数据结构

刘晓梅

1、使用教材

《数据结构(C语言版)》 严蔚敏等 清华大学出版社

《数据结构(第2版)》 陈越 高等教育出版社

2、学习方法、教学方法

第一章 绪论

1.1 什么是数据结构

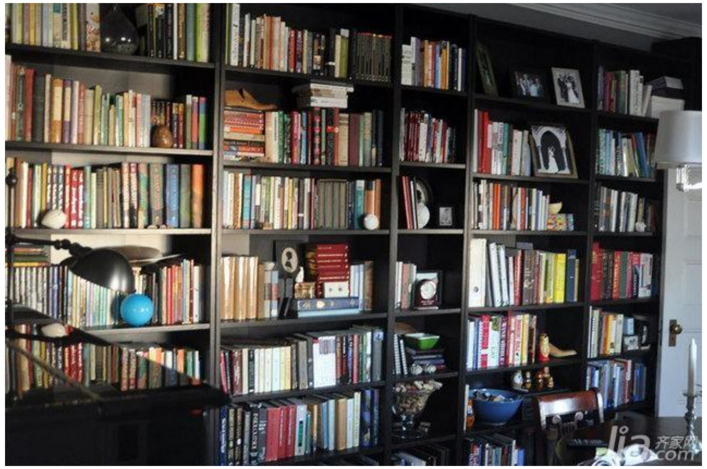
1.2 抽象数据类型

1.3 什么是算法

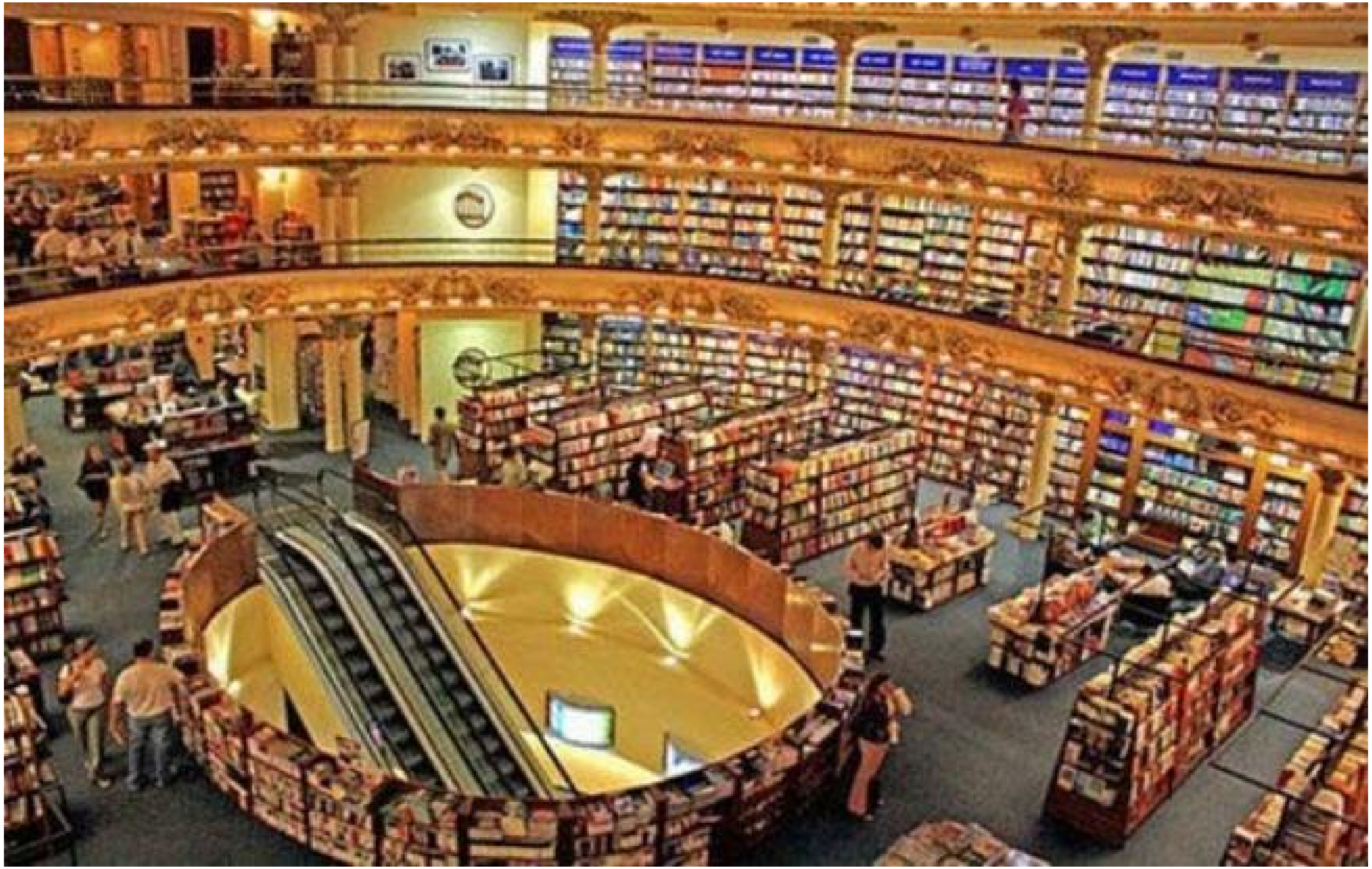
1.1 什么是数据结构

- “数据结构是数据对象，以及存在于该对象的实例和组成实例的数据元素之间的各种联系。这些联系可以通过定义相关的函数来给出。”
  - Sartaj Sahni，《数据结构、算法与应用》
- “数据结构是ADT（抽象数据类型Abstract Data Type）的物理实现。”
  - Clifford A.Shaffer，《数据结构与算法分析》
- “数据结构（data structure）是计算机中存储、组织数据的方式。通常情况下，精心选择的数据结构可以 带来最优效率的算法。”
  - 中文维基百科

例1：如何在书架上摆放图书？



例1：如何在书架上摆放图书?



例1：如何在书架上摆放图书？

图书的摆放要使得2个相关操作方便实现：

- 操作1：新书怎么插入？

- 操作2：怎么找到某本指定的书？

例1：如何在书架上摆放图书?

- 方法1：随便放
  - 操作1：新书怎么插入?
    - 哪里有空放哪里，一步到位!
  - 操作2：怎么找到某本指定的书?
    - ……累死

例1：如何在书架上摆放图书？

- 方法2：按照书名的拼音字母顺序排放
  - 操作1：新书怎么插入？
    - 新进一本《阿Q正传》……

  - 操作2：怎么找到某本指定的书？
    - 二分查找！

例1：如何在书架上摆放图书？

- 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
  - 操作1：新书怎么插入？
    - 先定类别，二分查找确定位置，移出空位
  - 操作2：怎么找到某本指定的书？
    - 先定类别，再二分查找

总结一：解决问题方法的效率，跟数据的组织方式有关

例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

