算法设计与分析

主讲教师：邵蓥侠

联系方式：shaoyx@bupt.edu.cn

个人主页：https://shaoyx.github.io/

渐进性复杂性分析

递归方程渐近阶的求解

编程作业说明

- CCF计算机软件能力认证（简称CCF CSP认证），鼓励学生报名参加CSP考试。
- CSP成绩抵平时的编程作业成绩。
  - CSP成绩300分及以上可替代编程实验成绩。
  - 300分对应编程实验成绩80分，380分对应满分。
  - 对CSP成绩在300分以上但没达到380的学生，允许选做一部分上机作业，以提高平时成绩。

递归方程解的渐近阶的求法

1. **代入法**：先推测递归方程的显式解，然后用数学归纳法证明这一推测的正确性。那么显式解的渐近阶即为所求。
2. **迭代法**：通过反复迭代，将递归方程的右端变成一个级数，然后求级数的和，再估计和的渐近阶；或者不求级数和而直接估计级数的渐近阶，从而达到对递归方程解的渐近阶的估计。
3. **套用公式法**：针对形如：$\mathrm{T}(n) = aT\left(\frac{n}{b}\right) + f(n)$的递归方程，给出三种情况下方程解的渐近阶的三个相应估计公式。
4. **差分方程法**：有些递归方程可看成一个差分方程，因而可以用解差分方程（初值问题）的方法解递归方程。然后对得到的解作渐近阶的估计。
5. **母函数法**：这是一个有广泛适用性的方法。它不仅可以用来求解线性常系数高阶齐次和非齐次的递归方程，而且可以用来求解线性变系数高阶齐次和非齐次的递归方程，甚至可以用来求解非线性递归方程。方法的基本思想是设定递归方程解的母函数，建立一个关于母函数的可解方程，将其解出，然后返回递归方程的解。

代入法

- 猜测技术：对递推关系式估计一个上限，然后（用数学归纳法）证明它正确，关键是寻找一个边界条件。

求证$\mathrm{T}(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$ 的渐近阶。

取整函数的若干性质

取整函数

- $\lfloor x \rfloor$：不大于x的最大整数；$\lceil x \rceil$：不小于x的最小整数。
- $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$;
- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;
- 对于$n \ge 0$，$a, b > 0$，有：
- $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$; $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;
- $\lceil a/b \rceil \le (a+(b-1))/b$; $\lfloor a/b \rfloor \ge (a-(b-1))/b$;
- $f(x)=\lfloor x \rfloor$, $g(x)=\lceil x \rceil$为单调递增函数。

对数函数的若干性质

对数函数

- $\log n = \log_2 n$;
- $\lg n = \log_{10} n$;
- $\ln n = \log_e n$;
- $\log^k n = (\log n)^k$;
- $\log \log n = \log(\log n)$;
- for a>0, b>0, c>0, $a = b^{\log_b a}$

对数函数的若干性质

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

代入法

- 猜测技术：对递推关系式估计一个上限，然后（用数学归纳法）证明它正确，关键是寻找一个边界条件。

求证$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$ 的渐近阶$O(n\log n)$。

- 推测T(n)=O(n *log* n)，即推测存在正的常数C和自然数n_0，使得当$n \geq n_0$时有

$$T(n) \leq Cn\log n \quad (1-1)$$

- 假设假设当$2^{k-1}n_0 \leq n \leq 2^k n_0$ ，$k \geq 1$时，(1-1)成立。
- 那么，当$2^k n_0 \leq n \leq 2^{k+1} n_0$时，有：

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2C\lfloor n/2 \rfloor \cdot \log(\lfloor n/2 \rfloor) + n \\ &< 2C \cdot \frac{n}{2} \cdot \log\frac{n}{2} + n \\ &= Cn\log n - Cn + n \\ &= Cn\log n - (c-1)n \\ &\leq Cn\log n \end{aligned}$$

