



## 第3章

# 渐进记法



# 本章内容

---

- 正确使用渐进符号 ( $O, \Omega, \Theta, o$ ) , 描述程序的时间复杂性和空间复杂性。
- 折半查找(二分查找)算法



# 渐进符号的引入

- 确定程序的操作计数和步数有两个重要的原因：
  - 比较两个完成同一功能的程序的时间复杂性；
  - 预测随着实例特征的变化，程序运行时间的变化量。
- 操作计数和步数都不能够非常精确地描述时间复杂性。
  - 操作计数：把注意力集中在某些“关键”的操作上，而忽略了所有其他操作。
  - 执行步数：概念本身就不精确。



# 渐进符号的引入

- 引入渐进符号的目的：
  - 描述**大型实例特征下**，时间复杂性和空间复杂性的具体表现。
  - 渐进符号**O**使用最普遍



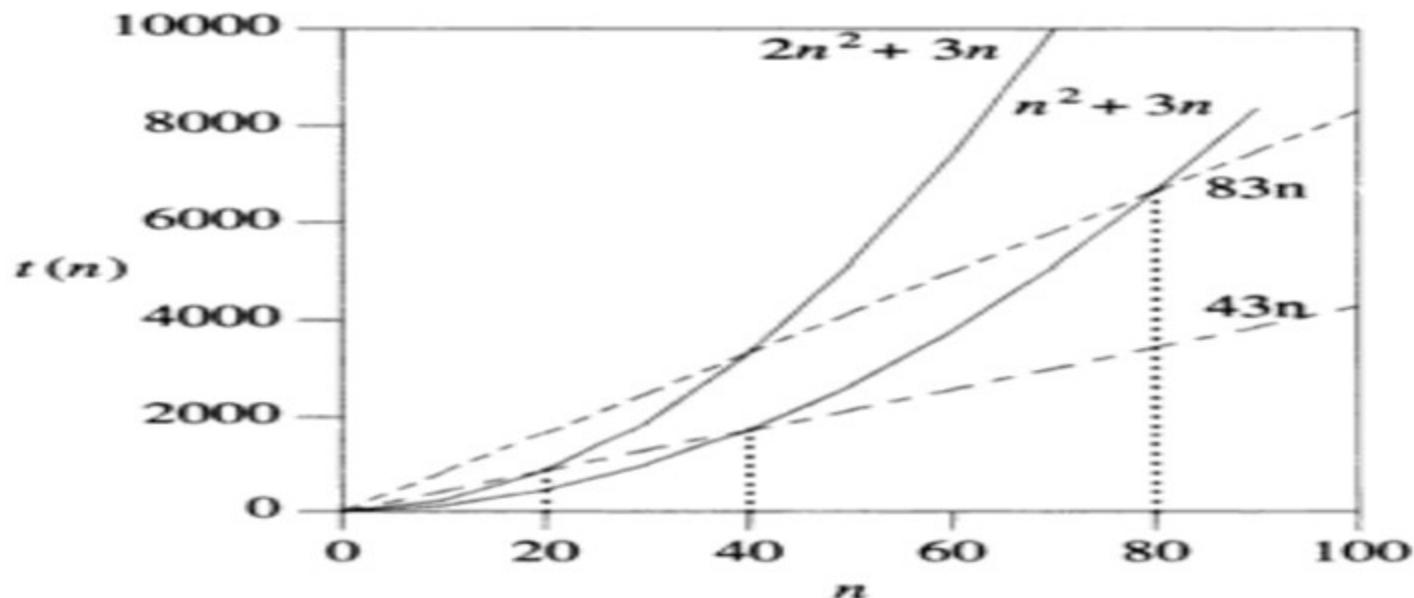
# 时间复杂性比较

- 两个程序的时间复杂性分别为:
  - $c_1n^2 + c_2n$  和  $c_3n$ ,
- 对于足够大的 $n$ , 复杂性为 $c_3n$ 的程序将比复杂性为 $c_1n^2 + c_2n$  的程序运行得快。
- 对于比较小的 $n$ 值, 两者都有可能成为较快的程序(取决于 $c_1$ ,  $c_2$ 和 $c_3$ )。
- 如果:  $c_1=1$ ,  $c_2=2$ ,  $c_3=100$ , 则有
  - $c_1n^2 + c_2n \leq c_3n \quad n \leq 98$
  - $c_1n^2 + c_2n > c_3n \quad n > 98$



# 时间函数比较示例

- $t_A(n) = n^2 + 3n$ ;  $t_B(n) = 43n$ ;
- $t_A(n) = 2n^2 + 3n$ ;  $t_B(n) = 83n$ ;



- $t_A(n) = c_1n^2 + c_2n + c_3$ ;  $t_B(n) = c_4n$ ;



