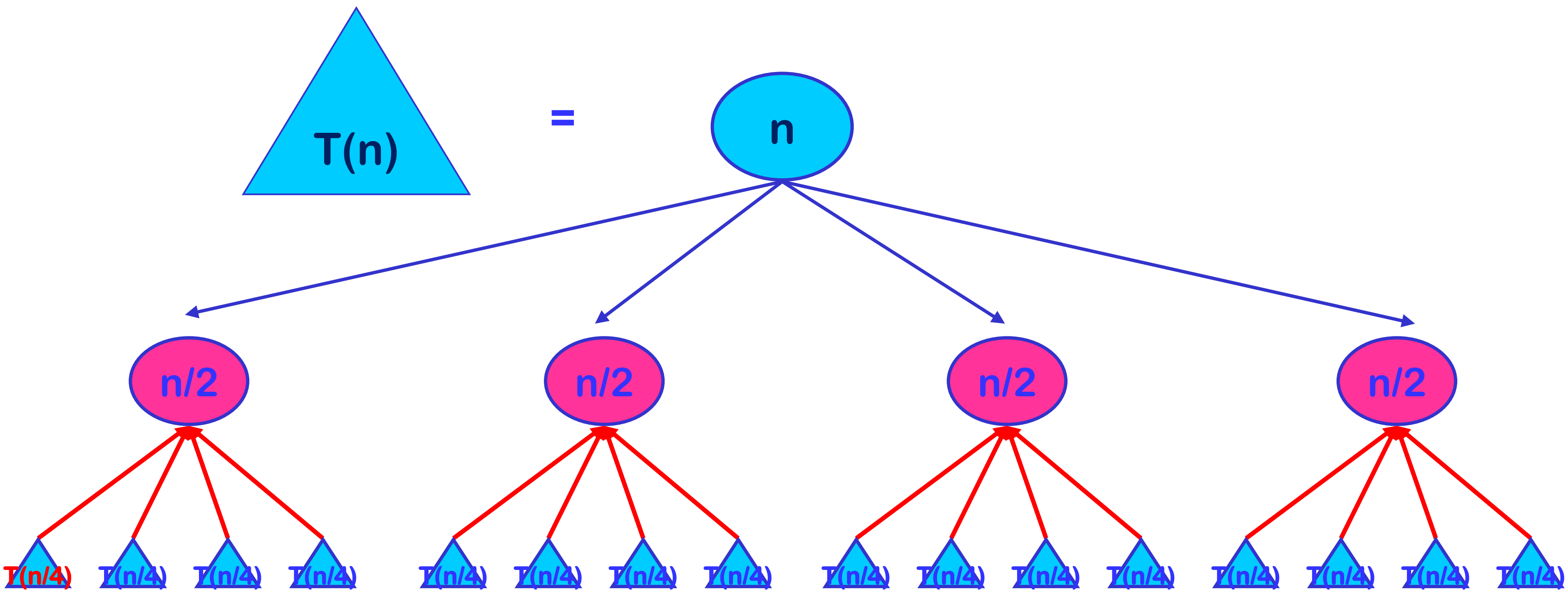
递归的总体思想

- 将求解的较大规模的问题分割成k个更小规模的子问题。
- 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。



递归的总体思想

- 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。



递归的总体思想

- 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。

分治法的设计思想：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，再各个击破，分而治之。

凡治众如治寡，分数是也。

----孙子兵法

递归的概念

- 直接或间接地调用自身的算法称为递归算法(直接递归、间接递归)。用函数自身给出定义的函数称为递归函数。
- 由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。
- 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。