算法复杂性分析中常见函数

- 这个方法的局限性在于它只适合容易推测出答案的递归方程或善于进行推测的高手。推测递归方程的正确解，没有一般的方法，得靠经验的积累和洞察力。

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

迭代法

迭代展开递归方程的右端，使之成为一个非递归的和式，然后通过对和式的估计来达到对方程左端的解的估计。

求证$T(n) = 3T\left(\frac{n}{4}\right) + n$ 的渐近阶为$O(n)$

$$T(n) \le n + 3(n/4 + 3(n/4^2 + 3(n/4^3 + \cdots + 3(n/4^i + 3T(0))\cdots)))$$

$$= n + \frac{3}{4}n + \frac{3^2}{4^2}n + \frac{3^3}{4^3}n + \cdots + \frac{3^i}{4^i}n + 3^{i+1} \cdot T(0)$$

$$< 4n + 3^{i+1} \cdot T(0)$$

知$i \leqslant \log_4 n$ ，从而

$$3^{i+1} \le 3^{\log_4 n + 1} = 3^{\log_3 n \cdot \log_4 3 + 1} = 3n^{\log_4 3},$$

代入

$$T(n) < 4n + 3n^{\log_4 3} \cdot T(0)$$

$$S_n = \frac{a_1(1-q^n)}{1-q} \ (q \neq 1)$$

等比数列求和公式

迭代法

$$T(n)=\begin{cases}7 & n=1\\ 2T(n/2)+5n^2 & n>1\end{cases}$$

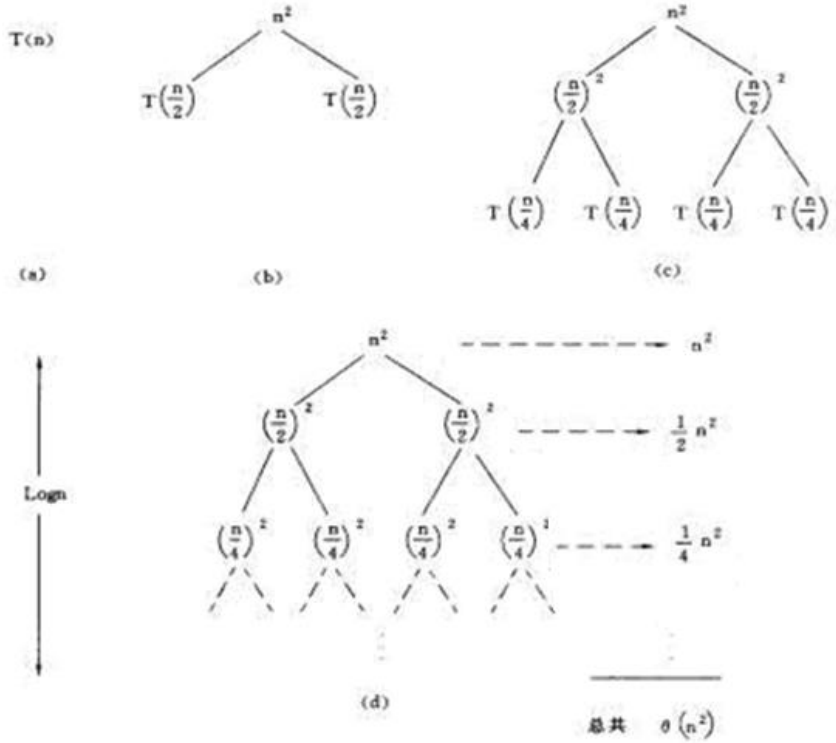
$$\begin{aligned}T(n) &= 2T(n/2)+5n^2\\ &= 2(2T(n/4)+5(n/2)^2)+5n^2\\ &= 2(2(2T(n/8)+5(n/4)^2)+5(n/2)^2)+5n^2\\ &= 2^k T(1)+2^{k-1}5(\frac{n}{2^{k-1}})^2+\cdots+2\times5(\frac{n}{2})^2+5n^2\end{aligned}$$

$$T(n)=7n+5\sum_{i=0}^{k-1}\left(\frac{n}{2^i}\right)^2=7n+5n^2(2-\frac{1}{2^{k-1}})=10n^2-3n\le 10n^2=O(n^2)$$

迭代法－递归树法

求证$\mathrm{T}(n)=2T\left(\frac{\mathrm{n}}{2}\right)+n^2$ 的渐近阶为$O(n^2)$

表格化的办法：先按横向求出每层结点的值之和，并记录在各相应层右端顶格处，然后从根到叶逐层地将顶格处的结果加起来便是我们要求的结果。照此，我们得到解的渐近阶为$\theta(n^2)$。



