数据结构和算法

作者：小甲鱼

让编程改变世界

Change the world by program

算法效率的度量方法

- 上一讲中我们提到设计算法要尽量的提高效率，这里效率高一般指的是算法的执行时间。
- 那么我们如何来度量一个算法的执行时间呢？
- 所谓“是骡子是马拉出来遛遛”，比较容易想到的方法就是我们把算法跑若干次，然后拿个“计时器”在旁边计时。
- 这种事后统计方法看上去的确不错，并且也并非真的要你拿个计算器在那里计算，因为计算机都有计时功能。

算法效率的度量方法

- 事后统计方法：这种方法主要是通过设计好的测试程序和数据，利用计算机计时器对不同算法编制的程序的运行时间进行比较，从而确定算法效率的高低。
- 但这种方法显然是有很大缺陷的：
  - 必须依据算法事先编制好测试程序，通常需要花费大量时间和精力，完了发觉测试的是糟糕的算法，那不是功亏一篑？赔了娘子又折兵？
  - 不同测试环境差别不是一般的大！

算法效率的度量方法

- 我们把刚刚的估算方法称为事后诸葛亮。我们的计算机前辈们也不一定知道诸葛亮是谁，为了对算法的评判更为科学和便捷，他们研究出事前分析估算的方法。
- 事前分析估算方法：在计算机程序编写前，依据统计方法对算法进行估算。
- 经过总结，我们发现一个高级语言编写的程序在计算机上运行时所消耗的时间取决于下列因素：

算法效率的度量方法

- 1. 算法采用的策略，方案
- 2. 编译产生的代码质量
- 3. 问题的输入规模
- 4. 机器执行指令的速度

• 由此可见，抛开这些与计算机硬件、软件有关的因素，一个程序的运行时间依赖于算法的好坏和问题的输入规模。（所谓的问题输入规模是指输入量的多少）

• 我们搬回搞死先生的那个算法来跟大家谈谈：

算法效率的度量方法

- 第一种算法：

```
int i, sum = 0, n = 100;   // 执行1次
for( i=1; i <= n; i++ )    // 执行了n+1次
{
    sum = sum + i;         // 执行n次
}
```

- 第二种算法：

```
int sum = 0, n = 100;      // 执行1次
sum = (1+n)*n/2;           // 执行1次
```

算法效率的度量方法

- 第一种算法执行了1+(n+1)+n=2n+2次。
- 第二种算法，是1+1=2次
- 如果我们把循环看做一个整体，忽略头尾判断的开销，那么这两个算法其实就是n和1的差距。
- 有些喜欢跟真理死磕的朋友可能对小甲鱼这观点意见不是一般的大！
- 因为循环判断在算法1里边执行了n+1次，看起来是个不小的数量，凭什么说忽略就能忽略？
- 淡定，请接着继续看延伸的例子：

算法效率的度量方法

```
int i, j, x=0, sum=0, n=100;
for( i=1; i <= n; i++ )
{
    for( j=1; j <= n; j++ )
    {
        x++;
        sum = sum + x;
    }
}
```

