算法与算法分析

一、算法的基本概念

二、算法分析

一、算法

1、算法定义

2、算法特性

3、算法的形式

4、算法举例

算法定义

- 对特定问题的求解步骤的描述，称为算法。

算法特性

- 输入 有0个或多个输入
- 输出 有一个或多个输出(处理结果)
- 确定性 每步定义都是确切无歧义的
- 有穷性 算法应在执行有穷步后结束
- 能行性 每一条运算应足够基本

算法形式

- 自然语言
- 流程图
- 框图
- C++的函数
  - 自顶向下，逐步求精

算法举例

写出将整型数组a中数据元素实现就地逆置的C++算法。

```
void reverse(int a[],int n){
    int temp;
    for (int i=0, j=n-1; i<j; i++,j--){
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;
    }
}
```

二、算法分析

1、算法的性能标准

2、算法时间效率的度量

3、时间复杂度及计算方法

4、时间复杂度简化计算方法

5、常见的时间复杂度及比较

6、其它情况

算法的性能标准

- 正确性 (Correctness) 算法应满足具体问题的需求。
- 可读性(Readability) 算法应该容易阅读。以有利于阅读者对程序的理解。
- 效率 效率指的是算法执行的时间和空间利用率。通常这两者与问题的规模有关。
- 健壮性 (Robustness) 算法应具有容错处理的功能。当输入非法数据时，算法应对其作出反应，而不应产生莫名其妙的输出结果。

时间性能的后期测试

- 在算法中的某些部位插装时间函数 **time ( )**，测定算法完成某一功能所花费时间。
- 例如，测算前面数组元素原地逆置算法 reverse的运行时间：

插装 time( ) 的计时程序

```
………//包括数组a的初始化等
double start, stop;
 time(&start);
 int k = reverse (a, n);
 time(&stop);
 double runTime = stop − start;
 cout << " " << n << " " << runTime << endl;
```

后期测试并不是客观的算法性能评估方法

- 程序运行绝对时间与软硬件环境有关，如编译程序生成的目标代码的质量、计算机程序指令系统的品质和速度等制约。

缺点： 1．必须执行程序

2．其它因素可以掩盖算法本质

时间性能的事前估计

- 一个特定算法的性能应与算法运行的环境无关

程序段	语句执行次数
m=0;	1
for (i=1;i<=n;i++)	1+n+1+n
m=m+1;	n
时间函数	T(n)=3n+3
渐近时间复杂度	T(n)=O(n)

程序段	语句执行次数
m=0;	1
for (i=1;i<=n;i++)	1,n+1,n
for (j =i; j<=n; j++)	n,n(n+1)/2+1,n(n+1)/2
m=m+1;	n(n+1)/2
时间函数	$T(n)=1.5n^2+4.5n+4$
渐近时间复杂度	$T(n)=O(n^2)$

例 求两个n阶方阵的乘积C = A×B

```
void MatrixMultiply (int A[n][n], int B[n][n],
        int C[n][n]) {
    for (int i = 0; i < n; i++)                 …2n+2
      for (int j = 0; j < n; j++) {             … n(2n+2)
        C[i][j] = 0;                            … n^2
        for (int k = 0; k < n; k++)             … n^2(2n+2)
          C[i][j] = C[i][j] + A[i][k] * B[k][j];
                                                … n^3
      }
}
```

