数据结构与算法 概述

算法复杂度分析

算法复杂度分析

- 引入
- 时间复杂度分析
- 空间复杂度分析


问题的理解
数据结构设计
算法设计
算法分析
程序实现与测试

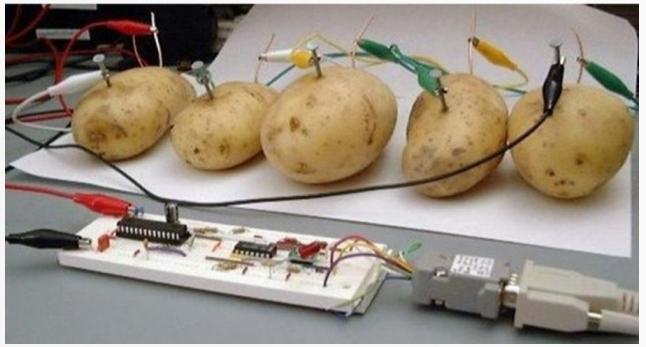
- 什么是一个“好”算法
  - 正确性
  - 更高的运行效率（时间复杂度）
  - 更低的资源占用（空间复杂度）

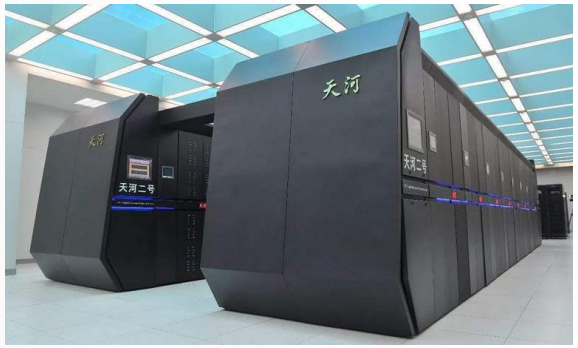
如何测量算法的时间/空间复杂度?

算法性能比较方法

- 编程后测试运行时间
- 编程前分析可能的运行时间

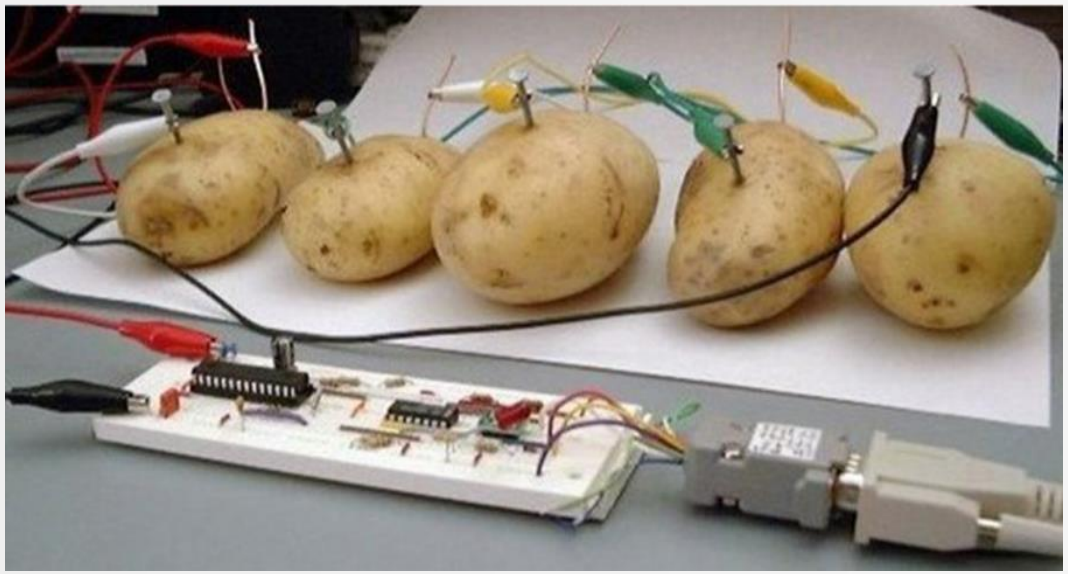
- 为什么事后测量法不可行?
  - 计算机性能差异
  - 编程语言性能差异
  - 编译器及解释器优化
  - 无法事后再统计（导弹制导算法、核弹拦截算法）





