数据结构与算法

陈越编写

高等教育出版社出版

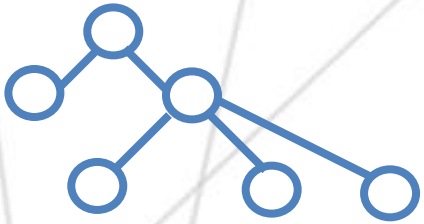
讲 解 人：张家琦

日　　期：hnzhangjq@126.com

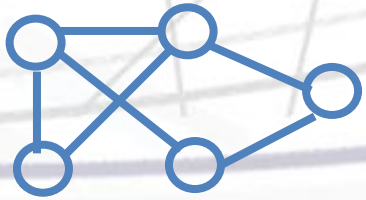
- 集合——数据元素间除“同属于一个集合”外，无其它关系

- 线性结构——一个对一个，如线性表、栈、队列

- 树形结构——一个对多个，如树



- 图形结构——多个对多个，如图



◆ 计算机如何解决一个具体问题？

- 从具体问题抽象出一个适当的数学模型；
- 设计一个解此数学模型的算法；
- 编出程序，进行测试、调整直至得到最终解答。

◆ 其中，建立数学模型是分析具体问题的重要过程，其步骤包括：

- 分析具体问题中操作对象；
- 找出这些对象间的关系，并用数学语言描述。

◆ 对于数值计算问题的解决方法，主要是用各种数学方程建立数学模型，例如：

- 求解梁架结构中应力的数学模型为线性方程组；
- 人口增长的数学模型为常微分方程。

◆ 对于非数值计算问题——数学模型？算法？

[例1.1] 该如何摆放书，才能让读者很方便地找到你手里这本《数据结构》？

【分析】

[方法1] 随便放----任何时候有新书进来，哪里有空就把书插到哪里。

查找效率极低！

[方法2] 按照书名的拼音字母顺序排放。

有时插入新书很困难！

[方法3] 把书架划分成几块区域，每块区域指定摆放某种类别的图书；在每种类别内，按照书名的拼音字母顺序排放。

可能造成空间的浪费！

图书馆的数目检索系统自动化问题

线性表

- 需要建立的数学模型：各种书目表。
  - 书目信息如：书名、作者、出版社、出版日期、书号、分类号、内容提要等。
  - 问题：如何表示和组织书目信息？
- 按照某个特定要求（如给定书名）对相关书目表进行查询的方法。

书目文件

001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
……	……		……

索引表

按书名

高等数学	001，003……
理论力学	002，……..
线性代数	004，……
……	……..

按作者名

樊映川	001, …
华罗庚	002, ….
栾汝书	004, ….
…….	…….

