INT202W01_复杂度分析

课程信息

考核：

- ICT1-W06 10%
- ICT2-W13 10%
- Final 80% 可带1张小抄

教师：

- Module leader: Dr. Rui Yang
  - • Email: R.Yang@xjtlu.edu.cn
  - • Week 1-6
- Co-Lecturer: Dr. Yaran Chen
  - • Email: Yaran.Chen@xjtlu.edu.cn
  - • Week 8-13

算法和数据结构

- 数据结构：组织和访问数据的一种系统方式（systematic way）
- 算法：在有限时间内执行任务的一系列步骤

> **算法分析 Algorithm Analysis**
>
> 如何衡量算法效率和好坏？
>
> - 首要兴趣：算法运行的时间（时间复杂度）和对数据结构的行为
> - 次要兴趣：内存使用（空间复杂度）

实验分析（Experimental Analysis）：

运行时间或内存要求对输入大小的依赖性。需要选择良好的样本输入和适当数量的测试，从而有统计确定性。取决于 input 的大小和实例、使用的算法，以及运行它的软件和硬件环境。

实验的局限性：

- 在一组有限的测试输入上进行；
- 需要在同样的硬件和软件执行测试；
- 需要运行算法。

理论分析（Theoretical Analysis）：

优势：可以考虑所有可能的输入；不受软硬件环境的影响，比较多种算法；涉及研究算法的高级表达（伪代码pseudocode）

需要：

- 一种用于描述算法的语言。
- 执行算法的计算模型。
- 用于衡量性能的指标。
- 一种描述性能的方法。

> 本课程不考察书写伪代码，但要求读懂伪代码。
>
> 定义了一组高级基元操作，很大程度上独立于所使用的编程语言。
>
> 基元操作（Primitive Operations）包括以下内容：变量赋值；调用方法；算数运算；比较数字；索引数组；对象引用；方法返回。