# 渐进符号**O**, **Ω**, **Θ**, **o**

- 设 $f(n)$  表示程序的时间复杂性或空间复杂性 ( $n$ 为实例特征)。
- **O(Big Oh)**符号给出了函数f的一个上限。
- **Ω(Omega)**符号给出了函数f的一个下限。  
**Θ(Theta)**符号， 函数f的上限与下限相同。
- **o(Little oh)**符号。



# 渐进的大于、小于、等于

■ 定义3-1，令 $p(n)$ 和 $q(n)$ 是两个非负函数

**$p(n)$  漐进地大于  $q(n)$**

当且仅当  $\lim_{n \rightarrow \infty} \frac{q(n)}{p(n)} = 0$

**$q(n)$  漐进地小于  $p(n)$**

当且仅当  **$p(n)$  漐进的大于  $q(n)$**

**$p(n)$  漐进地等于  $q(n)$**

当且仅当 任何一个都不是渐进的大于另一个



## 例3-1

- $\lim_{n \rightarrow \infty} \frac{10n+7}{3n^2+2n+6} = 0$
  - $3n^2 + 2n + 6$  漐进地大于  $10n + 7$
  - $10n + 7$  漐进地小于  $3n^2 + 2n + 6$
- 
- $8n^4 + 9n^2$  漐进地大于  $100n^3 - 3$
  - $2n^2 + 3n$  漐进地大于  $83n$
  - $12n + 6$  漐进地等于  $6n + 2$



# $f(n)$ 中的项

- $f(n)$  : 表示程序的时间复杂性或空间复杂性( $n$ 为实例特征)。
- $f(n)$  一般为若干项之和
- 例:  $f(n) = 3n^2 + 2n + 6$ ,
  - 项:  $3n^2, 2n, 6$
  - 最大项:  $3n^2$



# $f(n)$ 中通常出现的项

<u>项</u>	<u>名称</u>
1	常数
$\log n$	对数
$n$	线性
$n \log n$	$n$ 个 $\log$
$n^2$	平方
$n^3$	立方
$2^n$	指数
$n!$	阶乘