迭代法－递归树法

求证$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$ 的渐近阶为*O(nlogn)*

当我们累计递归树各层的值时，得到每一层的和都等于n，从根到叶的最长路径是

$$n \to \frac{2}{3}n \to (\frac{2}{3})^2 n \to \cdots \to 1$$

设最长路径的长度为k，则应该有

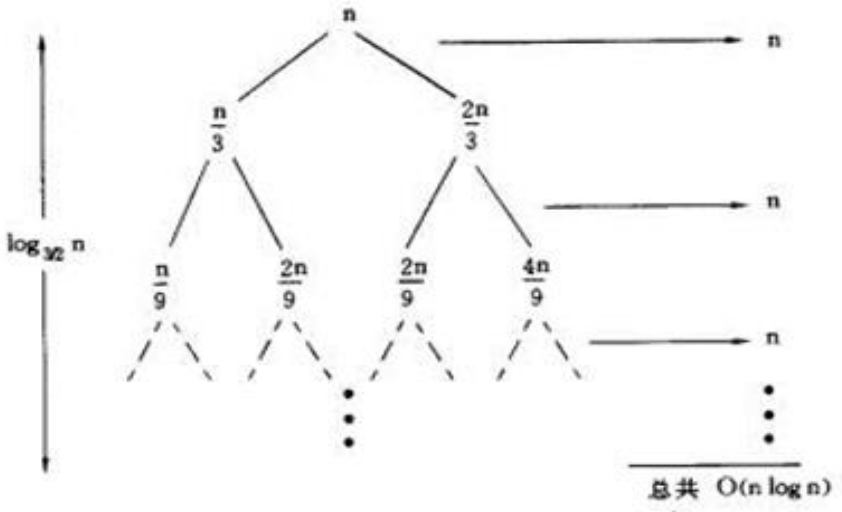
$$(\frac{2}{3})^k n = 1$$

得

$$k = \log_{3/2} n$$

于是

$$T(n) \le \sum_{i=0}^{k} n = (k+1)n = n(\log_{3/2} n + 1)$$



套用公式法

大小为n的原问题分成若干个大小为n/b的子问题，其中a个子问题需要求解，而cn^k是合并各个子问题的解需要的工作量。

$$T(n)=\begin{cases} c & n=1 \\ aT(n/b)+cn^k & n>1 \end{cases}$$

$$T(n)=\begin{cases} O(n^{\log_b a}) & a>b^k \\ O(n^k \log_b n) & a=b^k \\ O(n^k) & a<b^k \end{cases}$$

套用公式法 (一般化)

套用公式法

若递归方程形如：$T(n)=aT(n/b)+f(n)$，可根据$f(n)$的不同情况套用公式

1）若$\exists \varepsilon>0$，使得$f(n)=O(n^{\log_b a-\varepsilon})$，则$T(n)=\theta(n^{\log_b a})$

2）若$f(n)=\theta(n^{\log_b a})$，则$T(n)=\theta(n^{\log_b a}\cdot \log n)$

3）若$\exists \varepsilon>0$，使得$f(n)=\Omega(n^{\log_b a+\varepsilon})$，且$\exists\ c<1$，当n充分大时有

$a\,f(n/b) \le c\,f(n)$，则 $T(n)=\theta(f(n))$

母函数法

母函数法：设$a_0, a_1, \ldots a_n \ldots$是任意数列，称 $f(x)=a_0+a_1x+\ldots+a_nx^n+\ldots$为数列的母函数。若取数列为算法的时间复杂性函数{ T(n) },则其母函数为：　$f(x)=T(0)+T(1)\,x+\ldots+T(n)\,x^n\cdots$

若能由T(n)的递归方程求出T(n)的母函数,其展开式第n项系数即为T(n).

常系数线性递归关系

K阶常系数线性递归关系

$$c_0 f(m) + c_1 f(m-1) + \cdots + c_k f(m-k) = \varphi(m)$$

其中c_i为常数，并且c_0， c_k不为0；当

$$\varphi(m) = 0$$

时，称它为**K阶常系数线性齐次递归方程**。

齐次方程

齐次递归方程 $\sum_{i=0}^{k} c_i f(m-i) = 0$

其特征方程为：

$$c_0 x^k + c_1 x^{k-1} + \cdots + c_{k-1} x + c_k = 0$$

引理1： 若α是上式的r重根，则对每个$0 \le i < r, Am^i \alpha^m$是**递归方程**的几个解。

定理： 设$\alpha_1, \alpha_2, \ldots, \alpha_t$,是特征方程的$t$个不同的根，其重数分别为$r_1, r_2, \ldots, r_t$,则