递归的例子：n的阶乘

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

边界条件与递归方程是递归函数的二个要素，递归函数只有具备这两个要素，才能在有限次计算后得出结果。

递归的例子：Fibonacci数列

Fibonacci数列：无穷数列1，1，2，3，5，8，13，21，34，55，… 它可以递归地定义为：

边界条件

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

递归方程

第n个Fibonacci数可递归地计算如下：

```
public static int fibonacci(int n)
{
    if (n <= 1) return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

递归的例子：全排列问题

设计一个递归算法生成n个元素$\{r_1, r_2, \ldots, r_n\}$的**全排列**(从n个元素中取出m个元素进行排列，当n = m时叫做permutations)

设$R=\{r_1,r_2,\ldots,r_n\}$是要进行排列的n个元素，$R_i=R-\{r_i\}$。集合X中元素的全排列记为perm(X)。(r_i)perm(X)表示在全排列perm(X)的每一个排列前加上前缀得到的排列。R的全排列可归纳定义如下：

当n=1时，perm(R) = (r)，其中r是集合R中唯一的元素；
当n>1时，perm(R)由(r_1)perm(R_1)，(r_2)perm(R_2)，…，(r_n)perm(R_n)构成。

Input: [1,2,3]
Output: [1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]

递归的例子：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。

例子：正整数6有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。

递归的例子：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。

在本例中，如果设p(n)为正整数n的划分数，则难以找到递归关系，因此考虑增加一个自变量：将最大加数n_1不大于m的划分个数记作q(n, m)。可以建立q(n, m)如下递归关系。

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

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

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

正整数n的最大加数n_1不大于m的划分由$n_1 = m$的划分和$n_1 \leq m-1$ 的划分组成。

递归的例子：整数划分问题

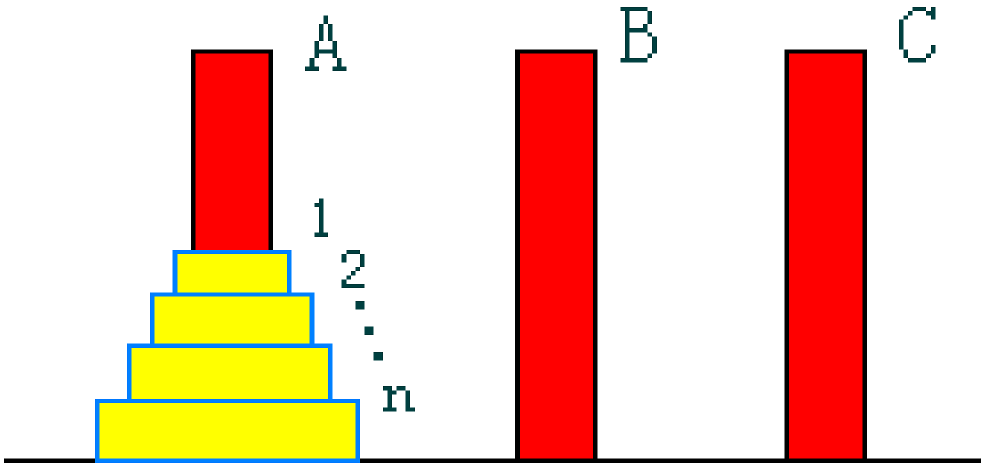
在本例中，可以建立q(n,m)如下递归关系。

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

正整数n的划分数p(n) = q(n, n)。

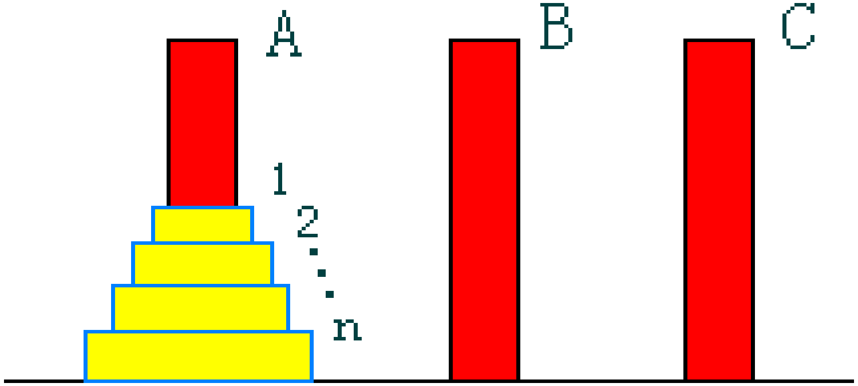
递归的例子：Hanoi塔问题

- 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圆盘压在较小圆盘之上；
- 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。



递归的例子：Hanoi塔问题

