Efficiency的多种测量方法：

使用更少的内存

更快的运转时间·

更快的释放分配资源

等等

效率的测量应该与

所用的软件（编译器，语言等等）无关

和硬件无关（CPU等等）

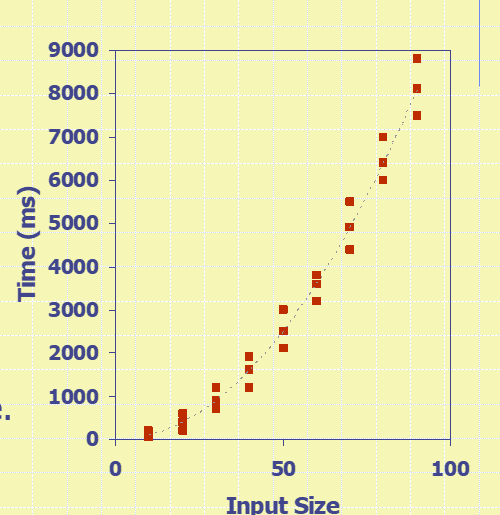
利用程序实验来评价算法效率：

1.Write a program implementing the algorithm 写一个体现算法的程序

2. Run the program with inputs of varying size and composition 使用不同大小和组合的input运行程序

3.利用System.currentTimeMillis() 来确认实际运行时间

4.把每次不同的结果plot出来



程序实验法的缺陷：

1. 必须要实际执行这个算法，有可能是困难/高花费的
2. 为了比较两个不同算法，必须使用同一个硬件和软件环境

因此为了测量效率我们更多用抽象法

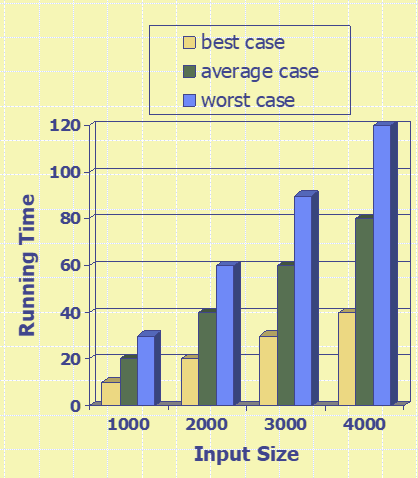
