复杂度分析（下）

数据结构与算法

复杂度除了大 O 表示法，还有**最好时间复杂度**，**最坏时间复杂度**，**平均时间复杂度**，**均摊时间复杂度。**

```
//n 表示数组 array 的长度
int find(int[] array,int n,int x){
    int i = 0;
    int pos = -1;
    for(; i < n; ++i){
        if(array[i] == x){
        pos = i;
        break;}
        }
    return pos;
}
```

这个程序是查找一个无序数组中 x 变量出现的位置，如果没有找到，就返回 -1 ,此时代码的市价复杂度就不为 O(n)，因为变量 x 有可能出现在数组中的任何位置，所以不同情况下，这段代码的时间复杂度是不一样的。

最好时间复杂度：在最理想的情况下，执行这段代码的时间复杂度。此代码为 O(1) 。
最坏时间复杂度：在最糟糕的情况下，执行这段代码的时间复杂度。此代码为 O(n) 。
平均时间复杂度：要查找的变量 x ，要么在数组中，要么不在，假设两者概率都为 1/2 。而在数组中 n 个位置上的概率相同，为 1/n ，所以要查找的数据出现在数组中的概率为 1/(2n)，此时平均时间复杂度的计算过程为：

$$1\times\frac{1}{2n}+2\times\frac{1}{2n}+3\times\frac{1}{2n}+\cdots+n\times\frac{1}{2n}+n\times\frac{1}{2}$$

$$=\frac{3n+1}{4}$$

即期望值。通过大 O 表示法，克制平均时间复杂度为 O(n)。

通常只有同一块代码在不同的情况下，时间复杂度有量级的差距，才会通过这三种复杂度表示法区分。

均摊时间复杂度

```
// array 表示一个长度为 n 的数组
// 代码中的 array.length 等于 n
int count = 0;

void insert(int val){
    if (count == array.length){
        int sum = 0;
        for(int i = 0; i < array.length; ++i){
            sum = sum + array[i];
            }
            array[0] = sum;
            count = 1;
            }
    array[count] = val;
    ++count;
}
```

这段代码实现一个往数组中插入数据的功能。当数组满了之后，通过 for 循环将数组遍历求和，并清空数组，将求和之后的 sum 值放到数组第一个位置，再插入新的数据。但是如果数组一开始就有位置，则直接插入。这段代码的最小时间复杂度为 O(1) ， 最大时间复杂度为 O(n) ,平均时间复杂度为 O(1)。

insert() 函数，O(1) 时间复杂度的插入和 O(n) 时间复杂度的插入，出现的频率是有规律的，一个 O(n) 插入之后，紧跟着 n-1 个 O(1) 插入。针队这种特殊场景的时间复杂度分析，采用均摊时间复杂度分析。

每一此 O(n) 插入操作，都会跟着 n-1 次 O(1) 的插入操作，所以把耗时最多的那次操作均摊到接下来的 n-1 次操作行，这一组连续的均摊时间复杂度就是 O(1)。

参考自：极客时间《数据结构与算法之美 》专栏