- **为什么事后测量法不可行?**

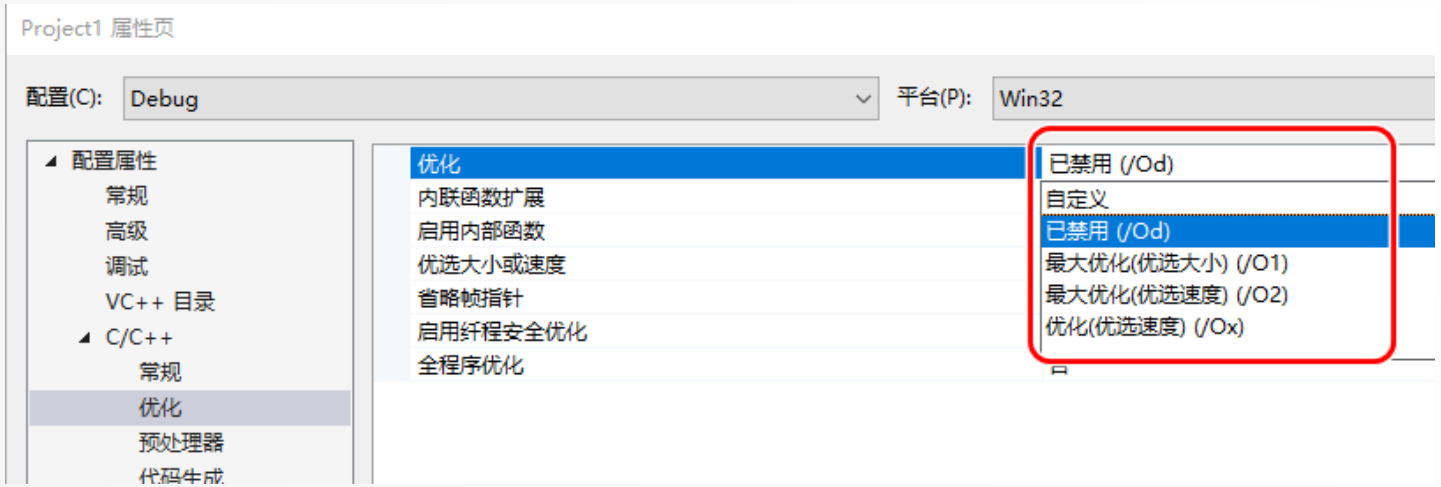
  - **计算机性能差异**



- 为什么事后测量法不可行?
  - 计算机性能差异
  - 编程语言性能差异

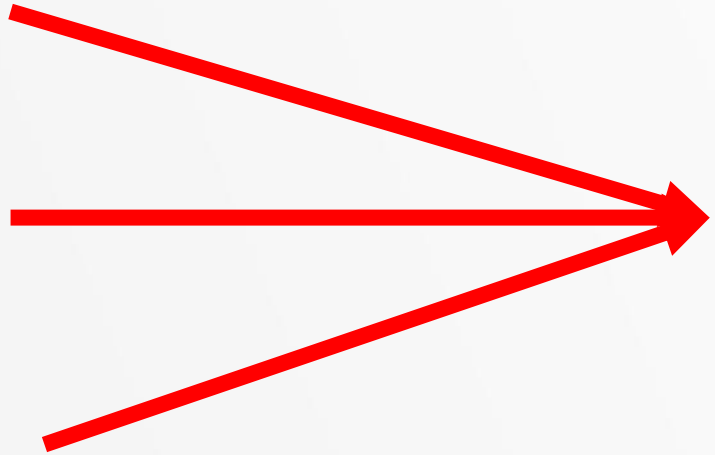
Programming Language
Python
C
C++
Java
C#
JavaScript
Visual Basic
SQL
Assembly language
PHP

- **为什么事后测量法不可行?**
  - **计算机性能差异**
  - **编程语言性能差异**
  - **编译器及解释器优化**



- 为什么事后测量法不可行?
  - 计算机性能差异
  - 编程语言性能差异
  - 编译器及解释器优化
  - 无法事后再统计（导弹制导算法、核弹拦截算法）

- 为什么事后测量法不可行?
  - 计算机性能差异
  - 编程语言性能差异
  - 编译器及解释器优化



排除外界因素干扰

  - 无法事后再统计（导弹制导算法、核弹拦截算法）

- 为什么事后测量法不可行?
  - 计算机性能差异
  - 编程语言性能差异
  - 编译器及解释器优化

  → 排除外界因素干扰

  - 无法事后再统计 → 事先估计算法复杂度

预估算法复杂度与问题规模n的关系

算法复杂度分析

- **算法复杂度：算法解决问题规模n所需的计算机资源的量**
  - **时间复杂度：算法解决问题规模n所需的时间资源的量**
  - **空间复杂度：算法解决问题规模n所需的空间资源的量**

算法复杂度数学描述

- **算法复杂度C：算法解决问题规模n所需的计算机资源的量**
  - **C=F(N, I, A)**
  - **N：问题规模**
  - **I：输入**
  - **A：算法**
  - **通常算法A是确定的，可省略**

算法复杂度数学描述简化

- **算法复杂度C：算法解决问题规模n所需的计算机资源的量**
  - **C=F(N, I)**
  - **N：问题规模**
  - **I：输入**

算法复杂度分析

- **时间复杂度：算法解决问题规模n所需的时间资源的量**
  - **T=T(N, I)**
  - **N：问题规模**
  - **I：输入**

算法复杂度分析

- **空间复杂度：算法解决问题规模n所需的空间资源的量**
  - **S=S(N, I)**
  - **N：问题规模**
  - **I：输入**
  - **I：通过划分等价类来去掉，后面详细介绍**

简化复杂度函数

核心思想：排除外在干扰，让算法运行在一台抽象的计算机上

◆该计算机支持k种元运算，分别记为O_1, O_2, ...O_k

◆该计算机每次执行各类元运算的时间分别为t_1, t_2, ... t_k

◆统计目标算法A中用到元运算O_i的次数，记为e_i

则$T = t_1 e_1 + t_2 e_2 + \ldots + t_k e_k$

简化复杂度函数1

核心思想：排除外在干扰，让算法运行在一台抽象的计算机上

◆该计算机支持k种元运算，分别记为O_1, O_2, ...O_k

◆该计算机每次执行各类元运算的时间分别为t_1, t_2, ... t_k

◆统计目标算法A中用到元运算O_i的次数，记为e_i

$e_i = e_i(N, I)$　　因此

$$T(N,I) = \sum_{i=1}^{k} t_i e_i(N,I)$$

简化复杂度函数2

计算步：算法中的语句执行次数

核心思想：统计抽象计算机中各类语句的执行次数

- ◆算法花费的时间与算法中语句的执行次数成正比
- ◆算法的计算步是问题规模n的一个函数T(n)
- ◆问题规模越大，算法的计算步越多，算法的耗时越大，复杂度也越高