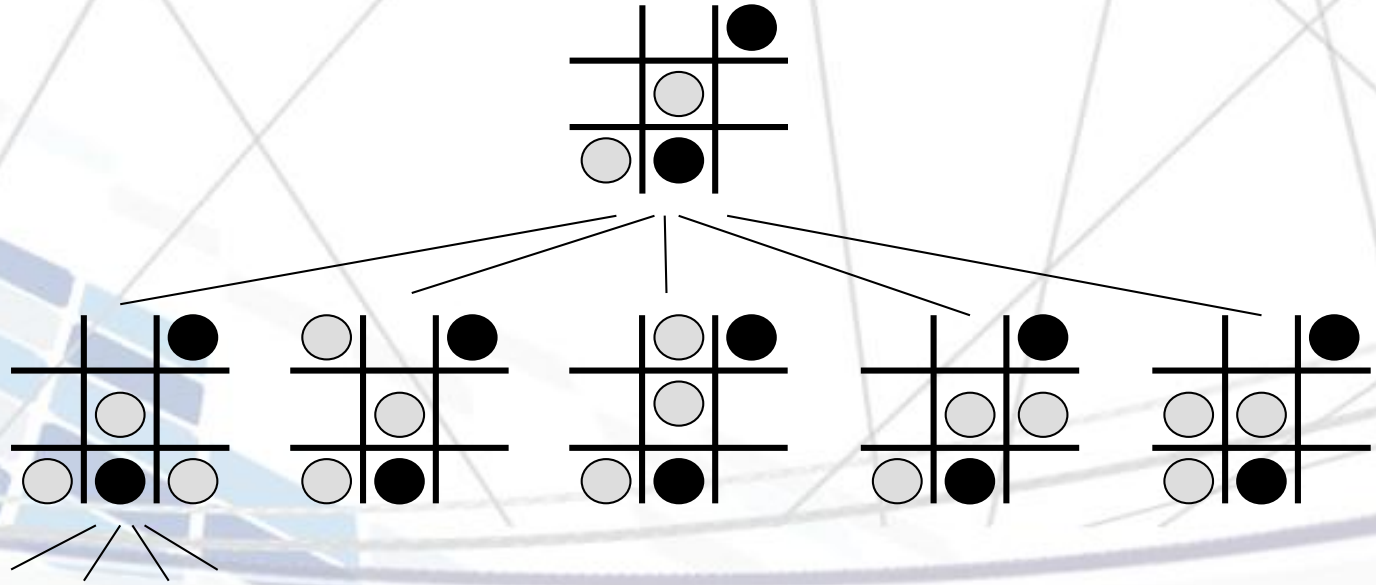
按分类号

L	002, …
S	001, 003,
……	……

树

补充案例1 ：计算机和人对弈问题

- 需要建立的数学模型：对弈树。
  - 操作对象：对弈过程中可能出现的棋盘状态（格局）。
  - 问题：如何表示、组织并保存动态变化的棋盘状态（格局）？
- 需要的算法：
  - 对弈／走棋操作的算法——使格局发生变化，由一种格局派生出另一种格局，并从多种可选路径中选择一种最优／合理路径，最后达到输或赢的状态。



图

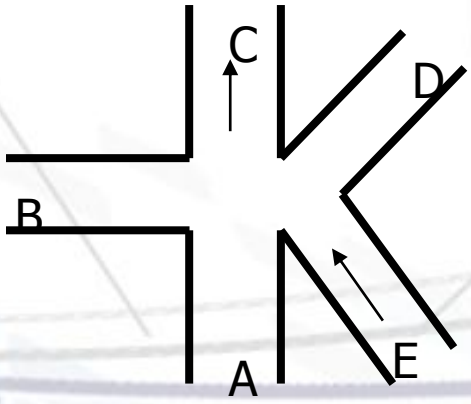
补充案例2：多叉路口交通灯的管理问题

- 目标：如何保证多叉路口的交通畅通有序，并使交通流量达到最大，且不会发生交通事故。
- 需要建立的数学模型：通路状态图。

需解决的问题：

- ①当某一条通路通行时，有哪些通路不能同时通行？
- ②当某一条通路通行时，有哪些通路可同时通行？
- ③如何表示和组织通路状态信息？

- 需要的算法：
  - 图的顶点的染色问题，每个顶点染一种颜色，有线相连的两个顶点不能具有相同颜色，总颜色数尽可能少。



[例1.2]：写程序实现一个函数PrintN，使得传入一个正整数为N的参数后，能顺序打印从1到N的全部正整数。

```
void PrintN ( int N )
{ int i;
  for ( i=1; i<=N; i++ )
    printf("%d\n", i );
  return;
}
```

```
void PrintN ( int N )
{ if ( N > 0 ){
    PrintN( N-1 );
    printf("%d\n", N );
  }
}
```

[例1.3] 多项式的标准表达式可以写为：

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

现给定一个多项式的阶数 n，并将全体系数存放在数组 a[ ] 里。请写程序计算这个多项式在给定点 x 处的值。

[方法1] 计算多项式函数值的直接法

```
double f( int n, double a[], double x )
{  /* 计算阶数为n，系数为a[0]...a[n]的多项式在x点的值 */
    int i;
    double p = a[0];
    for ( i=1; i<=n; i++ )
        p += a[i]*pow(x, i);
        return p;
}
```

