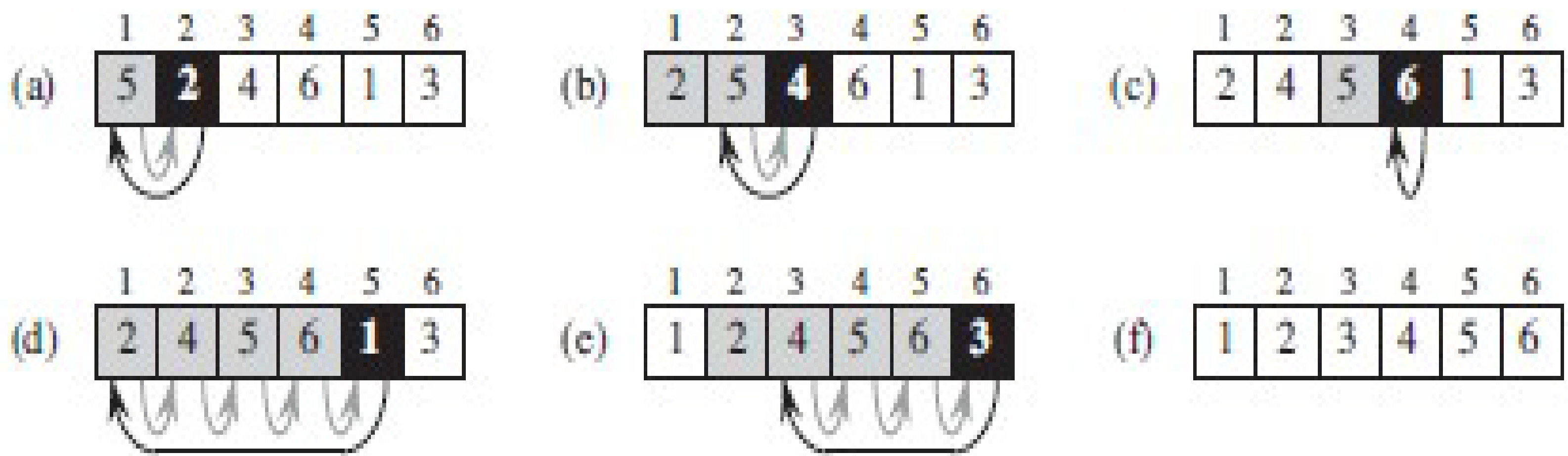
递归数列及函数增长—算法分析初步

离散数学教学组

排序算法–插入排序



遍历所有元素：

　　构造已排序的子序列

　　将待排序元素插入子序列中的合适位置

插入排序的伪代码

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

插入排序的时间复杂度

* 最坏情形下（用比较次数来衡量）
  * $(2-1)+(3-1)+\ldots+(n-1)=n(n-1)/2$
  * $O(n^2)$
* 最坏情形：待排序元素完全逆序！
  * 比如：6 5 4 3 2 1

算法的执行步骤数：算法分析初步

- 算法的正确性 VS 算法的效率
- 如何去评判一个算法的效率？
  - 时间开销: steps
  - 空间开销: memory
- 算法的执行步骤数是主要手段
  - 算法的执行步骤数不是简单的算法语句条数！

算法的执行步骤数

N (数据集规模)	T(n) (算法执行步数)
10	550
50	63750
100	505000

算法执行步数随着数据规模的变化而变化
不同的算法，变化的“剧烈程度”不同

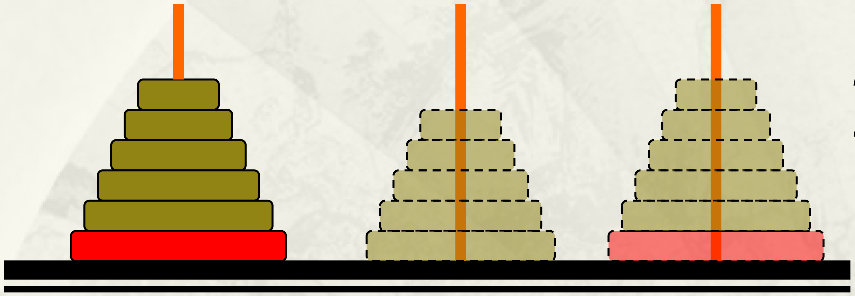
引入一个数学工具来刻画这种变化并尝试判断其规律

算法执行步骤函数

- 针对每个算法，可以定义该算法的执行步骤函数 T:N->N：
  - 数据规模->算法执行步骤数
- 该函数
  - 最坏情形/平均情形复杂度
  - 基本代表一个算法的执行效率
  - 随着数据规模变化，考察该函数的“增长”速度

