

# 计算机算法设计与分析

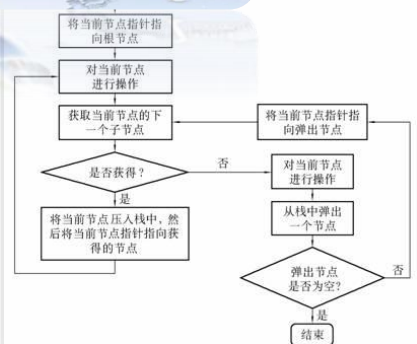


中国科学技术大学  
信息科学技术学院自动化系

王子磊

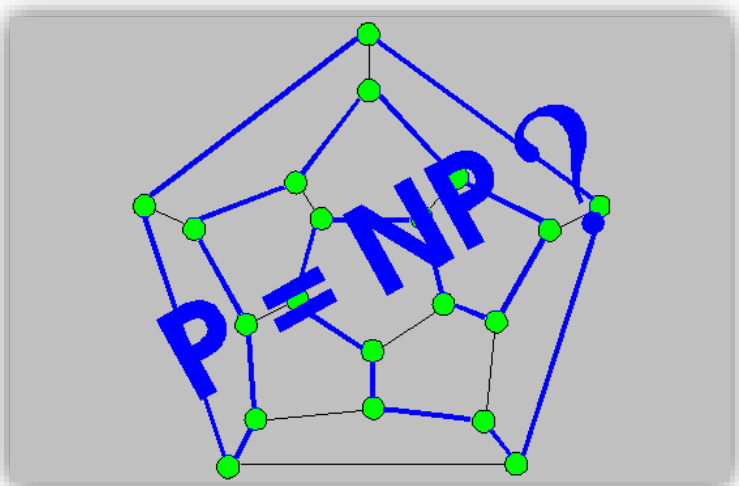
[zlwang@ustc.edu.cn](mailto:zlwang@ustc.edu.cn)

<http://vim.ustc.edu.cn/?product=algorithm>



教材

# 《计算机算法设计与分析》（第4版）



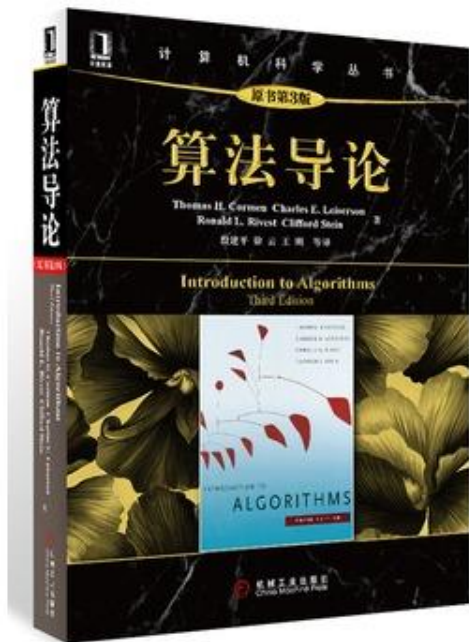
王晓东 编著

电子工业出版社

<http://www.phei.com.cn>

- ❖ 教辅: <http://www.hxedu.com.cn>
- ❖ 教学: <http://algorithm.fzu.edu.cn>

# 参考书



## 《算法导论》

Thomas H.Cormen, Charles E.Leiserson,  
Ronald L.Rivest, Clifford Stein  
(CLRS)

机械工业出版社

<http://www.cmpbook.com>

- 《计算机程序设计艺术》：The Art of Computer Programming  
编著：Donald Ervin Knuth  
国防工业出版社

# 课程要点

- 算法设计思想
- 算法分析方法
- 数据结构及其算法优化
- 算法思想在实际问题中的应用
- 经典算法的设计与分析

# Chapter 0

## Introduction to algorithm



王子磊 (Zilei Wang)

Email: [zlwang@ustc.edu.cn](mailto:zlwang@ustc.edu.cn)

<http://vim.ustc.edu.cn/>

# 学习要点

- 理解算法的概念
- 理解什么是程序，程序与算法的区别和内在联系
- 掌握算法的计算复杂性概念
- 掌握算法渐近复杂性的数学表述
- 掌握用C++语言描述算法的方法

# 算法 (Algorithm)

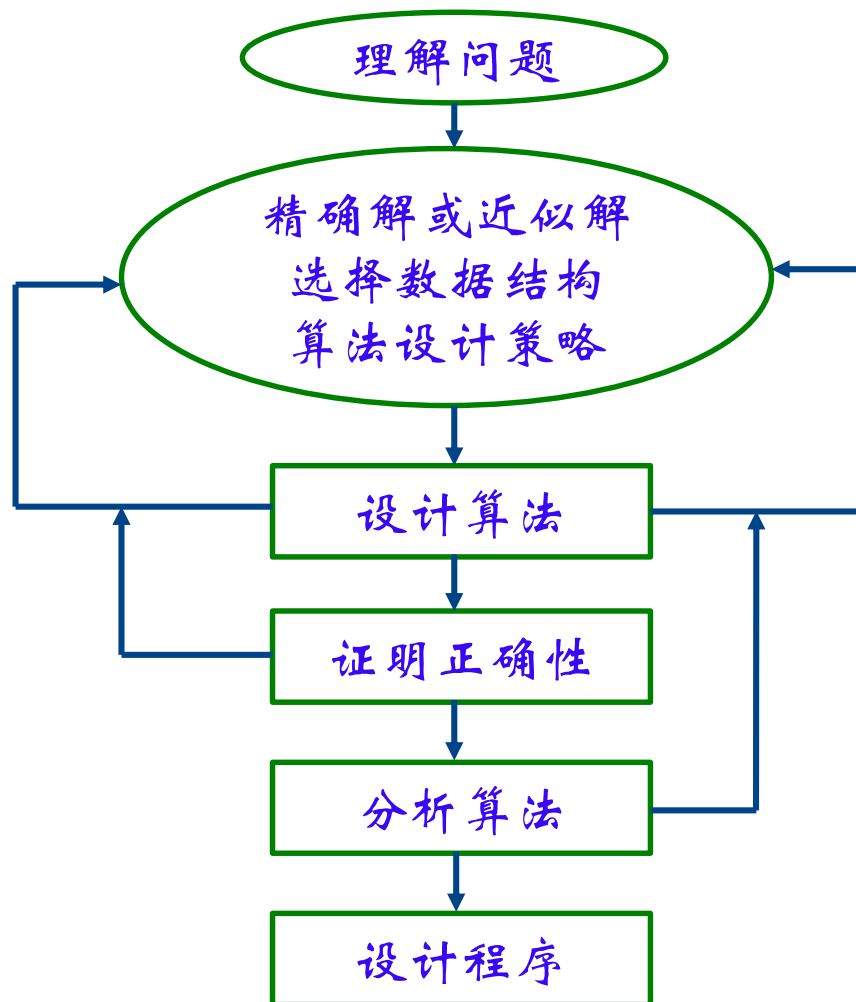
- ❖ 算法是指解决**问题**的一种方法或一个过程
- ❖ 算法是若干指令的有穷序列，满足性质：
  - **输入**：有外部提供的量作为算法的输入
  - **输出**：算法产生至少一个量作为输出
  - **确定性**：组成算法的每条指令是清晰，无歧义的
  - **有限性**：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的