$$\sum_{i=1}^{t} \sum_{j=0}^{r_i-1} A_{ij}\, m^j \alpha_i^m$$

是**递归方程**的一个通解，其中A_{ij}为常数，可由边界条件确定。

非齐次方程特解

两种形式的非齐次方程**特解**

（1）$\varphi(m)$ 为幂级数形式 $\varphi(m)=m^r$ ，则特解为

$$p_0+p_1m+p_2m^2+\cdots+p_rm^r$$

（2）$\varphi(m)$ 为指数函数类型 $\varphi(m)=a^m$

A: 若a不为其特征根，则特解为

$$pa^m$$

B: 若a为r重特征根，则特解为

$$\sum_{i=0}^{r}p_im^ia^m$$

Fibnoacci数列

$$\begin{cases} a_{n+2} = a_n + a_{n+1} & n \geq 0 \\ a_0 = 1, a_1 = 1 \end{cases}$$

该线性递归关系的特征方程 $x^2 = x + 1$

解得其根为 $$x = \frac{1 \pm \sqrt{5}}{2}$$

从而 $$a_n = c_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n$$

将边界条件代入，得

$$\begin{cases} c_1 = \frac{5+\sqrt{5}}{10} \\ c_2 = \frac{5-\sqrt{5}}{10} \end{cases}$$

代入并化简后即得

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

求解常系数线性递归关系

$$\begin{cases} f(n) = 8f(n-1) + 9 \cdot 10^{n-2} \\ f(1) = 8 \end{cases}$$

方程的特征根为8，对应的通解为 $c \cdot 8^n$

设 $p \cdot 10^{n-2}$ 为一个特解，将它代入到递归关系中，得

$$p \cdot 10^{n-2} = 8p \cdot 10^{n-3} + 9 \cdot 10^{n-2}$$

解得 p=45。故非齐次递关系的通解为

$$f(n) = c \cdot 8^n + 45 \cdot 10^{n-2}$$

利用边界条件 $f(1)$=8求得 $c = \frac{7}{16}$

得 $f(n) = \frac{7}{16} \cdot 8^n + 45 \cdot 10^{n-2} = 7 \cdot 8^{n-2} + 45 \cdot 10^{n-2}$

母函数法

例1 考虑线性变系数二阶齐次递归方程

$(n-1)T(n)=(n-2)T(n-1)+2T(n-2)$ ，$n\geqslant 2$

和初始条件$T(0)=0$，$T(1)=1$。根据初始条件，可计算$T(2)=0$，$T(3)=T(1)=1$。

设$\{T(n)\}$的母函数为：

$$A(x)=\sum_{n=0}^{\infty}T(n)x^n$$

由于$T(0)=T(2)=0$，$T(1)=1$，有：

$$\begin{aligned}A(x)&=x+T(3)x^3+T(4)x^4+\cdots+T(n)x^n+\cdots\\&=x(1+T(3)x^2+T(4)x^3+\cdots+T(n)x^{n-1}+\cdots)\end{aligned}$$

令 $B(x)=A(x)/x$，即：

$$B(x)=1+T(3)x^2+T(4)x^3+\cdots+T(n)x^{n-1}+\cdots$$

母函数法

$$B(x)=1+T(3)x^2+T(4)x^3+\cdots+T(n)x^{n-1}+\cdots$$

那么：

$$B'(x)=2T(3)x+3T(4)x^2+\cdots+(n-1)T(n)x^{n-2}+\cdots$$

$$xB'(x)=2T(3)x^2+3T(4)x^3+\cdots+(n-1)T(n)x^{n-1}+\cdots$$

$$B'(x)-xB'(x)=2T(3)x+(3T(4)-2T(3))x^2+(4T(5)-3T(4))x^3+\cdots+((n-1)T(n)-(n-2)T(n-1))x^{n-2}+\cdots$$

代入T（3）= 1，得

$$\begin{aligned}B'(x)(1-x)&=2T(1)x+2T(2)x^2+2T(3)x^3+\cdots+2T(n-2)x^{n-2}+\cdots\\&=2x(1+T(3)x^2+T(4)x^3+\cdots+T(n)x^{n-1}+\cdots)\\&=2xB(x)\end{aligned}$$

即

$$\frac{B'(x)}{B(x)}=\frac{2x}{1-x}=-2+\frac{2}{1-x}$$

指数函数的若干性质

指数函数

对于正整数m, n和实数$a>0$:

- $a^0=1$; $a^1=a$; $a^{-1}=1/a$;
- $(a^m)^n = a^{mn}$; $(a^m)^n = (a^n)^m$; $a^m a^n = a^{m+n}$;
- $a>1 \Rightarrow a^n$为单调递增函数;
- $a>1 \Rightarrow \lim_{n\to\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n)$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

基本初等函数求导公式

(1) $(C)'=0$

(2) $(x^{\mu})'=\mu x^{\mu-1}$

(3) $(\sin x)'=\cos x$

(4) $(\cos x)'=-\sin x$

(5) $(\tan x)'=\sec^2 x$

(6) $(\cot x)'=-\csc^2 x$

(7) $(\sec x)'=\sec x\tan x$

(8) $(\csc x)'=-\csc x\cot x$

(9) $(a^x)'=a^x\ln a$

(10) $(e^x)'=e^x$

(11) $(\log_a x)'=\dfrac{1}{x\ln a}$

(12) $(\ln x)'=\dfrac{1}{x}$,

(13) $(\arcsin x)'=\dfrac{1}{\sqrt{1-x^2}}$

(14) $(\arccos x)'=-\dfrac{1}{\sqrt{1-x^2}}$

(15) $(\arctan x)'=\dfrac{1}{1+x^2}$

(16) $(\operatorname{arc}\cot x)'=-\dfrac{1}{1+x^2}$

函数的和、差、积、商的求导法则

设 $u=u(x)$， $v=v(x)$ 都可导，则

(1) $(u\pm v)'=u'\pm v'$

(2) $(Cu)'=Cu'$（C 是常数）

(3) $(uv)'=u'v+uv'$

(4) $\left(\dfrac{u}{v}\right)'=\dfrac{u'v-uv'}{v^2}$

反函数求导法则

若函数 $x=\varphi(y)$ 在某区间 I_y 内可导、单调且 $\varphi'(y)\neq 0$，则它的反函数 $y=f(x)$ 在对应区间 I_x 内也可导，且

$$f'(x)=\frac{1}{\varphi'(y)} \quad \text{或} \quad \frac{dy}{dx}=\frac{1}{\dfrac{dx}{dy}}$$

母函数法

两边同时沿[0, x]积分，并注意到$B(0)=1$，有：

$$\ln B(x) = -2x - 2\ln(1-x)$$

$$B(x) = e^{-2x}/(1-x)^2$$

把$B(x)$展开成幂级数，得

$$B(x) = \sum_{n=0}^{\infty} \frac{(-2x)^n}{n!} \sum_{n=0}^{\infty} \binom{n+1}{1} x^n$$

$$= \sum_{i=0}^{\infty} \frac{(-1)^i \cdot 2^i}{i!} x^i \cdot \sum_{j=0}^{\infty} (j+1) \cdot x^j$$

$$= \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \frac{(-1)^i \cdot 2^i \cdot (j+1)}{i!} x^{(i+j)}$$

$$= \sum_{n=0}^{\infty} (\sum_{i=0}^{n} \frac{(-1)^i \cdot 2^i \cdot (n-i+1)}{i!}) \cdot x^n \qquad 0 \le x < 1$$

A）*麦克劳林公式*

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots$$

B）$\frac{1}{(1-x)^2} = \left(\frac{1}{1-x}\right)'$ *(导数)*

母函数法

从而

$$A(x)=\sum_{n=0}^{\infty}\left(\sum_{i=0}^{n}\frac{(-1)^i\cdot 2^i\cdot(n-i+1)}{i!}\right)\cdot x^{n+1}$$

$$=\sum_{n=1}^{\infty}\left(\sum_{i=0}^{n-1}\frac{(-1)^i\cdot 2^i\cdot(n-i)}{i!}\right)\cdot x^{n} \qquad 0\le x<1$$

最后得

$$T(n)=\sum_{i=1}^{n-1}\frac{(-1)^i\cdot 2^i\cdot(n-i)}{i!} \qquad n\ge 1$$

实例算法复杂性分析

顺序搜索算法

```
int  seqSearch ( const int A[ ],  int  N, int k )
{
/* 1*/          for( i = 0; i < N; i++ )
/* 2*/              if ( A[i] == k )
/* 3*/                  return i;
/* 4*/              }  /* end for-i */
/* 5*/          return  -1;
}
```