算法的时间复杂度计算

```
3   void work_hard(int n) {    // n: 问题规模
4       int day = 1;           // day: 天数
5       while (day <= n) {
6           printf("第%d天坚持学习\n", day);
7           day++;
8       }
9       printf("第%d天我出师了\n", day);
10  }
```

```
int main()
{
    work_hard(140);
}
```

```
第133天坚持学习
第134天坚持学习
第135天坚持学习
第136天坚持学习
第137天坚持学习
第138天坚持学习
第139天坚持学习
第140天坚持学习
第141天坚持学习
第141天我出师了
```

语句步：

④：1次　　⑤：141次　⑥及⑦：140次　⑨：1次

T(140) = 1 + 141 + 2*140 + 1　　**T(n) = 3n + 3**

思考：能不能进一步优化？

复杂度渐进分析法

忽略低阶项

$T_1(n) = 3n + 3$

$T_2(n) = 5n^2 + 10n + 233$

$T_3(n) = 9n^3 + 2n^2 + 10n + 66666$

当 n = 1000时，	当 n = 1000时，去掉低阶项	继续去掉系数
$T_1(n) = 3003$	$T_1(n) \approx 3000$	$T_1(n) \approx 1000$
$T_2(n) = 501,0233$	$T_2(n) \approx 500,0000$	$T_2(n) \approx 100,0000$
$T_3(n) = 90,0207,6666$	$T_3(n) \approx 90,0000,0000$	$T_3(n) \approx 10,0000,0000$

渐近分析理论基础

当n→ ∞时，T(n) → ∞

此时若存在F(n)，使得 $\lim_{n\to\infty} \frac{T(n)-F(n)}{T(n)} = 0$

则 F(n) 为 T(n) 的渐进性态

渐进性态F(n)就是T(n)中增长最快的部分

直观上， F(n) 为 T(n) 中忽略低阶项后留下的主项

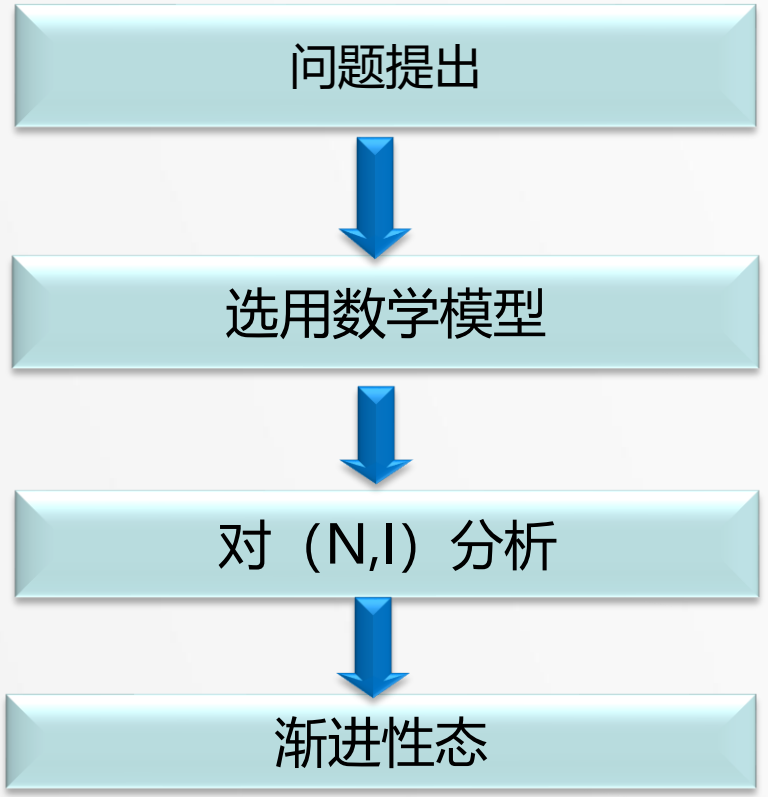
继续简化

- **时间复杂度：算法解决问题规模n所需的时间资源的量**
  - **T=T(N, I)**
  - **N：问题规模**
  - **I：输入**

思考：能不能把I简化掉?

