Version 2.00

编译日期: **2019-09-17**

任何建议及错误信息请发送至邮箱
1049188593@qq.com

目　录

1

时间复杂度的计算

- 知识点：讲解相关知识点。
- 题型：直接上真题。

1.1 知识点和方法论

1.1.1 知识点

- 时间复杂度常用大 O 符号表示
- 时间复杂度是去掉最高项多项式前面的系数, 且不包括函数的低阶项。
- 常见时间复杂度 $(1) < (log_2 n) < (n) < (nlog_2 n) < (n^2) < (n^3) < (2^n) < O(n!) < O(n^n)$

1.1.2 方法论

- 简单计算相关的要计算的函数关键语句执行的次数。
- 对于关键语句使用令关键语句执行的次数是 t 次。

1.2 真题实战

1.2.1 2011 年 408

设 n 是描述问题规模的非负整数，下面程序片段的时间复杂度是 ( )

```
x = 2;
while(x < n/2)
    x=2*x; // 3
```

解:

(计算函数关键语句的执行次数)

令：第三行语句执行了 t 次，

可知条件不满足的情况是 $x * 2^t >= n/2$, 其中 x = 2

所以条件不满足的情况 $2^{t+1} >= n/2$

求解 t：可知 $t >= log_2(n) - 2$

舍去低阶项，可知 $t >= log_2(n)$

再用大 O 表示，可得时间复杂度为 $O(log_2 n)$

1.2.2 2014 年 408

下面程序片段的时间复杂度是 ( )

```
count=0;
for(k=1; k<=n; k*=2)// 1
    for(j=1; j<=n; j++) // 2
        count++;// 3
```

解:

(计算函数关键语句的执行次数)

计算 3 语句的频度

已知：for 里面 for 循环导致 3 语句执行次数是两个 for 次数相乘

令 1 语句中 k*=2 执行了 t 次。1 语句可以执行 $k * 2^t > n$

可得 $t > log_2(n)$

1 的每个循环中，易知 2 语句中 j++ 执行了 n 次。

易知，3 语句的执行次数和 2 语句中 j++ 是一样的的。

得到总执行次数 $log_2(n) * n$ (前面的系数忽略)

易知，时间复杂度为 $log_2(n) * n$

1.2.3 2017 年 408

下列函数的时间复杂度是 ( )

```
int func(int n){
    int i=0; sum=0;
    while(sum < n)
        sum += ++i;\\ 4
    return i;
}
```

解:

(计算函数关键语句的执行次数)

可知 i 的变化是 1,2,3,4,5...

令 4 语句执行了 t 次

sum = 1 + 2 + 3 + ... + t

可知当条件不满足时 $sum = \frac{t*(1+t)}{2} >= n$

得知 $t + t^2 >= 2n$

忽略低次项 $t >= \sqrt{2 * n}$

忽略常数项的系数 $t >= \sqrt{n}$

得知时间复杂度是 $O(\sqrt{n})$