**一：时间复杂度讲解（大O表示法）**

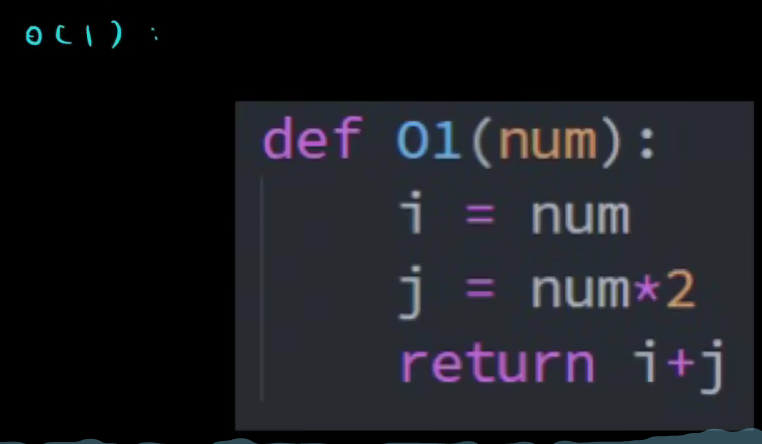
**1.**什么是时间复杂度： 算法的执行效率

算法的执行时间与算法的输入值的关系

**2.**常见的复杂度以及分析：

O（1）， O（log n）， O（n）， O（nlog n），O（n²）

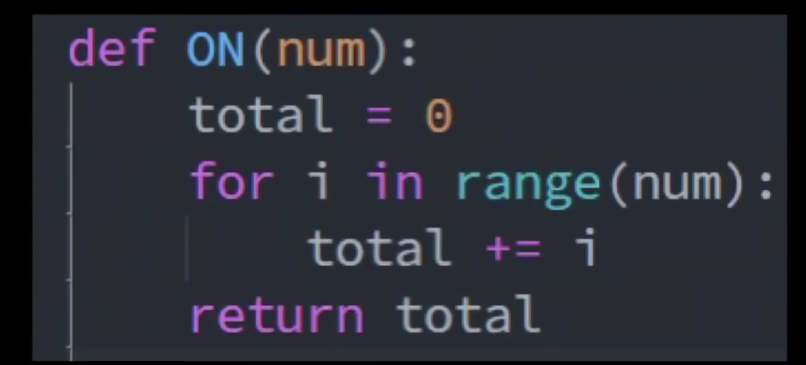
O（1）：



O（1）的分析，我们发现不管num是多少这个算法的时间还是从头到尾进行下去。所以时间复杂度就是

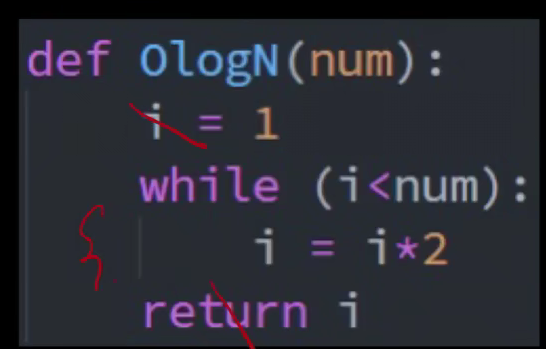
O（1），一般没有for和while循环。

O（N）：



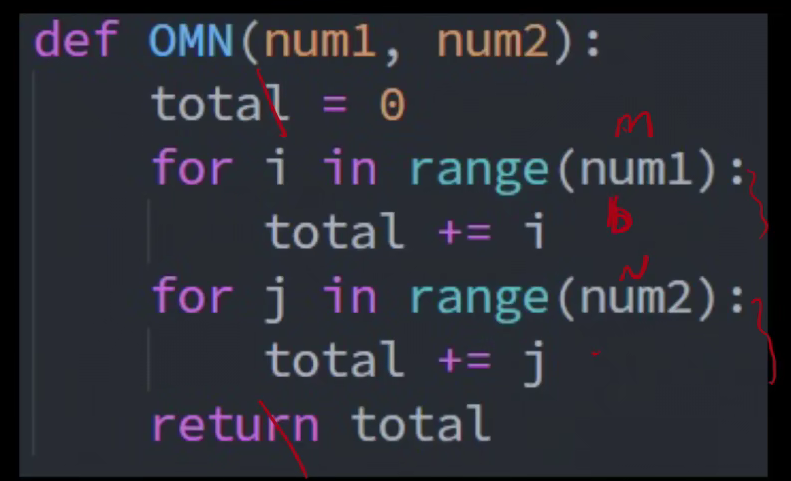
你看如果num值为1这个就要循环1次，如果num是100这个循环就执行100，如果num为n这个循环就执行n次，因为这个算法主要耗时是在循环里面的，假设这个for循环耗时一次为b，这个算法的时间就是nb，然后这个O表示的就是主要耗时b，于是这个算法时间就是O（N）。