继续简化

- **思路：划分等价类**
  - **最好输入**
  - **最坏输入**
  - **平均输入**



T(N, I)

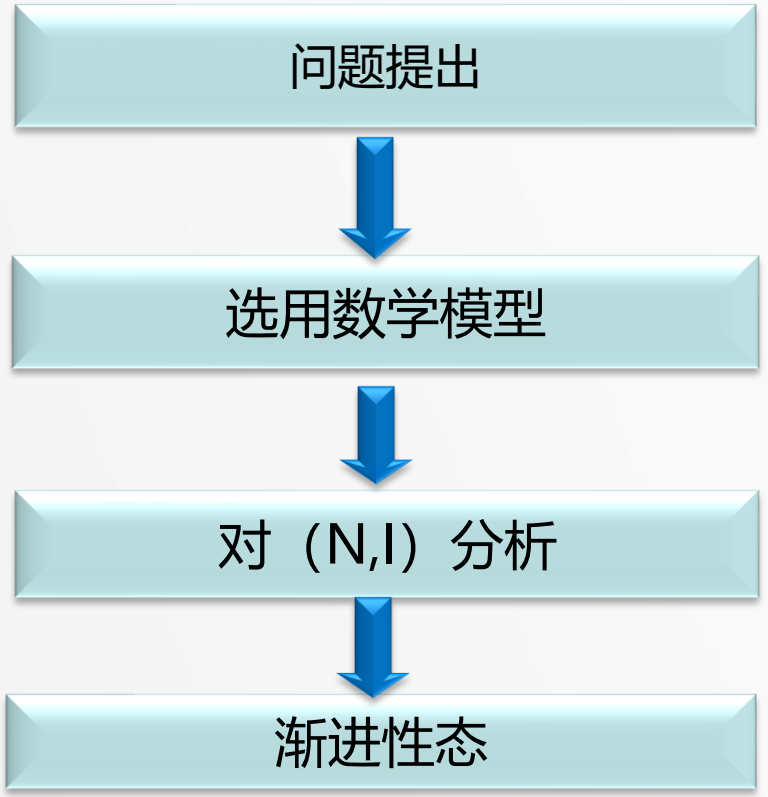
$$T(N,I)=\sum_{i=1}^{k}t_i e_i(N,I)$$

I：最坏，最好，平均

$\tilde{T}(N)$ F(N, I)

渐近符号引入

- **思路：划分等价类**
  - **最好输入**
  - **最坏输入**
  - **平均输入**



T(N, I)

$$T(N,I)=\sum_{i=1}^{k} t_i e_i(N,I)$$

I：最坏，最好，平均

$\tilde{T}(N)$ F(N, I)

渐近符号引入

忽略低阶项，忽略高阶项系数

$T_1(n) = 3n + 3$

$T_2(n) = 5n^2 + 10n + 233$

$T_3(n) = 9n^3 + 2n^2 + 10n + 66666$

当 n = 1000时，	当 n = 1000时，去掉低阶项	继续去掉系数
$T_1(n) = 3003$	$T_1(n) \approx 3000$	$T_1(n) \approx 1000$
$T_2(n) = 501,0233$	$T_2(n) \approx 500,0000$	$T_2(n) \approx 100,0000$
$T_3(n) = 90,0207,6666$	$T_3(n) \approx 90,0000,0000$	$T_3(n) \approx 10,0000,0000$

渐近符号引入

忽略低阶项，忽略高阶项系数

$T_1(n) = 3n + 3$

$T_2(n) = 5n^2 + 10n + 233$

$T_3(n) = 9n^3 + 2n^2 + 10n + 66666$

当 n = 1000时，

$T_1(n) = 3003$

$T_2(n) = 501,0233$

$T_3(n) = 90,0207,6666$

去掉系数

$T_1(n) \approx 1000$

$T_2(n) \approx 100,0000$

$T_3(n) \approx 10,0000,0000$

去掉系数

$T_1(n) = \Theta(n)$

$T_2(n) = \Theta(n^2)$

$T_3(n) = \Theta(n^3)$

平均情况渐近分析

当n→ ∞时，T(n) → ∞

此时若存在F(n)，使得 $\lim_{n\to\infty} \frac{T(n)}{F(n)} = c\ (c > 0)$

则 T(n) = Θ(F(n))