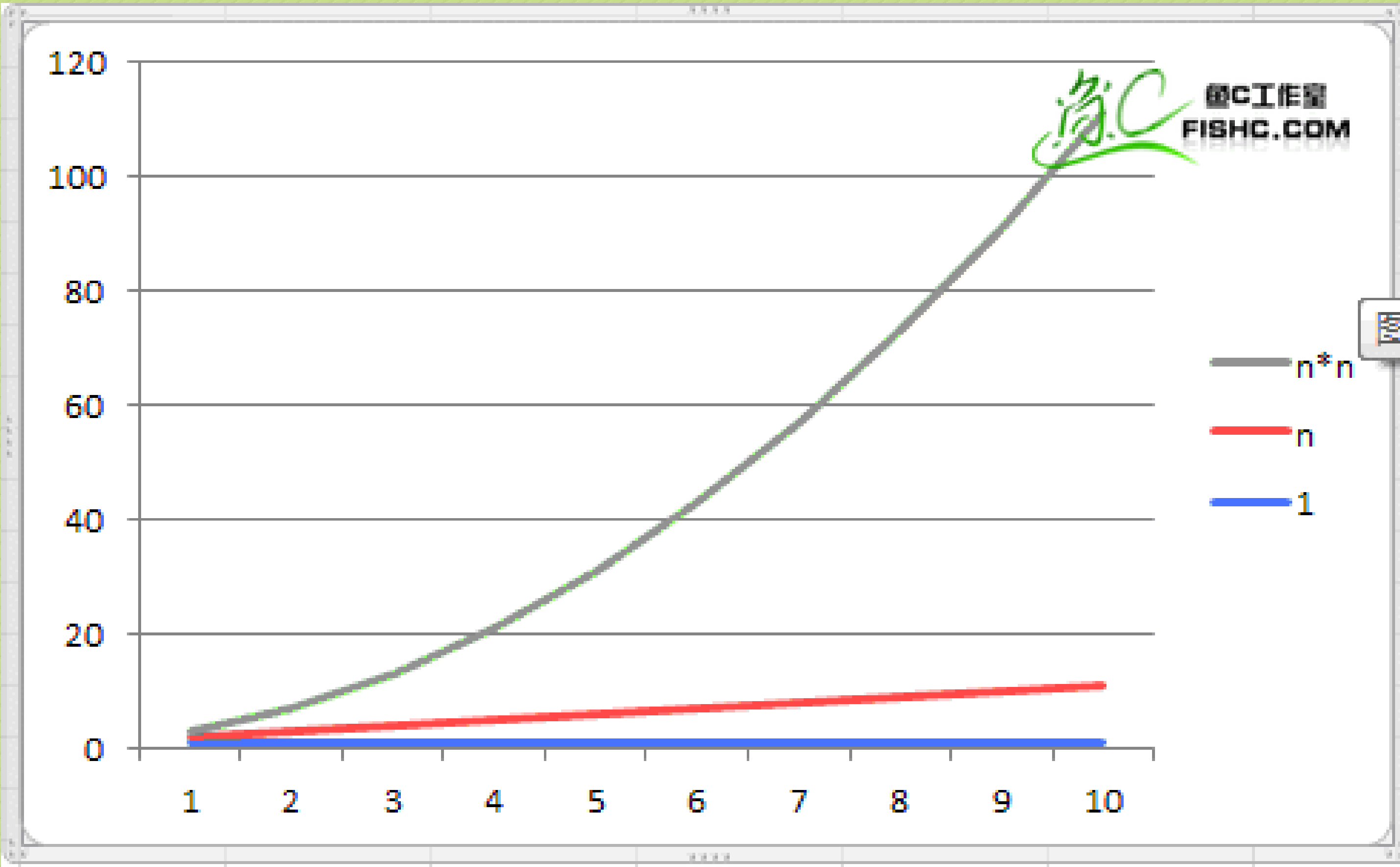
算法效率的度量方法

- 这个例子中，循环条件i从1到100，每次都要让j循环100次，如果非常较真的研究总共精确执行次数，那是非常累的。
- 另一方面，我们研究算法的复杂度，侧重的是研究算法随着输入规模扩大增长量的一个抽象，而不是精确地定位需要执行多少次，因为如果这样的话，我们就又得考虑回编译器优化等问题，然后，然后就永远也没有然后了！
- 所以，对于刚才例子的算法，我们可以果断判定需要执行100^2次。

算法效率的度量方法

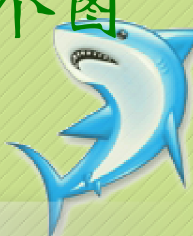
- 我们不关心编写程序所用的语言是什么，也不关心这些程序将跑在什么样的计算机上，我们只关心它所实现的算法。
- 这样，不计那些循环索引的递增和循环终止条件、变量声明、打印结果等操作。最终，在分析程序的运行时间时，最重要的是把程序看成是独立于程序设计语言的算法或一系列步骤。
- 我们在分析一个算法的运行时间时，重要的是把基本操作的数量和输入模式关联起来。

