算法复杂度理论

我们用复杂度来量化一个算法的时间，空间。在这一小节中，我们讲学习什么是复杂度，什么是时间复杂度，什么是空间复杂度。

在面试中，时间复杂度是问得比较多的，空间复杂度一般不会问。

时间复杂度是面试中必问的问题。学好时间复杂度，有如下的帮助：

1. 面试官会问你的算法时间复杂度是什么
2. 当面试官说，有没有更好的方法时，你知道朝什么样的复杂度优化
3. 利用时间复杂度倒推算法是面试常用技巧。如 O(logN) 的算法几乎可以确定是二分法。

一个算法的运行时间与其所要执行的语句的数量成正比，而所要执行的语句与问题规模正相关。因此算法的时间复杂度可以表示为一个与问题规模 N相关的多项式。

例如下面的代码：

int sum = 0;

int n = 100;

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

sum += i;

}

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

sum += i;

}

这段代码的运行时间与n的大小正相关，因此，我们可以将其复杂度写成多项式：f(n) = 2n;

在计算机科学中，时间复杂度使用标记O(大写英文字母o)表示，**只包含上述多项式中的最高次项，且忽略最高次项的系数**  
上面这段代码的时间复杂度就是：O(n)。

时间复杂度定性的描述了一个算法的运行时间。（定义参照[WIKI](https://zh.wikipedia.org/wiki/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)）

更为通俗的定义是：程序执行了多少句语句，一条一句被多次执行，就要算多次。

时间复杂度计算的要点：

* **只包含多项式的最高次项。** 这是因为在复杂度计算中，最高次项对运行时间有决定性的作用，次高次项可以忽略不计。例如f(n) = n^2 + n， 此时n这一项对于多项式的值的影响想对于n^2可以忽略不计。在定性的描述中，我们只取最高次项。
* **不包含多项式最高次项的系数。** 对于最高次项，我们忽略它的系数，在算法中，我们称之为常数。上面代码中，常数是2,但是时间复杂度的计算，我们只取O(n)。

在计算算法的时间复杂度时，我们关注其中最耗时的部分，对于相对不那么耗时的部分就忽略不去考虑。也就是上面讲的”只保留最高次项“。

**例子:**

sum = 0;

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

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

sum++;

}

}

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

sum++;

}

对于上面这段代码，一个两重循环和一个一重循环。  
两重循环的时间复杂度是：**O(n^2)**,  
一重循环的时间复杂度是：**O(n)**。  
那么我们标记这段代码的时间复杂度为:**O(n^2)**， 而不去考虑那个小的O(n)。  
我们只关心算法中最耗时的部分，并且，假如代码中做了好几次两重循环，我们也不计较这个几次，而是只保留复杂度的“最高次项”，即O(n^2)。



在面试中，经常会涉及到时间复杂度的计算。当你在对于一个问题给出一种解法之后，面试官常会进一步询问，是否有更优的方法。此时就是在问你是否有时间复杂度更小的方法（有的时候也要考虑空间复杂度更小的方法），这个时候需要你对常用的数据结构操作和算法的时间复杂度有清晰的认识，从而分析出可优化的部分，给出更优的算法。

例如，给定一个已经排序的数组，现在有多次询问，每次询问一个数字是否在这个数组中，返回True or False.

* 方法1： 每次扫描一遍数组，查看是否存在。  
  这个方法，每次查询的时间复杂度是: O(n)*O*(*n*)。
* 方法2：由于已经有序，可以使用二分查找的方法。  
  这个方法，每次查询的时间复杂度是: O(logn)*O*(*logn*)。
* 方法3：将数组中的数存入Hashset。  
  这个方法，每次查询的时间复杂度是: O(1)*O*(1)。

可以看到，上述的三种方法是递进的，时间复杂度越来越小。

在面试中还有很多常见常用的方法，他们的时间复杂度并不是固定的，都需要掌握其时间复杂度的分析，要能够根据算法过程自己推算出时间复杂度。

用 T 函数表示法计算时间复杂度

T 函数推导法

**T 函数推导法**

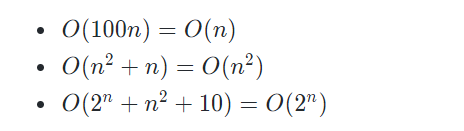
我们介绍一种时间复杂度的推导方法：T函数推导法  
比如二分法。二分法是每次通过 *O*(1) 的时间将规模为 *n* 的问题降低为规模为 *n*/2 的问题。  
这里我们用 *T*(*n*) 来表示规模为 n 的问题在该算法下的时间复杂度，那么我们得出推导公式：

