数据结构和算法

作者：小甲鱼

让编程改变世界

Change the world by program

函数调用的时间复杂度分析

- 如果我们把问题再实际化一点，大家是否能自己正确的分析出来呢？
- 我们来看下边这个例子：

```
int i, j;
for(i=0; i < n; i++) {
    function(i);
}
void function(int count) {
    printf("%d", count);
}
```

函数调用的时间复杂度分析

- 函数体是打印这个参数，这很好理解。function函数的时间复杂度是O(1)，所以整体的时间复杂度就是循环的次数O(n)。
- 假如function是下面这样，又该如何呢：

```
void function(int count) {
    int j;
    for(j=count; j < n; j++) {
        printf("%d", j);
    }
}
```

函数调用的时间复杂度分析

- 事实上，这和之前我们讲解平方阶的时候举的第二个例子一样：function内部的循环次数随count的增加(接近n)而减少，所以根据游戏攻略算法的时间复杂度为O(n^2)。

- 接着使出杀手锏，给鱼油们一个挑战的机会！

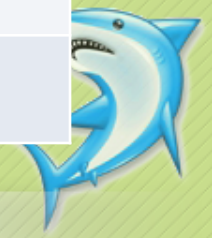
- 尝试自己分析以下程序的时间复杂度：

函数调用的时间复杂度分析

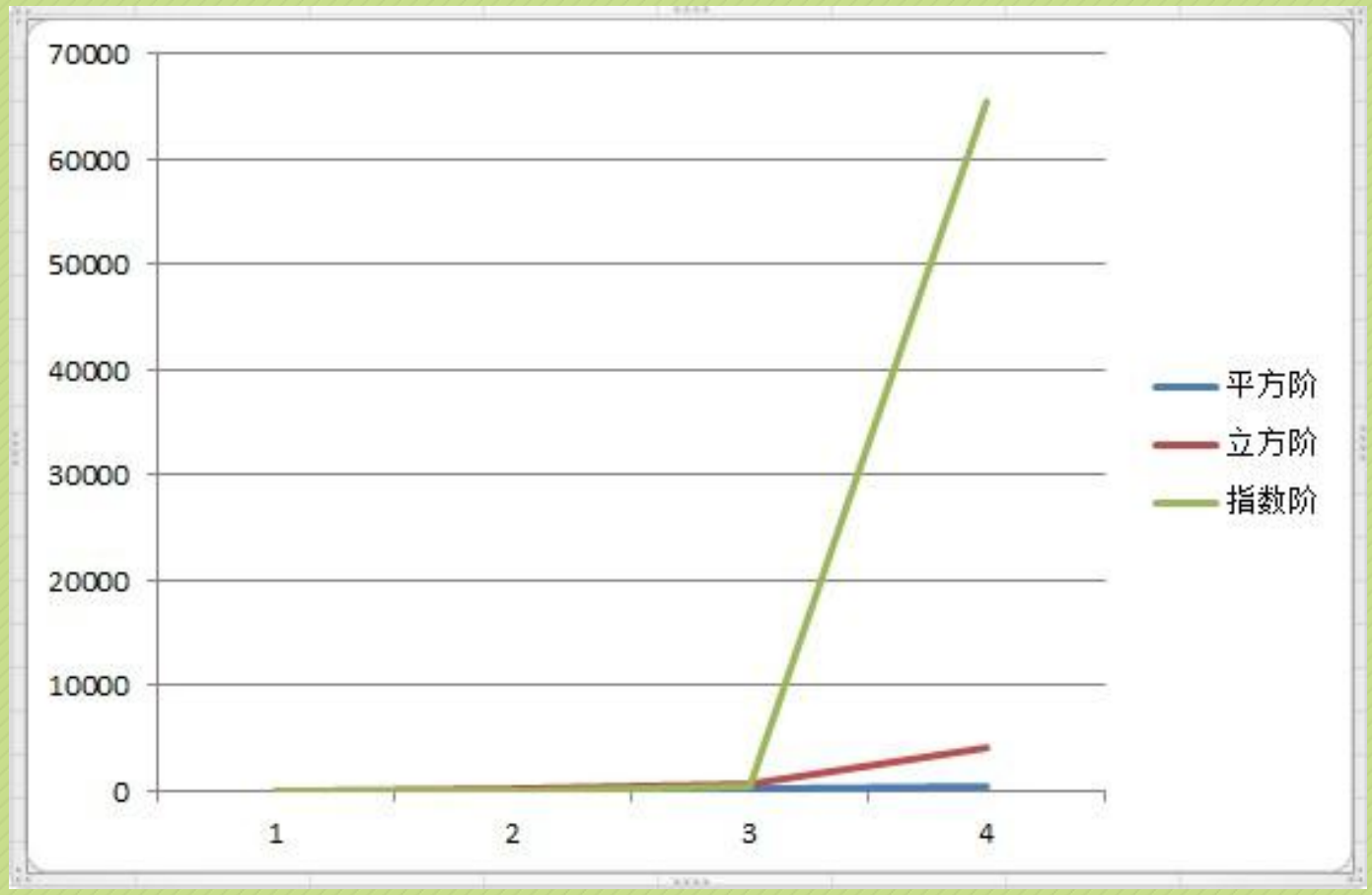
```
n++;
function(n);
for(i=0; i < n; i++) {
    function(i);
}
for(i=0; i < n; i++) {
    for(j=i; j < n; j++) {
        printf("%d", j);
    }
}
```

