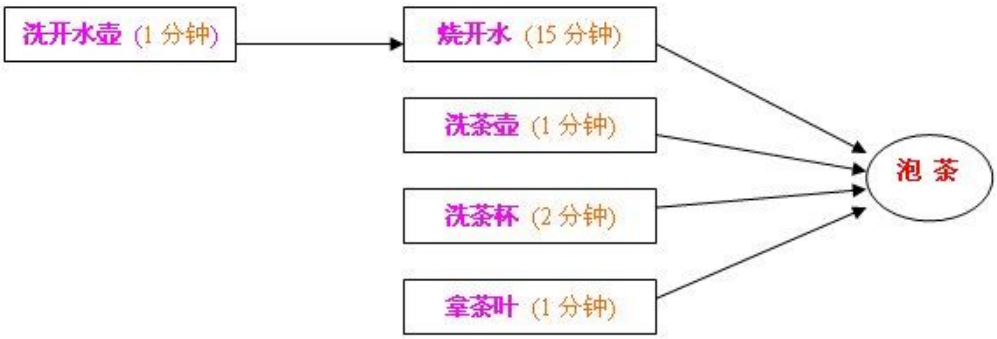
算法（algorithm），在数学和计算机科学中，为一系列具体计算步骤（instructions），常用于计算（calculation）、数据处理（data processing）、和自动推理（automatic reasoning）。作为一个有效方法（effective method），算法包含了一系列定义清晰的指令，并可于有限的时间及空间内用形式语言（formal language）表达出来。算法的指令从一个初始状态（initial state）和一个初始输入（initial input）出发，经过一系列有限连续的状态，最终到达终态（final state）并输出结果。其中不同状态之间的转移可以是决定性的也可以是随机的。（from Wikipedia）

小学语文课学过华罗庚写过的《统筹方法》，至今还记得文中开头非常生动的例子：

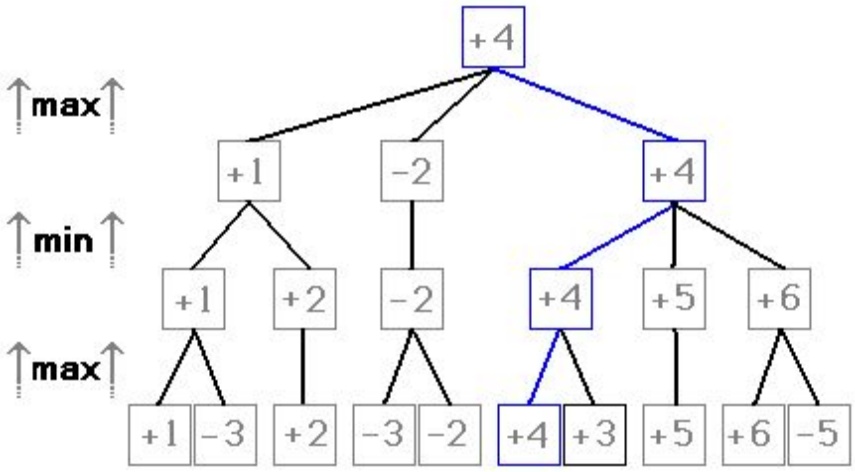
“比如，想泡壶茶喝。当时的情况是：开水没有；水壶要洗，茶壶茶杯要洗；火生了，茶叶也有了。怎么办？
办法甲：洗好水壶，灌上凉水，放在火上；在等待水开的时间里，洗茶壶、洗茶杯、拿茶叶；等水开了，泡茶喝。
办法乙：先做好一些准备工作，洗水壶，洗茶壶茶杯，拿茶叶；一切就绪，灌水烧水；坐待水开了泡茶喝。
办法丙：洗净水壶，灌上凉水，放在火上，坐待水开；水开了之后，急急忙忙找茶叶，洗茶壶茶杯，泡茶喝。
哪一种办法省时间？我们能一眼看出第一种办法好，后两种办法都窝了工。”



如果交给计算机程序解决这个问题，我们就可以把上述的每一项任务都标记上执行时间和等待时间，其中在执行时间里必须专注于该任务而无暇顾及其它任务，在等待时间里则可以去执行其它任务。这样一来，一个优先处理等待时间较长的任务的贪心算法就可以给这个问题最优解。

另一个例子是博弈论里的囚徒困境问题。它讲的是有两个嫌疑犯作案被抓而后各自被单独审讯，警察说，如果两人都抵赖则各判1年，如果一人坦白则坦白者获释而另一人判10年，如果双方坦白则两人均判8年。对于每个嫌疑犯来说，坦白是最优解，因为坦白后可能

的结果分别是0年和8年，而不坦白可能的后果分别是1年和10年。这一决策过程类似算法中的最小-最大准则（minimax theorem），即己方做选择时要考虑到选择之后对方会选择最不利于自己的选项。根据最大-最小准则，己方将首先计算出每个选择可能的后果里最差的情况（即“最小”），之后从这些最差的情况中选择最佳的情况（即“最大”）。