汉诺塔递归算法的性能分析

- Towers of Hanoi
  - How many moves are need to move all the disks to the third peg by moving only one at a time and never placing a disk on top of a smaller one.



```
void hanoi(int n,char one, two, three)
// 将n个盘从one座借助two座,移到three座
{
   void move(char x, char y);
   if(n==1)  then move(one,three);
      else  {
          hanoi(n-1,one,three,two);
          move(one,three);
          hanoi(n-1,two,one,three);
      }
}
```

$T(1) = 1$

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

Solution of Towers of Hanoi

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

$2T(n-1) = 4T(n-2) + 2$

$4T(n-2) = 8T(n-3) + 4$

…….

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

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

Recurrence relations (递推关系)

- Examples
  - 4,7,10,13,16,……
  - 1,1,2,3,5,8,13,21,34,…… (a)
- Problem
  - Recurrence relation: the recursive formula
  - e.g: $f_n = f_{n-1}+f_{n-2}$, $f_1=f_2=1$ for (a)
  - $f_1=f_2=1$: initial condition

Example

- Let A={0,1}.
- C_n: the number of strings of length n in A* that do not contain adjacent 0's
  - C_1=?; C_2=?;
  - C_3=?
  - C_n=?
- $C_n = C_{n-1} + C_{n-2}$

Finding an explicit formula

- Find an explicit formula for these sequences?
- Backtracking
  - E.g. 1:
    - $a_n = a_{n-1}+3$, $a_1 = 2$ => recurrence relation
    - $a_n = 2+3(n-1)$ => explicit formula
  - E.g. 2
    - $b_n = 2b_{n-1}+1$, $b_1 = 7$
    - $b_n = 2^{n+2}-1$

Linear Homogeneous Relation

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_m a_{n-k}$$

is called linear homogeneous relation of degree *k*.

$$c_n = (-2)c_{n-1}$$

$$f_n = f_{n-1} + f_{n-2}$$

Yes

$$a_n = a_{n-1} + 3$$

$$g_n = g_{n-1}^2 + g_{n-2}$$

No

Characteristic Equation

- For a linear homogeneous recurrence relation of degree k

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_m a_{n-k}$$

the polynomial of degree k

$$x^k = r_1 x^{k-1} + r_2 x^{k-2} + \cdots + r_k$$

is called its characteristic equation.

- The characteristic equation of linear homogeneous recurrence relation of degree 2 is: $\boxed{x^2 - r_1 x - r_2 = 0}$

Solution of Recurrence Relation

- If the characteristic equation $x^2 - r_1 x - r_2 = 0$ of the recurrence relation $a_n = r_1 a_{n-1} + r_2 a_{n-2}$ has two distinct roots s_1 and s_2, then

$$a_n = u s_1^n + v s_2^n$$

where u and v depend on the initial conditions, is the explicit formula for the sequence.

Solution of Recurrence Relation

* If the equation has a single root *s*, then,

$$a_n = us^n + vns^n$$

Solution of Recurrence Relation

- $c_n = 3c_{n-1} - 2c_{n-2}$, $c_1 = 5$, $c_2 = 3$
  - Characteristic equation:
    - $x^2 = 3x - 2$;
  - Get the root: 1,2
  - $C_n = u*1^n + v*2^n$
  - We have equations:
    - C1 = u+2v = 5
    - C2 = u+4v = 3
  - So: $C_n = 7-2^n$ (u = 7, v = -1)

Fibonacci Sequence

$f_1 = 1$

$f_2 = 1$

$f_n = f_{n-1} + f_{n-2}$

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Explicit formula for Fibonacci Sequence