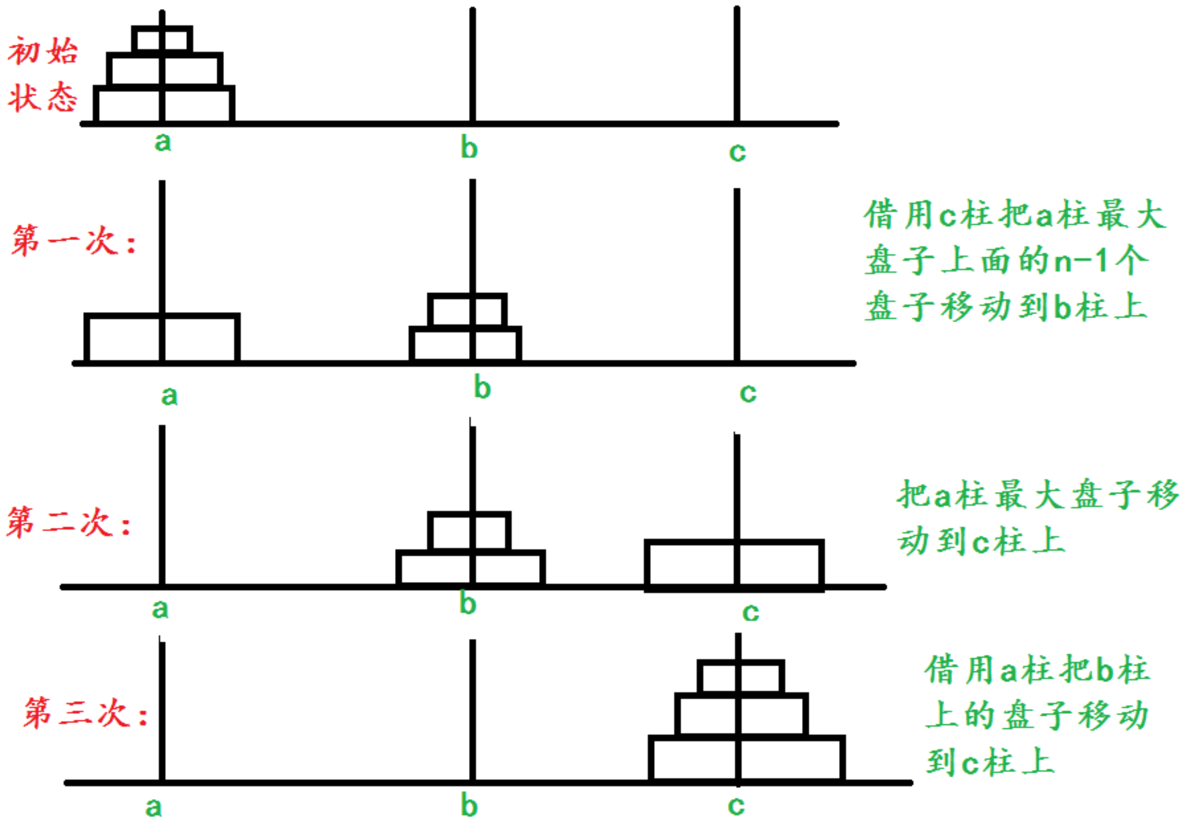
```
public static void hanoi(int n, int a, int b, int c)
{
      if (n > 0)
      {
         hanoi(n-1, a, c, b);
         move(a, b);
         hanoi(n-1, c, b, a);
      }
 }
```



三个步骤：

1） 把第n-1个盘子由a移到c；
2） 把第n个盘子由a移到b；
3） 把第n-1个盘子由c移到b；

递归的总结



思考题：如果塔的个数变为a, b, c, d四个，现要将n个圆盘从a全部移动到d，移动规则不变，求移动步数最小的方案。

递归的总结

- **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，能为设计算法、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。

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。

1、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。

2、用递推来实现递归函数。

3、通过变换能将一些递归转化为尾递归，从而迭代求出结果。

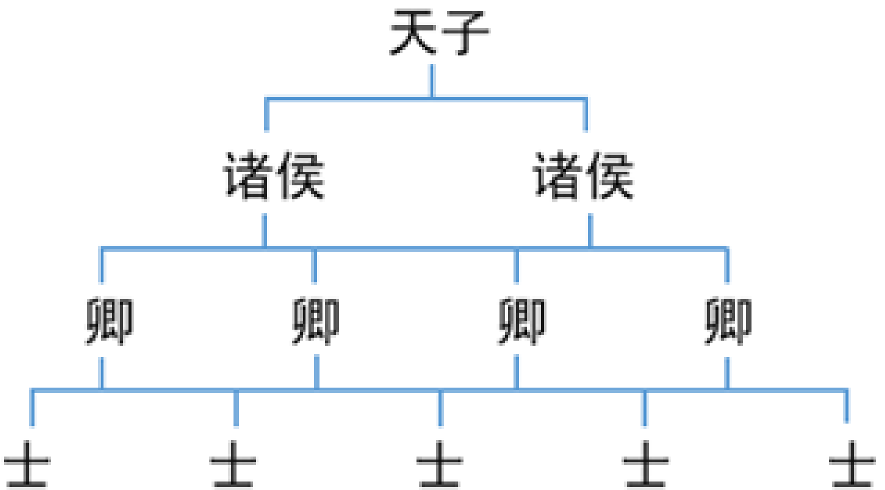
后两种方法在时空复杂度上均有较大改善，但其适用范围有限。

第三章 递归和分治策略

- 递归的定义、总体思想、特点；
- 通过具体例子理解递归策略与设计；
  - N的阶乘、Fibonacci数列、全排列、整数划分问题、Hanoi塔问题等
- 分治法的概念、步骤、复杂度分析；
- 通过几个范例学习分治策略的设计技巧；
  - 二分搜索、归并排序、乘法问题、找最大最小值问题、循环赛日程表等
- 掌握基于递归与分治策略的算法设计；

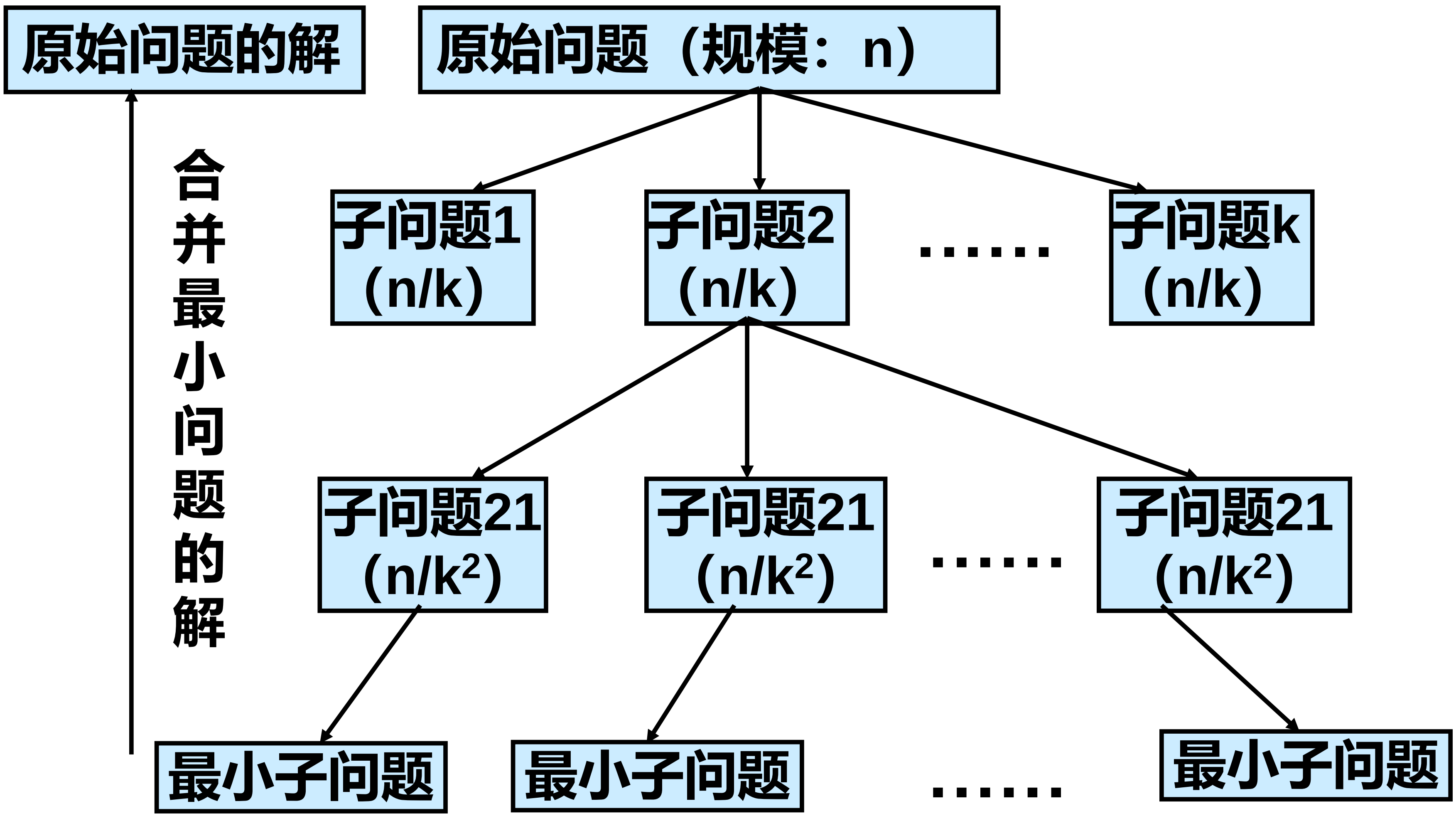
分治法的由来

- **分治法**，分而治之是一种古老的方法(周天子封邦建国)：



- 1. 分解，将原问题分解成若干个与原问题结构相同但规模较小的子问题；
- 2. 解决，解决这些子问题。如果子问题规模足够小，直接求解，否则递归地求解每个子问题；
- 3. 合并，将这些子问题的解合并起来，形成原问题的解。

分治法的概念


原始问题的解
原始问题（规模：n）
合并最小问题的解
子问题1 (n/k)
子问题2 (n/k)
……
子问题k (n/k)
子问题21 (n/k²)
子问题21 (n/k²)
……
子问题21 (n/k²)
最小子问题
最小子问题
……
最小子问题

分治法的概念

分治法所能解决的问题一般具有以下几个特征：

- **该问题的规模缩小到一定的程度就可以容易地解决；**
- **该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质**
- **利用该问题分解出的子问题的解可以合并为该问题的解；**
- **该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。**

因为问题的计算复杂性一般是随着问题规模的增加而增加，因此大部分问题满足这个特征。

这条特征是应用分治法的前提，它也是大多数问题可以满足的，此特征反映了递归思想的应用。

能否利用分治法完全取决于问题是否具有这条特征，如果具备了前两条特征，而不具备第三条特征，则可以考虑**贪心算法或动态规划**。

这条特征涉及到分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然也可用分治法，但一般用**动态规划**较好。

分治法的基本步骤