（from Wikipedia）

这样的例子还有很多，比如圣母大学计算机系某教授通过贪心算法找到被劫走的车。由此可见算法不光被应用在计算机领域，它在生活中用处也有很多。当遇到问题的时候，我们通常先想出几种方法，再从中选出一种目前最优的。如何从这些方法中选择需要一些分析，对计算机算法而言就是算法分析。

不同的机器运算速度不一样（即使慢的算法放到高级硬件上也运行得很快），结构不一样（随机存取（random access）与否可能会影响到一些指令的速度），存储效率不一样（缓存（cache）和硬盘（disk）），浮点数精度（floating point precision）不一样，指令不一样（加法和乘法的运算时间差异），参照物不一样（以输入元素（item）的数量为单位还是以输入字节（byte）的数量为单位）。在这么多变量中我们要如何进行算法分析呢？

因此我们需要固定一些对于算法分析较为次要的变量（比如硬件问题交给硬件工程师考虑）。在这篇连载中将统一使用如下假设：

1. 执行算法的计算机是通用计算机并支持随机存取；
2. 没有明确说明是并行计算的情况下默认所有指令都依次进行，即没有多个指令同时进行的情况；

3. 任何算数运算（arithmetic operation，如加减乘除），数据移动（data movement，如读取，存储，复制），控制语句（control，如条件语句（if statement），函数调用（function call），返回（return））均在常数时间内完成；
4. 不考虑浮点数精度带来的影响；
5. 数据单位（word of data）大小为常数，即一个大小为n的输入所占用的字节数为c lgn，其中c为常数；
6. 不考虑存储结构（memory hierarchy，如缓存优化）带来的影响。

基于这些假设我们可以有效排除一些变量，但仍然有很多问题值得讨论，例如复杂度类（complexity class），可计算性（computability），概率分析（probability analysis，包括随机性（randomization）分析），数据结构（data structure），并行计算（parallel computation），等等，同样希望在作者懒癌发作之前写完 :D 下面先介绍一下时间复杂度分析的基本概念。

以从小到大排序的插入排序为例，即依次遍历数组中第二个数到最后一个数，当该数字小于前面的数字时则和前面的数字交换，直到前面的数字小于等于该数字或该数字已位于第一个，伪代码如下（输入待排序的数组A，假设第一个数字序号（index）为1）：

```
i ← 1
while i < length(A)
    x ← A[i]
    j ← i - 1
    while j >= 0 and A[j] > x
        A[j+1] ← A[j]
        j ← j - 1
    end while
    A[j+1] ← x[4]
    i ← i + 1
end while
```

（from Wikipedia）

对于插入排序来说最好的情况是输入的数组已经是从小到大排序好的。此时内层while循环一经判断立刻跳出，所用时间为常数。这样一来外层while循环内部所有语句均可在常数时间内完成，代码整体花费的时间为输入大小（数组中元素个数）n的常数倍。

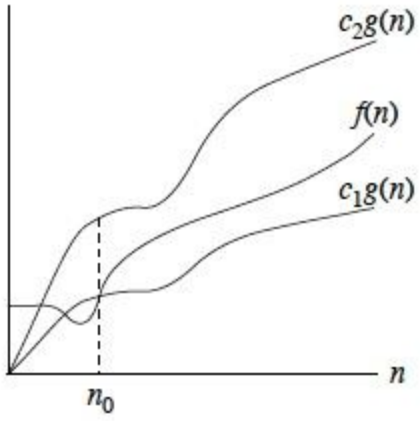
最差的情况是输入的数据是从大到小排序的。此时对于外层while循环的每个i值，内层while循环都要判断i次（包括A[i]是否处于数组第一位的判断），代码整体花费的时间为2

+ 3 + … + n = n(n + 1) / 2 - 1的常数倍，取其中最高次项并忽略常数项约为n^2的常数倍。

而平均情况在这里较难估计，取其中最高次项并忽略常数项后所需时间仍然约为n^2的常数倍，之后会在概率分析讲到。由此可见即使是通一种算法，在不同情况下所需都是显著不同的。当我们使用一种算法时，我们通常希望这个算法在最差的情况下不至于太差，因此当我们分析时间复杂度时通常都使用最差的情况，偶尔会使用平均情况（例如随机化算法），极少使用最佳情况（作为一个非洲人并不想让算法都变成欧皇检测器 :) ）。

可以注意到之前描述最差情况和平均情况的时候用到了“取最高次项并忽略常数项”，这是用于避免繁琐的精确计算，当输入大小n变得非常大的时候，低次项和常数带来的影响将变得不那么重要，由此引出了渐进表示法（asymptotic notation）：