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

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

循环实现　　　　　　　　　　递归

令 N = 100, 1000, 10000, 100000, ……

printN.cpp

总结二：解决问题方法的效率，
跟空间的利用效率有关

例3：写程序计算给定多项式在给定点x处的值

$$f(x) = a_{0+}a_1 x \qquad a_n \;\; {}_1 x^{n-1} \qquad a_n x^n$$

```
double f( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )
    p += (a[i] * pow(x, i));  return p;
}
```

$$f(x) = a_0 + x(a_1 + x(\cdots(a_{n-1} + x(a_n))\cdots))$$

```
double f( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )   p = a[i-1] + x*p;
  return p;
}
```

clock()：捕捉从程序开始运行到clock()被调用时所耗费的时间。这个 时间单位是clock tick，即“时钟打点”。

```
#include <stdio.h>  #include <time.h>

clock_t   start, stop;
/* clock_t是clock()函数返回的变量类型 */
double   duration;
/* 记录被测函数运行时间，以秒为单位 */
int main ()
{  /* 不在测试范围内的准备工作写在clock()调用之前*/

     start = clock();    /* 开始计时 */
     MyFunction();       /* 把被测函数加在这里 */
     stop = clock();      /* 停止计时 */
     duration = (double)(stop - start));
     /* 计算运行时间 */

     /* 其他不在测试范围的处理写在后面，例如输出duration的值 */
     return 0;
   }
```