[方法2] 秦九韶法

$$f(x) = a_0 + x\,(a_1 + x\,(a_2 + \ldots + x\,(a_n)\ \ldots)$$

```
double f( int n, double a[], double x )
{  /* 计算阶数为n，系数为a[0]...a[n]的多项式在x点的值 */
        int i;
        double p = a[n];
        for (i=n; i>0; i--)
                p = a[i-1] + x*p;
        return p;
}
```

- 即使解决一个非常简单的问题，往往也有多种方法，且不同方法之间的效率可能相差甚远
- 解决问题方法的效率
  - 跟数据的组织方式有关（如例1.1）
  - 跟空间的利用效率有关（如例1.2）
  - 跟算法的巧妙程度有关（如例1.3）

❖ 数据对象： 计算机要处理的事物，如例1中“图书” 。

❖ 操作：处理事物的动作集合，如例1中的“查找”和“插入”，例2的函数“求值”等。

❖ 算法： 操作的实现方法，如例1的按字母序排放的“查找”和“插入”、例2的“直接法”和“秦九韶法”等；

通常一个算法用一个函数来实现。

❖ 逻辑结构：数据对象的逻辑组织关系。分为“线性”、“树”和“图”。例1中按方法1来处理，就是把图书集看成是线性的结构；按方法3来处理，就是把图书集看成是树型的结构。

❖ 物理结构：数据对象信息在计算机内存中的存储组织关系。一般分为“顺序存储”和“链式存储”。

❖ 数据类型： 数据对象的类型确定了其“操作集”和“数据定义域”。

❖ 抽象数据类型： “抽象”的意思，是指我们描述数据类型的方法是不依赖于具体实现的，即数据对象集和操作集的描述与存放数据的机器无关、与数据存储的物理结构无关、与实现操作的算法和编程语言均无关。简而言之，抽象数据类型只描述数据对象集和相关操作集“是什么”，并不涉及“如何做到”的问题。

【定义】一个算法是解决某一类问题的步骤的描述。一般而言，算法应该符合以下五项要求：

(1) 输入：它接受一些输入（有些情况下不需要输入）；

(2) 输出：至少产生一个输出；

(3) 确定性：算法的每一步必须有充分明确的含义，不可以有歧义；

(4) 有限性：算法是一个有限指令集，并一定在有限步骤之后终止；

(5) 可行性：算法的每一步必须在计算机能处理的范围之内

- 另外，算法的描述可以不依赖于任何一种计算机语言以及具体的实现手段。可以用自然语言、流程图等方法来描述。
- 但是，用某一种计算机语言进行伪码描述往往使算法容易被理解，本书即采用C语言的部分语法作为描述算法的工具。

❖ 什么是“好”的算法？

❖ 具体衡量、比较算法优劣的指标主要有两个：

➢ 空间复杂度S(n) ——根据算法写成的程序在执行时占用存储单元的长度。这个长度往往与输入数据的规模 有关。空间复杂度过高的算法可能导致使用的内存超限，造成程序非正常中断。

○ 例1.2 的实现函数PrintN的递归算法S(n)太大。

➢ 时间复杂度 T(n) ——根据算法写成的程序在执行时耗费时间的长度。这个长度往往也与输入数据的规模 有关。时间复杂度过高的低效算法可能导致我们在有生之年都等不到运行结果。

○ 例1.3 的秦九韶算法的T(n)比较小。

- 我们经常关注下面两种复杂度：
  - 最坏情况复杂度： $T_{worst}(n)$
  - 平均复杂度： $T_{avg}(n)$
  - 显然：$T_{avg}(n) \le T_{worst}(n)$。
    对 $T_{worst}(n)$ 的分析往往比对 $T_{avg}(n)$的分析容易。
- 如果：

程序A执行了(3N+4)步，
程序B执行了(2N+2)步，
A一定比B慢吗？

- No！
- Why?

