时间复杂度计算

▶时间复杂度：

●算法中所有语句的频度之和记作 T(n)；

●时间复杂度主要分析 T(n)的数量级；

●通常采用算法中基本运算的频度 f(n)来分析算法的时间复杂度；

●T(n)=O(f(n));

▶三部曲：

●找出算法最重要的操作，即所谓的“基本操作”，它们对总运行时间的贡献最大。

●分析并计算基本操作的运行次数。

●去掉运行次数的常系数，并且保留次数最高的含 n 项，剩下的即是该算法的时间复杂度。

▶循环累加型：

①x=0；

```
while(n>=(x+1)*(x+1))    x=x+1;
```

解：设循环次数为 t，则

t	0	1	2	3	4	...	t
x	0	1	2	3	4	...	t

所以有(t+1)^2>n，t+1=$\sqrt{n}$，所以时间复杂度为 O($\sqrt{n}$)

②void aFunc(int n) {

```
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
```

```
            printf("Hello World\n");
        }
    }
}
```

解：当 i = 0 时，内循环执行 n 次运算，当 i = 1 时，内循环执行 n - 1 次运算……当 i = n - 1 时，内循环执行 1 次运算。

所以，执行次数 T(n) = n + (n - 1) + (n - 2)……+ 1 = n(n + 1) / 2 = n^2 / 2 + n / 2。

根据上文说的三部曲，可以知道，此时时间复杂度为 O(n^2)。

```
③int m=0，i，j；
 for（i=l；i<=n；i++）
   for（j=1；j<=2*i；j++）
         m++；
```

解析：当 i=1 时，内循环执行 2 次运算；当 i=2 时，内循环执行 4 次运算；...；当 i=n 时，内循环执行 2n 次运算。

所以执行次数 T(n)=2+4+...+2n=2*(1+2+3+...+n)=(n+1)*n,即时间复杂度为 O(n^2)。

递归型

求解递归函数时间复杂度的关键点在于——①找关系　②初始条件(递归出口）

```
例：int func (int n){     //n>=0
     if(n<=1) return 1;
     return n*func(n-1);
```

解：设执行时间为 T(n)。当 n=0 或 1 时，时间复杂度为 O(1)，即 T(0)=T(1)=O(1)；当 n>1 时，T(n)=O(1)+T(n-1）——在运行过程中分为两个部分，一个是*，一个是子函数的递归，因此时间叠加起来。这里的 n 与执行时间并没有关系，切勿加进去。

$$T(n)=\begin{cases} O(1) & n<=1 \\ O(1)+T(n-1) & n>1 \end{cases}$$

所以 $T(n)=O(1)+T(n-1)=2O(1)+T(n-2)=3O(1)+T(n-3)=...=(n-1)O(1)+T(1)=O(n)$

所以时间复杂度为 O(n）