效率很大程度上取决于METHOD是怎么写的，所以根据Method我们能算出效率

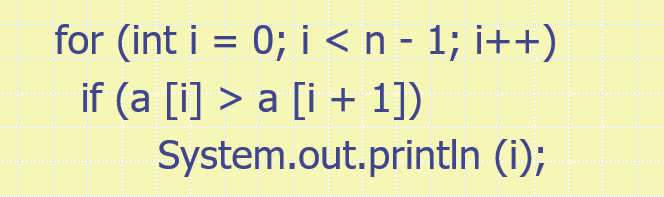
忽略各种限制，例如CPU速度，存储大小（无限大，比如ARRAY长度可以无限大）

既然方法与环境无关，我们叫他算法

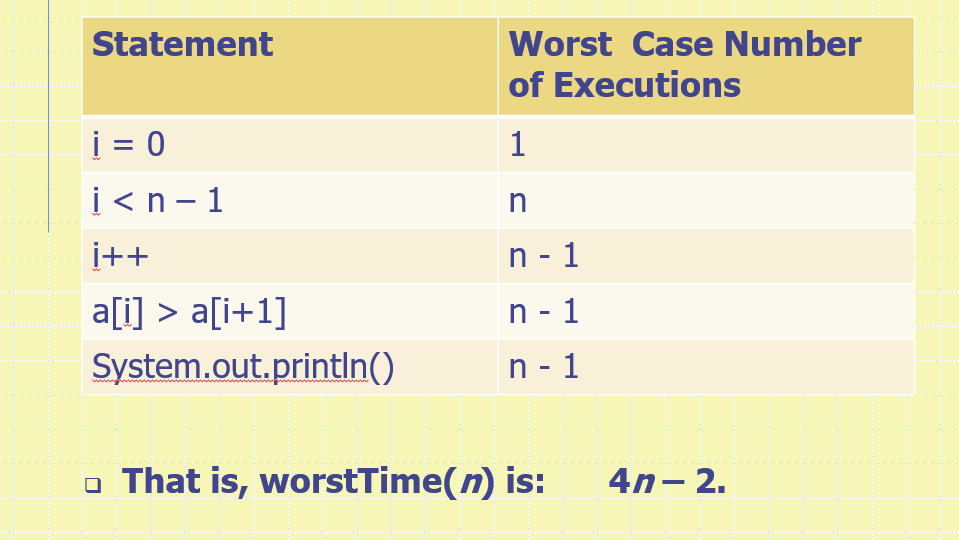
算法的运行时间通常随着input数据的增长而增长

我们通常关注最差情况，对于一个size为n的问题，***worstTime(n)既最大运行语句***



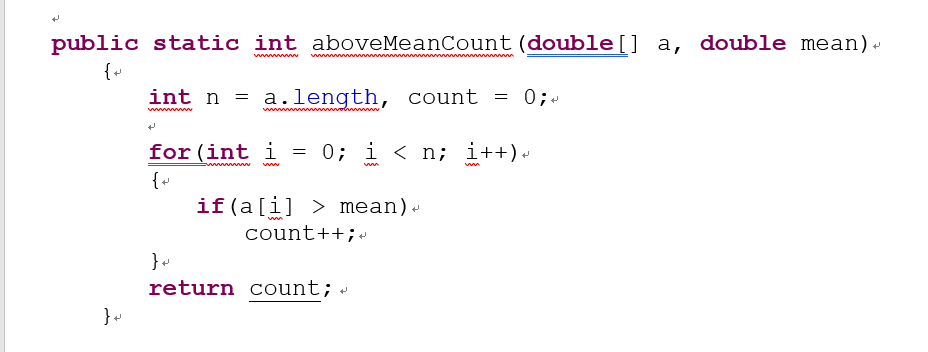


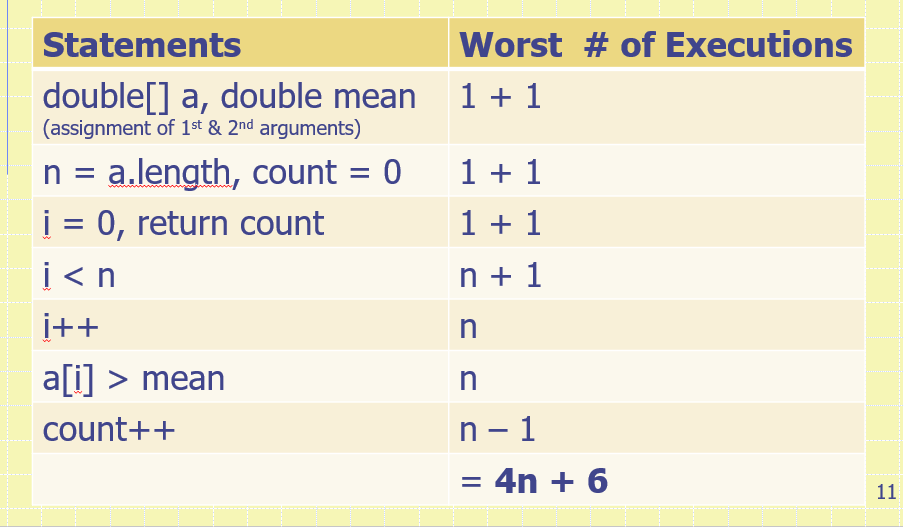
求worstTime(n)



小于的有N-1次，等于的有一次

自增只有在小于的时候进行，其他的同理





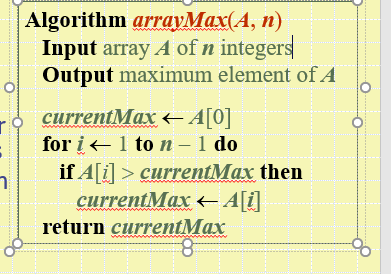
读入几个，进行几次，int算一次，int return各算一次， 其实就是进行几次算几次