❖ 如何来“度量”一个算法的时间复杂度呢？

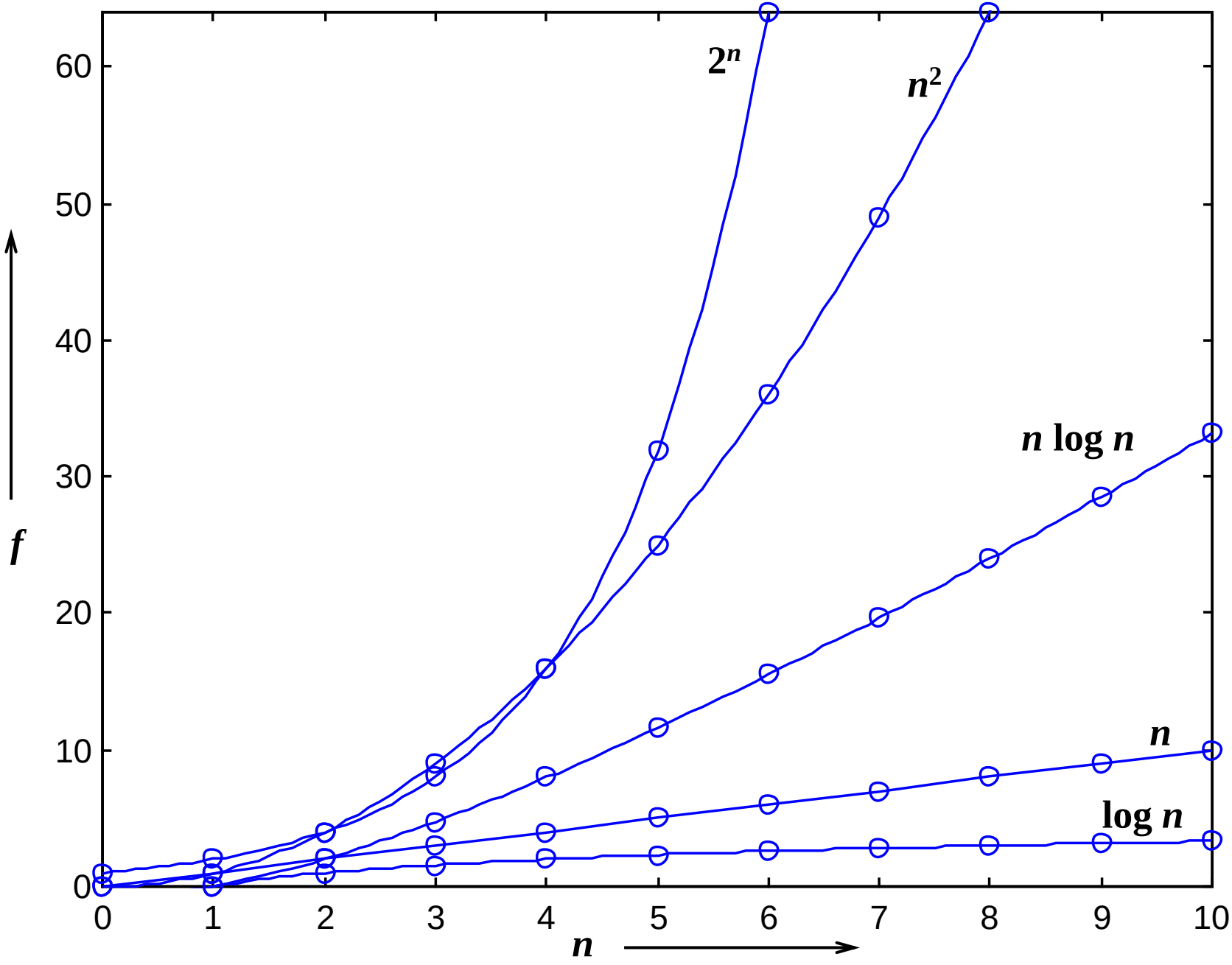
- 首先，它应该与运行该算法的机器和编译器无关；
- 其次，它应该与要解决的问题的规模 n 有关；
（有时，描述一个问题的规模需要多个参数）
- 再次，它应该与算法的“1步”执行需要的工作量无关！
- 所以，在描述算法的时间性能时，人们只考虑宏观渐近性质，
即当输入问题规模 n“充分大”时，观察算法复杂度随着 n 的“增长趋势”：
当变量n不断增加时，解决问题所需要的时间的增长关系。