# 程序 (Program)

- ❖ 程序是算法用某种程序设计语言的具体实现
- ❖ 程序可以不满足算法的性质-有限性
- ❖ 例如：操作系统
  - 是一个在无限循环中执行的程序，因而不是一个算法
  - 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现，该子程序得到输出结果后便终止



# 问题求解 (Problem solving)



# 算法复杂性分析

## ❖ 计算机程序的性能和所用资源的理论分析

- 算法复杂性 = 算法所需要的计算机资源
- 算法的时间复杂性  $T(n)$
- 算法的空间复杂性  $S(n)$ 
  - 其中  $n$  是问题的规模 (输入大小)

## ❖ 性能之外

- |                     |                   |               |
|---------------------|-------------------|---------------|
| ■ Modularity        | ■ Maintainability | ■ Robustness  |
| ■ User-friendliness | ■ Simplicity      | ■ Reliability |
| ■ Correctness       | ■ Functionality   |               |
| ■ Programmer time   | ■ Extensibility   |               |

# 为什么要研究算法复杂性

- 帮助我们理解算法的**可扩展性**
- 性能通常刻画了可行与不可行之间的**界限**
- 算法的数学分析为讨论算法行为提供了一种**工具**
- 程序性能分析的经验能够**推广**到其他计算资源的分析上
- 速度分析很**有意思**！

# 排序问题举例

**Input:** sequence  $\langle a_1, a_2, \dots, a_n \rangle$  of numbers.

**Output:** permutation  $\langle a'_1, a'_2, \dots, a'_n \rangle$  such that

$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

**Example:**

**Input:** 8 2 4 9 3 6

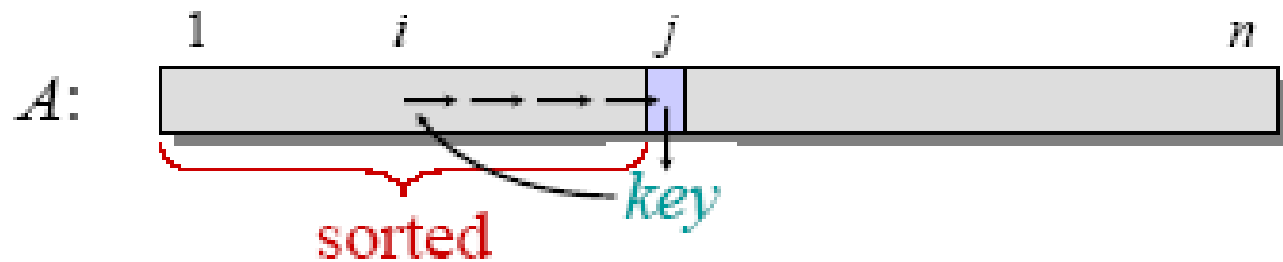
**Output:** 2 3 4 6 8 9

# 插入排序



“pseudocode”

```
INSERTION-SORT ( $A, n$ )  ▷  $A[1 \dots n]$   
  for  $j \leftarrow 2$  to  $n$   
    do  $key \leftarrow A[j]$   
       $i \leftarrow j - 1$   
      while  $i > 0$  and  $A[i] > key$   
        do  $A[i+1] \leftarrow A[i]$   
           $i \leftarrow i - 1$   
       $A[i+1] = key$ 
```