- $1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$
- $<$ : 演进地小于



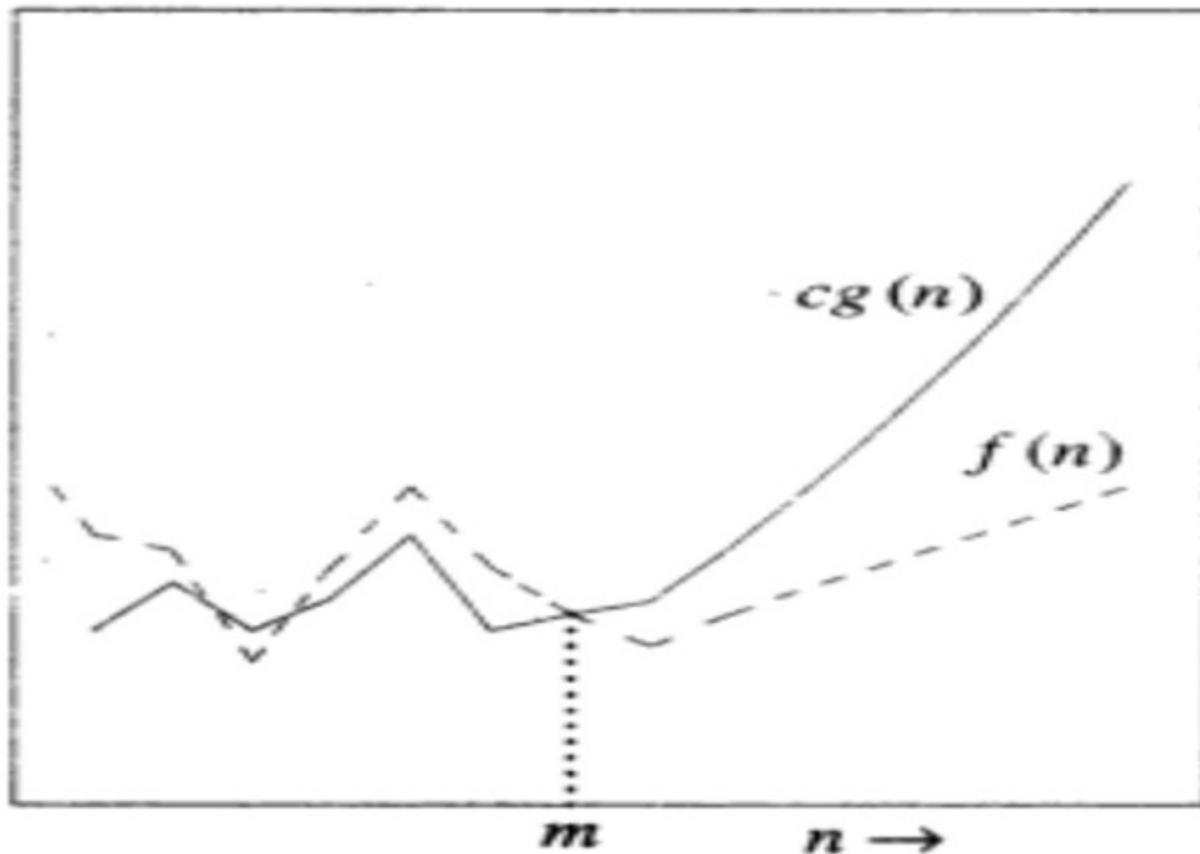
# 大O记法

- 定义3-3[大O记法] :
- $f(n)=O(g(n))$  (读作 “ $f(n)$  是  $g(n)$  的大O”), 当且仅当存在正的常数  $c$  和  $n_0$ , 使得对于所有的  $n, n \geq n_0$ , 有  $f(n) \leq cg(n)$  【 $f(n)=O(g(n))$  表示  $f(n)$  渐进地小于或等于  $g(n)$ 】。
- $g$  是  $f$  的一个上限(不考虑常数因子  $c$ )
  - $O(g(n))$  表示当  $n$  增大时,  $f(n)$  至多将以正比于  $g(n)$  的速度增长。
  - $n$  足够大时,  $f(n)$  不大于  $g(n)$  的一个常数倍。



$$f(n) = O(g(n))$$

- $g$  是  $f$  的一个上限(不考虑常数因子  $c$ )





# 线性函数

## ■ 例 3-7

$$f(n)=\mathbf{3n+2}$$

当  $n \geq n_0 = 2$  时,  $f(n) = 3n + 2 \leq 3n + n = 4n$

$$f(n)=\mathbf{O(n)}$$

$$f(n)=100n+6,$$

当  $n \geq n_0 = 6$ ,  $f(n) = 100n + 6 \leq 100n + n = 101n$

$$f(n)=\mathbf{100n+6=O(n)}.$$



# 平方函数

## ■ 例 3-8

$$f(n) = 10n^2 + 4n + 2$$

$$n \geq 2, f(n) \leq 10n^2 + 5n$$

$$n \geq 5, 5n \leq n^2$$

$$n \geq n_0 = 5,$$

$$\begin{aligned}f(n) &\leq 10n^2 + n^2 \\&= 11n^2,\end{aligned}$$

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

$$10n^2 + 4n + 2 = O(n^2)$$



# 指数函数

## ■ 例 3-9

$$f(n) = 6 \cdot 2^n + n^2$$

$$n \geq 4, \quad n^2 \leq 2^n,$$

$$n \geq 4, \quad f(n) \leq 6 \cdot 2^n + 2^n = 7 \cdot 2^n$$

$$6 \cdot 2^n + n^2 = O(2^n)$$



# 常数函数

## ■ 例 3-10

$$f(n)=c$$

$$f(n)=O(1)$$



# 最小上限

- 例 3-11

$$f(n)=3n+3$$

$$n \geq 3, f(n)=\mathbf{3n+3} \leq 3n+n=4n=\mathbf{O(n)}$$

$$n \geq 2, f(n)=\mathbf{3n+3} \leq 3n^2 = \mathbf{O(n^2)} \text{(不是最小上限)}$$

- 语句  $f(n)=O(g(n))$  仅表明对于所有的  $n \geq n_0$ ,  $c g(n)$  是  $f(n)$  的一个上限。它并未指出该上限是否为最小上限。
- 为了使语句  $f(n)=O(g(n))$  有实际意义, 其中的  $g(n)$  应尽量地小。



# 渐进复杂性分析

- $f(n) = O(g(n))$
- $f(n) = 0, g(n) = 0$
- 除 $f(n)=0$ 以外,  $g(n)$ 通常是
  - 令 $f(n)=O(g(n))$ 为真的最小单位项 (系数为1)
  - $f(n) = 10n + 7 = O(3n^2 + 2n + 6)$
  - $f(n) = 10n + 7 = O(n^2)$
  - $f(n) = 10n + 7 = O(n) \quad \checkmark$
- 
- $f(n) = 8n^4 + 9n^2 = O(n^4)$   
 $f(n) = 100n^3 - 3 = O(n^3)$   
 $f(n) = 3n^2 + 2n + 6 = O(n^2)$   
 $f(n) = 12n + 6 = O(n)$