$3n^3 + 5n^2 + 4n + 2$

- 算法中所有语句的执行次数之和是关于n的函数

  $T(n) = 3n^3 + 5n^2 + 4n + 2$

- 当n趋于无穷大时，时间复杂度的数量级称为算法的渐进时间复杂度

  $T(n) = O(n^3)$ — 大O表示法

算法渐进时间复杂度计算方法

1、将所有语句的执行次数之和作为算法的运行时间函数T(n)。

2、当$n\rightarrow\infty$时，时间函数的数量级表示称为渐进时间复杂度，简称时间复杂度。记为$T(n)=O(f(n))$

n指的是问题的规模，如被排序的数组中元素的个数，矩阵的阶数等。

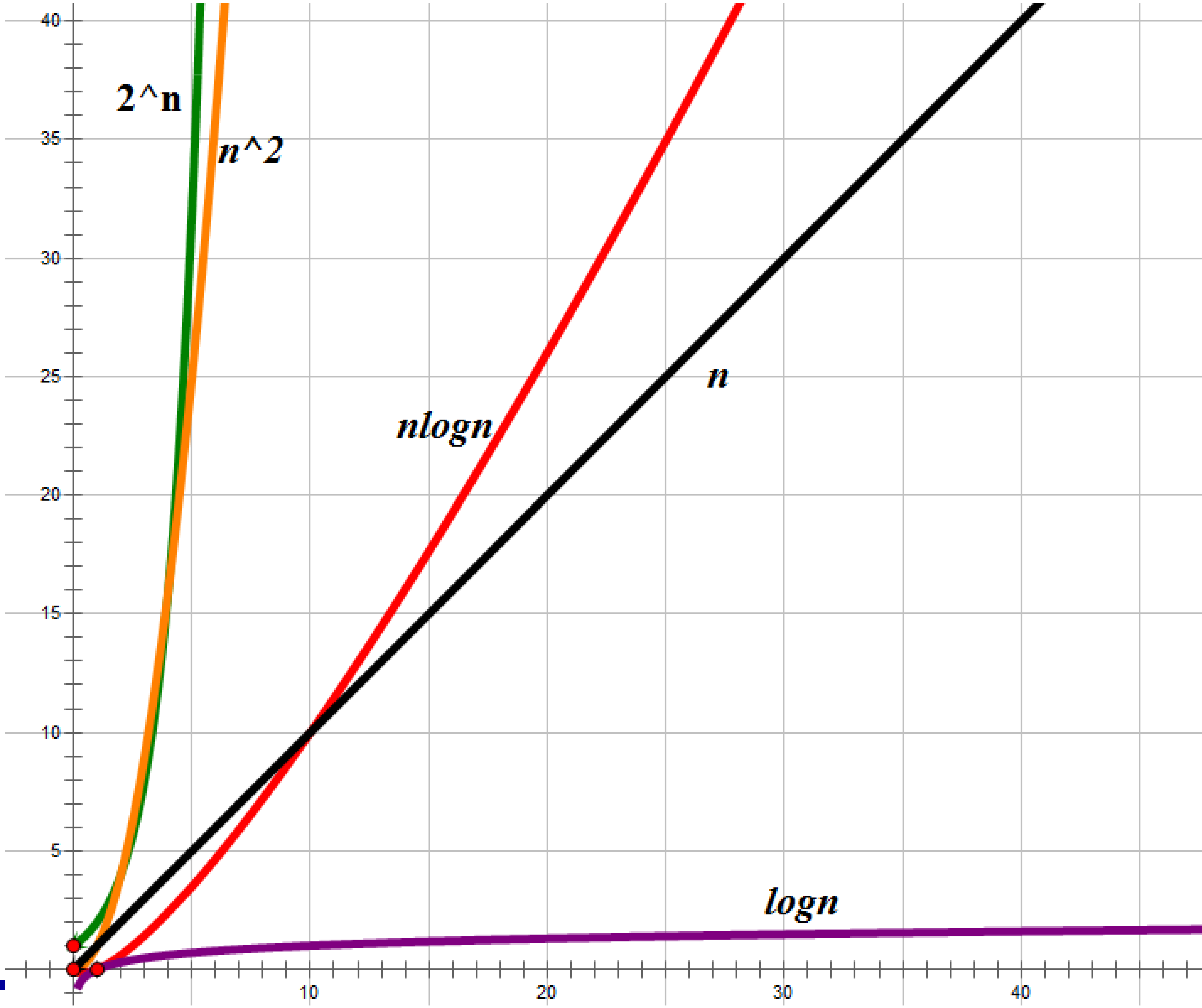
时间复杂度计算的简化方法

- 将某个关键操作的执行次数的数量级作为时间复杂度。
- 比如在矩阵相乘算法时，可选取这句（C[i][j] = C[i][j] + A[i][k] * B[k][j];）相乘累加的语句作为关键语句；
- 关键语句执行次数为n^3次；
- 算法间复杂度为 $T(n)=O(n^3)$。