O（log N）：



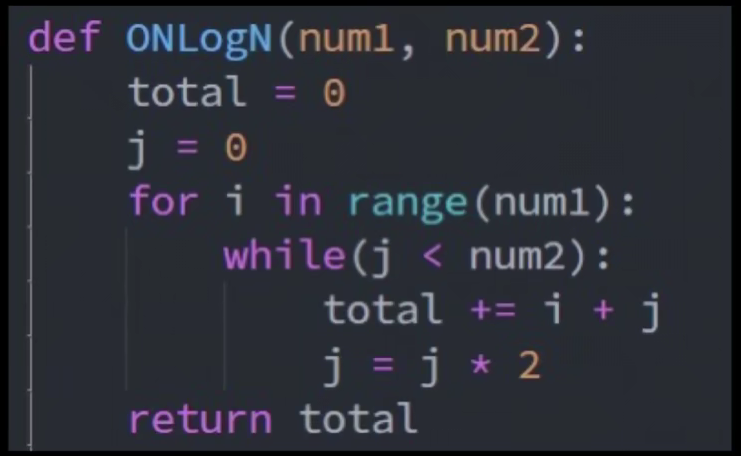
我们看这个循环啊，这个循环告诉我们如果i<num,这个i就不停的乘2直到i》=num的时候就停止。所以假设啊这个num为50，这个循环好执行多少次呢？执行log ₂50次，当num为N次的时候呢那就是log ₂N次，这个2也不是每个算法都会有的，所以这种算法的时间复杂度就为：O（log n）。