*T*(*n*)=*T*(*n*/2)+*O*(1)

我们来逐个说明一下这个公式的意义。

首先 T 代表的是 Time Complexity, *n* 代表的是问题规模（二分法里就是数组的大小）。  
那么 T(n) 代表的就是：求处理问题规模为n的数据的时间复杂度是多少。注意这里是一个问句，不是一个答案。  
*T*(*n*) 根据算法的不同可以是 *O*(*n*) 也可以是 *O*(*nlogn*) 或任何值，而 *O*(*n*) 就是 *O*(*n*)。

然后 O 代表的是时间复杂度。O(1) 就意味着，你大概用一个 if 语句，或者简单的加加减减，就可以完成。O 在这里的意思是数量级约等于。在 O 的世界里，我们只考虑最高项是什么，不考虑系数和常数项。比如：



**如何推导 T 函数**

我们可以使用不断展开的方法进行推导：

T(n) = T(n/2) + O(1)

= T(n/4) + O(1) + O(1)

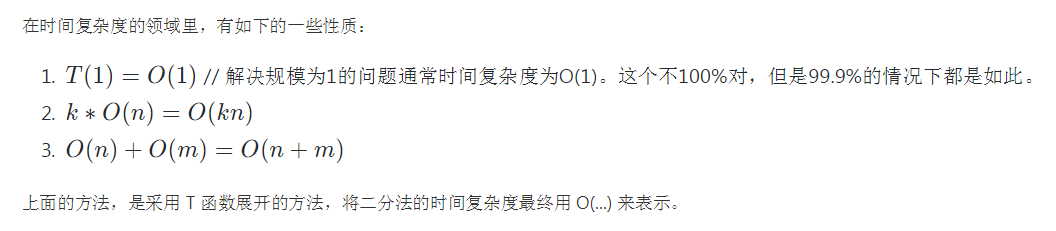
= T(n/8) + O(1) \* 3

= T(n/16) + O(1) \* 4

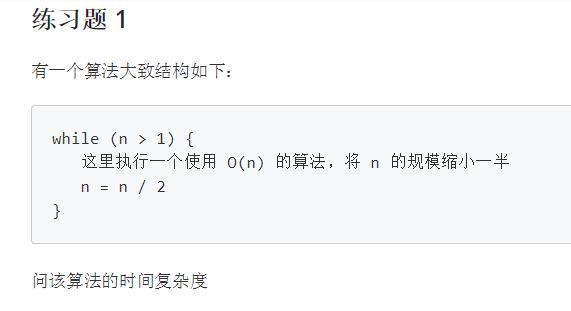
...

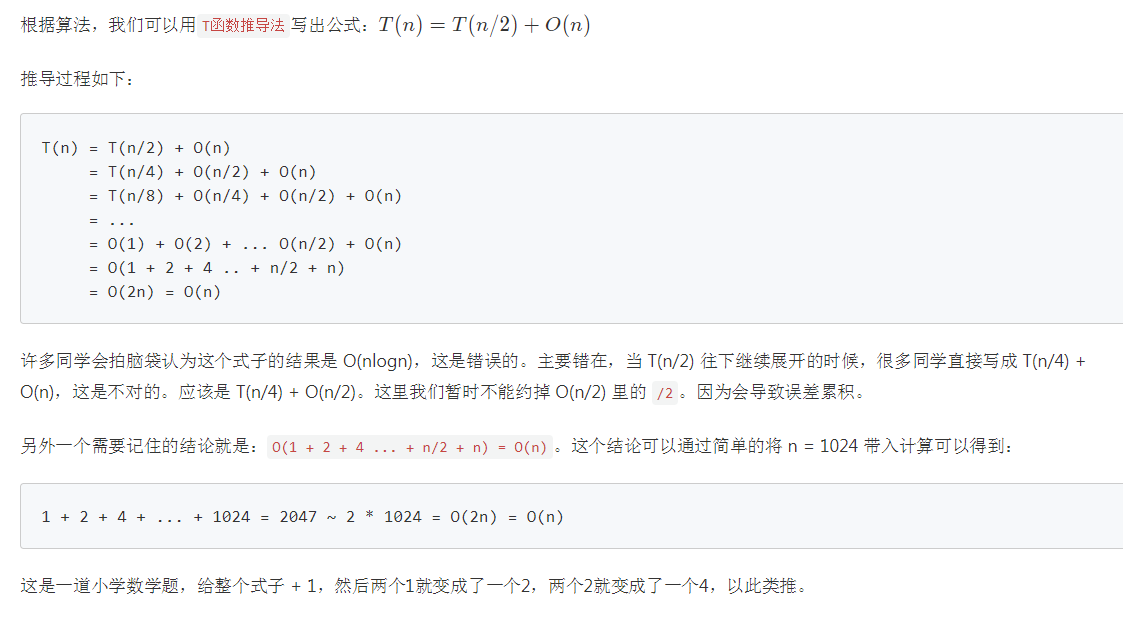
= T(1) + O(1) \* logn

= O(logn)