常见的时间复杂度

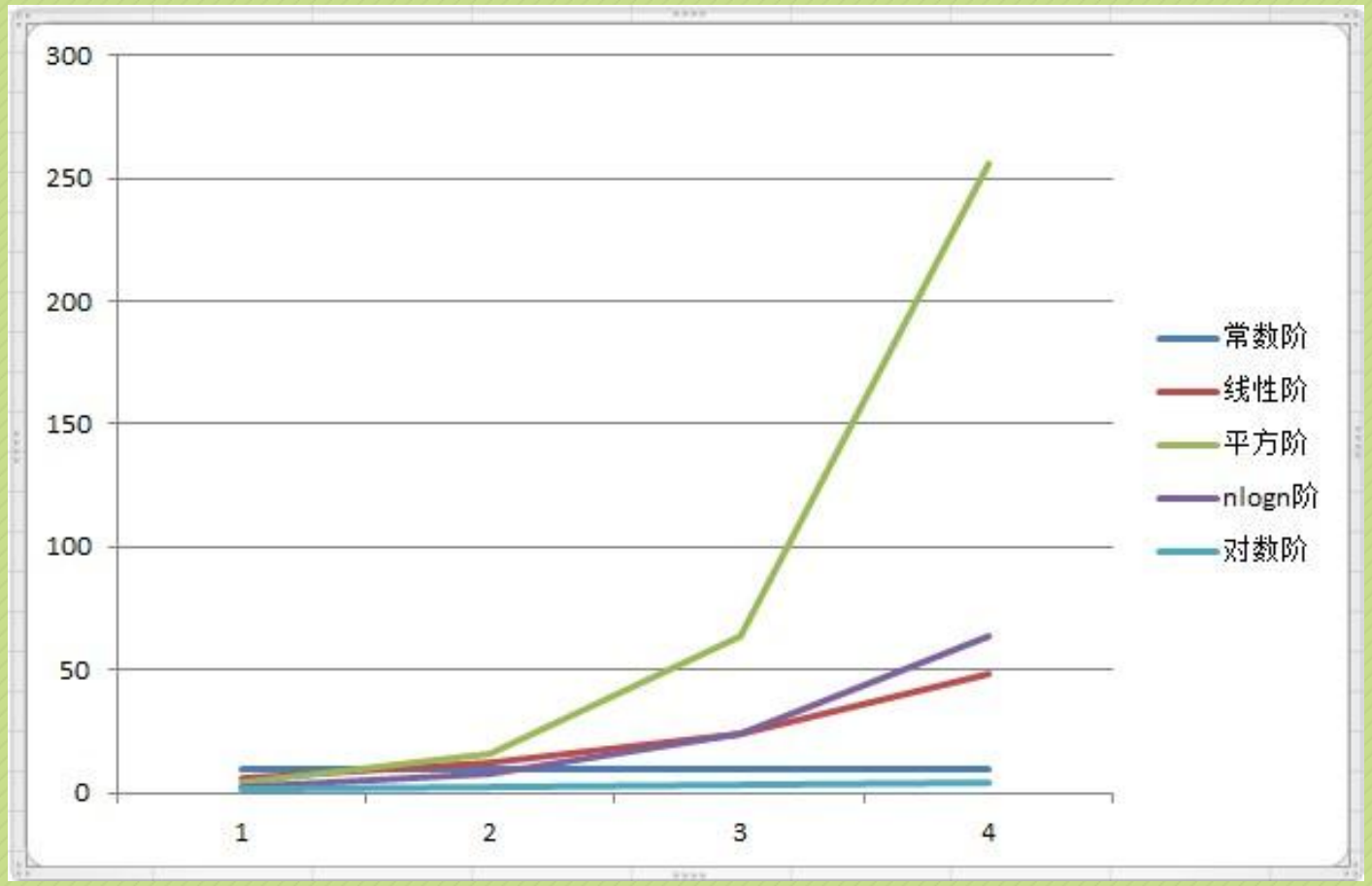
例子	时间复杂度	装逼术语
5201314	O(1)	常数阶
3n+4	O(n)	线性阶
3n^2+4n+5	O(n^2)	平方阶
3log(2)n+4	O(logn)	对数阶
2n+3nlog(2)n+14	O(nlogn)	nlogn阶
n^3+2n^2+4n+6	O(n^3)	立方阶
2^n	O(2^n)	指数阶