随机存取机（Random Access Machine）

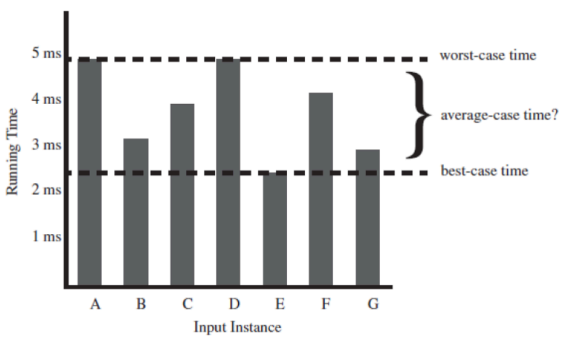
计算模型：CPU连接到一组储存单元，每个储存单元可存储一个数字，字符，地址等。

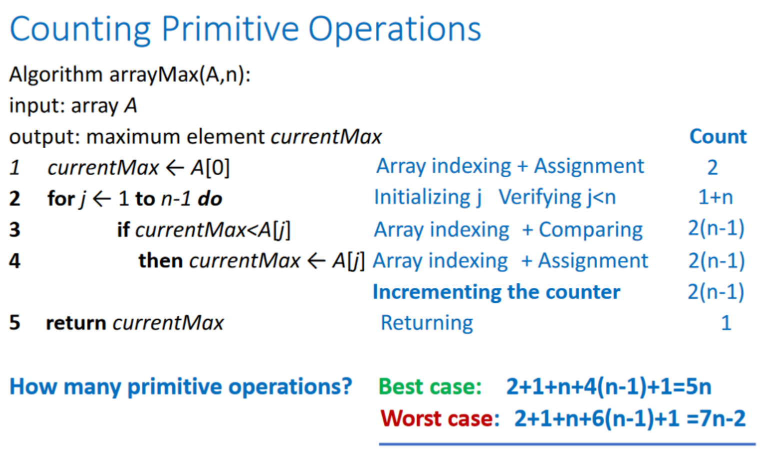
我们假设原始运算（如加减乘除，比较）花费相同的时间执行。

平均复杂度与最坏复杂度（Average-/ Worst-Case Complexity）

- 平均复杂度：所有相同大小输入的运行时间的平均值。
- 最坏复杂度：所有相同大小输入的运行时间的最大值。

我们会对最坏复杂度感兴趣。



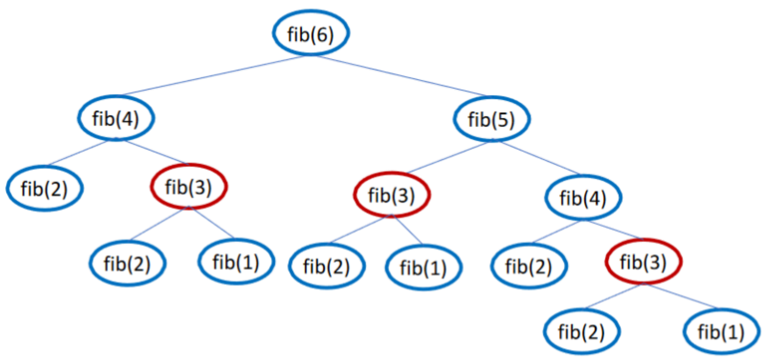


我们不会考察对基元操作的计数（比如Incrementing the counter的count），只要数量级对就行。

递归算法

虽然递归算法的编写比非递归版简单，但在很多情况下递归算法需要重复解决子问题，故不高效。

如在计算斐波那契数列的过程中，fib(5)和fib(4)都重复调用了fib(3)：



对于较大的输入值，重复的函数调用可能会耗尽机器的内存。

渐进表示法（Asymptotic notation）

渐进表示法允许我们描述影响运行时间的主要因素。

用于简化估计常数前执行原始操作的分析。

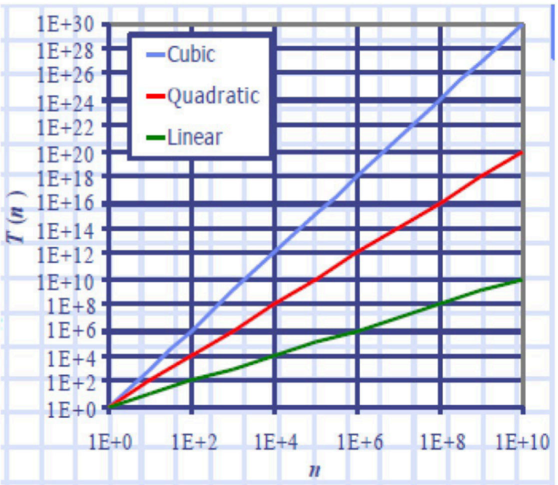
从而可以比较两种算法的运行时间。

假设机器频率为1MHz，下表展示了1秒，1分钟，1小时可以解决的问题大小：

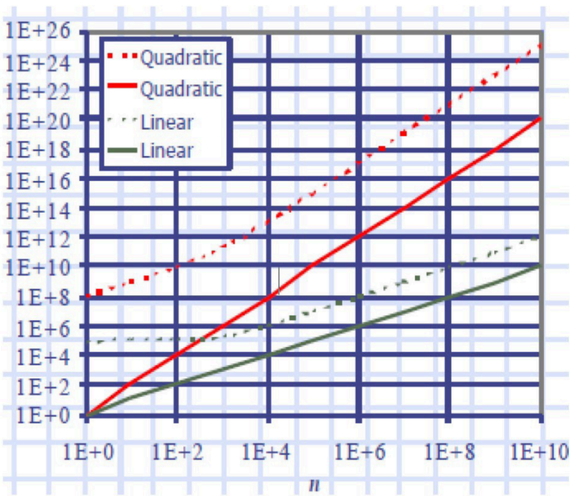
Running Time	Maximum problem size (n)		
	1 second	1 minute	1 hour
$400n$	2,500	150,000	9,000,000
$20n \log n$	4,096	166,666	7,826,087
$2n^2$	707	5,477	42,426
n^4	31	88	244
2^n	19	25	31

可见，运行时间渐近较慢的算法会被运行时间渐近较快的算法击败。

在log-log图下，斜率展示了增长率（growth rate）：



增长率不受常数或低阶量影响（即最终斜率会相同）：



大O表示法（"Big-Oh" Notation）

最常用的渐近表示法形式。

若两个给定正函数$f(n) = O(g(n))$则有任意常数c, n_0满足$f(n) \le c \cdot g(n)\ for\ all\ n \ge n_0$.