顺序搜索算法

(1) $T_{\max}(n) = \max\{\ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n\ \} = O(n)$

(2) $T_{\min}(n) = \min\{\ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n\ \} = O(1)$

(3) 在平均情况下，假设：

(a) 搜索成功的概率为p ($0 \le p \le 1$)；

(b) 在数组的每个位置i ($0 \le i < n$)搜索成功的概率相同，均为 p/n。

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

$$= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}\right) + n \cdot (1-p)$$

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

最大子段和

给定由n个整数(可能为负整数)组成的序列$a_1,a_2,\ldots,a_n$,求该序列形如 $\sum_{k=i}^{j} a_k$ 的子段和的最大值。当所有整数均为负整数时定义其最大子段和为 0 。依此定义，所求的最优值为:

$$\max\left\{0, \max_{1\le i\le j\le n}\sum_{k=i}^{j} a_k\right\}$$

例如当$(a_1, a_2, a_3, a_4, a_5, a_6)=(-2, 11, -4, 13, -5, -2)$时，最大子段和为 $\sum_{k=2}^{4} a_k = 20$ 。

最大子段和

1, -5, 3, 4, -2, -3, 10

以 1 开始的子数组：

1

1,-5

1,-5,3

1,-5,3,4

1,-5,3,4,-2

1,-5,3,4,-2,-3

1,-5,3,4,-2,-3,10

以-5 开始的子数组：

-5

-5,3

-5,3,4

-5,3,4,-2

-5,3,4,-2,-3

-5,3,4,-2,-3,10

以 3 开始的子数组：略...

以 4 开始的子数组：略...

以-2 开始的子数组：略...

以-3 开始的子数组：略...

以 10 开始的子数组：略...

枚举后再对所有的子数组求和，可以得到如下代码

最大子段和

Algorithm 1

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
                int  ThisSum,  MaxSum,  i,  j,  k;
/* 1*/          MaxSum = 0;   /* initialize the maximum sum */
/* 2*/          for( i = 0; i < N; i++ )  /* start from A[ i ] */
/* 3*/              for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 4*/                      ThisSum = 0;
/* 5*/                      for( k = i; k <= j; k++ )
/* 6*/                          ThisSum += A[
/* 7*/                      if ( ThisSum > M
/* 8*/                          MaxSum = Th
                    }  /* end for-j and for-i */
/* 9*/          return  MaxSum;
}
```

to A[j] */

um */

Detailed analysis is your first homework.

$$T(N) = O(N^3)$$

最大子段和

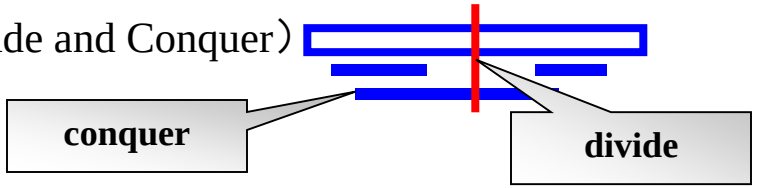
Algorithm 2

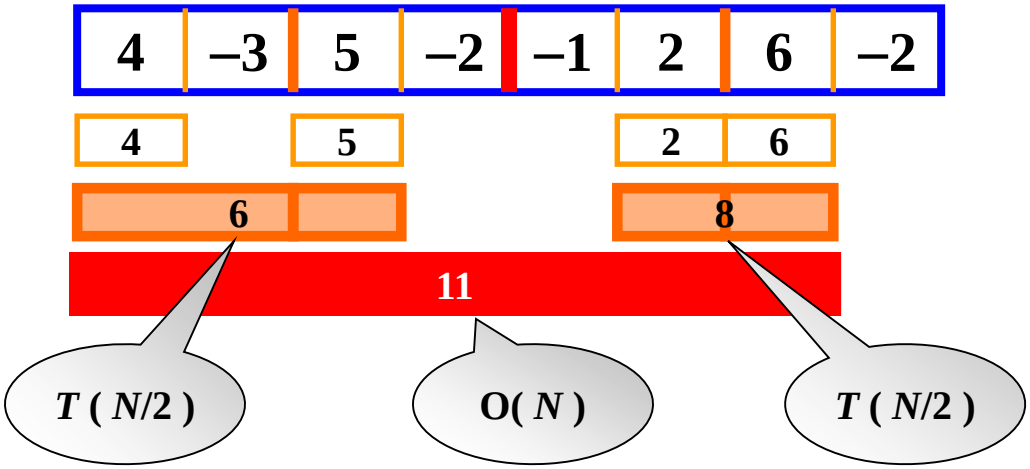
```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
            int  ThisSum,  MaxSum,  i,  j;
/* 1*/      MaxSum = 0;   /* initialize the maximum sum */
/* 2*/      for( i = 0; i < N; i++ )  {   /* start from A[ i ] */
/* 3*/          ThisSum = 0;
/* 4*/          for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 5*/                  ThisSum += A[ j ];  /* sum from A[ i ] to A[ j ] */
/* 6*/                  if ( ThisSum > MaxSum )
/* 7*/                      MaxSum = ThisSum;  /* update max sum */
            }  /* end for-j */
        }  /* end for-i */
/* 8*/  return  MaxSum;
}
```

$$T(N) = O(N^2)$$

最大子段和

Algorithm 3 分治法（Divide and Conquer）





$T(N) = 2\,T(N/2) + cN, \quad T(1) = O(1)$

$= 2\,[2\,T(N/2^2) + cN/2] + cN$

$= 2^k\,O(1) + ckN \quad$ where $N/2^k$

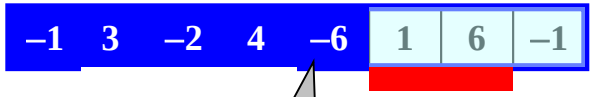
$= O(N \log N)$

The program would be discussed later.

最大子段和

Algorithm 4 On-line Algorithm