O（M+N）：



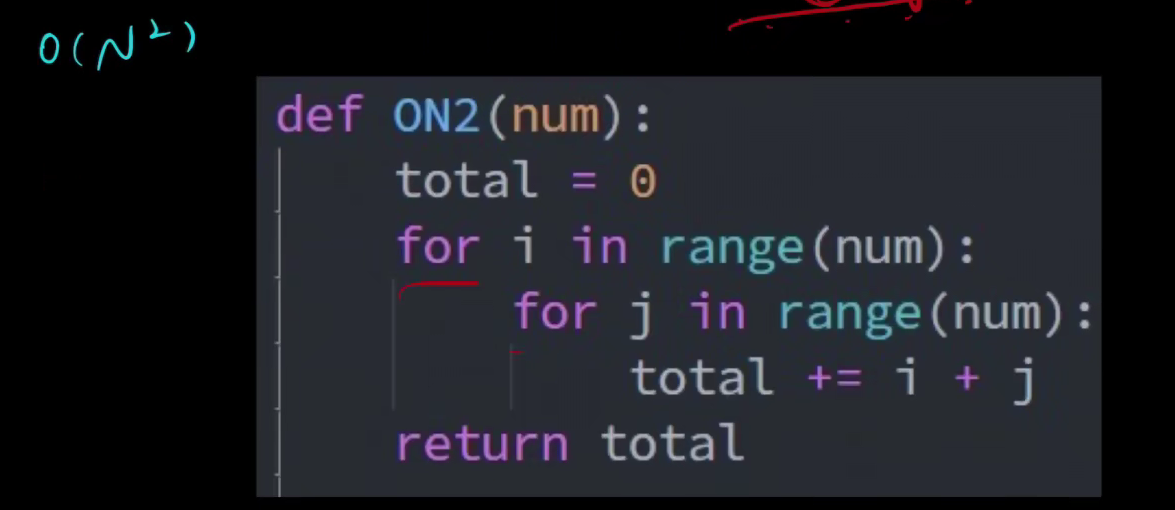
表示有两个循环，但是这两个循环是**并列**的所以就是相加。

O（M logN）：



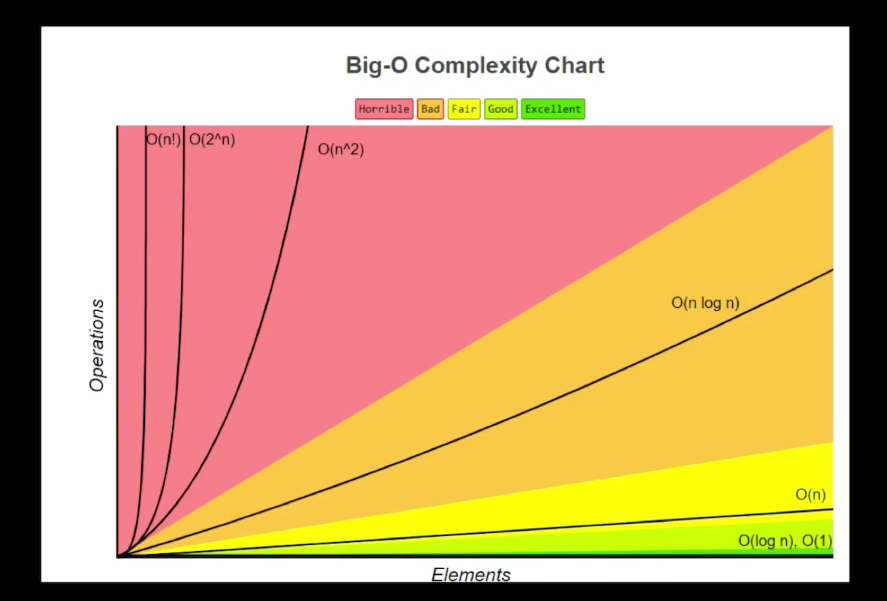
也表示有两个循环，但是这**两个循环是嵌套**的所以这就是相乘。

O（M\*N）：



也是两个循环，因为这两个循环都是以num的次数，所以就是N²。

**3.时间复杂度之间的对比：**



**O（1）<O(logN)<O（N）<O(NlogN)<O(N²)<O(2的n次方)<O(N!) 做算法尽量减少时间复杂度。**

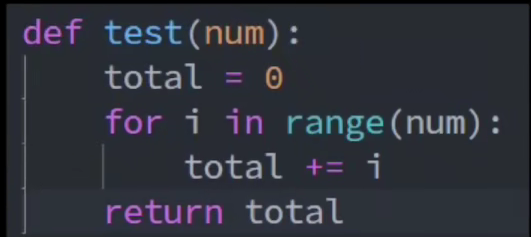
O(logN)一般存在于二分查找法，O(NlogN)一般存在于排序里面

**二：空间复杂度讲解（大O表示法）**

什么是空间复杂度：算法的存储空间与输入值之间的关系

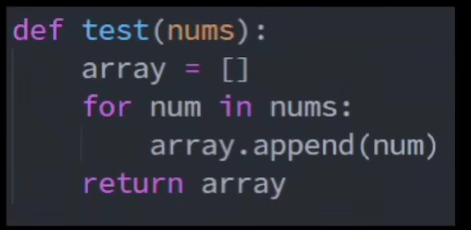
1. **常用空间复制度分析：**

**（1）O（1）**



你看啊，他只开辟了一个空间来存储这个值，不管num的值为多少最后也只是这一个空间用来存储值，所以这个时间复杂度就为O（1）。

**（2）O（N）：**



这个算法用到的是数组，然后他把得到的值放在这个数组里面，这个循环随着这个num的一个一个向数组里传值进去，所要的空间也会越来越大，所以空间复杂度为O（N）。

一般而言空间复杂度就是这两个，看到数组或者集合就是O（N）也可能是O（N²）【O（N²）一般也很少出现）】，O（logN）和O（NlogN）基本上就看不到。

分析空间复杂度要分析两点：

①：分析变量（如果是数组或者多个数集合在一起的那种变量就是O（N）或者O（N²），其他的一个存储空间的话就是O（1））

②：分析是否递归（每一层的信息都保留在递归栈里一般就是O（N））

空间复杂度排序：

O（1）<O(N)<O(N²)

Tips：

时间空间只能二选一，有的时候想要保留时间复杂度低一点就要牺牲空间复杂度相反如果空间复杂度低一些的话就要牺牲时间复杂度。

面试时问别人您想要了解时间复杂度低的还是空间复杂度低一点的方式。

工作时一般要牺牲空间复杂度来换取时间。

**三：数据结构**

**1：数组Array**

数组：在连续的内存空间中，存储一组相同类型的元素。

区分元素和索引：元素是数组存储的东西，索引代表的元素在内存空间中相对的位置，第一个元素索引永远是0第二个就是1等等等....

数组的访问和搜索：数组的访问是通过索引访问某一个元素，搜索是查找某一个元素，要从头到尾遍历数组然后返回一个索引。

数组的方法以及他们的时间复杂度：

1. ：访问 Access O（1）
2. ：搜索 Seach O（N）
3. ：插入 Insert O（N）
4. ：删除 Delete O（N）

因为搜索，插入，删除都要遍历数组然后进行操作，所以复杂度为O（N），但是访问只要通过索引来找元素，所以复杂度为O（1）

数组的常规操作：

1. ：创建数组：

方法一：int[] arr = {1,2,4, …};

方法二：int[] arr;

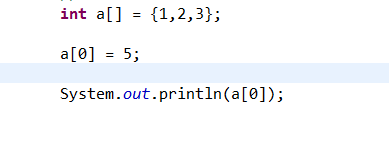
arr = new int[]{1,2,3, …};