Θ表示 F(n) 和 T(n) 同阶

直观上， F(n) 为 T(n) 中忽略低阶项后留下的主项

平均情况渐近分析

存在正常数c_1、c_2，使得对所有的$n \geq n_0$，

$0 \leq c_1F(n) \leq T(n) \leq c_2F(n)$ 恒成立，

则记 $T(n) = \Theta(F(n))$

直观上，T(n)能被“夹入” c1F(n) 与 $c_2F(n)$ 之间

Θ：渐进紧确界

平均情况渐近分析

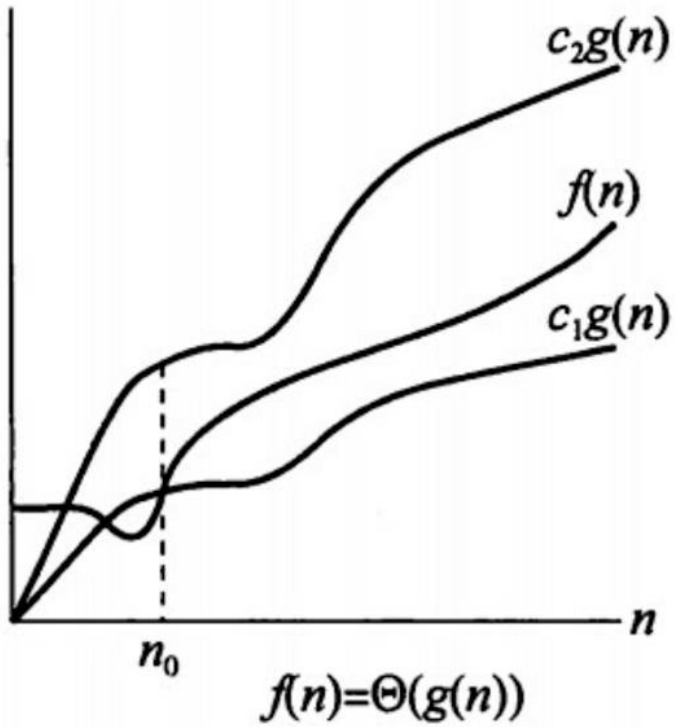
存在正常数c_1、c_2，使得对所有的$n \geq n_0$，

$0 \leq c_1F(n) \leq T(n) \leq c_2F(n)$ 恒成立，

则记 $T(n) = \Theta(F(n))$

直观上，T(n)能被“夹入” c1F(n) 与 $c_2F(n)$ 之间

Θ：渐进紧确界



T(n) = Θ(F(n))
T(n) = F(n)

最坏情况渐近分析

存在正常数 c 和 n_0，使得对所有的n $\geq n_0$，

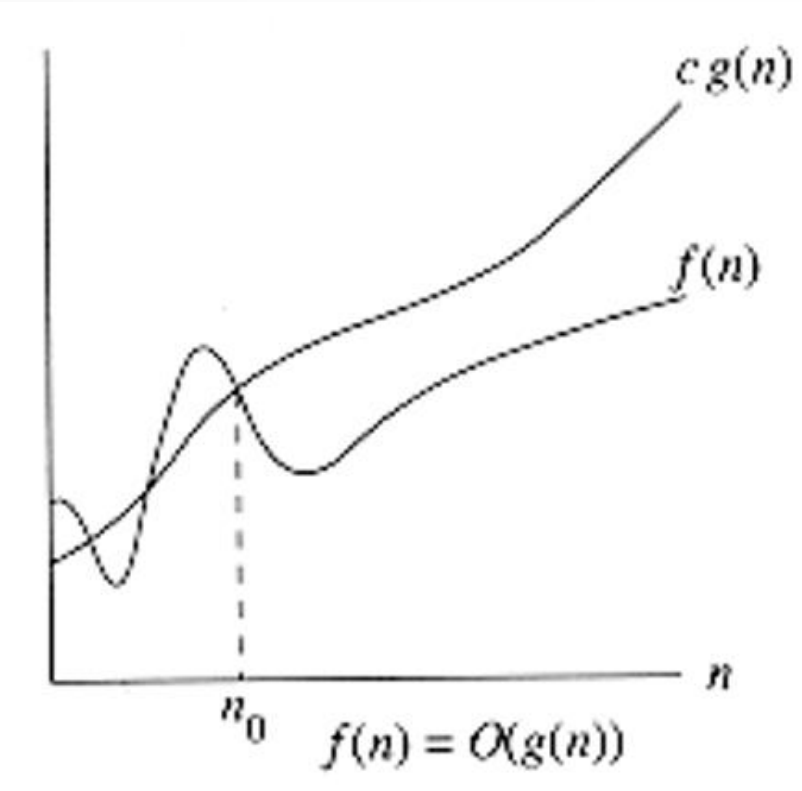
$0 \leq T(n) \leq cF(n)$ 恒成立，

则记 $T(n) = O(F(n))$

直观上，T(n)的阶不高于F(n)

大O：**渐进上界，上限，值越低越精确**

算法规模足够大时的上界



$T(n) = O(F(n))$
$T(n) \leq F(n)$

最好情况渐近分析

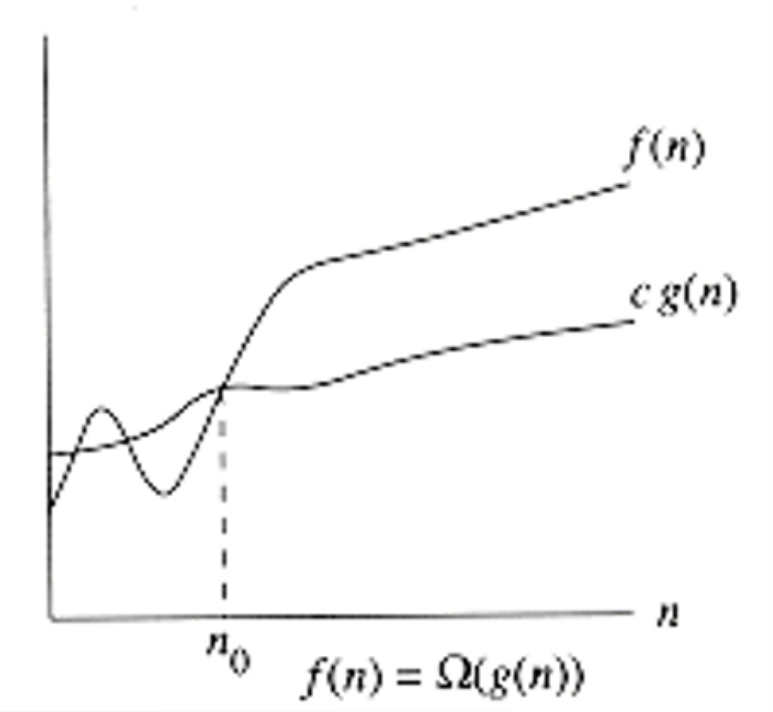
存在正常数 c 和 n_0，使得对所有的n $\geq n_0$，

$0 \leq cF(n) \leq T(n)$ 恒成立，

则记 $T(n) = \Omega(F(n))$

直观上，T(n)的阶不低于F(n)