# 渐进复杂性分析

- 渐进复杂性分析，用步数中渐进最大的项来描述复杂度。
  - $f(n)$ : 步数函数（或空间复杂性或时间复杂性）
  - $g(n)$ : 步数函数中单位项(系数为1的单项)中的最大项(渐进最大的项)
- 例： $f(n)=3n^2+6n\log n+7n+5$
- 单位项： $n^2$ 、 $n\log n$ 、 $n$ 、 $1$
- 最大单位项： $n^2$
- $f(n)=3n^2+6n\log n+7n+5$   
 $=O(n^2)$



# $\Omega$ 符号

- 定义3-4[ $\Omega$ 符号]:
- $f(n)=\Omega(g(n))$  (读作 “ $f(n)$  是  $g(n)$  的  $\Omega$  ”) 当且仅当存在正的常数  $c$  和  $n_0$ , 使得对于所有的  $n$ ,  $n \geq n_0$ , 有  $f(n) \geq cg(n)$  。 [ $f(n)=\Omega(g(n))$  表示  $f(n)$  漐进地大于或等于  $g(n)$ ]

- $g$  是  $f$  的一个下限(不考虑常数因子  $c$ )。

$$f(n)=3n+2 \geq 3n = \Omega(n)$$

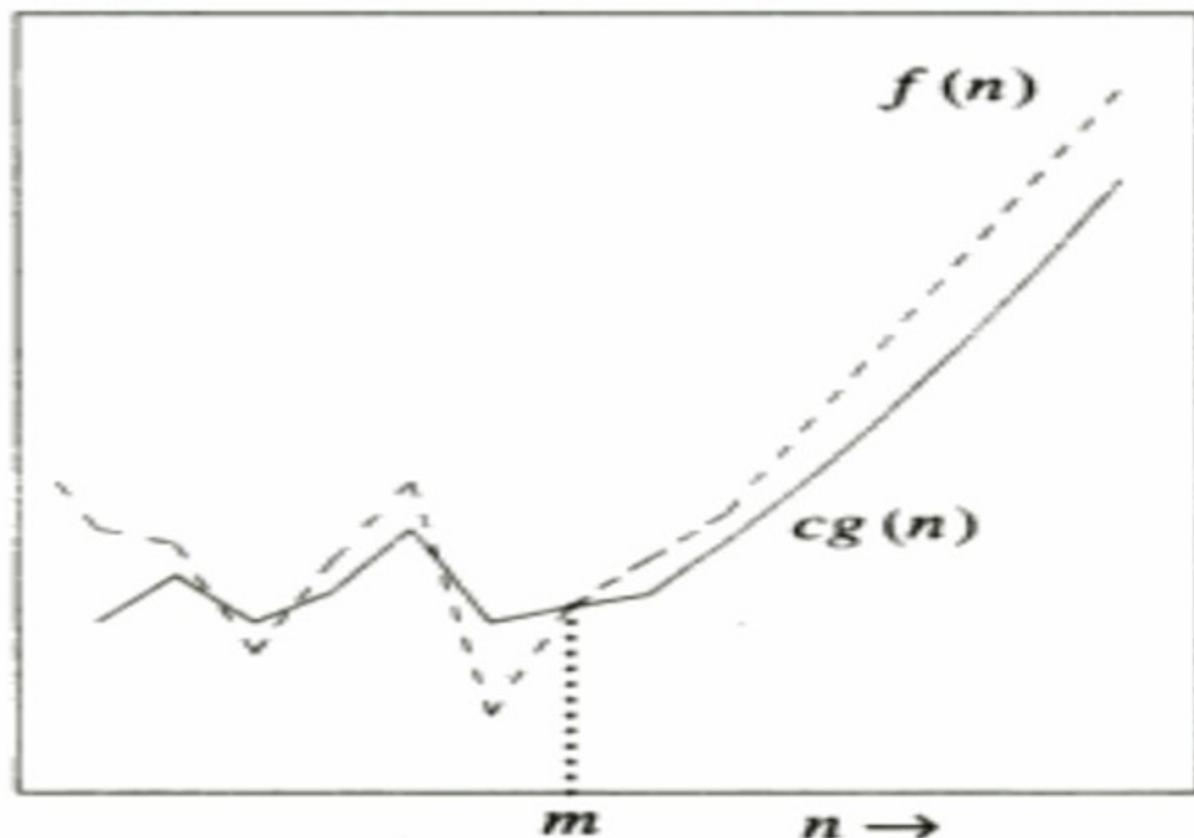
$$f(n)=10n^2 + 4n + 2 \geq 10n^2 = \Omega(n^2)$$

$$f(n)=6*2^n + n^2 \geq 6*2^n = \Omega(2^n)$$



$$f(n) = \Omega(g(n))$$

- $g$  是  $f$  的一个下限(不考虑常数因子  $c$ )