# 插入排序示例

8

2

4

9

3

6

# 插入排序示例



# 插入排序示例

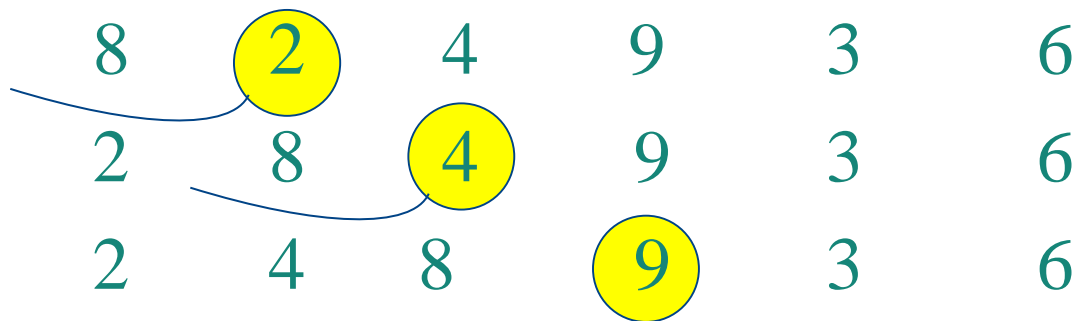




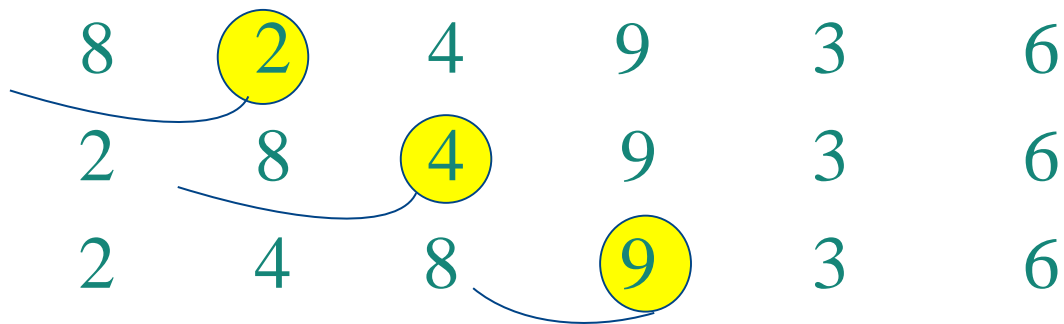
# 插入排序示例



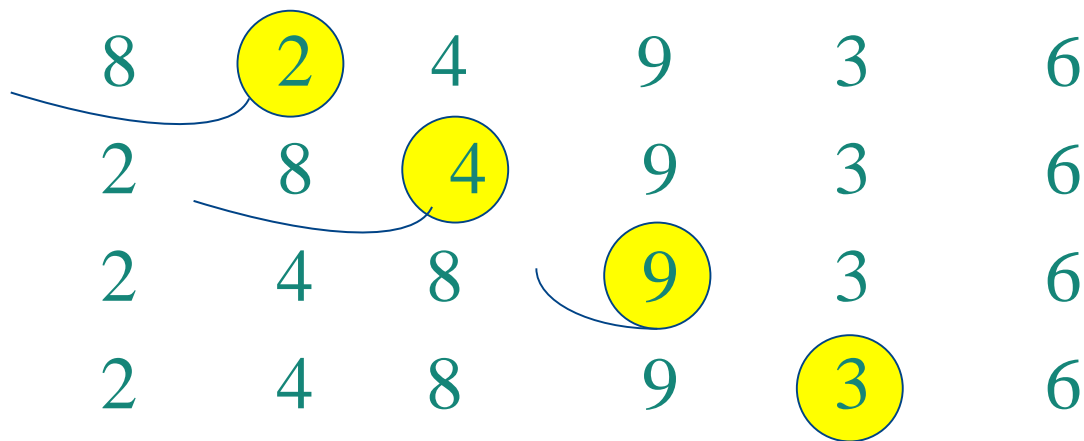
# 插入排序示例



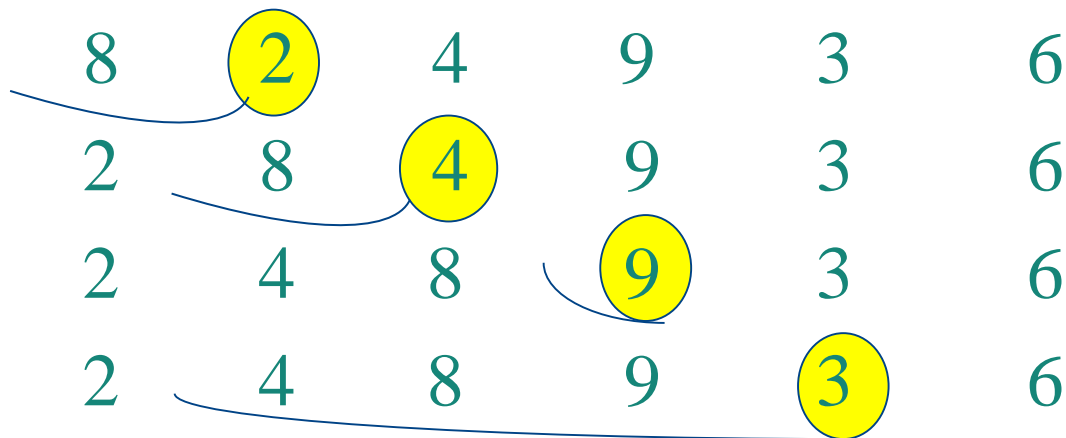
# 插入排序示例



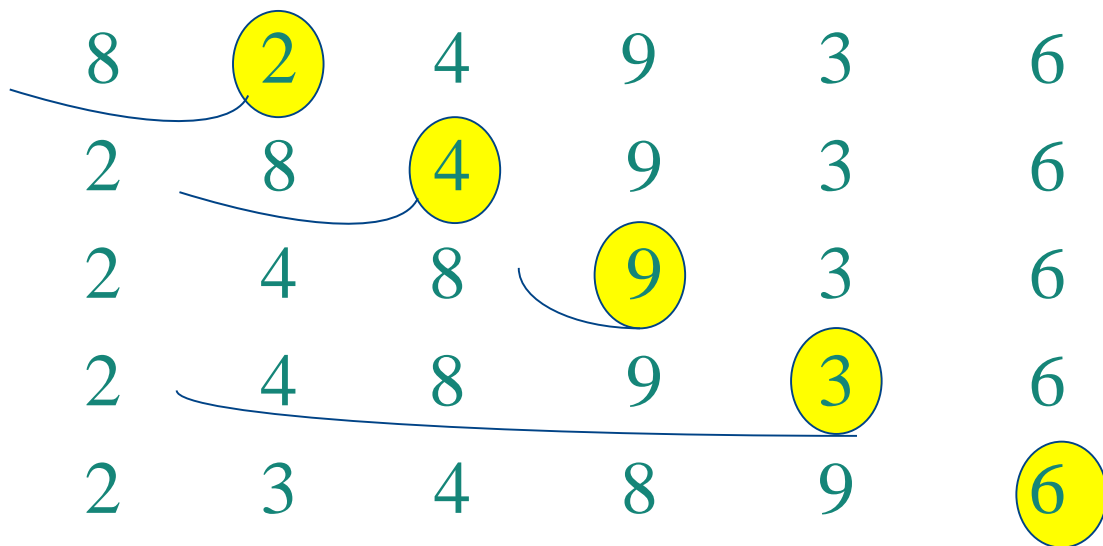
# 插入排序示例



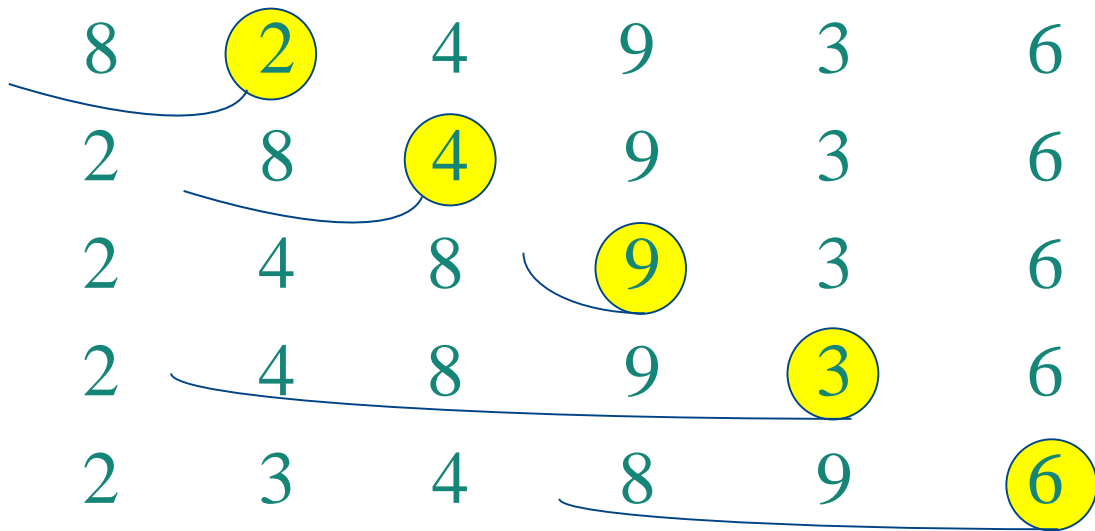
# 插入排序示例



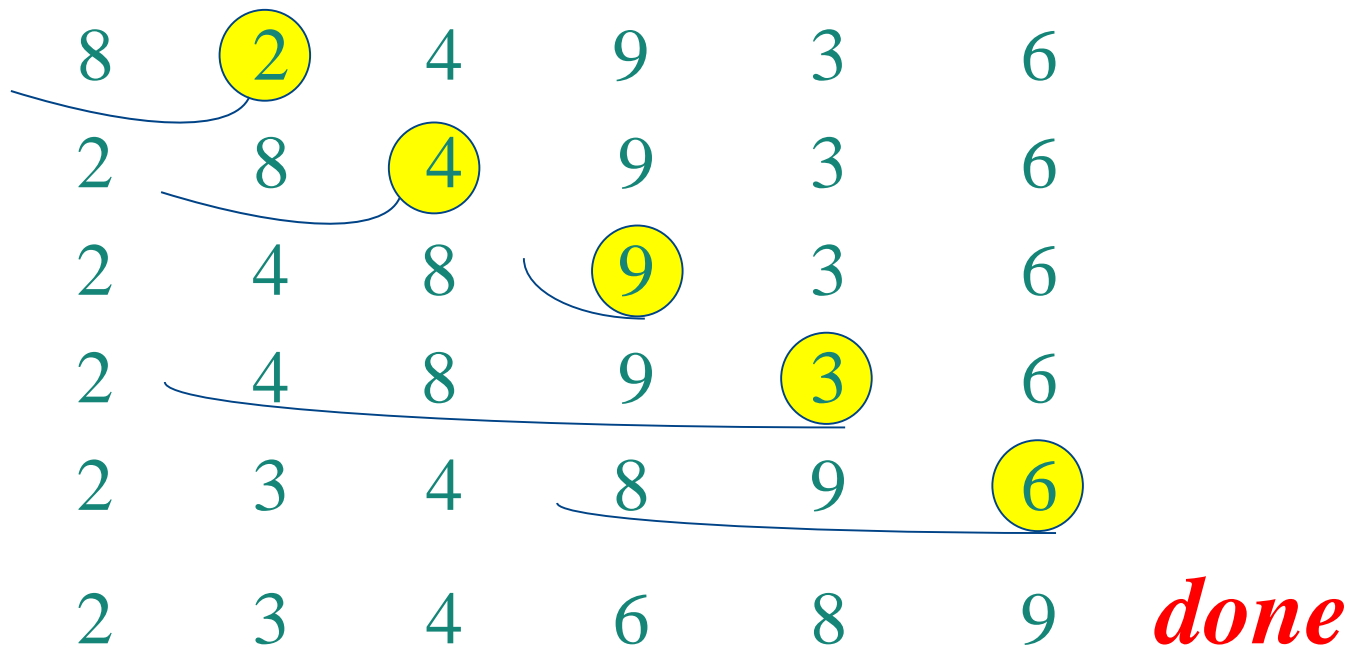
# 插入排序示例



# 插入排序示例



# 插入排序示例





# 运行时间

## ❖ 运行时间依赖于输入（输入驱动算法执行）

- 一个已经排好序的序列更容易排序
- 通常情况下，较短的序列更容易排序，因此，我们用输入的大小来参数化运行时间

## ❖ 通常，我们想要获取的是运行时间的上界

- 实际上每个人都更倾向于获得一种保证

# 算法的时间复杂性

## ❖ 最坏情况下的时间复杂性

$$T_{\max}(n) = \max\{ T(I) \mid \text{size}(I)=n \}$$

## ❖ 最好情况下的时间复杂性

$$T_{\min}(n) = \min\{ T(I) \mid \text{size}(I)=n \}$$

## ❖ 平均情况下的时间复杂性

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

- 其中  $I$  是问题的规模为  $n$  的实例， $p(I)$  是实例  $I$  出现的概率

# 机器独立的算法时间

插入排序算法的最坏情况需要多少时间呢？

- 这通常依赖于我们使用的计算机
  - 相对速度 (在同一机器上)
  - 绝对速度 (在不同的机器上)

## BIG IDEA:

- 忽略机器相关的常数
- 只考察  $T(n)$  随着  $n \rightarrow \infty$  的增长

渐进复杂性 (“Asymptotic Analysis”)

# 算法渐近复杂性

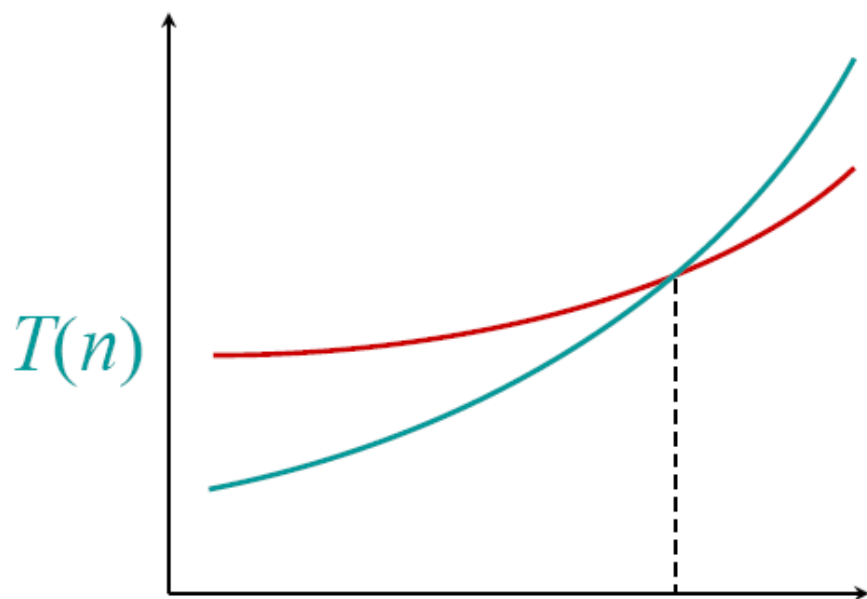