例3：写程序计算给定多项式 在给 $f(x)=\sum_{i=0}^{9} i \cdot x^i$
定点 x = 1.1 处的值 f(1.1)

```
double f1( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )
    p += (a[i] * pow(x, i));  return p;
}
```

```
double f2( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )   p = a[i-1] + x*p;
  return p;
}
```

DXS.cpp

总结三：解决问题方法的效率，跟算法的巧妙程度有关

所以到底什么是数据结构？？？

- **数据对象**是相同性质的数据元素的集合
- 数据结构是相互之间存在一种或多种特定关系的数据元素
  - 逻辑结构
    - 线性结构
    - 树形结构
    - 图形结构
  - 物理存储结构
    - 连续存储
    - 非连续存储
- 数据对象必定与一系列加在其上的**操作**相关联
- 完成这些操作所用的方法就是**算法**

1.2 抽象数据类型（Abstract Data Type）

- 数据类型
  - 数据对象集
  - 数据集合相关联的操作集
- 抽象：描述数据类型的方法不依赖于具体实现
  - 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

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

例4：“矩阵”的抽象数据类型定义

- **类型名称：** 矩阵（Matrix）
- **数据对象集：** 一个M× N的矩阵$A_{M\times N}=(a_{ij})(i=1, \cdots, M; j=1, \cdots, N)$由M× N个三 元组< a, i, j >构成，其中a是矩阵元素的值，i是元素所在的行号，j是元素所在的列号。
- **操作集：** 对于任意矩阵A、B、C ∈ Matrix，以及整数i、j、M、N
  - Matrix Create(int M, int N)：返回一个M× N的空矩阵；
  - int GetMaxRow(Matrix A)：返回矩阵A的总行数；
  - int GetMaxCol(Matrix A)：返回矩阵A的总列数；
  - ElementType GetEntry(Matrix A, int i, int j)：返回矩 阵A的第i行、第j列的元素；
  - Matrix Add(Matrix A, Matrix B)：如果A和B的行、列数一 致，则返回矩阵C=A+B，否则返回错误标志；
  - Matrix Multiply(Matrix A, Matrix B)：如果A的列数等于B
  - 的行数，则返回矩阵C=AB，否则返回错误标志；

  …

1.3 什么是算法

- 算法（Algorithm）
  - 一个有限指令集
  - 输入（有些情况下不需要输入）
  - 输出
  - 有穷性：一定在有限步骤之后终止
  - 确定性：每一条指令必须
    - 有充分明确的目标，不可以有歧义
    - 计算机能处理的范围之内
    - 描述应不依赖于任何一种计算机语言以及具体的实现 手段

例5：选择排序算法的伪码描述

```
void SelectionSort ( int List[], int N )
{ /* 将N个整数List[0]...List[N-1]进行非递减排序 */
  for ( i = 0; i < N; i ++ ) {
     /* 从List[i]到List[N-1]中找最小元，并将其位置赋给MinPosition
     /* 将未排序部分的最小元换到有序部分的最后位置 */
   }
}
```

例5：选择排序算法的伪码描述

```
void SelectionSort ( int List[], int N )
{ /* 将N个整数List[0]...List[N-1]进行非递减排序 */
  for ( i = 0; i < N; i ++ ) {
     MinPosition = ScanForMin( List, i, N–1 );
     /* 从List[i]到List[N–1]中找最小元，并将其位置赋给MinPosition
     */  Swap( List[i], List[MinPosition] );
     /* 将未排序部分的最小元换到有序部分的最后位置 */
  }
}
```

抽象 ——

List到底是数组还是链表（虽然看上去很像数组）？

Swap用函数还是用宏去实现?

