算法的五大特性

1. 输入: 算法具有0个或多个输入

2. 输出: 算法至少有1个或多个输出

3. 有穷性: 算法在有限的步骤之后会自动结束而不会无限循环，并且每一个步骤可以在可接受的时间内完成

4. 确定性：算法中的每一步都有确定的含义，不会出现二义性

5. 可行性：算法的每一步都是可行的，也就是说每一步都能够执行有限的次数完成

尝试：

```python
# 第一次
import time

start_time = time.time()

# 注意是三重循环
for a in range(0, 1001):
    for b in range(0, 1001):
        for c in range(0, 1001):
            if a**2 + b**2 == c**2 and a+b+c == 1000:
                print("a, b, c: %d, %d, %d" % (a, b, c))

end_time = time.time()
```

```
print("elapsed: %f" % (end_time - start_time))
print("complete!")
# 需要200多s
# 当前时间复杂度：
    T = 外层循环1000次*二层循环1000次*三层循环1000次 *2
    # 改为a+b+c =2000
    T = 外层循环2000次*二层循环2000次*三层循环2000次 *2
    假设 a+b+c = N
    T = N*N*N*2

    T（N）=N^3*2
    T（N）=N^3*10  #认为三者算是同一个数量级，趋势都差不多

    g(n) = T(N)*k   #g(n)可以说是T(N)的渐进函数，
    g（n） = N^3
    # O(g(n)) 就可以说是时间复杂度的大O表示法，
    # 为算法A的渐近时间复杂度，简称时间复杂度，记为T(n)


# 第二次
import time

start_time = time.time()

# 注意是两重循环
for a in range(0, 1001):
    for b in range(0, 1001-a): #总数不超过1000
        c = 1000 - a - b
        if a**2 + b**2 == c**2:
            print("a, b, c: %d, %d, %d" % (a, b, c))

end_time = time.time()
print("elapsed: %f" % (end_time - start_time))
print("complete!")

需要0.18s
```

```
50
51 # 时间复杂度计算
52 T(n) = n * n * (1 + max(1,0))
53  = n^2*2
54  =O(n^2) # 大 O表示法
55
```

算法效率衡量

同样的代码，在每台机器的总时间还是会不同的，但是执行的基本运算数量大体相同，因此可以作为衡量标准

时间复杂度与“大O记法”

见上面习题分析过程：

假设存在函数g，使得算法A处理规模为n的问题示例所用时间为T(n)=O(g(n))，则称O(g(n))为算法A的渐近时间复杂度，简称时间复杂度，记为T(n)

如何理解：

可以认为3n2和100n2属于同一个量级，如果两个算法处理同样规模实例的代价分别为这两个函数，就认为**它们的效率“差不多”，都为n2级，就是相当于忽略常数**

时间复杂度的几条基本计算规则

1. 基本操作，即只有常数项，认为其时间复杂度为O(1)
2. **顺序**结构，时间复杂度按**加法**进行计算
3. **循环**结构，时间复杂度按**乘法**进行计算
4. **分支**结构，时间复杂度**取最大值**
5. 判断一个算法的效率时，往往**只需要关注操作数量的最高次项**，其它次要项和常数项可以忽略
6. 在没有特殊说明时，我们所分析的算法的时间复杂度都是指**最坏时间复杂度**