❖  $T(n) \rightarrow \infty$ , as  $n \rightarrow \infty$

❖  $(T(n) - t(n)) / T(n) \rightarrow 0$ , as  $n \rightarrow \infty$

- $t(n)$  是  $T(n)$  的渐近性态，为算法的渐近复杂性
- 在数学上， $t(n)$  是  $T(n)$  的渐近表达式，是  $T(n)$  略去低阶项后留下的主项，它比  $T(n)$  简单

# 渐近性能

- 当  $n$  足够大时，一个  $\Theta(n^2)$  的算法  
总是能够打败一个  $\Theta(n^3)$  的算法



- 在实际应用中，我们不能直接忽视渐近慢的算法，因为，我们通常需要对工程目标进行仔细的平衡
- 渐近分析为结构化算法的思路提供了一种有用的工具

# 渐近分析的记号

❖ 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 $O$

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

有:  $0 \leq f(n) \leq cg(n) \}$

固定

## (2) 渐近下界记号 $\Omega$

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

有:  $0 \leq cg(n) \leq f(n) \}$

# 渐近分析的记号

## (3) 非紧上界记号 $o$

- $o(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0 \text{ 使得对所有 } n \geq n_0, \text{ 有: } 0 \leq f(n) < cg(n) \}$
- 等价于  $f(n) / g(n) \rightarrow 0$ , as  $n \rightarrow \infty$

## (4) 非紧下界记号 $\omega$

- $\omega(g(n)) = \{ f(n) \mid \text{对于任何正常数 } c > 0, \text{ 存在正数和 } n_0 > 0 \text{ 使得对所有 } n \geq n_0, \text{ 有: } 0 \leq cg(n) < f(n) \}$
- 等价于  $f(n) / g(n) \rightarrow \infty$ , as  $n \rightarrow \infty$
- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

## (5) 紧渐近界记号 $\Theta$

- $\Theta(g(n)) = \{ f(n) \mid \text{存在正常数 } c_1, c_2 \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } c_1 g(n) \leq f(n) \leq c_2 g(n) \}$
- **定理1:**  $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$



# 渐近分析记号在等式和不等式中的意义

❖  $f(n) = \Theta(g(n))$  的确切意义是:  $f(n) \in \Theta(g(n))$

❖ 一般情况下, 等式和不等式中的渐近记号  $\Theta(g(n))$  表示  $\Theta(g(n))$  中的某个函数

■ 例如:  $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$  表示

$2n^2 + 3n + 1 = 2n^2 + f(n)$ , 其中  $f(n)$  是  $\Theta(n)$  中某个函数

❖ 等式和不等式中渐近记号  $O, o, \Omega$  和  $\omega$  的意义是类似的

# 插入排序的复杂性分析

- **最坏情况下：**输入是反向排序的

$$T(n) = \sum_{j=2}^n \Theta(j) = \Theta(n^2)$$

- **平均情况下：**所有的顺序以等概率出现

$$T(n) = \sum_{j=2}^n \Theta(j/2) = \Theta(n^2)$$

- 插入排序是一个快速的排序算法吗？

- 当  $n$  较小时，性能还可以接受
- 当  $n$  较大时，比较差

```
INSERTION-SORT ( $A, n$ )    ▷  $A[1 \dots n]$ 
  for  $j \leftarrow 2$  to  $n$ 
    do  $key \leftarrow A[j]$ 
       $i \leftarrow j - 1$ 
      while  $i > 0$  and  $A[i] > key$ 
        do  $A[i+1] \leftarrow A[i]$ 
           $i \leftarrow i - 1$ 
       $A[i+1] = key$ 
```