算法效率的度量方法



函数的渐近增长

- 小甲鱼给大家做一个测试：判断以下两个算法A和B哪个更好？
- 假设两个算法的输入规模都是n，算法A要做2n+3次操作，你可以这么理解：先执行n次的循环，执行完成后再有一个n次的循环，最后有3次运算。
- 算法B要做3n+1次操作，理解同上，你觉得它们哪一个更快些呢？
- 在给大家解答问题之前，小甲鱼先给大家做个图表参考：

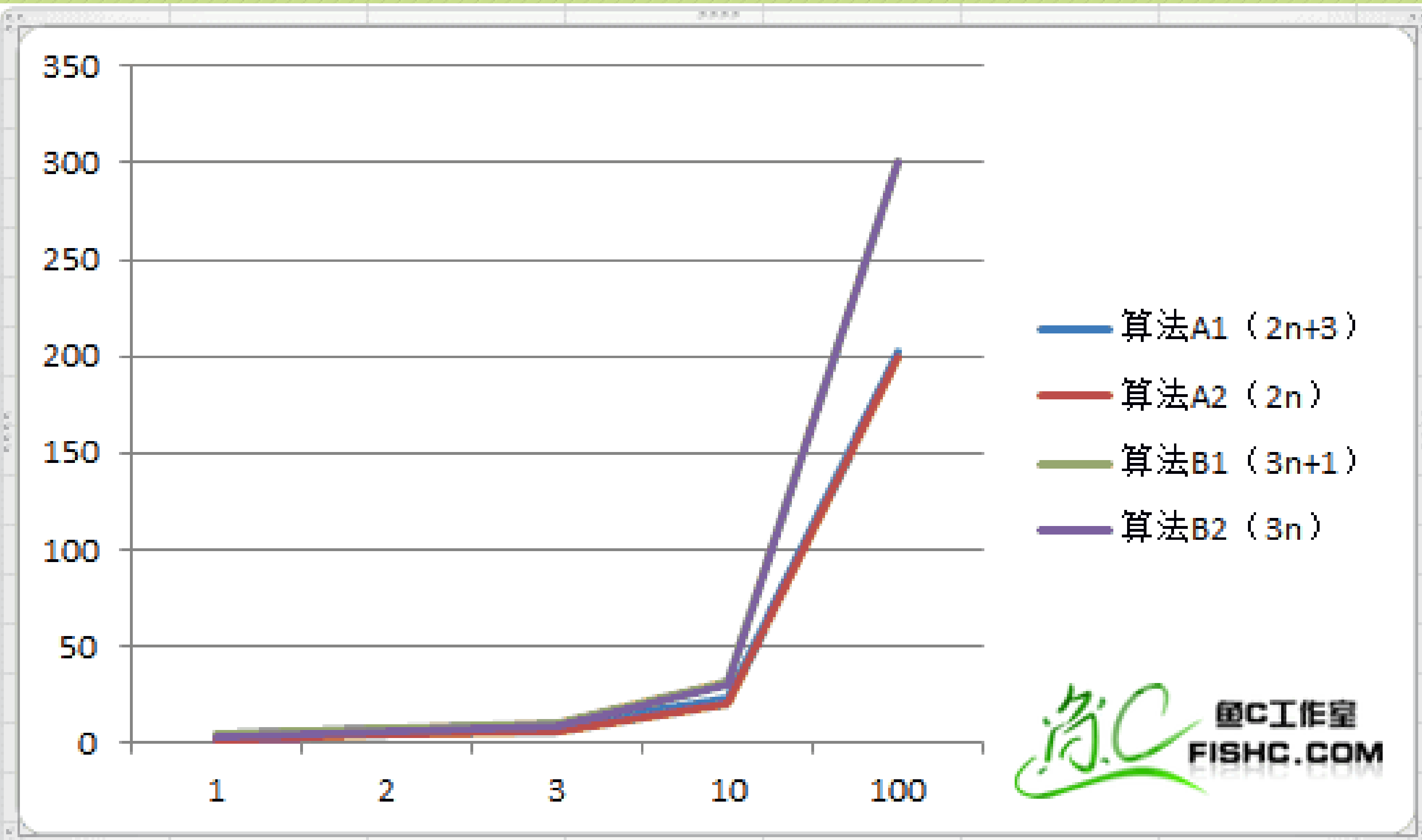


函数的渐近增长

规模	算法A1（2n+3）	算法A2（2n）	算法B1（3n+1）	算法B2（3n）
n=1	5	2	4	3
n=2	7	4	7	6
n=3	9	6	10	9
n=10	23	20	31	30
n=100	203	200	301	300

- 当n=1时，算法A1效率不如算法B1，当n=2时，两者效率相同；当n>2时，算法A1就开始优于算法B1了，随着n的继续增加，算法A1比算法B1逐步拉大差距。所以总体上算法A1比算法B1优秀。

函数的渐进增长


350
300
250
200
150
100
50
0
1
2
3
10
100
算法A1（2n+3）
算法A2（2n）
算法B1（3n+1）
算法B2（3n）
鱼C工作室
FISHC.COM

函数的渐近增长

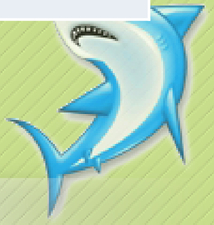
- 函数的渐近增长：给定两个函数f(n)和g(n)，如果存在一个整数N，使得对于所有的n>N，f(n)总是比g(n)大，那么，我们说f(n)的增长渐近快于g(n)。
- 从刚才的对比中我们还发现，随着n的增大，后面的+3和+1其实是不影响最终的算法变化曲线的。
- 例如算法A2，B2，在图中他们压根儿被覆盖了。所以，我们可以忽略这些加法常数。
- 后边我们给大家举多几个例子，会更明显。