考试需要写大O的证明过程：

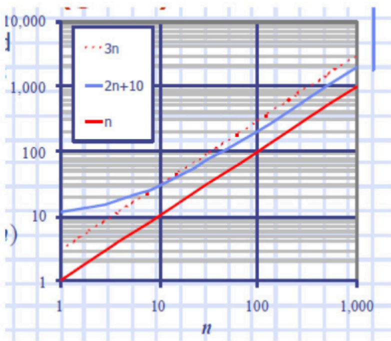
$13n^3 + 7nlog(n) + 3\ is\ O(n^3)$

Proof: $13n^3 + 7nlog(n) + 3\ \le\ 16n^3\ for\ n \ge 1$

“Big-Oh” Notation

Example: 2***n*** + 10 is ***O***(***n***)

- 2***n*** + 10 ≤ ***cn***
- (***c*** - 2) ***n*** ≥ 10
- ***n*** ≥ 10/(***c*** - 2)
- Pick ***c*** = 3 and $\boldsymbol{n_0}$ = 10

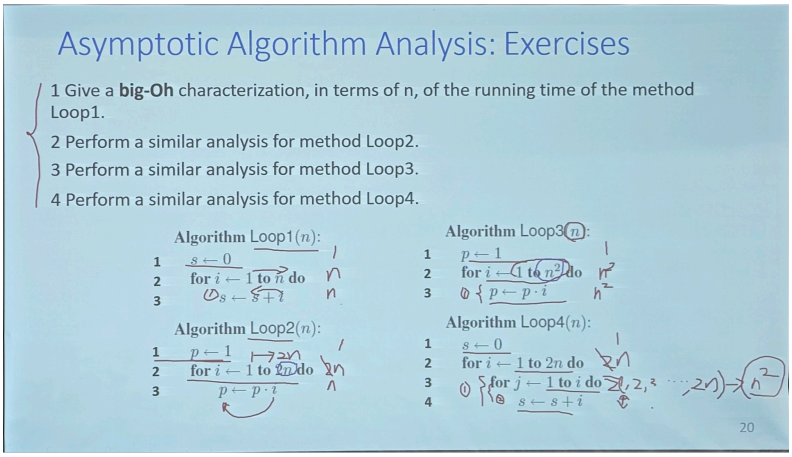


Asymptotic Algorithm Analysis渐进算法分析：

```
Algorithm prefixAverages1(X, n)
    Input array X of n integers
    Output array A of prefix averages of X      #operations
     A ← new array of n integers                 n
    for i ← 0 to n − 1 do                        n
        s ← X[0]                                 n
        for j ← 1 to i do                        1 + 2 + …+ (n − 1)
            s ← s + X[j]                         1 + 2 + …+ (n − 1)
        A[i] ← s / (i + 1)                       n
    return A                                     1
```

发现prefixAverages1 的运行时间是O(1 + 2 + …+ n)=O(n(n + 1) / 2)~O(n^2)

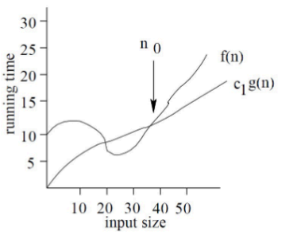
由于常数因子和低阶项最终都会被丢弃，因此我们在计算原始运算时可以忽略它们

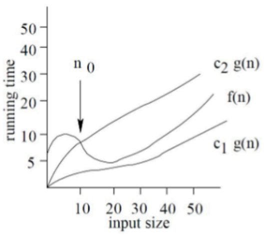


Ω(n)和Θ(n)记号

大Omega：若存在常数c和n_0，使得$f(n) \geq cg(n)\ for\ all\ n \geq n_0$

大Theta：$f(n) = \Theta(g(n))$，若存在$f(n) = O(g(n))$且$f(n) = \Omega(g(n))$





- 大O：Big-Oh

  f(n) is O(g(n)) if f(n) is asymptotically less than or equal to g(n)

  f(n) 渐近小于或等于 g(n)

- 大Ω：Big-Omega

  f(n) is Ω(g(n)) if f(n) is asymptotically greater than or equal to g(n)

  f(n) 渐近大于或等于 g(n)

- 大θ：Big-Theta

  f(n) is Θ(g(n)) if f(n) is asymptotically equal to g(n)

  f(n) 渐近等于 g(n)