算法效率的度量

- 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。

- 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2

```c
void PrintN ( int N )
{ if ( N ){
      PrintN( N – 1 );
      printf(“%d\n”, N );
  }
  return;
}
```

……	100000	99999	99998	…… ……	1	……

```
PrintN(100000)
  PrintN(99999)
    PrintN(99998)
      PrintN(99997)
        ……  ……
              PrintN
              (0)
```

$$S(\mathbf{N}) = C \cdot \mathbf{N}$$

例3

```c
double f( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )
    p += (a[i] * pow(x, i));
  return p;
}
```

(1+2+……+n) =(n²+n)/2次乘法

$T(n) = C_1 n^2 + C_2 n$

```c
double f( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )
    p = a[i-1] + x*p;
  return p;
}
```

n次乘法!

$T(n) = C \cdot n$

算法的时间复杂度

算法中问题的规模以某种单位由1增到n时，运行算法中基本操作重复执行的次数是f(n)，算法的时间复杂度是T(n)=O(f(n))。

算法的时间复杂度

用算法表白--“爱你n遍”

```
void loveYou（int n）
{
    int i=1；
    while（i<=n）
    {
        i++;
        printf("I love you %d\n",n);
    }
    printf("I love you more than %d\n",n)
}
```

n=3000

T(3000)=1+3001+2*3000+1

时间开销与问题规模n的关系：T(n)=3n+3

- 问题一：是否可以忽略表达式的某些部分?

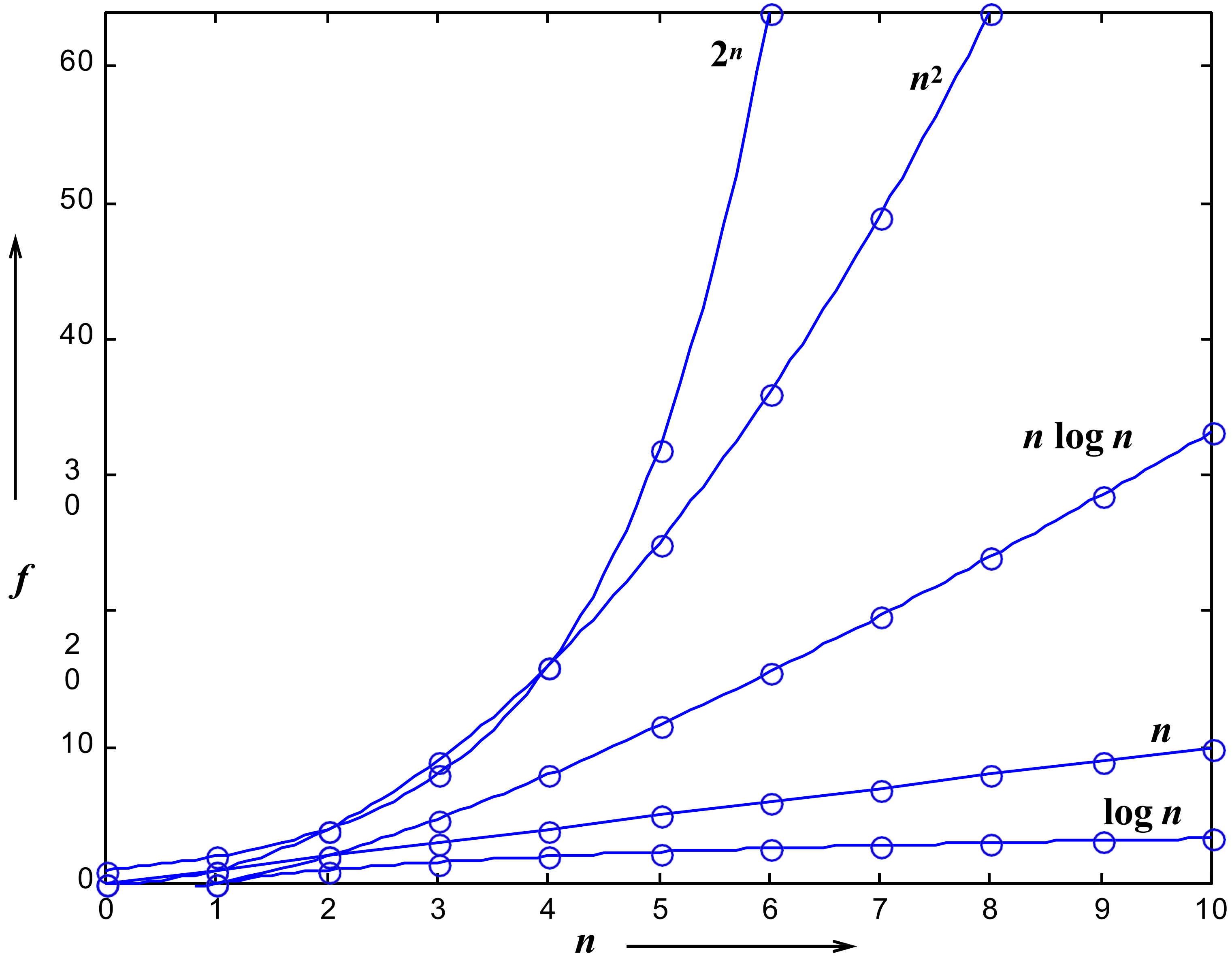
- 问题二：如果有好几千行代码，按这种方法需要一行一行数?