# 最大下限

- 为了使语句 $f(n)=\Omega(g(n))$ 更有实际意义，其中的 $g(n)$ 应足够地大。

- $f(n)=3n+2 \geq 3n = \Omega(n)$
- $f(n)=3n+2 = \Omega(1)$

→ 使用 $3n+2=\Omega(n)$

- $6*2^n+n^2 = \Omega(2^n);$
- $6*2^n+n^2 = \Omega(1)$

→ 使用 $6*2^n+n^2=\Omega(2^n)$



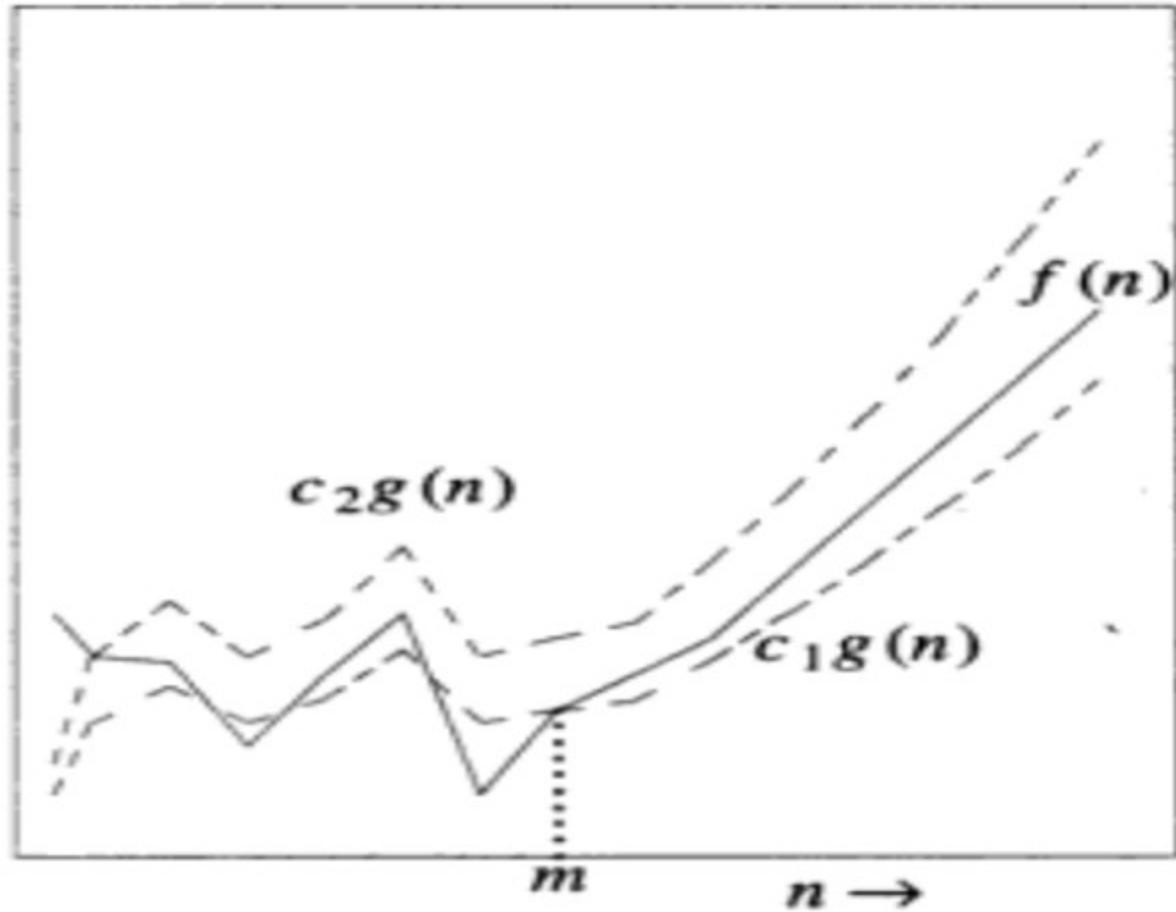
## Θ符号

- 对于所有足够大的 $n$  (如 $n \geq n_0$ ),  $g$ 既是 $f$ 的上限也是 $f$ 的下限(不考虑常数因子 $c$ )。
- 定义3-5[ $\Theta$ 符号]:  $f(n)=\Theta(g(n))$  (读作 “ $f(n)$  是 $g(n)$  的 $\Theta$ ”), 当且仅当存在正常数 $c_1, c_2$ 和 $n_0$ , 使得对于所有的 $n$ ,  $c_1g(n) \leq f(n) \leq c_2g(n)$  【 $f(n)=\Theta(g(n))$  表示  $f(n)$  渐进地等于  $g(n)$ 】
- 函数 $f$ 介于函数 $g$ 的 $c_1$ 倍和 $c_2$ 倍之间, 除非 $n$ 小于 $n_0$ 。