```
divide-and-conquer(P)
{// P是问题的规模，n0是阈值
    if ( |P| <= n0) adhoc(P);  // 基本子算法，解决小规模的问题
    divide P into smaller subinstances P1,P2,...,Pk；// 分解问题
    for (i=1; i<=k; i++)  // k通常为2
        yi=divide-and-conquer(Pi); // 递归的解各子问题
    return merge(y1,...,yk); // 将各子问题的解合并为原问题的解
}
```

人们从大量实践中发现，在用分治法设计算法时，最好使子问题的规模大致相同。即将一个问题分成大小相等的k个子问题的处理方法是行之有效的。这种使子问题规模大致相等的做法是出自一种**平衡(balancing)子问题**的思想，它几乎总是比子问题规模不等的做法要好。

分治法的复杂性分析

一个分治法将规模为n的问题分成k个规模为n／m的子问题去解。设分解阀值$n_0=1$，且adhoc解规模为1的问题耗费1个单位时间。再设将原问题分解为k个子问题以及用merge将k个子问题的解合并为原问题的解需用f(n)个单位时间。用T(n)表示该分治法解规模为|P|=n的问题所需的计算时间，则有：

$$T(n)=\begin{cases} O(1) & n=1 \\ kT(n/m)+f(n) & n>1 \end{cases}$$

通过迭代法求得方程的解：$T(n)=n^{\log_m k}+\sum_{j=0}^{\log_m n-1} k^j f(n/m^j)$

注意：递归方程及其解只给出n等于m的方幂时T(n)的值，但是如果认为T(n)足够平滑，那么由n等于m的方幂时T(n)的值可以估计T(n)的增长速度。通常假定T(n)是单调上升的，从而当$m^i \le n < m^{i+1}$时，$T(m^i) \le T(n) < T(m^{i+1})$。

*运用主定理Master Theorem (《算法导论》第4章节有具体的推导与证明)容易求解出递归式的时间复杂度。

思考题：分治法的优缺点？

- 能简单地求解复杂的问题
- 并行性 (并行计算、多处理器系统)
- 内存访问 (利用内存缓存机制，不需要访问存取速度较慢的主存)

- 分治法不能适应于所有问题！
- 递归的效率较慢 (具体的实现方式)
- 分治法比迭代方法更复杂 (例子：n个数求和)

分治法的常用例子

