一。最好、最坏情况时间复杂度

```
// 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;
}
```

最好情况时间复杂度就是，在最理想的情况下，要查找的变量 x 正好是数组的第一个元素。
最坏情况时间复杂度就是，如果数组中没有要查找的变量 x，我们需要把整个数组都遍历一遍才行

二。平均情况时间复杂度

要查找的变量 x 在数组中的位置，有 n+1 种情况：在数组的 0～n-1 位置中和不在数组中
我们把每种情况下，查找需要遍历的元素个数累加起来，然后再除以 n+1，就可以得到需要遍历的元素个数的平均值，即

得到的平均时间复杂度就是 O(n)。

考虑概率情况：我们假设在数组中与不在数组中的概率都为 1/2。
数据出现在 0～n-1 这 n 个位置的概率也是一样的，为 1/n。所以，根据概率乘法法则，要查找的数据出现在 0～n-1 中任意位置的概率就是 1/(2n)。

得到的平均时间复杂度就是 O(n)。
这个值就是概率论中的加权平均值，也叫作期望值，所以平均时间复杂度的全称应该叫加权平均时间复杂度或者期望时间复杂度。

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，我们才才会使用这三种复杂度表示法来区分。

三。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 就等于 n
int[] array = new int[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;
}
```

假设数组的长度是 n，根据数据插入的位置的不同，我们可以分为 n 种情况，每种情况的时间复杂度是 O(1)。除此之外，还有一种“额外”的情况，就是在数组没有空闲空间时插入一个数据，这个时候的时间复杂度是 O(n)。而且，这 n+1 种情况发生的概率一样，都是 1/(n+1)。所以，根据加权平均的计算方法，我们求得的平均时间复杂度就是：

首先，find() 函数在极端情况下，复杂度才为 O(1)。但 insert() 在大部分情况下，时间复杂度都为 O(1)。只有个别情况下，复杂度才比较高，为 O(n)。这是 insert()第一个区别于 find() 的地方。

我们再来看第二个不同的地方。对于 insert() 函数来说，O(1) 时间复杂度的插入和 O(n) 时间复杂度的插入，出现的频率是非常有规律的，而且有一定的前后时序关系，一般都是一个 O(n) 插入之后，紧跟着 n-1 个 O(1) 的插入操作，循环往复。

我们还是继续看在数组中插入数据的这个例子。每一次 O(n) 的插入操作，都会跟着 n-1 次 O(1) 的插入操作，所以把耗时多的那次操作均摊到接下来的 n-1 次耗时少的操作上，均摊下来，这一组连续的操作的均摊时间复杂度就是 O(1)。这就是均摊分析的大致思路。你都理解了吗？