```
int MaxSubsequenceSum( const int  A[ ],  int  N )
{
            int  ThisSum, MaxSum, j;
/* 1*/      ThisSum = MaxSum = 0;
/* 2*/      for ( j = 0; j < N; j++ ) {
/* 3*/          ThisSum += A[ j ];
/* 4*/          if  ( ThisSum > MaxSum )
/* 5*/                  MaxSum = ThisSum;
/* 6*/          else if ( ThisSum < 0 )
/* 7*/                  ThisSum = 0;
            }  /* end for-j */
/* 8*/      return MaxSum;
}
```



At any point in time, the algorithm can correctly give an answer to the subsequence problem for the data it has already read.

$T(N) = O(N)$

A[ ] is scanned once only.

最大子段和——算法比较

Running times of several algorithms for maximum subsequence sum (in seconds)

Algorithm		1	2	3	4
Time		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
Input Size	N =10	0.00103	0.00045	0.00066	0.00034
	N =100	0.47015	0.01112	0.00486	0.00063
	N =1,000	448.77	1.1233	0.05843	0.00333
	N =10,000	NA	111.13	0.68631	0.03042
	N =100,000	NA	NA	8.0113	0.29832

Note: The time required to read the input is not included.

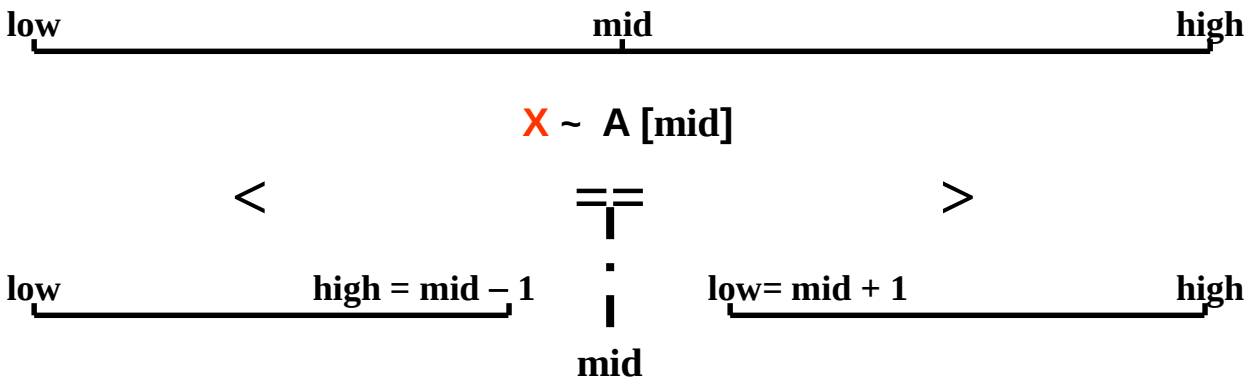
二分搜索Binary Search

Given: $A[0] \le A[1] \le \ldots\ldots \le A[N-1]$; X

Task: Find X

Output: i if X = = A [i]

–1 if X is not found



二分搜索Binary Search