练习题1





Takeaways:

1. O(1)+O(2)+…+O(n)=O(1+2+….+n)

2．另外一个需要记住的结论就是：O(1 + 2 + 4 ... + n/2 + n) = O(n)。 因为如果再最前面+1

1+ 1 + 2 + 4 + ... + 1024 = 2048 = 2 \* 1024 = O(2n) = O(n)

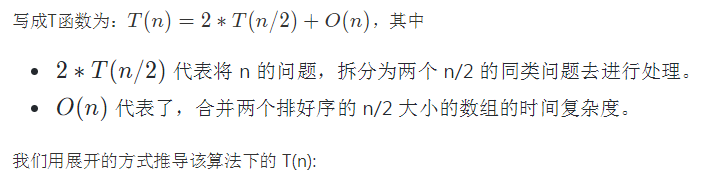
一个数+他所有前面的数之和

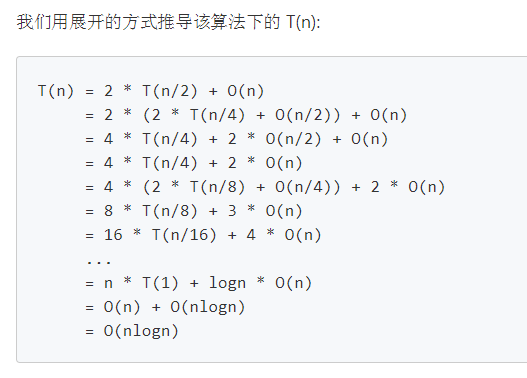
练习题2

请用 T 函数来推导归并排序(merge sort)算法的时间复杂度

归并排序(merge sort)算法的步骤为：

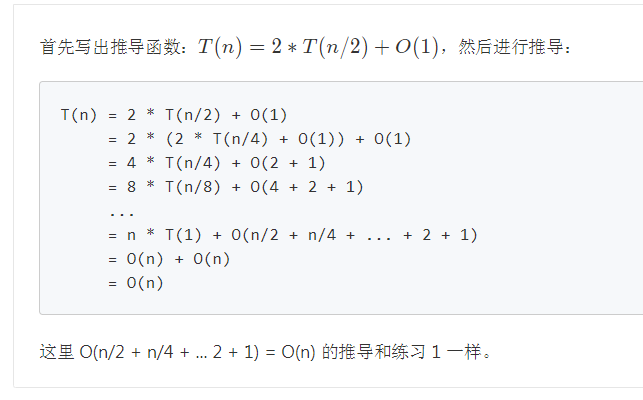
1. 找出数组的中点
2. 将数组分成两个部分递归执行该算法，分别排序两个部分
3. 合并两个排序好的子数组到一个大数组





**练习题 3**

如果一个算法，每次通过 O(1) 的时间将 n 的问题拆分为两个 O(n/2) 的问题，求时间复杂度。



**Extra 练习**

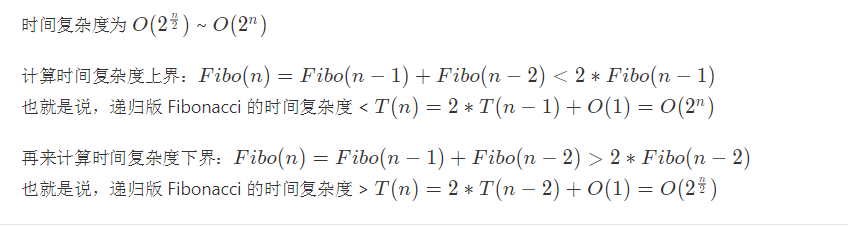
**练习 3**

int Fibo(int n) {

if (n == 0 || n == 1) return 1;

return Fibo(n - 1) + Fibo(n - 2);

}



**练习 6（难）**

int j = 0;

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

while (j < n && nums[j] - nums[i] < window) {

j++;

}

}