函数的渐近增长

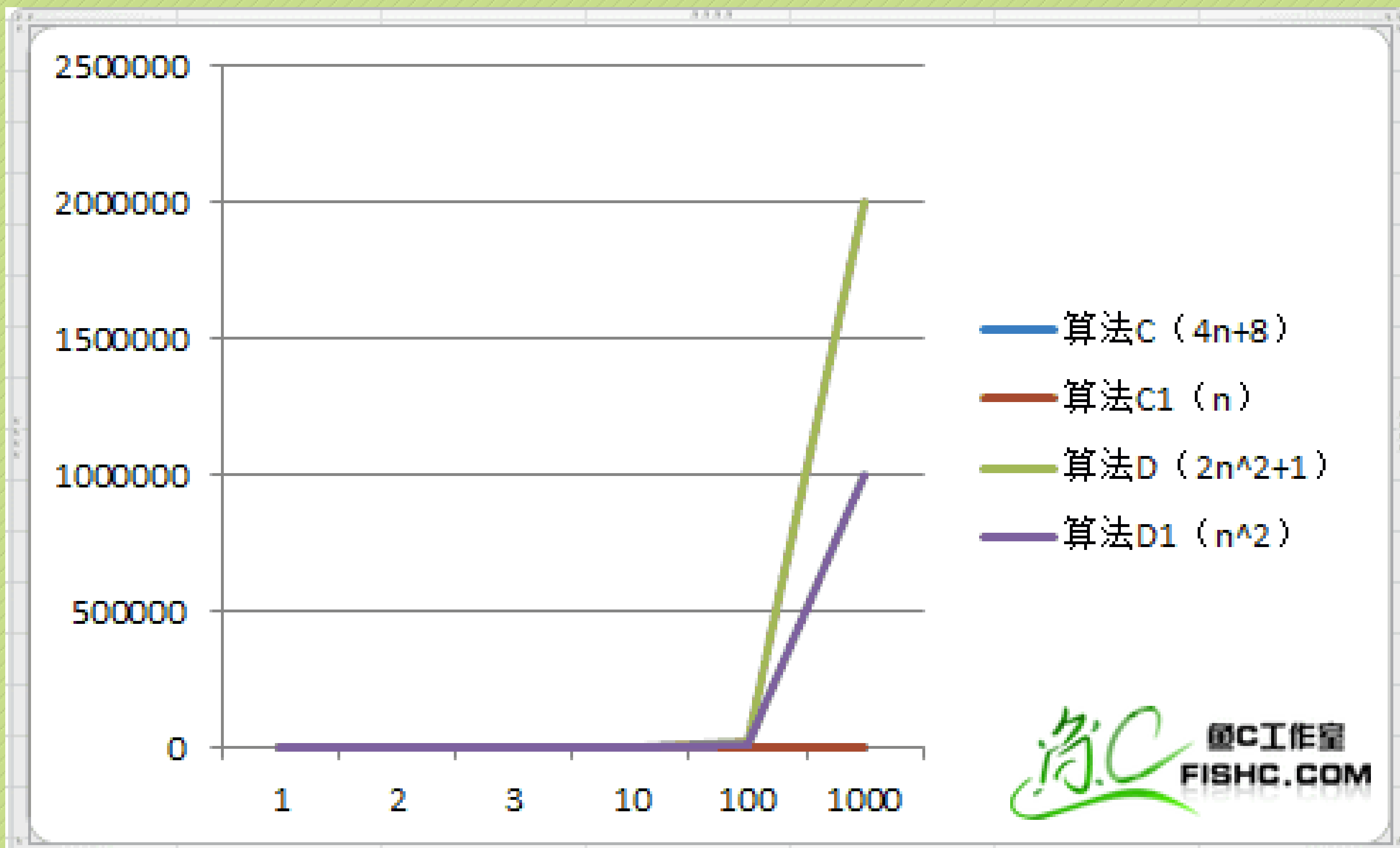
- 第二个测试，算法C是4n+8，算法D是2n^2+1。

次数	算法C1（4n+8）	算法C2（n）	算法D1（2n^2+1）	算法D2（n^2）
n=1	12	1	3	1
n=2	16	2	9	4
n=3	20	3	19	9
n=10	48	10	201	100
n=100	408	100	20001	10000
n=1000	4008	1000	2000001	1000000

- 再来看一下线性图。



函数的渐近增长


2500000
2000000
1500000
1000000
500000
0
1
2
3
10
100
1000
算法C（4n+8）
算法C1（n）
算法D（2n^2+1）
算法D1（n^2）
鱼C工作室
FISHC.COM

函数的渐近增长

- 我们观察发现，哪怕去掉与n相乘的常数，两者的结果还是没有改变，算法C2的次数随着n的增长，还是远小于算法D2。
- 也就是说，与最高次项相乘的常数并不重要，也可以忽略。