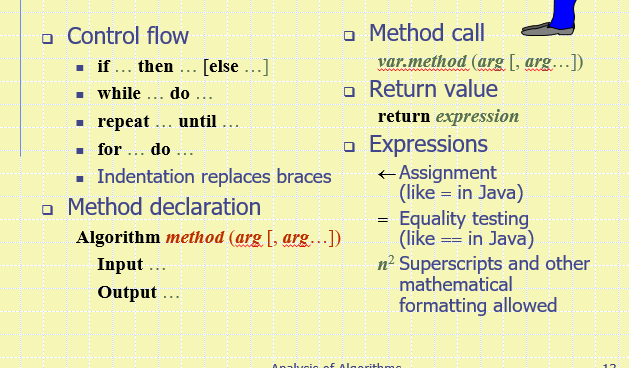
为啥count是n-1，最多有N-1大于mean(平均)

Pseudocode伪代码

高度概括algorithm算法，比英文描述更有结构性

比具体程序信息更少。





Ram模型

一个：CPI

一个潜在的没有边界的存储单元，每一个都能存储任意数量的character

存储单元是排好序的并且进入任何单元需要一个单位的时间

T（n）是具体运算时间，

A是最快基本操作时间，b是最慢基本操作时间，ab取决于电脑运行环境

改变硬件软件会改变ab,但不会改变T(n)增长速率

注意growth rate不会被constant factor所影响，或低级的term影响

比如说r^3+r^2,不会被r^2影响

Big-O,我们不用算具体的最坏情况，只用算近似值big-o

Big-O会给算法/function决定一个upper bound上限，换句话说，他决定了这个算法可以得到的最坏情况

我们可以把它写成f(n)is O（g（n））因为我们不在乎系数



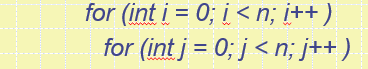


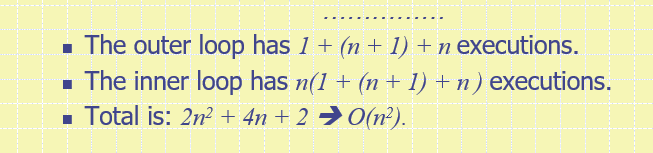
 

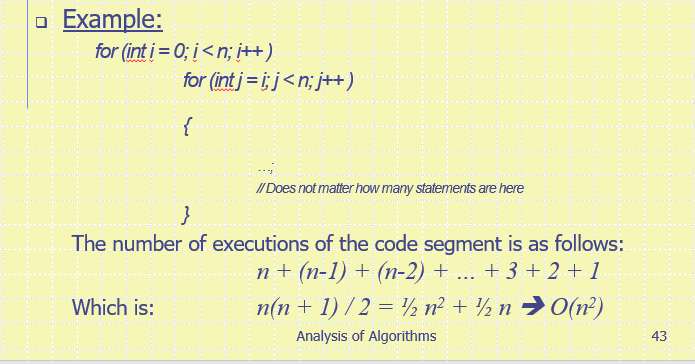
对于对数log，我们可以忽略他的基（最下面那个小的数）

，然后logab又是C，C是被忽略的