The characteristic equation is $x^2 - x - 1 = 0$, which has roots:

$$s_1 = \frac{1+\sqrt{5}}{2} \quad and \quad s_2 = \frac{1-\sqrt{5}}{2}$$

Note: (by initial conditions) $f_1 = us_1 + vs_2 = 1 \ \ and \ \ f_2 = us_1^{\ 2} + vs_2^{\ 2} = 1$

which results:

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

函数的增长-算法分析初步

- 集合A上的关系R，令|A| = n
- 求该关系的传递闭包算法有：S1算法,S2算法
- 如何去判断哪个算法更好一些？
  - 时间开销: steps
    - T_{S1}函数；T_{S2}函数
  - 如何比较时间开销？
    - 看谁“增长得快”！

函数增长

N (数据集规模)	S1 (算法执行步数)	S2 (算法执行步数)
10	550	1250
50	63750	781250
100	505000	12500000

两个算法执行步数随着数据规模的变化而变化
不同的算法，变化的“剧烈程度”不同

需要一种数学工具通过执行步骤函数的处理来反映上述“剧烈程度”

函数的增长

- 定义函数T:N(或R)→R：
  - 数据规模->算法执行步骤数
- 针对上述两个算法：
  - $T_{S1}(n) = n^3/2 + n^2/2$ for algorithm S1
  - $T_{S2}(n) = n^4/8$ for algorithm S2

哪个好一些？

函数的增长速度

- 给定f和g是整数或实数集合到实数集合的函数
  - 如果存在正常数 c 和k，使得对于所有大于k的x，都有$|f(x)| \le C|g(x)|$
  - 我们称：
    - f 是 $O(g)$
    - f增长速度不高于g

实际上

- 可以做如下判断：
  - 函数 f 是$O(g)$ if $\lim_{n\to\infty}[f(n)/g(n)]=c<\infty$
  - if there exists constants $c \in N$ and $k \in N$ such that for all n, $f(n)\le cg(n)$
- 例如: let $f(n)=n^2$, $g(n)=n\lg n$，*则*:
  - f 不是$O(g)$, 因为$\lim_{n\to\infty}[f(n)/g(n)]= \lim_{n\to\infty}[n^2/n\lg n]= \lim_{n\to\infty}[n/\lg n]= \lim_{n\to\infty}[1/(1/n\ln 2)]=\infty$
  - g 是$O(f)$, 因为$\lim_{n\to\infty}[g(n)/f(n)]=0$

再例

- let $f(n)=n^2$, $g(n)=7n^2+9n-1$
  - $\lim_{n\to\infty}[f(n)/g(n)] = \lim_{n\to\infty}[n^2/(7n^2+9n-1)] = 1/7$
  - 所以： f 是O(g)

  - $\lim_{n\to\infty}[g(n)/f(n)] = \lim_{n\to\infty}[(7n^2+9n-1)/n^2] = 7$
  - 所以： g 是O(f)
- 我们称：f和g增长得一样快(同阶)