❖ 比如：线性增长： $T(n) = c\,n$
即问题规模n增长到2倍、3倍……时，解决问题所需要的时间T(n)也是增长到2倍、3倍……（ 与c无关 ）

❖ 平方增长： $T(n) = c\,n^2$
即问题规模n增长到2倍、3倍……时，解决问题所需要的时间T(n)增长到4倍、9倍…… （ 与c无关 ）

❖ 表1.1 ： 常用函数增长表：

	输入规模n					
函数	1	2	4	8	16	32
1	1	1	1	1	1	1
$\log_2 n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log_2 n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40326	2092278988000	26313×10^{33}


2^n
n^2
$n \log n$
n
$\log n$
f
n
0
10
20
30
40
50
60
0
1
2
3
4
5
6
7
8
9
10

	每秒10亿指令计算机的运行时间表						
n	$f(n)=n$	$n \log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01μs	.03μs	.1μs	1μs	10μs	10sec	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84hr	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1sec
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121.36d	18.3min
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1yr	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171yr	$4*10^{13}$yr
1,000	1.00μs	9.96μs	1ms	1sec	16.67min	$3.17*10^{13}$yr	$32*10^{283}$yr
10,000	10μs	130.03μs	100ms	16.67min	115.7d	$3.17*10^{23}$yr	
100,000	100μs	1.66ms	10sec	11.57d	3171yr	$3.17*10^{33}$yr	
1,000,000	1.0ms	19.92ms	16.67min	31.71yr	$3.17*10^{7}$yr	$3.17*10^{43}$yr	

μs = 微秒 = 10^{-6} 秒　　min = 分　　yr = 年

ms = 毫秒 = 10^{-3} 秒　　sec = 秒　　hr = 小时　　d = 天

复杂度的渐进表示法

(4) 若干层嵌套循环的时间复杂度等于各层循环次数的乘积再乘以循环体代码的复杂度。

例如下列2层嵌套循环的复杂度是$O(N^2)$：

```
for ( i=0; i<N; i++ )
    for ( j=0; j<N; j++ )
    {   x = y*x + z; k++;   }
```

(5) if-else 结构的复杂度取决于if的条件判断复杂度和两个分枝部分的复杂度，总体复杂度取三者中最大。即对结构：

```
    if (P1)    /* P1的复杂度为 O(f1) */
        P2;    /* P2的复杂度为 O(f2) */
    else
        P3;    /* P3的复杂度为 O(f3) */
```

总复杂度为 max($O(f_1)$, $O(f_2)$, $O(f_3)$) 。

〖问题〗 给定 n个整数(可以是负数)的序列$\{a_1, a_2, \ldots, a_n\}$，求函数 $f(i, j) = \max(0, \sum_{k=i}^{j} a_k)$ 的最大值。