1. Θ （Big Theta notation）
   对于两个函数（function）f(n)和g(n)，当存在正常数（positive constant）c1, c2, n0，使得对于所有n ≥ n0，有0 ≤ c1 * g(n) ≤ f(n) ≤ c2 * g(n)，如下图所示。

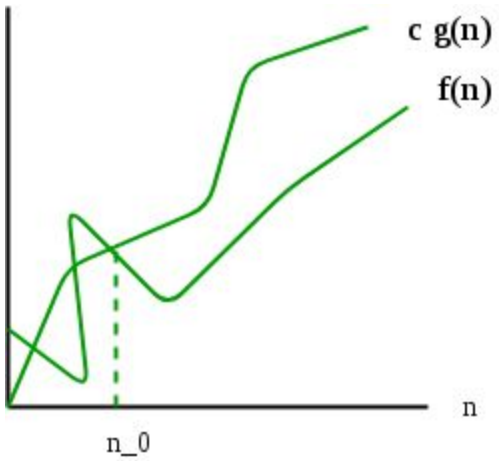


   上图中在c2 * g(n)和c1 * g(n)之间有着一片区域，它不一定只包括f(n)，因此Θ(g(n))可以看作一些函数的集合，这个集合包括f(n)，写作 f(n) ∈ Θ(g(n))，也可

以说g(n)是f(n)的渐进紧密边界（asymtotically tight bound）。为了方便起见，定义要求f(n)为非负函数。

例子：lgn = Θ(logn)

2. O （Big O notation）
   对于两个函数（function）f(n)和g(n)，当存在正常数（positive constant）c和n0，使得对于所有n ≥ n0，有0 ≤ f(n) ≤ c * g(n)，如下图所示。

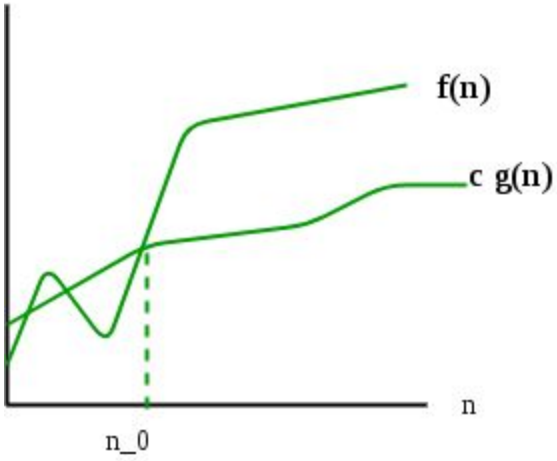


f(n) = O(g(n))

（from GeeksforGeeks）

例子：n = O(n^2)

3. Ω （BIg Omega notation）
   对于两个函数（function）f(n)和g(n)，当存在正常数（positive constant）c和n0，使得对于所有n ≥ n0，有0 ≤ c * g(n) ≤ f(n)，如下图所示。



f(n) = Omega(g(n))

（from GeeksforGeeks）

例子：n^2 = Ω(n)

4. o （Little O notation）
   o和O差不多，但不包括 f(n) = c * g(n)的情况，即对于两个函数（function）f(n)和g(n)，当存在正常数（positive constant）c和n0，使得对于所有n ≥ n0，有0 ≤ f(n) < c * g(n)。

5. ω （Little Omega notation）
   ω和Ω差不多，但不包括 f(n) = c * g(n)的情况，即对于两个函数（function）f(n)和g(n)，当存在正常数（positive constant）c和n0，使得对于所有n ≥ n0，有0 ≤ c * g(n) < f(n)。

以上是五种常用的渐进表示法，其中O尤为常用（毕竟要经常考虑最差的情况）。它们本身存在一些性质，比如一旦f(n) = O(g(n))和f(n) = Ω(g(n))同时成立，则f(n) = Θ(g(n))。这些性质中，

- Θ, O, Ω, o, ω都满足传递性（transitivity，即已知aRb和bRc可以推出aRc，R代表一种关系（relation）），例如由f(n) = Θ(g(n)), g(n) = Θ(h(n))得到f(n) = Θ(h(n))；
- Θ, O, Ω满足自反性（reflexivity，即对于任意变量a，aRa都成立），例如f(n) = Θ(f(n))；
- Θ满足对称性（symmetry，即aRb成立当且仅当bRa），例如f(n) = Θ(g(n))当且仅当g(n) = Θ(f(n))；
- O和Ω，o和ω满足转置对称性（transpose symmetry），即f(n) = O(g(n))当且仅当g(n) = Ω(f(n))，f(n) = o(g(n))当且仅当g(n) = ω(f(n))。

相关材料：

本章很多内容来自算法导论（Introduction to Algorithms），购买链接：
https://www.amazon.com/Introduction-Algorithms-3rd-MIT-Press/dp/0262033844（此条零元，替算法圣经传福音怎么能赚钱 :)）
https://en.wikipedia.org/wiki/Algorithm