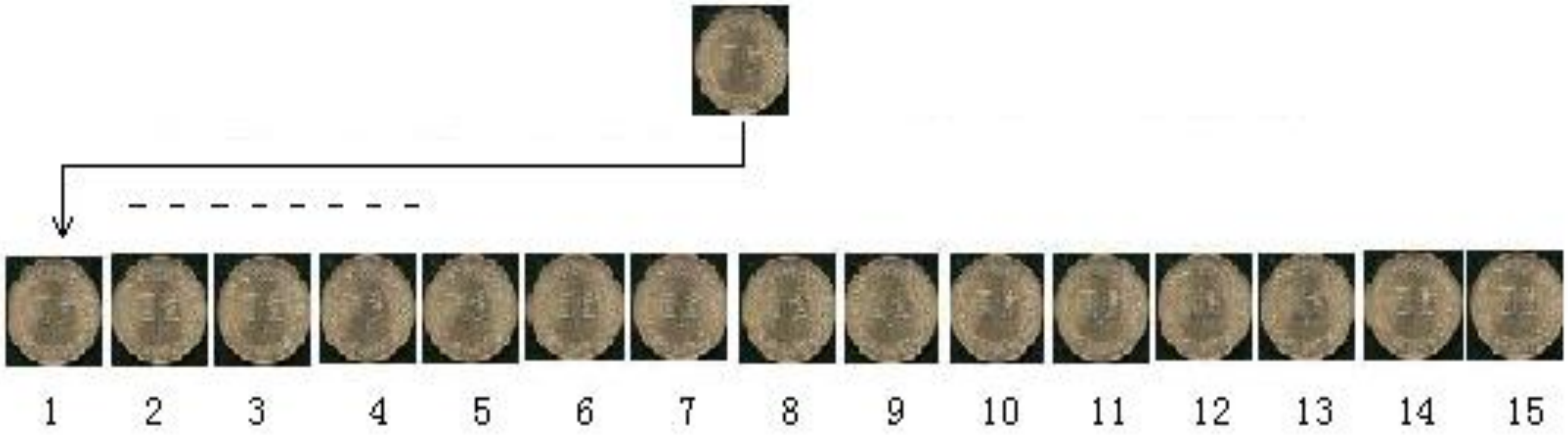
- **Binary search**二分搜索/二分查找
- **Merge sort** 归并排序
- **Quick sort** 快速排序
- **Matrix multiplication** 矩阵乘法
- **Multiplication of two numbers** 乘法问题
- **Multiplication of two matrices** 乘法问题
- **Finding Minimum and Maximum** 找最大最小值
- **循环赛日程表**

- **分治法不能适应于所有问题！**

二分查找：找假币实例

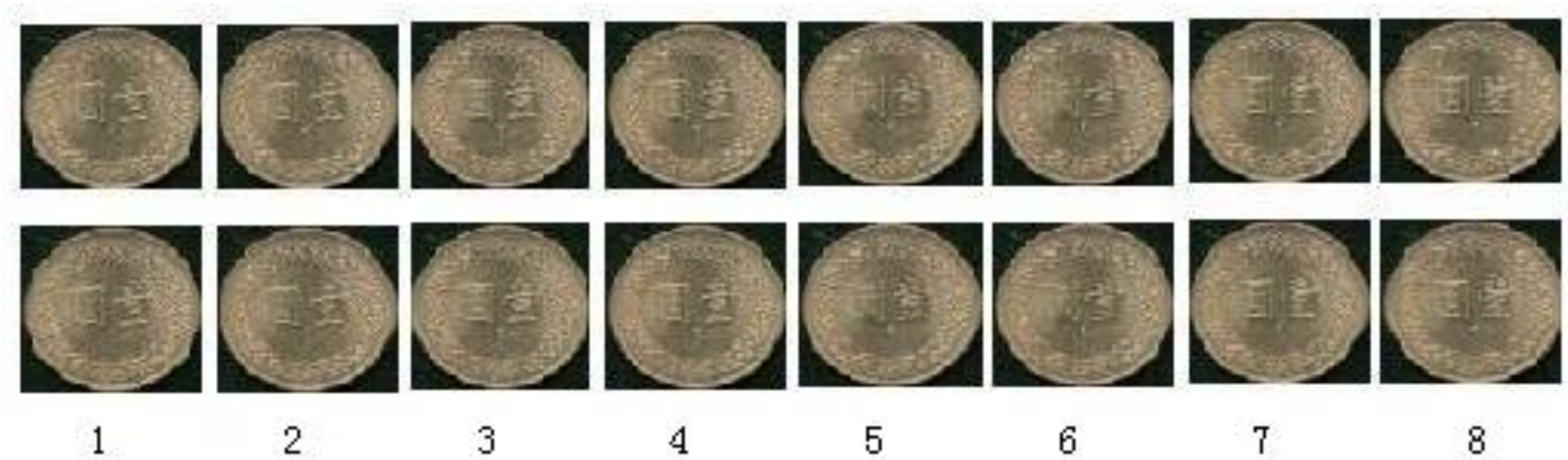
- 1 6枚硬币中有一个是伪造的，伪造的硬币比真的硬币要轻一些。你的任务是利用天平找出这枚伪造的硬币。

- **任意取1枚硬币，与其他硬币进行比较，若发现轻者，这那枚为伪币。最多可能有15次比较。**

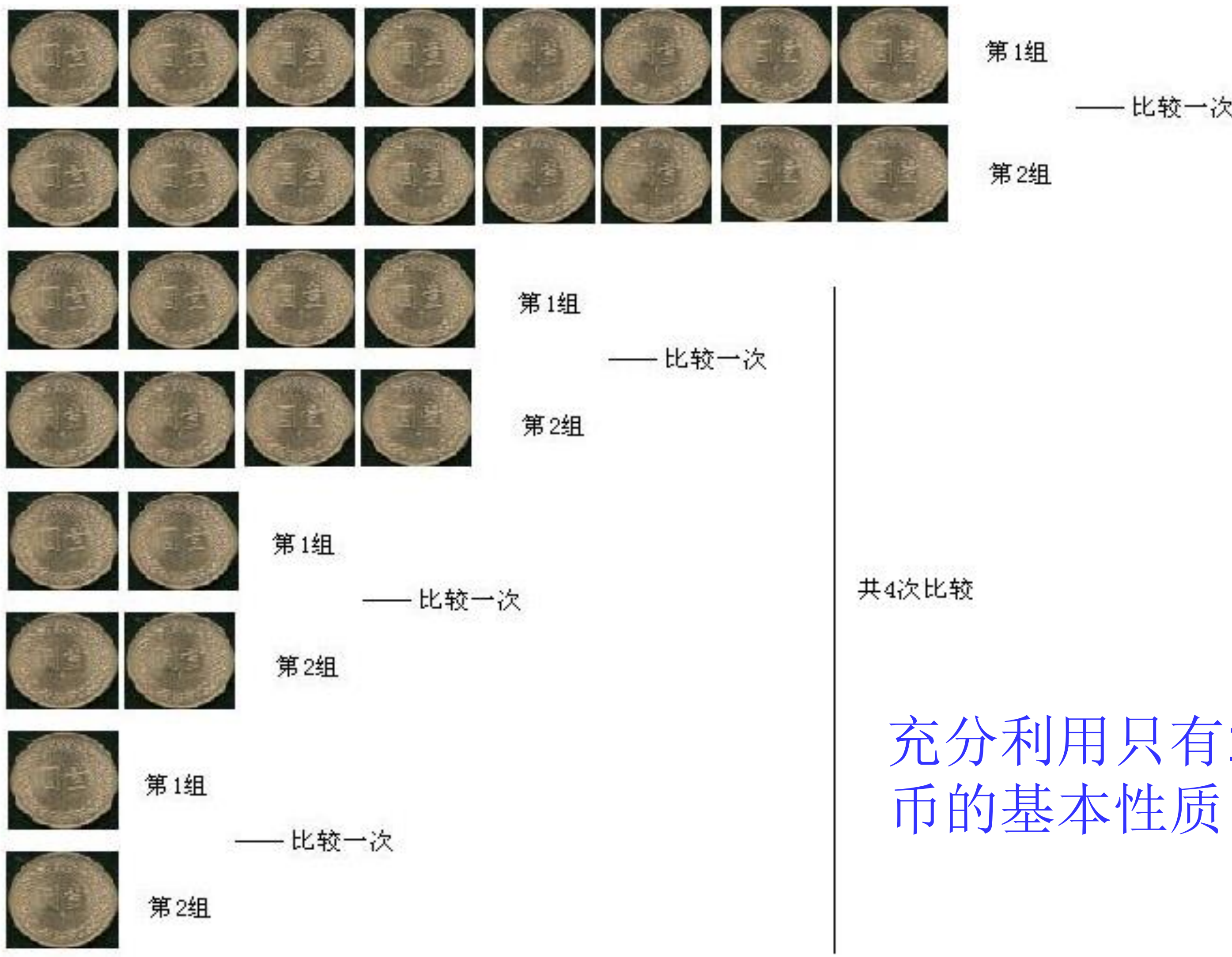


二分查找：找假币实例

- 将硬币分为8组，每组2个，每组比较一次，若发现轻的，则为伪币。最多可能有8次比较。



二分查找：找假币实例



充分利用只有1枚假币的基本性质

分治法的例子：二分搜索

- 给定已按升序排好序的n个元素a[0:n-1]，现要在这n个元素中找出一特定元素x。
- 分析：
  - ✓ 该问题的规模缩小到一定的程度就可以容易地解决；
  - ✓ 该问题可以分解为若干个规模较小的相同问题;
  - ✓ 分解出的子问题的解可以合并为原问题的解；
  - ✓ 分解出的各个子问题是相互独立的。

分析：如果n=1即只有一个元素，则只要比较这个元素和x就

分析：很显然此问题分解出的子问题相互独立，即在a[i]的前面或后面查找x是独立的子问题，因此满足分治法的第四个适用条件。

分治法的例子：二分搜索

给出一组有序的数：8，11，19，23，27，33，45，55，67，98，如何查找出19？

应用场景及局限性

- 二分查找依赖顺序表结构，如数组；
- 二分查找针对的是有序数据，如果无序，则要先排序；
- 数据量太小不适合二分查找；
- 数据量太大也不适合二分查找。

分治法的例子：二分搜索

- 给定已按升序排好序的n个元素a[0:n-1]，现要在这n个元素中找出一特定元素x。设计出二分搜索算法：