```
int BinarySearch ( const ElementType  A[ ],
                                      ElementType  X,  int  N )
{
            int  Low, Mid, High;
/* 1*/      Low = 0;  High = N - 1;
/* 2*/      while ( Low <= High ) {
/* 3*/          Mid = ( Low + High ) / 2;
/* 4*/          if ( A[ Mid ] < X )
/* 5*/                  Low = Mid + 1;
               else
/* 6*/                  if ( A[ Mid ] > X )
/* 7*/                      High = Mid - 1;
                        else
/* 8*/                      return  Mid; /* Found */
            }  /* end while */
/* 9*/      return  NotFound; /* NotFound is defined as -1 */
}
```

$T_{worst}(N) = O(\log N)$

Very useful in case the data are **static** and is in **sorted** order (e.g. find words from a dictionary).

Euclid's Algorithm

Euclid's Algorithm: computing the greatest common divisor (Gcd).

```
unsigned int Gcd ( unsigned int  M,  unsigned int  N )
{
            unsigned int Rem;
/* 1*/      while ( N > 0 ) {
/* 2*/          Rem = M % N;
/* 3*/          M = N;
/* 4*/          N = Rem;
            } /* end while */
/* 5*/      return M;
}
```

$T(N) = ?$

$T(N) < 1 + T(M/2)$

$< 2 + T(N/2)$

$\cdots\cdots$

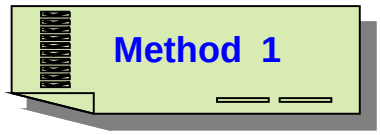
$< k + T(N/2^k)$

【Theorem】 If $M > N$, then (M mod N) $< M / 2$.

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

$$T_{avg}(N) = \frac{12 \ln 2 \ln N}{\pi^2} + 1.47$$

Checking Your Analysis

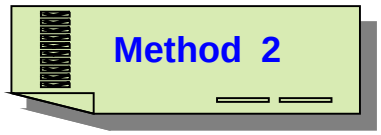


When $T(N) = O(N)$, check if $T(2N)/T(N) \approx 2$

When $T(N) = O(N^2)$, check if $T(2N)/T(N) \approx 4$

When $T(N) = O(N^3)$, check if $T(2N)/T(N) \approx 8$

… …



When $T(N) = O(f(N))$, check if

$$\lim_{N \to \infty} \frac{T(N)}{f(N)} \approx \text{Constant}$$

算法分析的基本法则

非递归算法：

（1）for / while 循环

- 循环体内计算时间*循环次数；

（2）嵌套循环

- 循环体内计算时间*所有循环次数；

（3）顺序语句

- 各语句计算时间相加；

（4）if-else语句

- if语句计算时间和else语句计算时间的较大者。

算法分析的基本法则

递归算法

运用递归方程进行求解。

1. **代入法**
2. **迭代法**
3. **套用公式法**
4. **差分方程法**
5. **母函数法**

最优算法

- 问题的计算时间下界为$\Omega(f(n))$，则计算时间复杂性为$O(f(n))$的算法是最优算法。
- 例如，
  - 最大子段和问题
  - 排序问题的计算时间下界为$\Omega(n\log n)$，计算时间复杂性为$O(n\log n)$的排序算法是最优算法。堆排序算法是最优算法。

本章小结

- 算法与程序的联系及区别
- 算法复杂性分析 – 时间和空间复杂性
- 渐近算法复杂性分析
  - $O, \Omega, \Theta, o, \omega$
  - 递归方程渐近阶求解

作业：证明与计算

- 证明关系
  - $3n^2 + 4n^3 = O(n^3)$
  - 推导递归关系$T(n) = \begin{cases} c, & n = 1 \\ aT\left(\frac{n}{b}\right) + cn^k, & n > 1 \end{cases}$的渐近阶
- 求解线性递归关系：
  - $T(n) =$

$$\begin{cases} T(n+1) = 3T(n) + 10T(n-1) + 6^{n-1}, n \geq 1 \\ T(0) = 2, \quad T(1) = 3 \end{cases}$$

作业：算法分析

1. 分析最大字段和算法1的时间复杂度。