空间复杂度

类似于时间复杂度的讨论，一个算法的**空间复杂度S(n)**定义为该算法所耗费的存储空间，它也是问题规模n的函数。

渐近空间复杂度也常常简称为**空间复杂度。**

空间复杂度(SpaceComplexity)是对一个算法在运行过程中临时占用存储空间大小的量度。

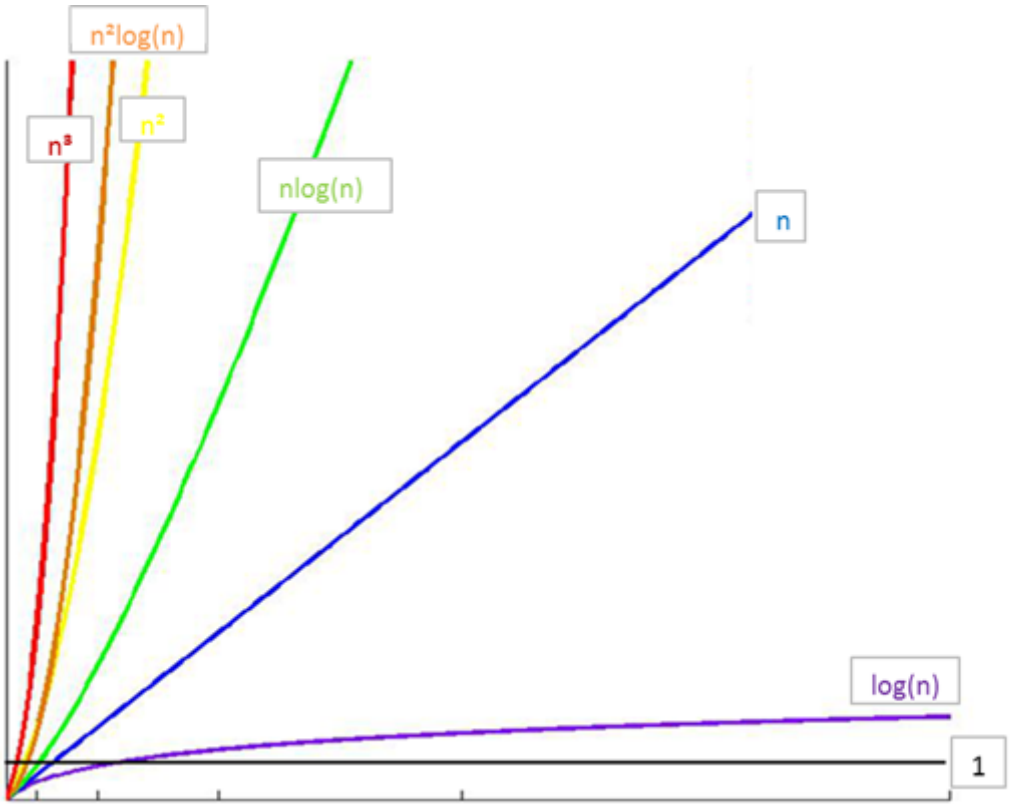
算法的时间复杂度和空间复杂度合称为算法的复杂度。

常见时间复杂度

执行次数函数举例	阶	非正式术语
12	O(1)	常数阶
2n+3	O(n)	线性阶
3n2+2n+1	O(n2)	平方阶
5log2n+20	O(logn)	对数阶
2n+3nlog2n+19	O(nlogn)	nlogn阶
6n3+2n2+3n+4	O(n3)	立方阶
2n	O(2n)	指数阶

注意，经常将log2n（以2为底的对数）简写成logn

常见时间复杂度之间的关系



所消耗的时间从小到大

O(1) < O(logn) < O(n) < O(nlogn) < O(n2) < O(n3) < O(2n) < O(n!) < O(nn)

list内置操作的时间复杂度

Operation	Big-O Efficiency
indexx[]	$O(1)$
index assignment	$O(1)$
append	$O(1)$
pop()	$O(1)$
pop(i)	$O(n)$
insert(i,item)	$O(n)$
del operator	$O(n)$
iteration	$O(n)$
contains (in)	$O(n)$
get slice [x:y]	$O(k)$
del slice	$O(n)$
set slice	$O(n+k)$
reverse	$O(n)$
concatenate	$O(k)$
sort	$O(n \log n)$
multiply	$O(nk)$

Table 2.2: Big-O Efficiency of Python List Operators

dict内置操作的时间复杂度

Operation	Big-O Efficiency
copy	$O(n)$
get item	$O(1)$
set item	$O(1)$
delete item	$O(1)$
contains (in)	$O(1)$
iteration	$O(n)$

Table 2.3: Big-O Efficiency of Python Dictionary Operations