```
public static int binarySearch(int [] a, int x, int n)
  {
   // 在 a[0] <= a[1] <= ... <= a[n-1] 中搜索 x
   // 找到x时返回其在数组中的位置，否则返回-1
   int left = 0; int right = n - 1;
   while (left <= right) {
     int middle = (left + right)/2;
     if (x == a[middle]) return middle;
     if (x > a[middle]) left = middle + 1;
     else right = middle - 1;
    }
   return -1; // 未找到x
  }
```

算法复杂度分析：
每执行一次算法的while循环， 待搜索数组的大小减少一半。因此，在最坏情况下，while循环被执行了O(logn) 次。循环体内运算需要O(1) 时间，因此整个算法在最坏情况下的计算时间复杂性为O(logn) 。

思考题：给定a，用二分法设计出求a^n的算法。

分治法的例子：**归并排序**

基本思想：将待排序元素分成大小大致相同的2个子集合，分别对2个子集合进行排序，最终将排好序的子集合合并成为

复杂度分析

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

T(n)=O(nlogn) 渐进意义下的最优算法

```
public                                                    right)
  {
    if
    int i=(left+right)/2;  //取中点
    mergeSort(a, left, i);
    mergeSort(a, i+1, right);
    merge(a, b, left, i, right);  //合并到数组b
    copy(a, b, left, right);    //复制回数组a
    }
  }
```

分治法的例子：归并排序

算法**Merge-Sort**的递归过程可以消去 [2-路归并]

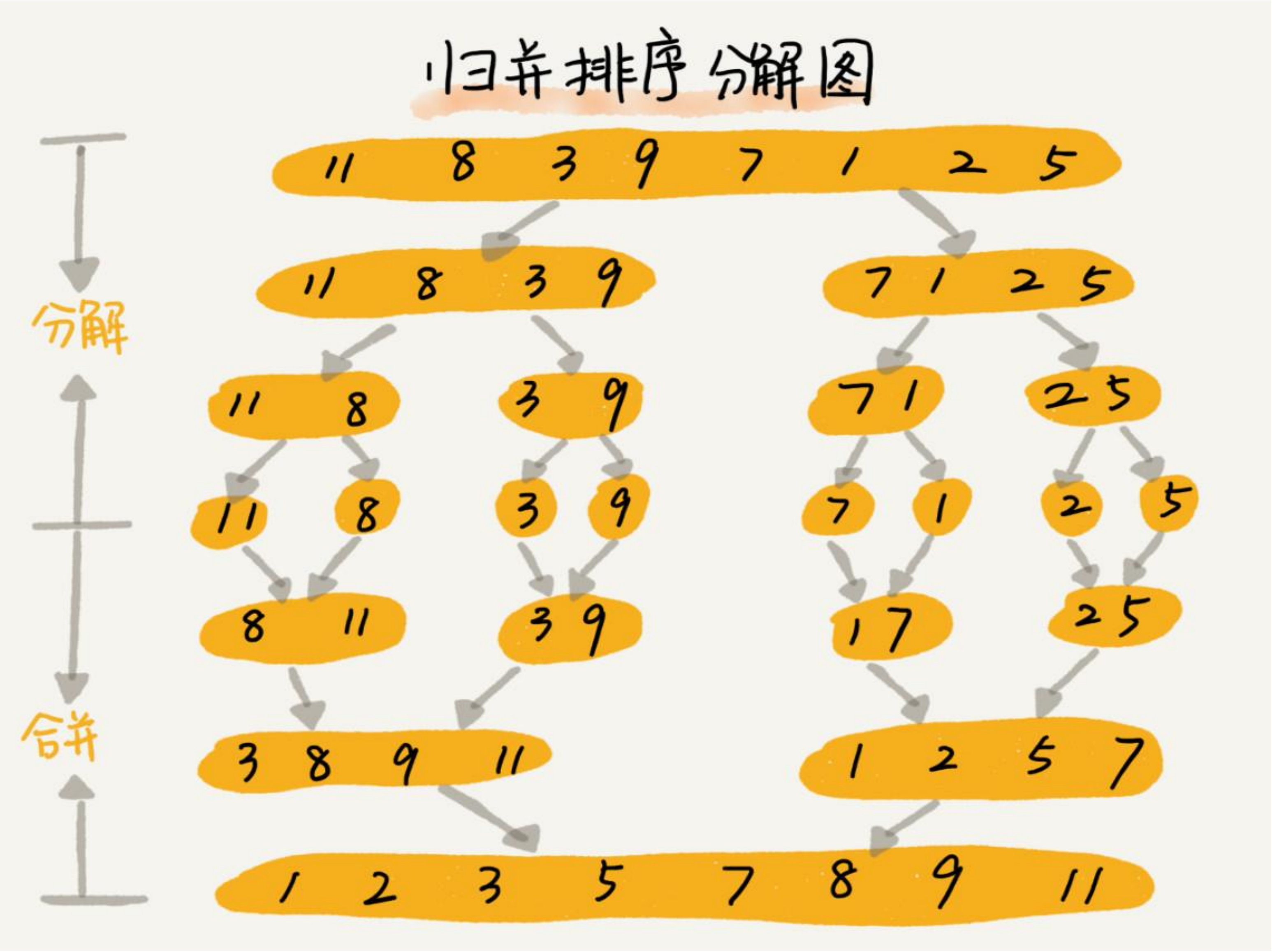
初始序列 [49] [38] [65] [97] [76] [13] [27]

第一步 [38 49] [65 97] [13 76] [27]

第二步 [38 49 65 97] [13 27 76]

第三步 [13 27 38 49 65 76 97]

分治法的例子：归并排序


归并排序分解图
分解
合并
11 8 3 9 7 1 2 5
11 8 3 9
7 1 2 5
11 8
3 9
7 1
2 5
11
8
3
9
7
1
2
5
8 11
3 9
1 7
2 5
3 8 9 11
1 2 5 7
1 2 3 5 7 8 9 11

分治法的例子：归并排序

- 最坏时间复杂度：O(nlogn)
- 平均时间复杂度：O(nlogn)
- 辅助空间：O(n)
- 稳定性：稳定

思考题：给定有序表A[1:n]，修改合并排序算法，求出该有序表的逆序对数。

分治法的例子：乘法问题

1. 整数相乘问题。

X和Y是两个n位的十进制整数，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$0 \le x_i, y_j \le 9$ (i, j=0, 1, … n-1)，设计一个算法求X×Y，并分析其复杂度。说明：算法中“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。

2. 矩阵相乘问题。

A和B是两个n阶实方阵，表示为 $\mathbf{A} = \begin{pmatrix} a_{11}\ldots a_{1n} \\ \ldots\ldots\ldots \\ a_{n1}\ldots a_{nn} \end{pmatrix}$，$\mathbf{B} = \begin{pmatrix} b_{11}\ldots b_{1n} \\ \ldots\ldots\ldots \\ b_{n1}\ldots b_{nn} \end{pmatrix}$

设计一个算法求A×B，并分析计算复杂度。

说明：算法中“基本操作”约定为两个实数相乘，或两个实数相加。

大整数的乘法 Multiplication of two numbers

two *n*-digit numbers X and Y, Complexity(X × Y) = ?

- **Naive (原始的) pencil-and-paper algorithm**