大O表示法:

- $T_1(n)=3n+3$
- $T_2(n)=n^2+3n+1000$
- $T_3(n)=n^3+n^2+9999999$
- 若n=3000，则
- $3n=9000$ vs $T_1(n)=9003$
- $n^2=9,000,000$ vs $T_2(n)=9,010,000$
- $n^2=9,000,000$ vs $T_2(n)=9,010,000$
- $n^3=27,000,000,000$ vs $T_3(n)=27,018,999,999$
- 结论**1**：可以只考虑阶数高的的部分
- $3n=9000$ vs $n=3000$ 同一个数量级
- 结论**2:**可以去掉系数
- $T_1(n)=O(n)$ $T_2(n)=O(n^2)$ $T_3(n)=O(n^3)$
- $T(n)=O(f(n))$

输入规模 n

函数	1	2	4	8	16	32
1	1	1	1	1	1	1
$\log n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log n$	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}



2n
n2
n log n
n
log n
f
n
0
10
2
0
3
0
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　　sec = 秒　　hr = 小时　　yr = 年

ms = 毫秒= 10^{-3}秒　　min = 分钟　　d = 日

```
void loveYou（int n）
{  //此处插入1000行顺序执行的代码。
   int i=1；
   while（i<=n）
   {
      i++;
      printf("I love you %d\n",n);
   }
   printf("I love you more than %d\n",n)
}
```

T(n)=3n+1003=O(n)

结论1：顺序执行的代码只会影响常数项，可以忽略

结论2:只需挑循环中的一个基本操作分析它的执行次数与n的关系即可

算法的时间复杂度

```
//算法2——嵌套循环型
void loveYou（int n）
{  //此处插入1000行顺序执行的代码。
   int i=1；
   while（i<=n）
   {
      i++;
      printf("I love you %d\n",n);
          for(int j=1;j<=n;j++)
               printf(“I am Iron Man”);
   }
   printf("I love you more than %d\n",n)
}
```

结论3:如果有多层嵌套循环，只需关注最深层循环循环了几次

算法的时间复杂度

```
//算法3void loveYou（int n)
{  //此处插入1000行顺序执行的代码。
   int i=1;
   while（i<=n)
   {
      i=i*2;
      printf("I love you %d\n",n);
   }
   printf("I love you more than %d\n",n)
}
```

结论3:如果有多层嵌套循环，只需关注最深层循环循环了几次

算法的时间复杂度

```
//算法4
void loveYou  (int flag[] , int n)
{   printf("I am Iron Man\n");
    for(int i=0;i<n;i++)
    {   if(flag[i]==n)
        {  printf("I love you  %d\n",n);
           break;}
    }
}
```