常见的时间复杂度

常量阶 $O(1)$
对数阶 $O(\log n)$
线性阶 $O(n)$
线性对数阶 $O(n\log n)$
多项式阶 $O(n^m)$
指数阶 $O(2^n)$

常见的时间复杂度



各种时间复杂度从好到坏排序

$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n) < O(3^n) < O(n!)$

其它情况

- 多个问题规模
- 并列程序段----加法原则
- 嵌套程序段----乘法原则
- 时间复杂度不仅依赖于问题规模 **n**，还与输入实例的初始排列有关

多个问题规模

如：对两个长度分别为m和n的数组进行合并，问题的规模就有2个m和n。

$$\mathbf{T}(n, m) = \mathbf{O}(f(n,m))$$

并列程序段----加法原则

$$T(n, m) = T_1(n) + T_2(m) = O(\max(f(n), g(m)))$$

- 例子

变量计数

```
x = 0;  y = 0;
```

$T_1(n) = O(1)$

```
for ( int k = 0; k < n; k ++ )
    x ++;
```

$T_2(n) = O(n)$

```
for ( int i = 0; i < n; i++ )
    for ( int j = 0; j < n; j++ )
        y ++;
```

$T_3(n) = O(n^2)$

$$T(n) = T_1(n)+T_2(n)+T_3(n) = O(\max(1, n, n^2)) = O(n^2)$$

嵌套程序段---乘法规则

$$T(n, m) = T_1(n) * T_2(m) = O(f(n) * g(m))$$

- 例子

变量计数

```
x = 0;  y = 0;
```

$T_1(n) = O(1)$

```
for ( int k = 0; k < n; k ++ ){
    x ++;
for ( int i = 0; i < n; i++ )
   for ( int j = 0; j < n; j++ )
       y ++;
}
```

$T_2(n) = O(n)$

$T_3(n) = O(n^2)$

$T(n) = T_1(n)+（T_2(n)*T_3(n)） = T1(n)+O(n* n^2))$
$=O(max(1, n^3)) = O(n^3)$

```
void test (float x[ ][ ], int m, int n) {
    float sum [ ];
    for (int i = 0; i < m; i++) {      //x中各行
        sum[i] = 0.0;                  //数据累加
        for (int j = 0; j < n; j++)
            sum[i] += x[i][j];
    }
    for (i = 0; i < m; i++)         //打印各行数据和
        cout << i << " : " <<sum [i] << endl;
}
```

渐进时间复杂度为 $O(max\ (m*n, m))=O(m*n)$

- 时间复杂度与输入实例的初始排列有关
- 例子
- 选取最坏情况下的时间复杂度

顺序查找

```
int  SeqSearch (int a[ ],  int n, int k ) {
  int i =0;
       while (i <n && A[i] != k)
              i++;
        if (i==n)
          return -1;
       else
          return i;
}
```

- 取i++作为关键操作，它的执行次数不仅与 n 有关，还与 A[ ] 中各元素的取值以及 k 的取值有关。
- 最好情况时间复杂度----O(1)
- 最坏情况时间复杂度----O(n)
- 平均情况时间复杂度----O(n)

- 通常选取最坏情况下的时间复杂度作为算法效率量度

空间复杂性

- 算法所需空间和问题规模的函数。记为 **S(n)**。当 **n→∞** 时的时间复杂性，称为渐进空间复杂性。

空间复杂度度量

- 存储空间的固定部分
  程序指令代码的空间，常数、简单变量、定长成分(如数组元素、结构成分、对象的数据成员等)变量所占空间
- 可变部分
  尺寸与实例特性有关的成分变量所占空间、引用变量所占空间、递归栈所用空间、通过**new**和**delete**命令动态使用空间

作业

- 编写在长度为n的整型数组a中找出最大值的算法，并分析算法的时间复杂度。