```
     12
   ×23
   ----
      6
     3
   - - - -
     4
    2
   ----
    276
```

```
          31415962
         ×27182818
         ---------
         251327696
         31415962
        251327696
        62831924
       251327696
       31415962
      219911734
      62831924
      ----------------
      853974377340916
```

- **Complexity analysis: n^2 multiplications and at most n^2-1 additions (加法). So, $T(n) = O(n^2)$.**

大整数的乘法 Multiplication of two numbers

two n-digit numbers X and Y, Complexity(X × Y) = ?

- **Divide and Conquer algorithm**

  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
  $m = \lceil n/2 \rceil$
  $a = \lfloor X/10^m \rfloor$; b=X mod 10^m
  $c = \lfloor Y/10^m \rfloor$; 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 $10^{2m}e + 10^m(g + h) + f$

Complexity analysis:
$T(1) = 1$,
$T(n) = 4T(\lceil n/2 \rceil) + O(n)$.
Applying Master Theorem, we have
$T(n) = O(n^2)$

分治法的例子：大整数的乘法

设计一个可以进行两个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:

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☹

$X = a\,2^{n/2} + b \qquad Y = c\,2^{n/2} + d$

$XY = ac\,2^{n} + (ad+bc)\,2^{n/2} + bd$

分治法的例子：大整数的乘法

- 小学的方法：$O(n^2)$ ✘效率太低
- 分治法:

$$XY = ac\,2^n + (ad+bc)\,2^{n/2} + bd$$

为了降低时间复杂度，必须减少乘法的次数。

1. $XY = ac\,2^n + ((a-c)(b-d)+ac+bd)\,2^{n/2} + bd$
2. $XY = ac\,2^n + ((a+c)(b+d)-ac-bd)\,2^{n/2} + bd$

分治法的例子：大整数的乘法

two n-digit numbers X and Y, Complexity(X × Y) = ?

- Divide and Conquer (Karatsuba's algorithm大数乘法)

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 $bc + ad = ac + bd - (a - b)(c - d)$