最好情况：T(n)

最坏情况：T(n)

平均情况：T(n)

算法的时间复杂度-应用举例

排序算法：

一、冒泡排序：

```
Void bubble_sort(int a[],int n)
{
        for(i=n-1,change=TRUE;i>=1&&change;--i)
    {
        change=FALSE;
        for(j=0;j<i;++j)
          if(a[j]>a[j+1])
                {a[j]↔a[j+1];change=TRUE;}
      }
}
```

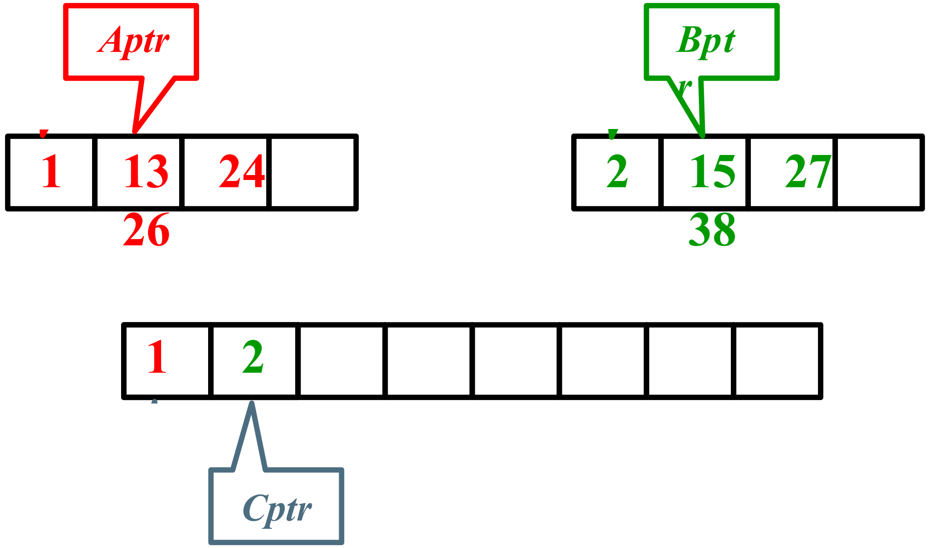
算法的时间复杂度-应用举例

排序算法：

二、归并排序：

```
Void bubble_sort(int a[],int n)
{
        for(i=n-1,change=TRUE;i>=1&&change;--i)
   {
        change=FALSE;
        for(j=0;j<i;++j)
          if(a[j]>a[j+1])
                {a[j]↔a[j+1];change=TRUE;}
      }
}
```

核心：有序子列的归并



如果两个子列一共有N个元素，则归并的时间复杂度是？

$$T(N) = O(N)$$

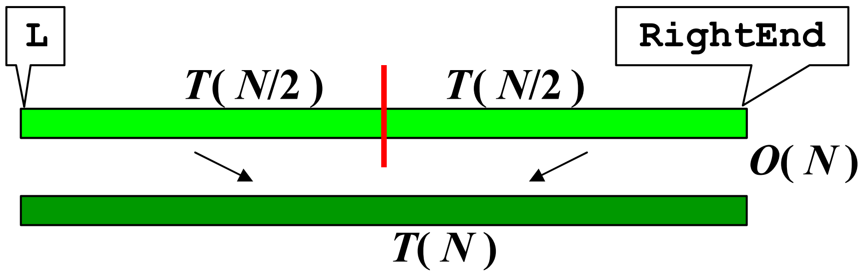
核心：有序子列的归并

```
/* L = 左边起始位置， R = 右边起始位置， RightEnd = 右边终点位置 */
void Merge( ElementType A[], ElementType TmpA[],
              int L, int R, int RightEnd )
{    LeftEnd = R - 1; /* 左边终点位置。假设左右两列挨着 */
     Tmp = L; /*  存放结果的数组的初始位置 */
     NumElements = RightEnd - L + 1;
     while( L <= LeftEnd && R <= RightEnd ) {
          if ( A[L] <= A[R] ) TmpA[Tmp++] = A[L++];

          else                TmpA[Tmp++] = A[R++];
          e
     }
          TmpA[Tmp++] = A[L++];
     while( L <= LeftEnd ) /* 直接复制左边剩下的 */
     while( R <= RightEnd ) /*直接复制右边剩下的 */
          TmpA[Tmp++] = A[R++];
     for( i = 0; i < NumElements; i++, RightEnd -- )
          A[RightEnd] = TmpA[RightEnd];
}
```