算法1

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
        int  ThisSum,  MaxSum,  i,  j,  k;
/* 1*/  MaxSum = 0;   /* 初始化最大子列和 */
/* 2*/  for( i = 0; i < N; i++ )  /* i是子列左端位置 */
/* 3*/      for( j = i; j < N; j++ ) {   /* j是子列右端位置*/
/* 4*/          ThisSum = 0;       /* ThisSum是从A[i]到A[j]的子列和 */
/* 5*/          for( k = i; k <= j; k++ )
/* 6*/              ThisSum += A[ k ];
/* 7*/          if ( ThisSum > MaxSum ) /* 如果刚得到的这个子列和更大 */
/* 8*/              MaxSum = ThisSum;  /*则更新结果*/
            } /* i，  j 循环结束  */
/* 9*/  return  MaxSum;
}
```

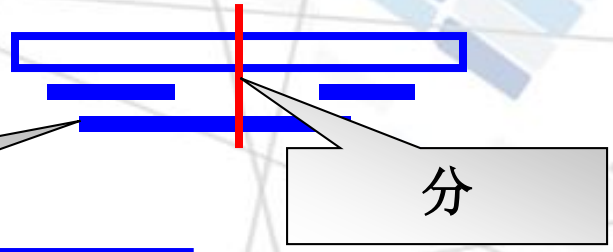
$$T(N) = O(N^3)$$

算法 2

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
        int  ThisSum,  MaxSum,  i,  j;
/* 1*/  MaxSum = 0;   /* 初始化最大子列和  */
/* 2*/  for( i = 0; i < N; i++ )  {     /* i是子列左端位置 */
/* 3*/      ThisSum = 0;        /* ThisSum是从A[i]到A[j]的子列和 */
/* 4*/      for( j = i; j < N; j++ ) {   /* j是子列右端位置*/
/* 5*/          ThisSum += A[ j ];
        /* 对于相同的i，不同的j，只要在j-1次循环的基础上累加1项即可 */
/* 6*/          if ( ThisSum > MaxSum )  /* 如果刚得到的这个子列和更大 */
/* 7*/              MaxSum = ThisSum;  /* 则更新结果 */
            } /* j循环结束 */
        } /* i循环结束 */
/* 8*/  return  MaxSum;
}
```

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

算法 3　分治法



4	–3	5	–2	–1	2	6	–2

4	5	2	6

6	8

11

$T(N/2)$　　$O(N)$　　$T(N/2)$

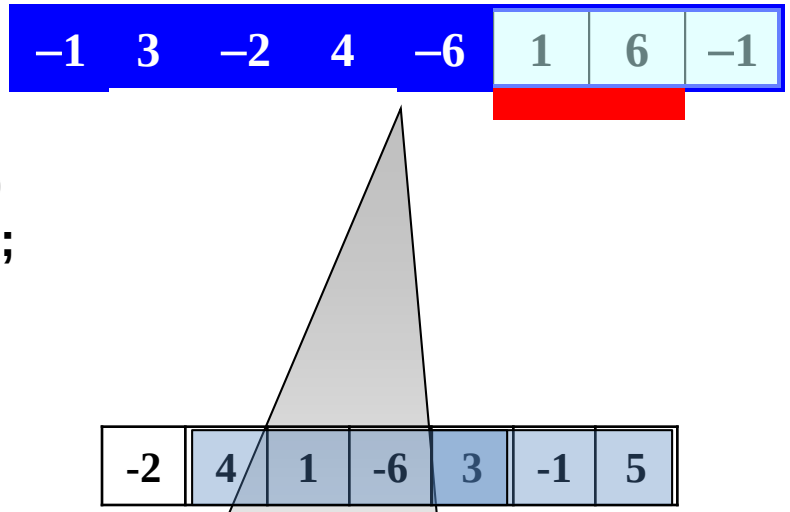
程序在教材 p.16-17.

$$T(N) = 2T(N/2) + cN, \quad T(1) = O(1)$$
$$= 2[2T(N/2^2) + cN/2] + cN$$
$$= 2^k O(1) + ckN \quad \text{此处 } N/2^k = 1$$
$$= O(N \log N)$$

结论对$N \neq 2^k$同样正确

算法 4 “在线”算法

```
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;
}
```



任何时刻，“在线”算法都可以对已经读入的数据序列给出正确的最大子列和答案.