```
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
```

Complexity analysis:

$T(1) = 1$,

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

Applying Master Theorem, we have

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

分治法的例子：大整数的乘法

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☺

细节问题：两个XY的复杂度都是$O(n^{\log 3})$，但考虑到a+c, b+d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。

分治法的例子：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。

➢最终该思想导致了**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，对于大整数乘法，它能在**O(nlogn)**时间内解决。

➢是否能找到线性时间的算法？？？目前为止还没有结果

矩阵相乘 Multiplication of two matrices

two $n \times n$ matrices A and B, Complexity(C=A×B) = ?

- **Standard method**

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{pmatrix} \cdots\cdots\ .\ .. \\ \cdots\cdots c_{ij} .. \\ \cdots\cdots\ .\ .. \\ \cdots\cdots\ .\ .. \end{pmatrix} = \begin{pmatrix} \cdots\cdots\cdots \\ ***** \\ \cdots\cdots\cdots \\ \cdots\cdots\cdots \end{pmatrix} \begin{pmatrix} \cdots\cdots * .. \\ \cdots\cdots * .. \\ \cdots\cdots * .. \\ \cdots\cdots * .. \end{pmatrix}$$

MATRIX-MULTIPLY(A, B)
for $i \leftarrow 1$ to n
  for $j \leftarrow 1$ to n
    $C[i, j] \leftarrow 0$
    for $k \leftarrow 1$ to n
      $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
return C

Complexity:
$O(n^3)$ multiplications and additions.
$T(n) = O(n^3)$.

矩阵相乘 Multiplication of two matrices

two $n \times n$ matrices A and B, Complexity(C=A×B) = ?

- **Divide and conquer**

An $n \times n$ matrix can be divided into four $n/2 \times n/2$ matrices,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},\ B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},\ C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$C_{11}=A_{11}B_{11}+A_{12}B_{21}$, $C_{12}=A_{11}B_{12}+A_{12}B_{22}$

$C_{21}=A_{21}B_{11}+A_{22}B_{21}$, $C_{22}=A_{21}B_{12}+A_{22}B_{22}$

Complexity analysis:

Totally, 8 multiplications (subproblems), and 4 additions ($n/2 \times n/2 \times 4$).

$T(1)=1$, $T(n) = 8T(\lceil n/2 \rceil)+n^2$.

Applying Master Theorem, we have

$T(n)= O(n^3)$.

矩阵相乘 Multiplication of two matrices

two $n \times n$ matrices A and B, Complexity(C=A×B) = ?

- **Divide and conquer (Strassen Algorithm)**

An $n \times n$ matrix can be divided into four $n/2 \times n/2$ matrices,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Define

$$P_1 = (A_{11}+A_{22})(B_{11}+B_{22})$$
$$P_2 = (A_{11}+A_{22})B_{11}$$
$$P_3 = A_{11}(B_{11}-B_{22})$$
$$P_4 = A_{22}(-B_{11}+B_{22})$$
$$P_5 = (A_{11}+A_{12})B_{22}$$
$$P_6 = (-A_{11}+A_{21})(B_{11}+B_{12})$$
$$P_7 = (A_{12}-A_{22})(B_{21}+B_{22})$$

Then

$C_{11}=P_1+P_4-P_5+P_7$, $C_{12}=P_3+P_5$

$C_{21}=P_2+P_4$, $C_{22}=P_1+P_3-P_2+P_6$

Complexity analysis:
Totally, 7 multiplications, and 18 additions.

$T(1)=1$,

$T(n) = 7T(\lceil n/2 \rceil) + cn^2$.

Applying Master Theorem

$$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$$

矩阵相乘 Multiplication of two matrices

- ◆传统方法：$O(n^3)$
- ◆分治法: $O(n^{2.81})$
- ◆更快的方法??

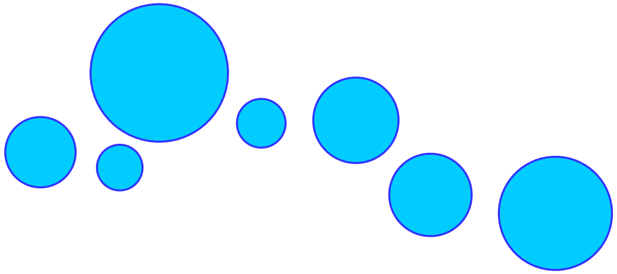
➢Hopcroft和Kerr已经证明(1971)，计算2个2×2矩阵的乘积，7次乘法是必要的。因此，要想进一步改进矩阵乘法的时间复杂性，就不能再基于计算2×2矩阵的7次乘法这样的方法了。或许应当研究3×3或5×5矩阵的更好算法。

➢在Strassen之后又有许多算法改进了矩阵乘法的计算时间复杂性。目前最好的计算时间上界是 $\mathbf{O(n^{2.376})}$

➢是否能找到$O(n^2)$的算法？

Finding Minimum and Maximum（最大最小值）

- **Background: Find the lightest and heaviest of *n* elements using a balance that allows you to compare the weight of 2 elements.**（对于一个具有 *n* 个元素的数组，用一个天平，通过比较 2个元素的重量，求出最轻和最重的一个）



- **Goal: Minimize the number of comparisons.**（正确找出需要的元素，最少的比较次数？）

Finding Minimum and Maximum（最大最小值）

金块问题：老板有一袋金块(共n块)，最优秀的雇员得到其中最重的一块，最差的雇员得到其中最轻的一块。假设有一台天秤，**用最少的比较次数找出最重的和最轻的金块。**

算法设计1：**比较简单的方法是逐个的进行比较查找。**先拿两块比较重量，留下重的一个与下一块比较，直到全部比较完毕，找到最重的金子。算法类似于一趟选择排序，算法如下：

```
maxmin( float a[],int n)
 {  max==min=a[1];
 for(i=2 i<=n i++ )
   if(max < a[i])  max=a[i];
    else if(min > a[i])   min=a[i];
 }
```

Finding Minimum and Maximum（最大最小值）

算法分析1：算法中需要n-1次比较得到max。最好的情况是全块由小到大取出，不需要进行与min的比较，共进行n-1次比较。最坏的情况是全块由大到小取出，需要再经过n-1次比较得到min, 共进行2*n-2次比较。在平均情况下，A(i)将有一半的时间比max大，因此平均比较数为3(n—1)/2。

算法设计2: 问题可以简化为：在含n (n是2的幂, 即n>=2))个元素的集合中寻找极大元和极小元。用分治法(二分法)可以用较少比较次数解决上述问题：

1) 将数据等分为两组(两组数据可能差1)，目的是分别选取其中的最大值。

2) 递归分解直到每组元素的个数≤2，可简单地找到最大值。

3) 回溯时将分解的两组解大者取大，小者取小，合并为当前问题的解。

Finding Minimum and Maximum（最大最小值）

算法2: 递归求取最大和最小元素

```
float a[n];
maxmin (int  i, int j ,float &fmax, float &fmin)
{int mid;  float lmax, lmin, rmax, rmin;
if (i==j)  {fmax= a[i];  fmin=a[i];} //一个元素
else if (i==j-1)  //两个元素
    {if(a[i]<a[j])  { fmax=a[j]；  fmin=a[i];}
      else {fmax=a[i]; fmin=a[j];}}
else     {mid=(i+j)/2;  //多个元素，  二分法
              maxmin (i，  mid，  lmax，  lmin);
              maxmin (mid+1，  j，  rmax，  rmin);
             if(lmax>rmax)    fmax=lmax;
        else       fmax=rmax;
              if(lmin>rmin)     fmin=rmin;
         else       fmin=lmin;
}
```

总结：分治法的优缺点

- 能简单地求解复杂的问题
- 并行性 (并行计算、多处理器系统)
- 内存访问 (利用内存缓存机制，不需要访问存取速度较慢的主存)

- 分治法不能适应于所有问题！
- 递归的效率较慢 (具体的实现方式)
- 分治法比迭代方法更复杂 (例子：n个数求和)

循环赛日程问题

设计一个满足以下要求的比赛日程表:
(1) 每个选手必须与其他n-1个选手各赛一次;
(2) 每个选手一天只能赛一次;
(3) 循环赛一共进行n-1天。

按分治策略，将所有的选手分为两半，n个选手的比赛日程表就可以通过为n/2个选手设计的比赛日程表来决定。递归地用对选手进行分割，直到只剩下2个选手时，比赛日程表的制定就变得很简单。这时只要让这2个选手进行比赛就可以了。

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

谢谢！

Q & A

作业： 3-1 3-2