# 渐近分析中函数比较

❖  $f(n) = O(g(n)) \approx a \leq b$

❖  $f(n) = \Omega(g(n)) \approx a \geq b$

❖  $f(n) = \Theta(g(n)) \approx a = b$

❖  $f(n) = o(g(n)) \approx a < b$

❖  $f(n) = \omega(g(n)) \approx a > b$

# 渐近分析记号的若干性质

## (1) 传递性:

- $f(n) = \Theta(g(n)), \quad g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
- $f(n) = O(g(n)), \quad g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
- $f(n) = \Omega(g(n)), \quad g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$
- $f(n) = o(g(n)), \quad g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$
- $f(n) = \omega(g(n)), \quad g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$

## (2) 反身性:

- $f(n) = \Theta(f(n))$
- $f(n) = O(f(n))$
- $f(n) = \Omega(f(n))$

## (3) 对称性:

- $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

#### (4) 互对称性:

- $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$
- $f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$

#### (5) 算术运算:

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

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

- 对于任意  $f_1(n) \in O(f(n))$ , 存在正常数  $c_1$  和自然数  $n_1$ , 使得对所有  $n \geq n_1$ , 有  $f_1(n) \leq c_1 f(n)$
- 类似地, 对于任意  $g_1(n) \in O(g(n))$ , 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  $c_3 = \max\{c_1, c_2\}$ ,  $n_3 = \max\{n_1, n_2\}$ ,  $h(n) = \max\{f(n), g(n)\}$
- 则对所有的  $n \geq n_3$ , 有
- $$\begin{aligned} f_1(n) + g_1(n) &\leq c_1 f(n) + c_2 g(n) \leq c_3 f(n) + c_3 g(n) = c_3 (f(n) + g(n)) \\ &\leq c_3 2 \max\{f(n), g(n)\} = 2c_3 h(n) = O(\max\{f(n), g(n)\}) \end{aligned}$$

# 算法渐近复杂性分析中常用函数

## (1) 单调函数

- 单调递增:  $m \leq n \Rightarrow f(m) \leq f(n)$
- 单调递减:  $m \leq n \Rightarrow f(m) \geq f(n)$
- 严格单调递增:  $m < n \Rightarrow f(m) < f(n)$
- 严格单调递减:  $m < n \Rightarrow f(m) > f(n)$

## (2) 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



# 取整函数的若干性质

- ❖  $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$
- ❖  $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$
- ❖ 对于  $n \geq 0$ ,  $a, b > 0$  (整数), 有:
  - $\lceil \lceil n/a \rceil / b \rceil = \lceil n/ab \rceil$
  - $\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor$
  - $\lceil a/b \rceil \leq (a+(b-1))/b$
  - $\lfloor a/b \rfloor \geq (a-(b-1))/b$
- ❖  $f(x) = \lfloor x \rfloor$ ,  $g(x) = \lceil x \rceil$  为单调递增函数

### (3) 多项式函数

- $p(n) = a_0 + a_1n + a_2n^2 + \dots + a_dn^d; \quad a_d > 0$
- $p(n) = \Theta(n^d)$
- $f(n) = O(n^k) \Leftrightarrow f(n)$  多项式有界
- $f(n) = O(1) \Leftrightarrow f(n) \leq c$
- $k \geq d \Rightarrow p(n) = O(n^k)$
- $k \leq d \Rightarrow p(n) = \Omega(n^k)$
- $k > d \Rightarrow p(n) = o(n^k)$
- $k < d \Rightarrow p(n) = \omega(n^k)$

## (4) 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

❖  $a > 1 \Rightarrow \lim_{n \rightarrow \infty} n^b / a^n = 0 \Rightarrow n^b = o(a^n)$

$$e^x = 1 + x + x^2/2! + x^3/3! + \dots = \sum_i x^i/i!$$

$$\diamond e^x \geq 1+x$$

$$\diamond |x| \leq 1 \Rightarrow 1+x \leq e^x \leq 1+x+x^2$$

$$\diamond e^x = 1+x + \Theta(x^2), \text{ as } x \rightarrow 0$$

$$\lim_{n \rightarrow \infty} (1 + x/n)^n = e^x$$

## (5) 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  $a > 0, b > 0, c > 0$ ,  $a = b^{\log_b a}$

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

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

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

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

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

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

❖  $|x| \leq 1 \Rightarrow \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \Lambda .$

❖ for  $x > -1$ ,  $\frac{x}{1+x} \leq \ln(1+x) \leq x$

❖ for any  $a > 0$ ,  $\lim_{n \rightarrow \infty} \frac{\log^b n}{(2^a)^{\log n}} = \lim_{n \rightarrow \infty} \frac{\log^b n}{n^a} = 0 \Rightarrow \log^b n = o(n^a)$

## (6) 阶乘函数

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

❖ Stirling's approximation

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$



$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\alpha_n}, \quad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$n! = o(n^n)$$

