复杂度分析：最好、最坏、平均、均摊时间复杂度

1. 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;
}
```

这段代码实现的功能是：在数组中查找一个元素x的位置，找到了就返回，未找到就返回-1。 通过时间复杂度分析，这段代码的时间复杂度应该为：O(N)

但是，现在不妨假设几种情况：

1.假如元素x的位置就在数组的第一个位置，那么这段代码的时间复杂度就是O(1),这就是最好情况时间复杂度。最好情况时间复杂度就是，在最理想的情况下，执行这段代码的时间复杂度。

2.假如数组中根本就不存在元素x，那么这段代码会将数组中的所有元素遍历一遍，那么这段代码的时间复杂度就是O(N),这就是最坏情况时间复杂度。最坏情况时间复杂度就是，在最糟糕的情况下，执行这段代码的时间复杂度。

平均情况时间复杂度：

为了更好的表平均情况下的复杂度，我们引入平均情况时间复杂度。

查找变量x的位置，一共有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：

$$\frac{1+2+3+\cdots+n+n}{n+1}=\frac{n(n+3)}{2(n+1)}$$

利用大O表示法，将这个公式简化后，得到的平均时间复杂度为O(n)。实际上，元素一共有n + 1种情况，但是每种情况出现的概率并不是一样的

由上可知，查找的元素x可能在数组中，也可能不在数组中，现在假设在数组中与不在数组中的概率都为1/2。另外，要查找的数据出现在 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)。

> 乘法原理又称分步计数原理：做一件事，完成它需要分成n个步骤，做第一步有$m1$种不同的方法，做第二步有$m2$种不同的方法，……，做第n步有mn种不同的方法，那么完成这件事共有$N=m1\times m2\times m3\times\ldots\times mn$种不同的方法。 乘法原理中的每一步都 不能独立完成任务，且各步都不可缺少，需要依次完成所有步骤才能完成一个独立事件，只有满足这个条件，才能用乘法原理。每次事件都是独立的互不影响，就用加法原理。

因此，前面的推导过程中存在的最大问题就是，没有将各种情况发生的概率考虑进去。如果我们把每种情况发生的概率也考虑进去，那平均时间复杂度的计算过程就变成了这样：

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

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

大 O 表示法来表示，去掉系数和常量，这段代码的加权平均时间复杂度仍然是 O(n)。