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

- $g$ 既是 $f$ 的上限也是 $f$ 的下限(不考虑常数因子 $c$ )





# Θ符号

$$f(n) = 3n + 2 = \Theta(n)$$

$$f(n) = 10n^2 + 4n + 2 = \Theta(n^2)$$

$$f(n) = 6 * 2^n + n^2 = \Theta(2^n)$$



# 小o记法

- 定义[小o符号]:
- $f(n)=o(g(n))$  (读作 “ $f(n)$  是  $g(n)$  的小  $O$ ”), 当且仅当  
 $f(n)=O(g(n))$  且  $f(n) \neq \Omega(g(n))$ .



# 渐进等式(P74)

	$f(n)$	渐近分析法
E1	$c$	$\Theta(1)$
E2	$\sum_{i=0}^k c_i n^i$	$\Theta(n^k)$
E3	$\sum_{i=1}^n i$	$\Theta(n^2)$
E4	$\sum_{i=1}^n i^2$	$\Theta(n^3)$
E5	$\sum_{i=1}^n i^k, k > 0$	$\Theta(n^{k+1})$
E6	$\sum_{i=0}^n r^i, r > 1$	$\Theta(r^n)$
E7	$n!$	$\Theta(\sqrt{n}(n/e)^n)$
E8	$\sum_{i=1}^n 1/i$	$\Theta(\log n)$
		$\Theta$ 可以表示 $O$ 、 $\Omega$ 和 $\Theta$ 中的任何一个



# 关于渐进符号的推理规则(**P74**)

$$I1 \quad \{f(n) = \Theta(g(n))\} \rightarrow \sum_{n=a}^v f(n) = \Theta(\sum_{n=a}^v g(n))$$

$$I2 \quad \{f_i(n) = \Theta(g_i(n)), 1 \leq i \leq k\} \rightarrow \sum_1^k f_i(n) = \Theta(\max_{1 \leq i \leq k} \{g_i(n)\})$$

$$I3 \quad \{f_i(n) = \Theta(g_i(n)), 1 \leq i \leq k\} \rightarrow \prod_1^k f_i(n) = \Theta(\prod_1^k g_i(n))$$

$$I4 \quad \{f_1(n) = O(g_1(n)), f_2(n) = \Theta(g_2(n))\} \rightarrow f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$$

$$I5 \quad \{f_1(n) = \Theta(g_1(n)), f_2(n) = \Omega(g_2(n))\} \rightarrow f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$$

$$I6 \quad \{f_1(n) = O(g(n)), f_2(n) = \Theta(g(n))\} \rightarrow f_1(n) + f_2(n) = \Theta(g(n))$$



# 复杂性分析举例：

```
template<class T>
T sum(T a[], int n)
    {//计算a[0:n - 1]中元素之和
    T theSum=0;
    stepCount++; //对应于theSum=0
    for (int i=0; i<n; i++) {
        stepCount++; //对应于for语句
        theSum +=a[i];
        stepCount++; //对应于赋值语句
    }
    stepCount++; //对应于最后一个for语句
    stepCount++; //对应于return语句
    return theSum;
}
```

步数： $2n+3$

$$t_{\text{sum}}(n) = 2n+3 = \Theta(n)$$



# 函数sum(程序1-30)的渐进复杂性

语句	s/e	频率	总步数
T sum(T a[], int n)	0	0	$\Theta(0)$
{	0	0	$\Theta(0)$
T theSum=0;	1	1	$\Theta(1)$
for(int i=0; i<n; i++)	1	n+1	$\Theta(n)$
theSum +=a[i];	1	n	$\Theta(n)$
return theSum;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{\text{sum}}(n) = \Theta(\max(g_i(n))) = \Theta(n)$$



# sequentialSearch的渐进复杂性

语句	s/e	频率	总步数
int sequentialSearch(T a[],T& x, int n)	0	0	$\Theta(0)$
{	0	0	$\Theta(0)$
int i;	1	1	$\Theta(1)$
for (int i=0; i<n && a[i]!=x; i++)	1	$\Omega(1), O(n)$	$\Omega(1), O(n)$
if (i==n) return -1;	1	1	$\Theta(1)$
return i;	1	$\Omega(0), O(1)$	$\Omega(0), O(1)$
}	0	0	$\Theta(0)$

$$t_{\text{SequentialSearch}}(n) = \Omega(1)$$

$$t_{\text{SequentialSearch}}(n) = O(n)$$



# 求排列(程序1-32)的渐进复杂性

程序1-32

```
template<class T>
void permutations(T list[], int k, int m)
{ //生成list[k:m]的所有排列方式，输出前缀是list[0:k-1] 后缀是
list[k:m]的所有排列方式
    int i;
    if (k == m) { // list[k:m]只有一个排列
        copy(list, list+m+1, ostream_iterator<T>(cout, " "));
        cout << endl;
    }
    else // list[k:m ]有多个排列方式，递归地产生这些排列方式
        for (i=k; i <= m; i++) {
            swap (list[k], list[i]);
            permutations (list, k+1, m);
            swap (list[k], list[i]);
        }
}
```