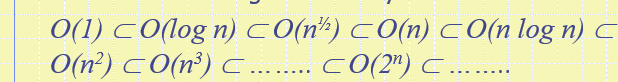
Nested

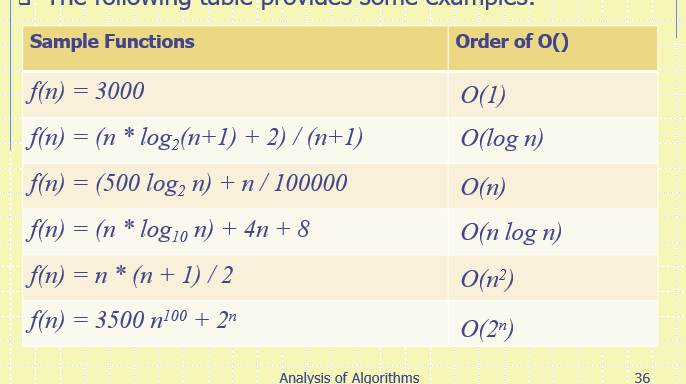






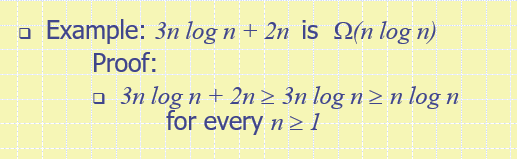
注意在这个例子中,n+n-1+n-2中已经蕴含了对上面的一个循环

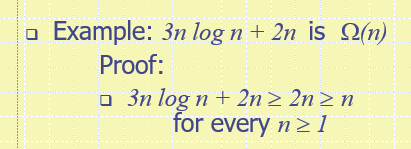




Big-O代表着最坏情况，Big Omega代表着最好情况







注意O可以无穷大，fN如果小于N，必然就小于NLOGN。。。

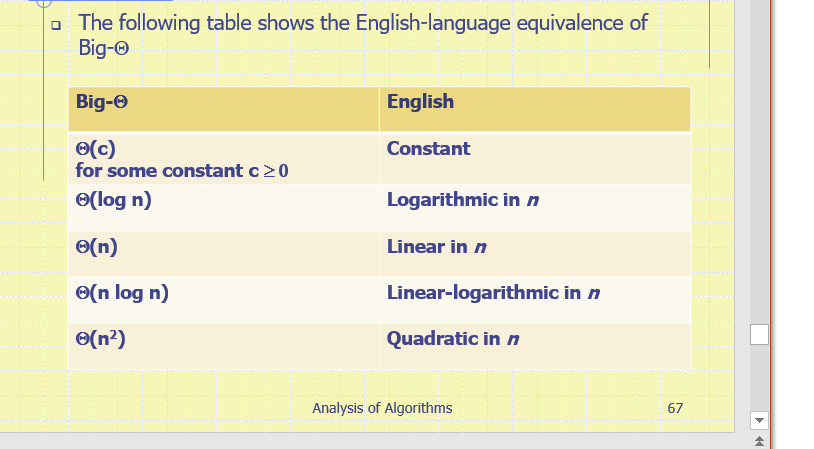
OMEGA可以无穷小，他如果大于N，必然就大于LOOGN。

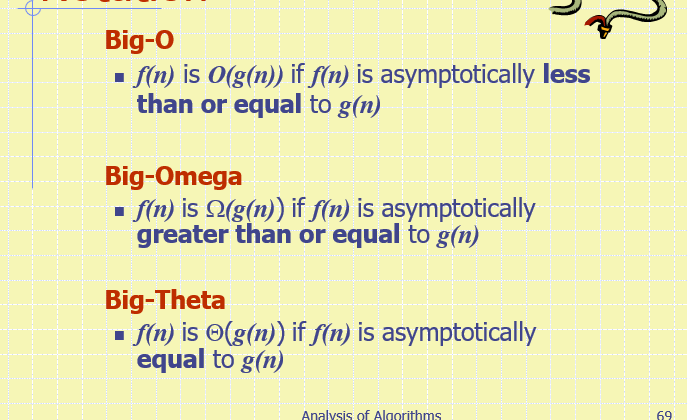
都是TRUE，但是不一定是good好的。不是最优解

只有O和OMEGA交界，得到的big-theta才是最优解



但是OMEGA本身是和O是distinct有区别的





3n^6-15n^8是不存在的，数字越大，时间越小，成了负数，创造了时间机器

答案可以写O（1）或Big（O）is not for tgus akgitgun或doesn’t exist

F(n)=1/n^4 是可以的，O（1）,越变越小没关系，关键是不能成为负数

最优解例子

把一个array从头到尾调换，你可医用temp，但是复杂程度是N^2,不如用swap，BIgO是N

怎么写简略程序