方法三：int[] arr1= new int[3];

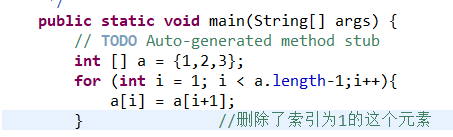
1. ：添加元素
2. ：访问元素

a[i]

1. ：修改元素



1. ：删除元素



1. ：遍历数组

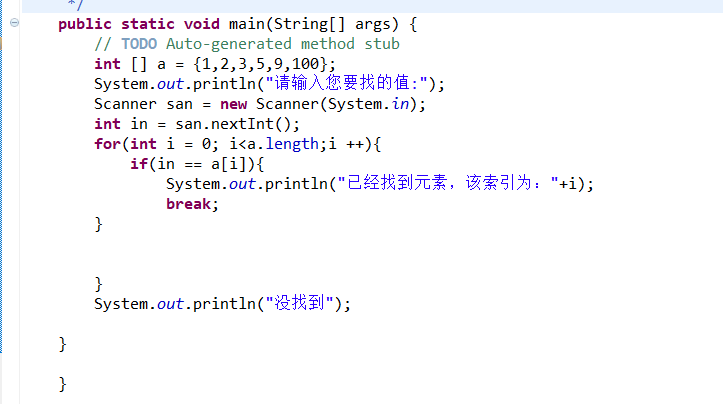
int[] is = new int[1,12,4,546]

for(int i=0; i<is.length; i++){

System.out.println(int[i] + "");

}

1. ：查找元素



1. ：数组的长度

Array.length

1. ：数组的排序（内置的排序方法）

Arrays.sort(数组名) 升序排法。

Arrays.sort(a, Collections.reverseOrder()); 降序排法

练习题：力扣 485 283 27

1. **链表**

链表特点：不连续的内存空间，同一类型。

一般单向链表就够用了

链表一般操作的时间复杂度：

访问：O（N）

查找：O（N）

插入：O（1）

删除：O（1）

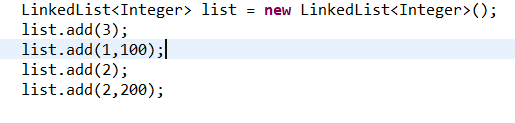
所以链表的特点：写的快，读的慢。（读少写多）

链表常用操作：

1. ：创建链表



1. ：添加元素



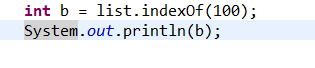
1. ：访问元素

根据索引访问内容：



1. ：查找元素

根据内容找在哪（索引）：



1. ：删除元素



1. ：链表的长度



题目：203 206

1. **队列（Queue）**

队列就好比你去排队，先排队的先拿完东西，所以队列的特点是：先入先出。

队列分两种：①：单端队列：只有一个口可以进，一个口可以出。（单端这种多些）

②：双端队列：两个口都可以进，也都可以出。（分左和右）

队列的时间复杂度：

访问：O（N）（队列一个顺序的，如果你要根据索引查找元素，所以要从开始执行）

搜索：O（N）（给你一个数要从头到尾查一遍在哪？根据元素找索引）

插入：O（1）（插入只能在队列的末尾端插入）

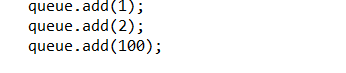
删除：O（1）（删除只能删除头元素）

基本操作：

1. ：创建队列。



1. ：添加元素。



1. ：获取即将出队的元素（第一个进队的元素）。



1. ：删除即将出队的元素（第一个进队的元素）



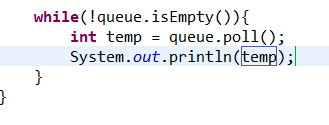
1. ：判断队列是否为空



1. ：队列的长度。



1. ：遍历队列。



力扣刷题：933

1. **栈（stack）**

栈的特点是：先进后出。 （应用于浏览器的后退功能）

栈的时间复杂度：

访问：O（1） （栈顶元素）

查找（search）：O（N）

插入：O（1）

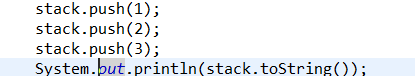
删除：O（1） （栈顶元素）

栈的常规操作：

1. ：创建栈



1. ：添加元素



1. ：查看栈顶元素



1. ：删除栈顶元素



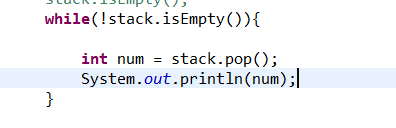
1. ：栈的长度



1. ：栈是否为空



1. ：遍历栈



20，496