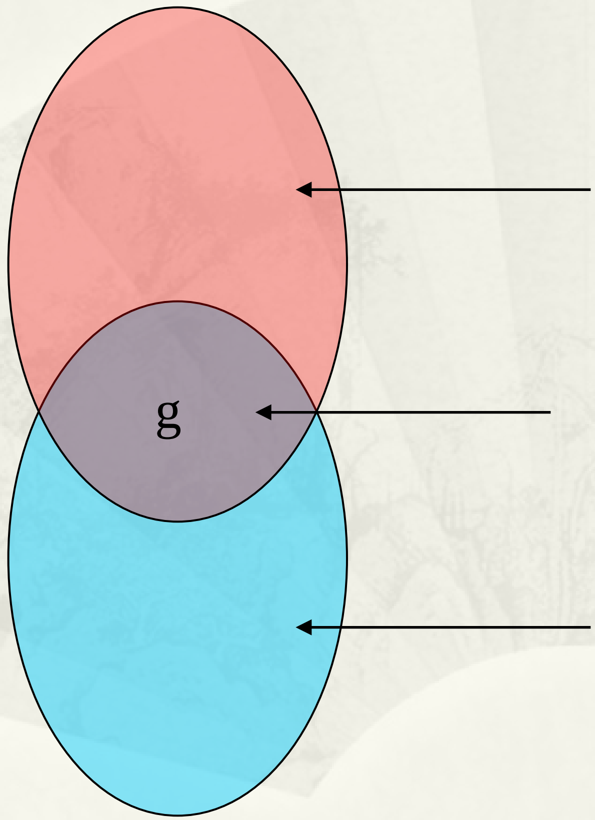
Ω和Θ

- 给定f 和g是整数或实数集合到实数集合的函数
  - 如果存在正常数 c 和k， 使得对于所有大于k的x，都有$|f(x)| \geq C|g(x)|$
  - 我们称：
    - f 是 $\Omega(g)$
    - f 增长速度不低于g

- 如果f 既是$O(g)$，又是$\Omega(g)$，则称 f 是$\Theta(g)$，即f和g是同阶的。

相对增长速度

给定函数g：



Ω(g):functions that grow at least as fast as g

Θ(g):functions that grow at the same rate as g

O(g):functions that grow no faster as g

Θ 关系

- $n^2/100+5n$ 是 $O(3n^4-5n^2)$, 它是$O(10n^4)$?
- $3n^4-5n^2$ 和$10n^4$增长得一样快
- 实际上，n^4 是所有和$3n^4-5n^2$同阶的函数中的最简形式
- $3n^4-5n^2$ 是$\Theta(n^4)$的
- 可以将Θ 看做一个等价关系

常见阶

* 一些常见的代表性阶
  * $\Theta(1)$, $\Theta(n)$, $\Theta(n^2)$, $\Theta(n^3)$, $\Theta(\log(n))$, $\Theta(n\log(n))$, $\Theta(2^n)$

范例

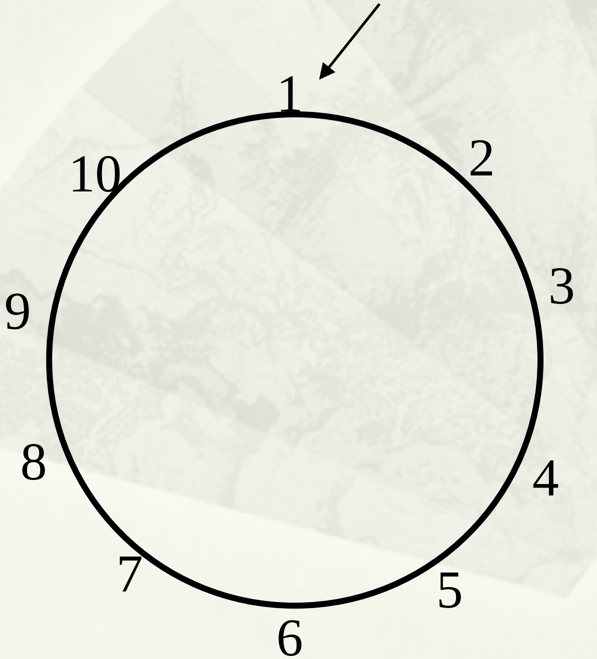
- 从低到高排列
  - $\Theta(1000000)$
  - $\Theta(n^{0.2})$
  - $\Theta(n+10^7)$
  - $\Theta(n\lg(n))$
  - $\Theta(1000n^2-n)$
  - $\Theta(1.3^n)$

教材和练习

- 教材：
  - 3.1.3；3.2；3.3；
  - 7.1
- 扩展阅读：7.3
- 练习：
  - P142：2；12；22；38
  - P349：7；14；24；36
  - P360：4(a,c,e,g)

Josephus's problem

- Given n people, *k*th man will be executed.
- Find the position to survive, J(n, k)



J(10,2) = 5;

J(11,2) = 7;

Josephus's problem

- $J(n, 2)$ is denoted by $J(n)$
- //Thinking recursively

$J(1) = 1$

$J(2n) = 2J(n)-1$

$J(2n+1) = 2J(n)+1$

$\boldsymbol{J(2^m+l) = 2l+1}$

$\boldsymbol{(0 \leq l < 2^m)}$