# 求排列(程序1-32)的渐进复杂性

- 假定 $m=n-1$ 。
- $k=m$ : 所需要的时间为 $cn$ ( $c$ 是一个常数)。
  - $t_{\text{permutations}}(k,m) = t_{\text{permutations}}(m,m) = cn$
- $k < m$ : 执行else语句，
  - for循环将被执行 $m-k+1$ 次
  - 每次循环所花费的时间:  $dt_{\text{permutations}}(k+1,m)$ ,  $d$ 是一个常数.
  - $t_{\text{permutations}}(k,m) = d(m-k+1)t_{\text{permutations}}(k+1,m)$ 。使用置换的方法, 可以得到:
- $t_{\text{permutations}}(0,m) = \Theta((m+1)*(m+1)!) = \Theta(n*n!)$ , 其中 $n \geq 1$ 。



## 例 3-24 折半搜索 (Binary Search)

- 在**有序数组**a中查找元素x: 折半搜索(查找)
- 折半搜索(查找)思想:
  - left指向数据段的左端, right指向数据段的右端
  - x与**搜索段 [left:right]**中间位置**middle**元素比较
  - 如果x**等于**中间元素, 则查找结束
  - 如果x**小于**中间元素, 则需要查找**搜索段的左半部分**, right被修改为**middle-1**, 继续在a[left:right]中查找。
  - 如果x**大于**中间元素, 则仅需要在**搜索段的右半部分**进行查找, left 将被修改为**middle+1** , 继续在a[left:right]中查找。



# 搜索过程示例

[05 13 19 21 37 56 64 75 80 88 92]



left



middle



right

[05 13 19 21 37] 56 64 75 80 88 92



left



middle



right

05 13 19 [21 37] 56 64 75 80 88 92



left



right

middle

查找 $x=21$ 的过程(查找成功)



## 搜索过程示例

[05 13 19 21 37 56 64 75 80 88 92]



05 13 19 21 37 56 [64 75 80 88 92]



05 13 19 21 37 56 64 75 80 [88 92]



05 13 19 21 37 56 64 75 80] [88 92]

查找 $x=85$ 的过程(查找失败)



# 折半搜索 (Binary Search)

程序3-1

```
template<class T>
int binarySearch(T a[], const T& x, int n)
{//在有序数组a中查找元素x
 //如果存在，就返回元素x的位置，否则返回-1
int left=0; //left指向数据段的左端
int right=n-1 ;//right指向数据段的右端
while (left≤right) {
    int middle=(left+right)/2;//数据段的中间
    if (x==a[middle]) return middle;
    if (x > a[middle]) left=middle + 1;
    else right=middle-1 ;
}
return -1; //没有找到x
```