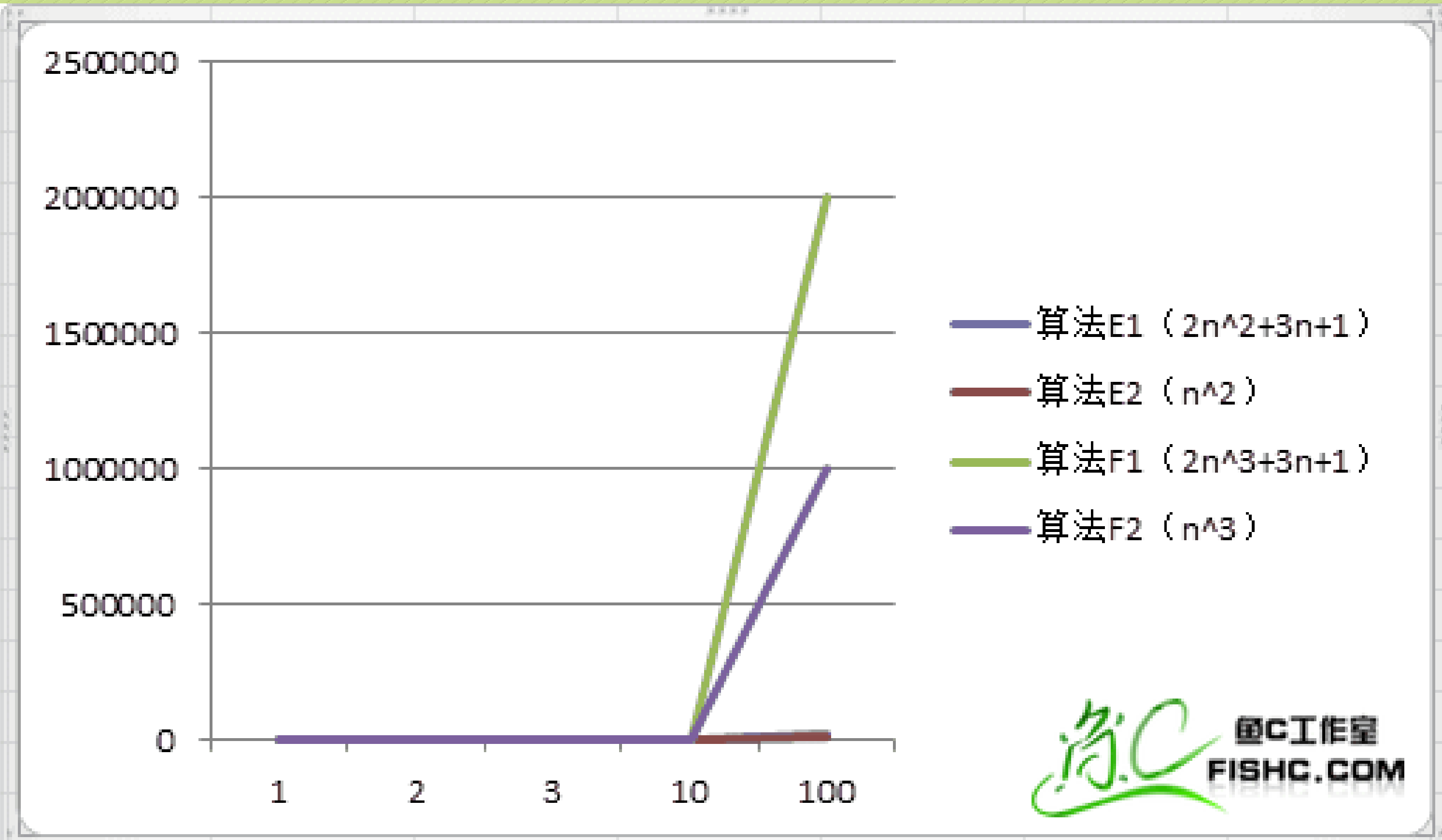
函数的渐近增长

- 我们再来看第三个测试，算法E是2n^2+3n+1，算法F是2n^3+3n+1。

次数	算法E1（2n^2+3n+1）	算法E2（n^2）	算法F1（2n^3+3n+1）	算法F2（n^3）
n=1	6	1	6	1
n=2	15	4	23	8
n=3	28	9	64	27
n=10	231	100	2031	1000
n=100	20301	10000	2000301	1000000

- 再来看一下线性图。

函数的渐近增长


2500000
2000000
1500000
1000000
500000
0
1
2
3
10
100
算法E1（2n^2+3n+1）
算法E2（n^2）
算法F1（2n^3+3n+1）
算法F2（n^3）