大O：**渐进下界，下限，值越高越精确**



$T(n) = \Omega(F(n))$
$T(n) \geq F(n)$

渐近分析的符号

在下面的讨论中，对所有n，$f(n) \geq 0$，$g(n) \geq 0$。

（1）渐近上界记号O

$O(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq f(n) \leq cg(n) \}$

（2）渐近下界记号Ω

$\Omega(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq cg(n) \leq f(n) \}$

（3）紧渐近界记号Θ

$\Theta(g(n)) = \{ f(n) \mid$ 存在正常数c_1, c_2和n_0使得对所有$n \geq n_0$有：$c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

如果 f(n)是集合 O(g(n))中的一个成员，我们说f(n) 属于 O(g(n))

更多渐近分析的符号

在下面的讨论中，对所有n，$f(n) \geq 0$，$g(n) \geq 0$。

（4）非紧上界记号o

$o(g(n)) = \{ f(n) \mid$ 对于任何正常数$c>0$，

存在正数和$n_0 > 0$使得对所有$n \geq n_0$有：$0 \leq f(n) < cg(n) \}$

等价于 $f(n) / g(n) \to 0$ ， as $n \to \infty$。

（5）非紧下界记号ω

$\omega (g(n)) = \{ f(n) \mid$ 对于任何正常数$c>0$，

存在正数和$n_0 > 0$使得对所有$n \geq n_0$有：$0 \leq cg(n) < f(n) \}$

等价于 $f(n) / g(n) \to \infty$ ， as $n \to \infty$。

渐近分析中符号小结

$f(n) = O(g(n)) \rightarrow f \leq g;$

$f(n) = \Omega(g(n)) \rightarrow f \geq g;$

$f(n) = \Theta(g(n)) \rightarrow f = g;$

$f(n) = o(g(n)) \rightarrow f < g;$

$f(n) = \omega(g(n)) \rightarrow f > g.$

课堂讨论

- **根据渐近分析方法，如下结论是否正确?**

例如： $f(n) = 32n^2 + 17n + 32$

$f(n) = \Theta(n)$? $f(n) = \Theta(n^3)$? 错误!

$f(n) = \Theta(n^2)$

$f(n)$ 属于 $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, $\Theta(n)$.

$f(n)$ 不属于 $O(n)$, $\Omega(n^3)$, $\Theta(n^2)$, or $\Theta(n^3)$.

渐近分析的符号运算

◆加法规则：$O(f(n))+O(g(n)) = O(f(n)+g(n)) = O(\max\{f(n),g(n)\})$

◆多项相加，只保留最高阶的项，且系数置为1

$T_1(n) = 9n^3$

$T_2(n) = 2n^2$

$T_3(n) = 10n$

$T_4(n) = 66666$

$$T(n) = 9n^3 + 2n^2 + 10n + 66666 = O(n^3)$$

证明规则 $O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$

对任意 $f_1(n) \in O(f(n))$，存在正常数 c_1 和自然数 n_1，使得对所有 $n \geq n_1$，有 $f_1(n) \leq c_1 f(n)$

对任意 $g_1(n) \in O(g(n))$，存在正常数 c_2 和自然数 n_2，使得对所有 $n \geq n_2$，有 $g_1(n) \leq c_2 g(n)$