有图有真相



有图有真相

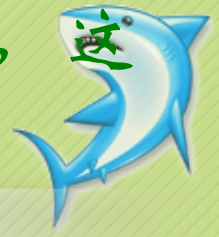


常见的时间复杂度

- 常用的时间复杂度所耗费的时间从小到大依次是：O(1) < O(logn) < (n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)
- O(1),O(logn),O(n),O(n^2)我们前边已经给大家举例谈过了，至于O(nlogn)我们将会在今后的课程中介绍。
- 而像O(n^3)之后的这些，由于n值的增大都会使得结果大得难以想象，我们没必要去讨论它们。

最坏情况与平均情况

- 从心理学角度讲，每个人对将来要发生的事情都会有一个预期。譬如看半杯水，有人会说：哇哦，还有半杯哦！但有人就会失望的说：天，只有半杯了。
- 一般人常出于一种对未来失败的担忧，而在预期的时候趋向做最坏打算。这样，即使最糟糕的结果出现，当事人也有了心理准备，比较容易接受结果，假如结局并未出现最坏的状况，这也会使人更加快乐，瞧，事情发展的还不错嘛！嗯，这是典型的自慰手法。



最坏情况与平均情况

- 算法的分析也是类似，我们查找一个有n个随机数字数组中的某个数字，最好的情况是第一个数字就是，那么算法的时间复杂度为O(1)，但也有可能这个数字就在最后一个位置，那么时间复杂度为O(n)。
- 平均运行时间是期望的运行时间。
- 最坏运行时间是一种保证。在应用中，这是一种最重要的需求，通常除非特别指定，我们提到的运行时间都是最坏情况的运行时间。

算法的空间复杂度

- 我们在写代码时，完全可以用空间来换去时间。
- 举个例子说，要判断某年是不是闰年，你可能会花一点心思来写一个算法，每给一个年份，就可以通过这个算法计算得到是否闰年的结果。
- 另外一种方法是，事先建立一个有2050个元素的数组，然后把所有的年份按下标的数字对应，如果是闰年，则此数组元素的值是1，如果不是元素的值则为0。这样，所谓的判断某一年是否为闰年就变成了查找这个数组某一个元素的值的问题。

算法的空间复杂度

- 第一种方法相比起第二种来说很明显非常节省空间，但每一次查询都需要经过一系列的计算才能知道是否为闰年。第二种方法虽然需要在内存里存储2050个元素的数组，但是每次查询只需要一次索引判断即可。

- 这就是通过一笔空间上的开销来换取计算时间开销的小技巧。到底哪一种方法好？其实还是要看你用在什么地方。

算法的空间复杂度

- 算法的空间复杂度通过计算算法所需的存储空间实现，算法的空间复杂度的计算公式记作：S(n)=O(f(n))，其中，n为问题的规模，f(n)为语句关于n所占存储空间的函数。
- 通常，我们都是用“时间复杂度”来指运行时间的需求，是用“空间复杂度”指空间需求。
- 当直接要让我们求“复杂度”时，通常指的是时间复杂度。
- 显然对时间复杂度的追求更是属于算法的潮流！