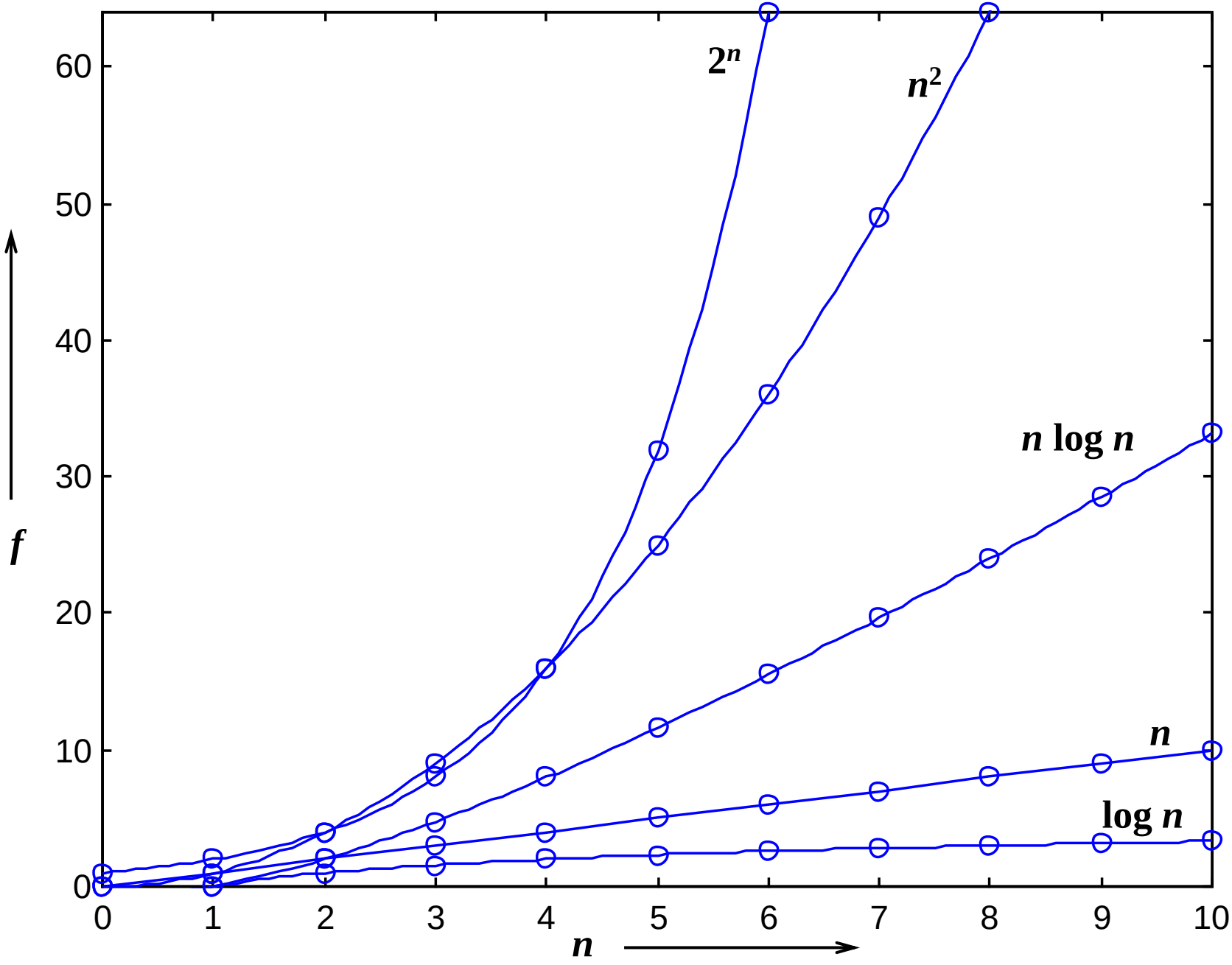
$T(N) = O(N)$

序列A[ ] *仅需*扫描一遍！

❖上述4种算法用于求最大子列和所需的运行时间比较 (秒)

算法		1	2	3	4
时间复杂性		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
子列大小	$N=10$	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

注: 不包括输入子列的时间.
NA – Not Acceptable, 不可接受的时间


2^n
n^2
$n \log n$
n
$\log n$
f
n
0
10
20
30
40
50
60
0
1
2
3
4
5
6
7
8
9
10