令 $c_3=\max\{c_1, c_2\}$，$n_3=\max\{n_1, n_2\}$，$h(n)=\max\{f(n),g(n)\}$

对所有的 $n \geq n_3$，有

$$f_1(n)+g_1(n) \leq c_1 f(n) + c_2 g(n)$$

$$\leq c_3 f(n) + c_3 g(n) = c_3(f(n) + g(n))$$

$$\leq c_3 2 \max\{f(n),g(n)\}$$

$$= 2c_3 h(n) = O(\max\{f(n),g(n)\})$$

渐近分析的符号运算

- 乘法规则：O(f(n))*O(g(n)) = O(f(n)*g(n))
- 多项相乘，都保留

$T_1(n) = n^2$

$T_2(n) = \log n$

$T_3(n) = n^2 \log n$

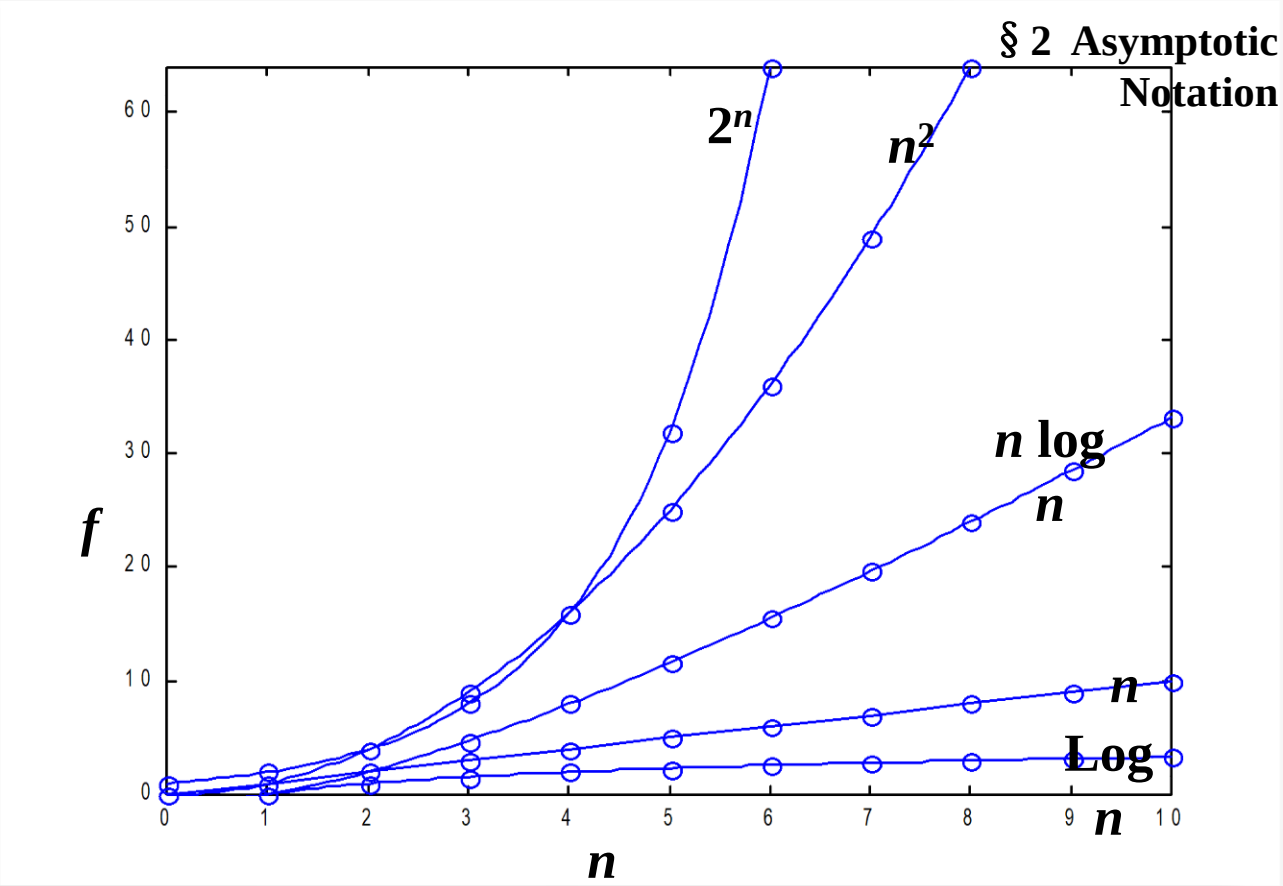
$T(n) = 9n^2 + 4n\log n$，O如何保留？

渐近分析的符号运算

- $O(f(n))+O(g(n)) = O(f(n)+g(n)) = O(\max\{f(n),g(n)\})$
- $O(f(n))*O(g(n)) = O(f(n)*g(n))$;
- $O(cf(n)) = O(f(n))$;
- $g(n)= O(f(n)) \Rightarrow O(f(n))+O(g(n)) = O(f(n))$ 。

渐近分析的常见关系

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



常对幂指阶

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

多项式. $a_0 + a_1 n + \ldots + a_d n^d = \Theta(n^d)$ 其中 $a_d > 0$.

对数. $O(\log_a n) = O(\log_b n)$ 其中 a, b > 0为常数.

底不重要

对数. 对任意 $x > 0$, $\log n = O(n^x)$.

指数. 对任意 $r > 1$ 和 $d > 0$, $n^d = O(r^n)$.

指数比多项式更高阶

渐近分析的符号运算

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

$T(n) = 9n^2 + 4n\log n$，O如何保留？

$$\lim_{n\to\infty}\frac{n^2}{n\log n}=\lim_{n\to\infty}\frac{n}{\log n}=\lim_{n\to\infty}\frac{1}{\frac{1}{n\ln 2}}=+\infty$$

洛必达法则 或 泰勒展开

课堂讨论

- **根据渐近分析方法，如下结论是否正确?**

例如： $f(n) = 32n^2 + 17n + 32$

$f(n) = \Theta(n)$? $f(n) = \Theta(n^3)$? 错误!

$f(n) = \Theta(n^2)$ 正确!

$f(n)$ 属于 $O(n^2), O(n^3), O(2n), O(n!), O(n^n)$ 上界

$f(n)$ 属于 $\Omega(n^2), \Omega(n\log n), \Omega(n), \Omega(1)$ 下界

例子

对下列函数按渐进关系O从小到大排列：

$f_1(n)=10^n$　$f_2(n)=n^{1/3}$　$f_3(n)=n^n$　$f_4(n)=\log_2 n$　$f_5(n)=2^{\sqrt{\log_2 n}}$

O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)

$f_4(n)=\log_2 n$　$f_2(n)=n^{1/3}$　$f_1(n)=10^n$　$f_3(n)=n^n$

$f_5(n)=2^{\sqrt{\log_2 n}}$

头重脚轻取对数！

例子

对下列函数按渐进关系O从小到大排列：

$$f_1(n)=10^n \quad f_2(n)=n^{1/3} \quad f_3(n)=n^n \quad f_4(n)=\log_2 n \quad f_5(n)=2^{\sqrt{\log_2 n}}$$

$$\log f_4(n)=\log_2 z \qquad \log f_5(n)=z^{1/2} \qquad \log f_2(n)=\frac{1}{3}z \quad z=\log_2 n$$