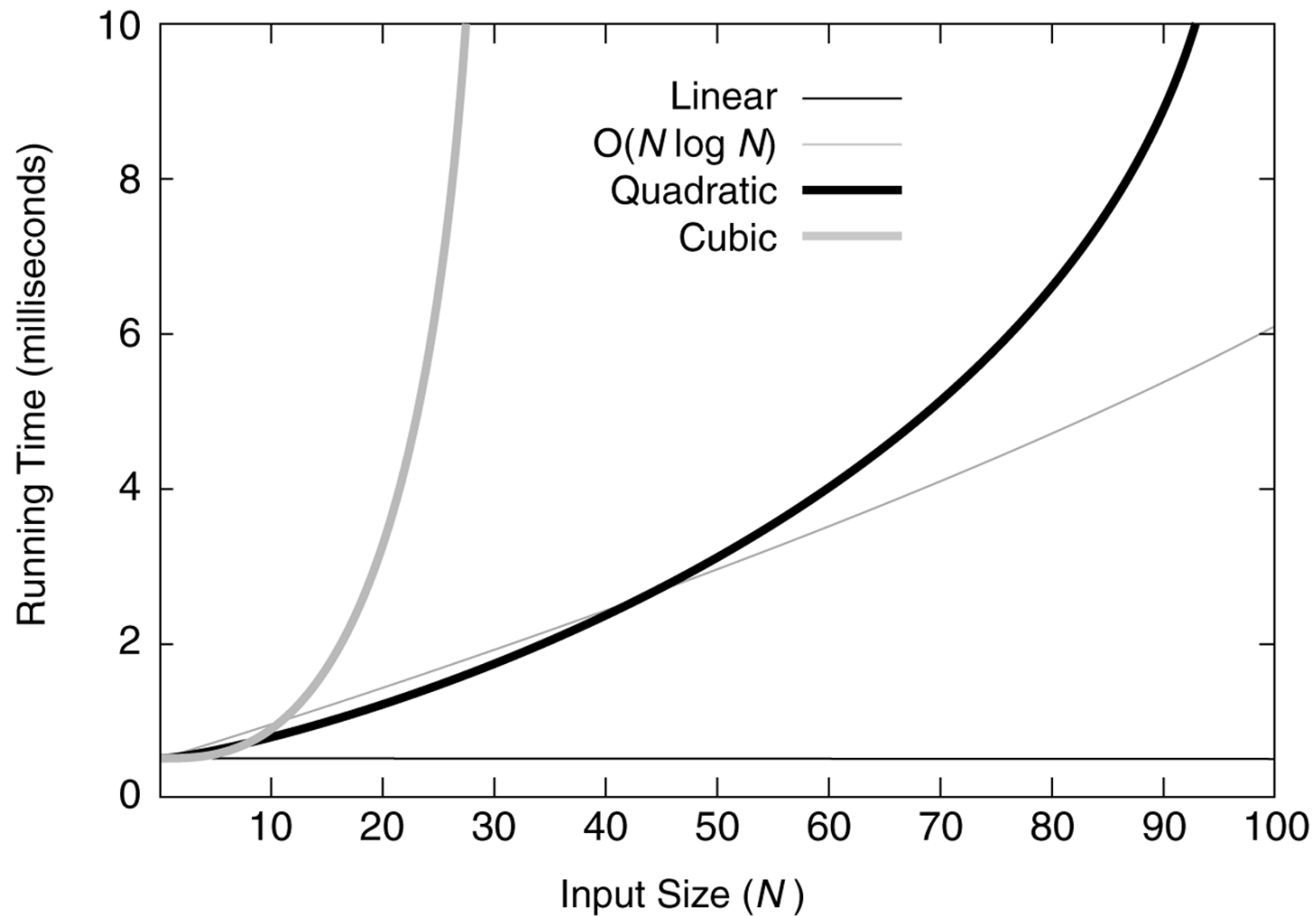
$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n \log n)$$