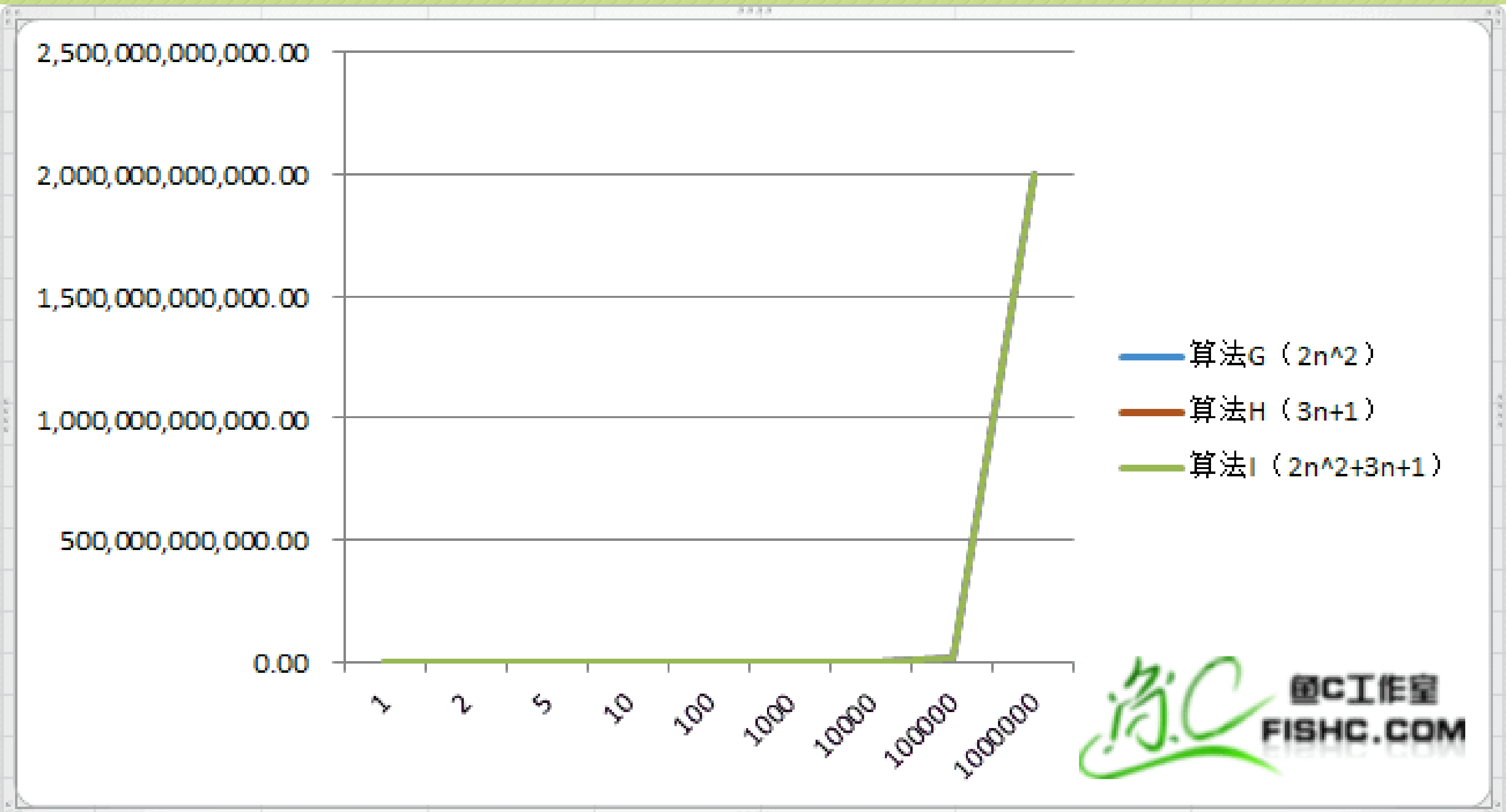
函数的渐近增长

- 这次我们又发现什么呢？
- 小甲鱼没有小鸡鸡？
- 不是的，我们通过观察又发现，最高次项的指数大的，函数随着n的增长，结果也会变得增长特别快。
- 恩，我们进行最后一个小测试，把这些概念都总结起来吧！
- 算法G是2n^2，算法H是3n+1，算法I是 2n^+3n+1。