递归算法

- **分而治之**

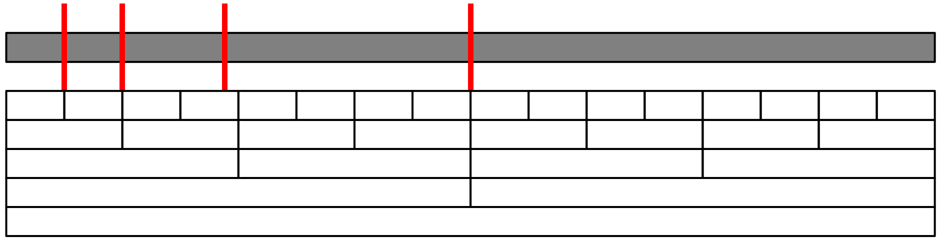



```
void MSort( ElementType A[], ElementType TmpA[],
            int L, int RightEnd )
{   int  Center;
    if ( L < RightEnd ) {
        Center = ( L + RightEnd ) / 2;
        MSort( A, TmpA, L, Center );
        MSort( A, TmpA, Center+1, RightEnd );
        Merge( A, TmpA, L, Center+1, RightEnd );
    }
}
```

$$T(N) = T(N/2) + T(N/2) + O(N) \Rightarrow T(N) = O(N \log N)$$

递归算法

- **如果只在Merge中声明临时数组**
  - void Merge(ElementType A[], int L, int R, int RightEnd)
  - void MSort(ElementType A[], int L, int RightEnd)



4 2 6 8 1 3 7 5

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

$\log_2^n$ $n\log_2^n$

$T(1)=0$ ———————————基本条件

$T(n)=2T(n/2) + n$

$T(16)= 2T(8) + 16$

$T(8)= 2T(4) + 8$

$T(4)= 2T(2) + 4$

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

$T(16)=64$

观看冒泡算法和归并算法动画 $n=50$

后续是作业

时间复杂度

以下算法的时间复杂度为（ ）

```
Void  fun(int n){
    int i=1;
    while(i<=n)
      i=i*2;
}
```

A. $O(n)$　　B. $O(n^2)$　　C. $O(n\log_2{}^{n})$　　D. $O(\log_2{}^{n})$

时间复杂度

设n是描述问题规模的非负整数，下面程序片段的时间复杂度是（ ）

```
X=2;
While(x<n/2)
  x=2*x;
```

A. $O(\log_2^n)$　B. $O(n^2)$　C. $O(n\log_2^n)$　D. $O(n)$

时间复杂度

求整数n（n>=0）阶乘的算法如下，其时间复杂度是（ ）

```
int fact(int n){
if(n<=1) return 1；
return n*fact(n-1);
}
```

A. $O(\log_2{}^n)$ B. $O(n)$ C. $O(n\log_2{}^n)$ D. $O(n^2)$

时间复杂度

已知两个长度分别为m和n的升序链表，若将他们合并为一个长度为m+n的降序链表，则最坏情况下的时间复杂度是（ ）

A. O(n)　B.O（m*n）　C. O(min(m,n))　D. O(max(m,n)

时间复杂度

下列程序的时间复杂度是（ ）

```
count=0;
for(k=1;k<=n;k*=2)
   for(j=1;j<=n;j++)
      count++;
```

A. $O(\log_2^n)$　　B. $O(n)$　　C. $O(n\log_2^n)$　　D. $O(n^2)$