第一讲 基本概念

浙江大学 陈 越

1.2 什么是算法

定义

- 算法（**Algorithm**）
  - 一个有限指令集
  - 接受一些输入（有些情况下不需要输入）
  - 产生输出
  - 一定在有限步骤之后终止
  - 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1：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？

什么是好的算法？

- 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。

1.1 例2

```
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)
```

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

1.1 例3

```
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

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

```
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！

$T(n) = C \cdot n$

什么是好的算法？

- 在分析一般算法的效率时，我们经常关注下面两种复杂度
  - 最坏情况复杂度 $T_{worst}(n)$
  - 平均复杂度 $T_{avg}(n)$

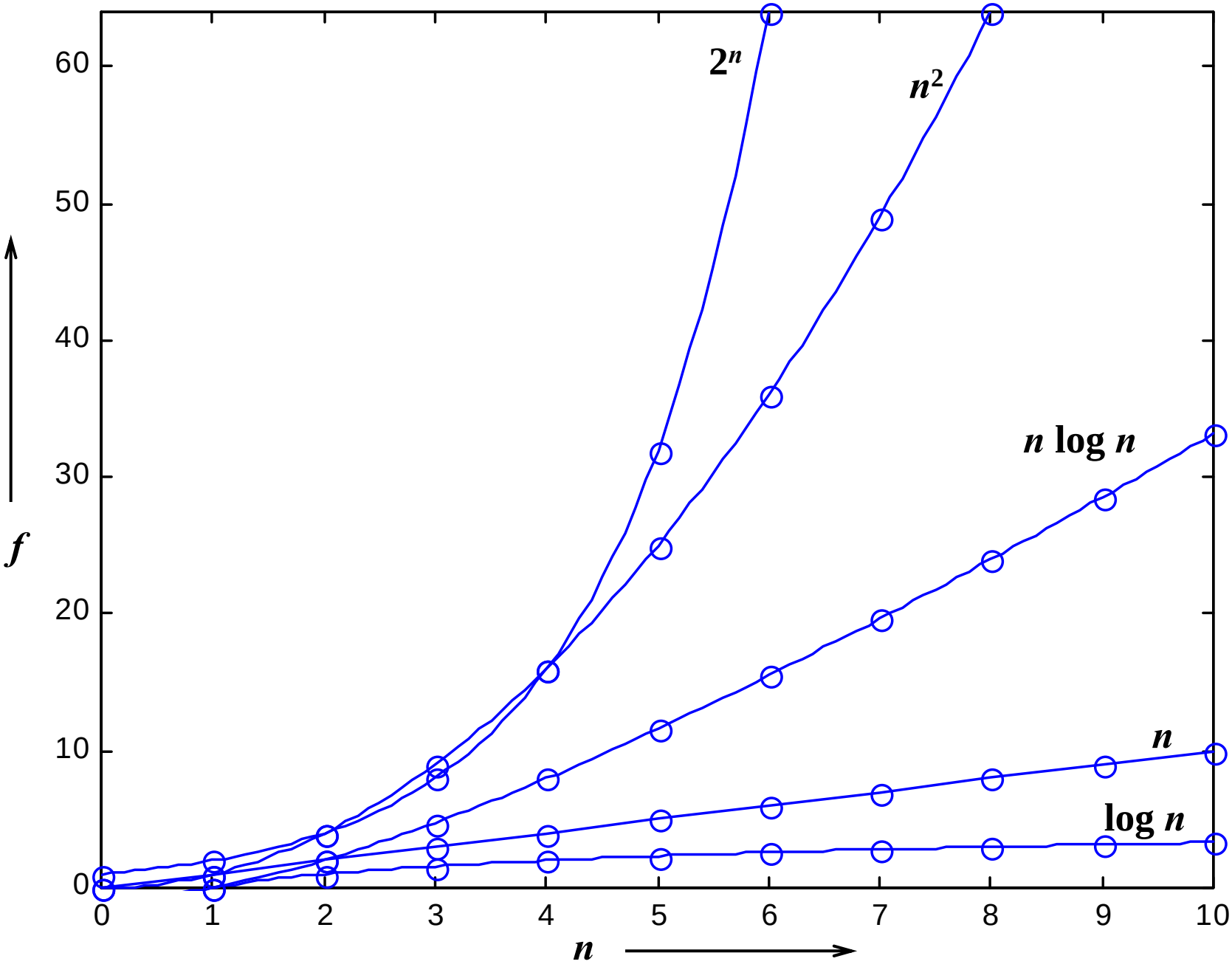
$$T_{avg}(n) \leq T_{worst}(n)$$

复杂度的渐进表示法

- $T(n) = O(f(n))$ 表示存在常数$C > 0, n_0 > 0$ 使得当 $n \geq n_0$ 时有$T(n) \leq C \cdot f(n)$
- $T(n) = \Omega(g(n))$ 表示存在常数$C > 0, n_0 > 0$ 使得当 $n \geq n_0$ 时有$T(n) \geq C \cdot g(n)$
- $T(n) = \Theta(h(n))$ 表示同时有$T(n) = O(h(n))$ 和 $T(n) = \Omega(h(n))$

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}


2^n
n^2
$n \log n$
n
$\log n$
f
n
0
10
20
30
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

浙江大学计算机科学与技术学院

复杂度分析小窍门

- 若两段算法分别有复杂度$T_1(n) = O(f_1(n))$ 和$T_2(n) = O(f_2(n))$，则
  - $T_1(n) + T_2(n) = \max(O(f_1(n)), O(f_2(n)))$
  - $T_1(n) \times T_2(n) = O(f_1(n) \times f_2(n))$
- 若$T(n)$是关于n的k阶多项式，那么$T(n)= \Theta (n^k)$
- 一个`for`循环的时间复杂度等于循环次数乘以循环体代码的复杂度
- `if-else` 结构的复杂度取决于`if`的条件判断复杂度和两个分枝部分的复杂度，总体复杂度取三者中最大