例子

对下列函数按渐进关系O从小到大排列：

$f_1(n)=10^n$ $f_2(n)=n^{1/3}$ $f_3(n)=n^n$ $f_4(n)=\log_2 n$ $f_5(n)=2^{\sqrt{\log_2 n}}$

O(1) < O(logn) < O(n) < O(nlogn) < $O(n^2)$ < $O(n^3)$ < $O(2^n)$ < O(n!) < $O(n^n)$

$f_4(n)=\log_2 n$ $f_5(n)=2^{\sqrt{\log_2 n}}$ $f_2(n)=n^{1/3}$ $f_1(n)=10^n$ $f_3(n)=n^n$

算法的时间复杂度计算

```
3   void work_hard(int n) {    // n: 问题规模
4       int day = 1;           // day: 天数
5       while (day <= n) {
6           printf("第%d天坚持学习\n", day);
7           day++;
8       }
9       printf("第%d天我出师了\n", day);
10  }
```

还需要逐行分析吗？

④/⑨：顺序及分支语句（无循环）只影响常数项

⑥/⑦：循环中语句条数只影响系数，分析循环次数即可

若存在多层循环，只需关注最深层循环执行次数

例一 累加求和

```
float  sum ( float  list[ ],  int  n )
{  /* add a list of numbers */
   float  tempsum = 0;
   int  i ;
   for ( i = 0; i < n; i++ )

       tempsum  += list [ i ] ;

   return  tempsum;
}
```

$O(n)$

例二 选择排序

```
void select_sort(int& a[], int n)
{ // 将 a 中整数序列重新排列成自小至大有序的整数序列。
    for ( i = 0;  i< n-1;  ++i )
    { j = i;  // 选择第 i 个最小元素
      for ( k = i+1;  k < n;  ++k )
          if (a[k] < a[j] ) j = k;

      if ( j != i )  a[j] ←→ a[i]


    }
} // select_sort
```

$O(n^2)$

例三 冒泡排序

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

            if (a[j] > a[j + 1]) {
                swap(&a[j], &a[j + 1]);
            }
        }
}
```

$O(n^2)$

例四 卷王

```
void daydayup(int n) {
    int i = 1;
    while (i <= n) {
        printf("每天努力一倍，当卷王");
        i = i * 2;
    }
}
```

O(log(n))

例5. 求平面上n个点(x1, y1), …, (xn, yn)中最近两个点之间的距离.

对每对点都尝试一下。

```
min ← (x1 - x2)^2 + (y1 - y2)^2
for i = 1 to n {
    for j = i+1 to n {
        d ← (xi - xj)^2 + (yi - yj)^2
        if (d < min)
            min ← d
    }
}
```

$$O(n^2)$$

例6. 计算平面上两点间的最短距离。看下面这段程序的时间复杂度

```
d[i*j]
for i = 1 to n {
    for j = i+1 to n {
        d[k] ← (x_i - x_j)^2 + (y_i - y_j)^2
    }
}
a=d[1]
for i = 1 to i*j {
    if (d[i] < a)
        a ← d[i]
}
```

$O(n^2)$

例七 查找

```
void FindTarget(int arr[], int n, int target)
{
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
```

最好、最坏、平均复杂度?

最好：第一个位置能找到，O(1)

最坏：最后一个位置找到或找不到，O(n)

例七 查找

```
void FindTarget(int arr[], int n, int target)
{
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
```

最好、最坏、平均复杂度?

平均：在n个位置找到的概率均等，求数学期望

$$(1+2+\cdots+n)\cdot\frac{1}{n}=\frac{(1+n)\cdot n}{2}\cdot\frac{1}{n}=\frac{n+1}{2}$$

O(n)

算法复杂度分析

- 引入
- 时间复杂度分析
- 空间复杂度分析

算法的空间复杂度计算

```
3   void work_hard(int n) {    // n: 问题规模
4       int day = 1;           // day: 天数
5       while (day <= n) {
6           printf("第%d天坚持学习\n", day);
7           day++;
8       }
9       printf("第%d天我出师了\n", day);
10  }
```

算法所需的存储空间开销随数据规模增长的变化情况

程序会占用哪些内存?

代码段、数据段（全局数据段、栈区）、堆区

算法的空间复杂度计算

```
void func(int n) {
    int* arr = (int *)malloc(n * sizeof(int));
    ...
}
```

O(n)

```
void func(int n) {
    int* arr1 = (int *)malloc(n * sizeof(int));
    int* arr2 = (int *)malloc(n * n * sizeof(int));
    ...
}
```

$O(n^2)$

常见排序算法的复杂度

排序方法	最坏时间	平均时间	最好时间	空间	稳定性
选择排序	n^2	n^2	n^2	1	不稳定
冒泡排序	n^2	n^2	n	1	稳定
插入排序	n^2	n^2	n	1	稳定
堆排序	nlogn	nlogn	nlogn	1	不稳定
希尔排序	n^2	$n^{1.3}$	n	1	不稳定
归并排序	nlogn	nlogn	nlogn	n	稳定
快速排序	n^2	nlogn	nlogn	logn	不稳定
桶排序	n^2	n+k	n	n+k	稳定
计数排序	n+k	n+k	n+k	n+k	稳定
基数排序	n*k	n*k	n*k	n+k	稳定