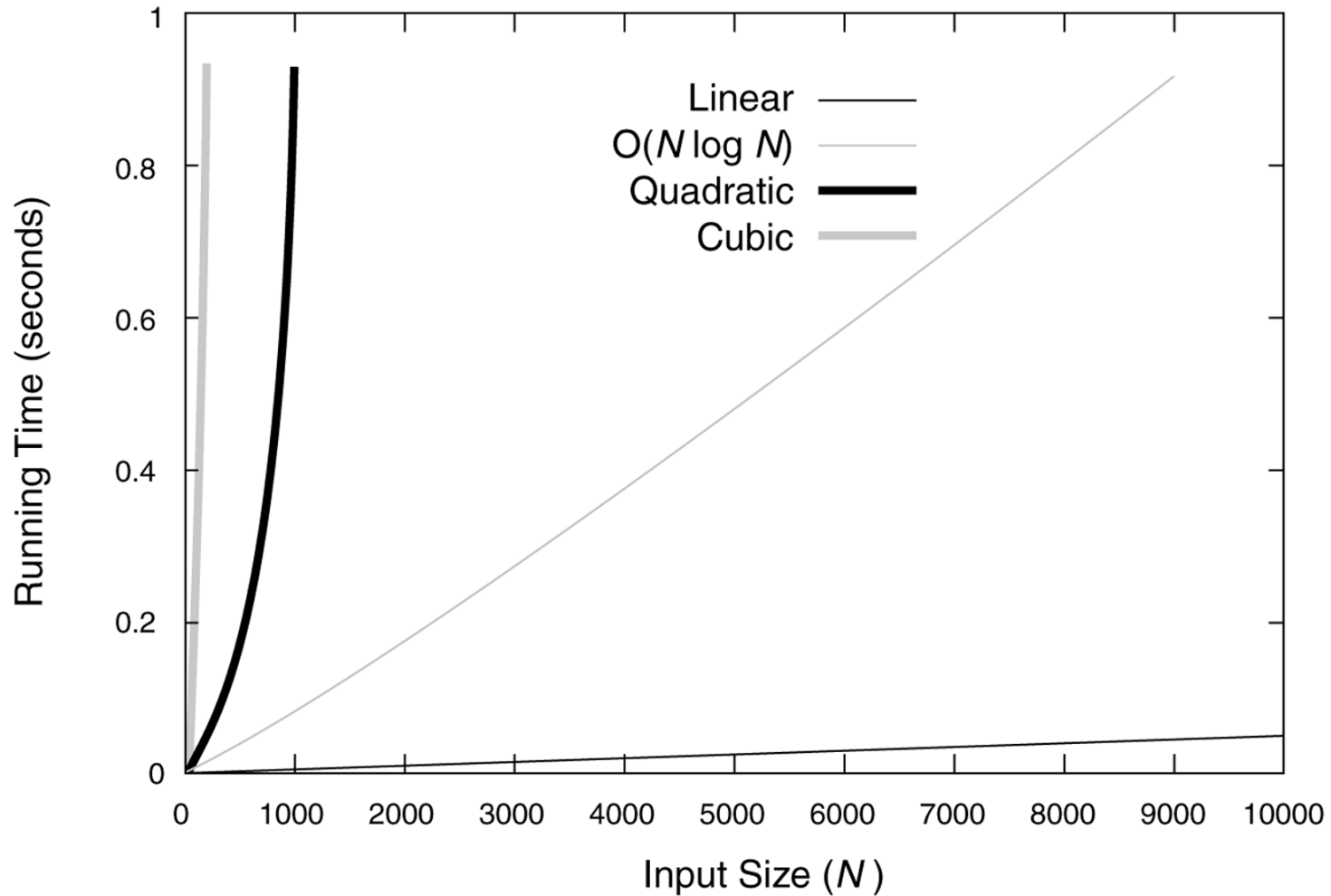
# 算法分析中常见的复杂性函数

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

# 小规模数据



# 中等规模数据



# 用C++描述算法

CATEGORY	EXAMPLES	ASSOCIATIVITY
Operations on References	. []	Left to right
Unary	++ -- ! - (type)	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift (bitwise)	<< >>	Left to right
Relational	< <= > >= instanceof	Left to right
Equality	== !=	Left to right
Boolean (or bitwise) AND	&	Left to right
Boolean (or bitwise) XOR	^	Left to right
Boolean (or bitwise) OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= *= /= %= += -=	Right to left

# (1) 选择语句

## (1.1) if 语句:

```
if (expression) statement;  
else statement;
```

## (1.2) ? 语句:

```
exp1?exp2:exp3  
y= x>9 ? 100:200;
```

等价于:

```
if (x>9) y=100;  
else y=200;
```

### (1.3) switch 语句:

```
switch (expression) {  
    case 1:  
        statement sequence;  
        break;  
    case 2:  
        statement sequence;  
        break;  
    ...  
    default:  
        statement sequence;  
}
```

## (2) 迭代语句

### (2.1) for 循环:

- `for (init; condition; inc) statement;`

### (2.2) while 循环:

- `while (condition) statement;`

### (2.3) do-while 循环:

- `do{`
- `statement;`
- `} while (condition);`



# (3) 跳转语句

## (3.1) return 语句：

- return expression;

## (3.2) goto 语句：

- goto label;
- ...
- label:

## (4) 函数

### ❖ 函数定义

```
return-type function_name(para-list)
{
    body of the function
}
```

### ❖ 例：

```
int max(int x,int y)
{
    return x>y?x:y;
}
```

## (5) 模板 template

### ❖ 定义与使用

```
template <class Type>  
Type max(Type x, Type y)  
{  
    return x > y ? x : y;  
}
```

```
int i = max(1, 2);  
double x = max(1.0, 2.0);
```

# (6) 动态存储分配

## (6.1) 运算符 new :

- 运算符 new 用于动态存储分配
- new 返回一个指向所分配空间的指针
- 例: `int * y; y=new int; *y=10;`
- 也可将上述各语句作适当合并如下:
  - `int * y=new int; *y=10;`
  - 或 `int *y=new int(10);`
  - 或 `int *y; y=new int(10);`

## (6.2) 一维数组

❖ 为了在运行时创建一个大小可动态变化的一维浮点数组  $x$ ，可先将  $x$  声明为一个 `float` 类型的指针，然后用 `new` 为数组动态地分配存储空间

❖ 例：

- `float *x=new float[n];`
- 创建一个大小为  $n$  的一维浮点数组，运算符 `new` 分配  $n$  个浮点数所需的空間，并返回指向第一个浮点数的指针
- 然后可用 `x[0]`，`x[1]`，`...`，`x[n-1]` 来访问每个数组元素

## (6.3) 运算符 delete

- ❖ 当动态分配的存储空间已不再需要时，应及时释放所占用的空间
- ❖ 用运算符 delete 来释放由 new 分配的空间
- ❖ 例：
  - delete y;
  - delete [ ]x;
  - 分别释放分配给 \*y 的空间和分配给一维数组 x 的空间

## (6.4) 动态二维数组

- ❖ 创建类型为 Type 的动态工作数组，这个数组有 rows 行和 cols 列

```
template <class Type>
void Make2DArray(Type** &x, int rows, int cols)
{
    x=new Type*[rows];
    for (int i=0; i<rows; i++)
        x[i]=new Type[cols];
}
```

❖ 当不再需要一个动态分配的二维数组时，可按以下步骤释放它的空间

- 首先释放在 for 循环中为每一行所分配的空间
- 然后释放为行指针分配的空间

```
template <class Type>
void Delete2DArray(Type** & x, int rows)
{
    for (int i=0 ;i<rows; i++)
        delete []x[i];
    delete []x;
    x=0;
}
```

❖ 释放空间后将 x 置为 0，以防继续访问已被释放的空间



# Next

## ❖ 算法复杂度分析方法

- Algorithm analysis method