最坏情况下，  
时间复杂性：

$O(\log n)$

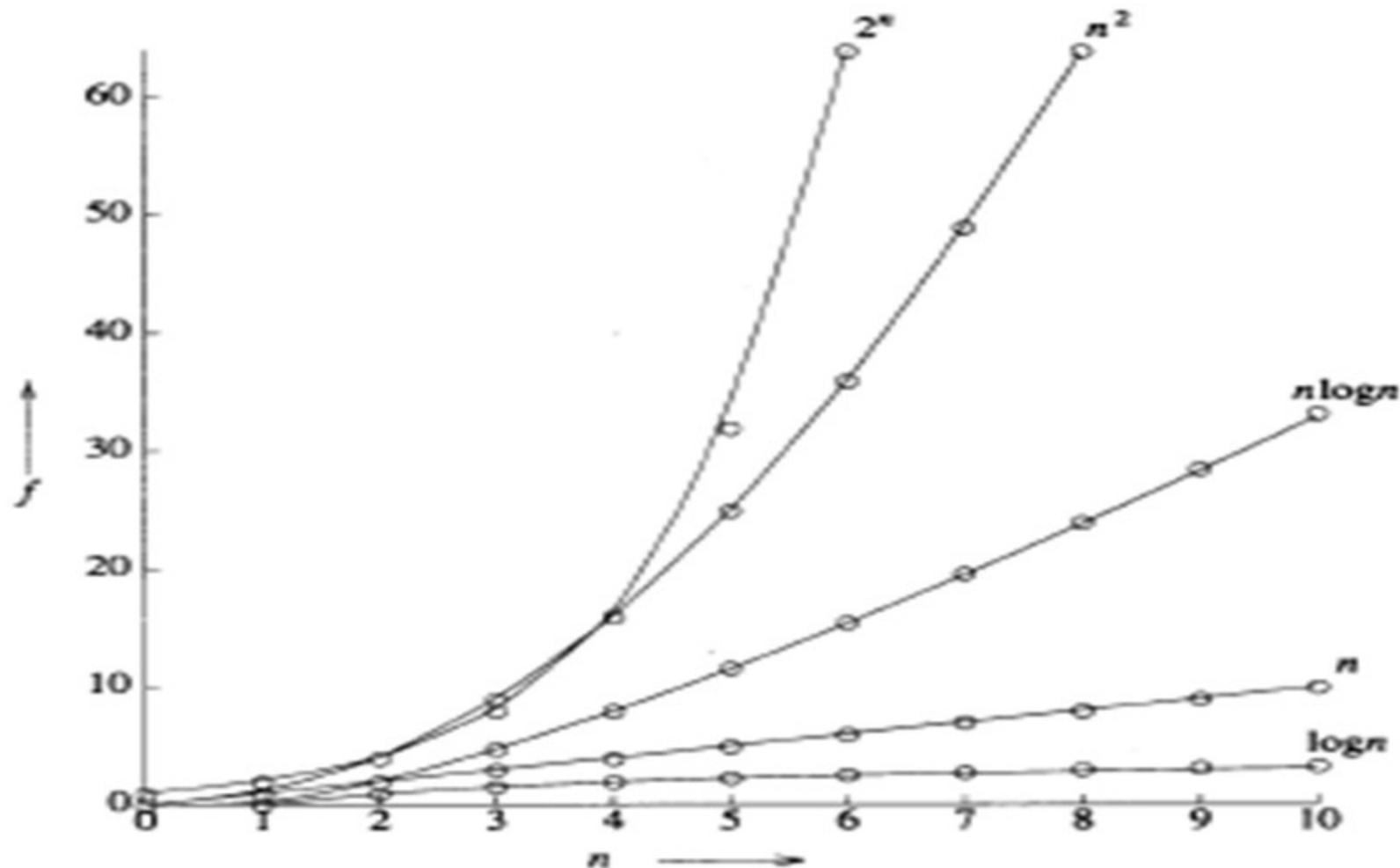


## 3.5 实际复杂性

log n	n	n log n	$n^2$	$n^3$	$2^n$
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296



# 实际复杂性





# 实际复杂性数据示例

- 如果程序执行步是 $n^3$ , 在每秒执行1000 000 000步的计算机执行
    - $n=1000$ , 需要执行1秒;
    - $n=10000$ , 需要110. 67分钟;
    - $n=100000$ , 需要11. 57天。
  - 如果程序执行步 $n^{10}$ , 在每秒执行1000 000 000步的计算机执行
    - $n=10$ , 需要10秒钟;
    - $n=100$ , 需要3171年;
    - $n=1000$ , 需要 $3.17 \times 10^{13}$ 年。
- 具有高次多项式复杂性的函数也必须限制使用。



# 实际复杂性数据示例

- 随着 $n$ 的增长， $2^n$ 的增长极快。
- 假设程序执行步 $2^n$ ，该程序在每秒执行1000 000 000步的计算机上执行，大约需要时间：
  - $n=40$ , 18. 3分钟;
  - $n=50$ , 13天;
  - $n=60$ , 310. 56年;
  - $n=100$ ,  $4*10^{13}$ 年。
- ➡ 具有指数复杂性的程序仅适合于小的 $n$ （典型地取 $n \leq 40$ ）。



## 第4章

# 性能测量



# 性能测量

- 性能测量 (performance measurement) 主要关注于得到一个程序实际需要的空间和时间。
- 空间密切相关：
  - 特定的编译器
  - 编译器选项
  - 执行程序的计算机
- 不能精确地测量一个程序运行时所需要的空间
- 程序的运行时间：使用C++函数clock()



# 时间测量

- 测量程序(以排序为例), 需要
  - 确定实例特征 $n$ 的一组值
  - 对于实例特征 $n$ 的每一个值, 设计测试数据
    - 可以人工设计或借助计算机设计相应的测试数据
  - 编写程序, 测量运行时间
    - 为了提高测量的精确度, 对于实例特征的每一个值, 可以重复求解若干次。
    - 实际测量时间包括: 排序的时间、额外时间(每次对 $a$  初始化等)