函数的渐近增长

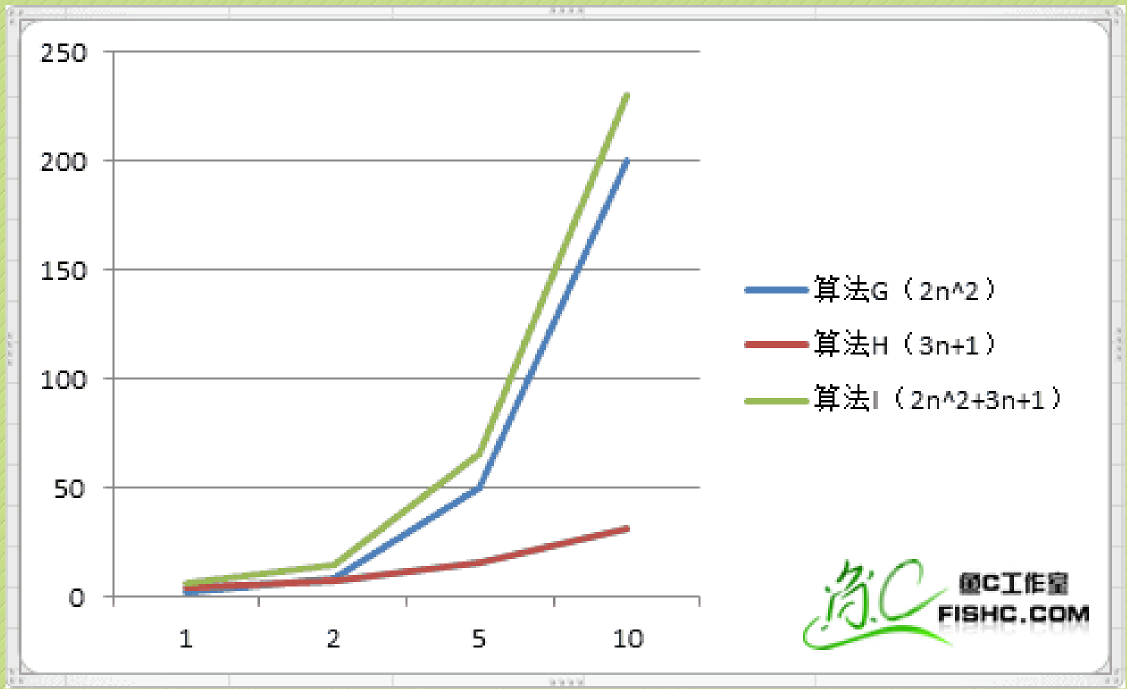
次数	算法G（2n^2）	算法H（3n+1）	算法I（2n^2+3n+1）
n=1	2	4	6
n=2	8	7	15
n=5	50	16	66
n=10	200	31	231
n=100	2000	301	20301
n=1000	2000000	3001	200301
n=10000	200000000	30001	200030001
n=100000	20000000000	300001	20000300001
n=1000000	2000000000000	3000001	2000003000001

函数的渐近增长


2,500,000,000,000.00
2,000,000,000,000.00
1,500,000,000,000.00
1,000,000,000,000.00
500,000,000,000.00
0.00
1
2
5
10
100
1000
10000
100000
1000000
算法G（2n^2）
算法H（3n+1）
算法I（2n^2+3n+1）

函数的渐近增长

- 看出啥？一条直线？当他们数据很小的时候是这样的：



函数的渐近增长

- 这组数据我们看得很清楚，当n的值变得非常大的时候，3n+1已经没法和2n^2的结果相比较，最终几乎可以忽略不计。而算法G在跟算法I基本已经重合了。
- 于是我们可以得到这样一个结论，判断一个算法的效率时，函数中的常数和其他次要项常常可以忽略，而更应该关注主项（最高项）的阶数。
- 注意，判断一个算法好不好，我们只通过少量的数据是不能做出准确判断的，很容易以偏概全。