时间复杂度

1.什么是时间复杂度？

表示的是一个算法执行效率与数据规模增长的变化趋势算法的执行效率：算法的执行时间与算法的输入值之间的关系

```
def test(num): //10
    total = 0        //a
    for i in  range(num):    //从0加到num  10b
        total += i  //b
    return total    //c
```

假设:三个语句的执行时间分别为 a,b,c num值为10，则for循环中的执行时间为10b，因此总体的执行时间是：a + 10b + c 。由此可见该语句的总体时间是由for循环决定的，因为a、c都是固定的（常量）。因此，通常在计算时间复杂度的时候，可以把a和c的时间去掉(忽略不计)。因此我们可以说该语句的时间复杂度是10b，10由num决定。

再次假设：num的值为n ，则该语句的时间复杂度是nb，由于b为系数，所以可以去掉，所有该语句的时间复杂度是：n

- 不关心系数

- 小的执行时间去掉
- 通常多观察是否由for循环和while循环
- 总的时间复杂度就等于量级最大的那段代码的时间复杂度

2.大O表示法

O(n) -> O(N)，N代表num是多少

3.常见的时间复杂度分析

1. O(1): 通常表示算法的执行时间和**num**值没有关系

```
def o1(num):
    i = num        //a
    j = num * 2    //b
    return i + j
```

总的执行时间是 a+b 与传入的num值无关

2. O(N)：有一个**for**循环，其他执行时间去掉

```
def ON(num):
    total = 0
    for i in range(num):   //n
        total += i         //b
    return total
```

总的执行时间是nb, 去掉系数b

3. O(logN): 二分查找法

```
def OlogN(num):
    i = 1
    while (i < num):    //n
        i = i * 2
    return i
```

从代码中可以看出，变量 i 的值从 1 开始取，每循环一次就乘以 2。当大于 n 时，循环结束。实际上，变量 i 的取值就是一个等比数列。如果我把它一个一个列出来，就应该是这个样子的：

所以，我们只要知道 x 值是多少，就知道这行代码执行的次数了。通过 2^x=n 求解 x , x=log2n，所以，这段代码的时间复杂度就是 O(log2n)。

4. O(M+N) : 两个**for**循环

```
def OMN(num1,num2):
    total = 0
    for i in range(num1):  //M
        total += i
    for j in range(nun2):  //N
        total += j
    return total
```

5. O(NlogN): 排序 **for**循环嵌套**while**循环

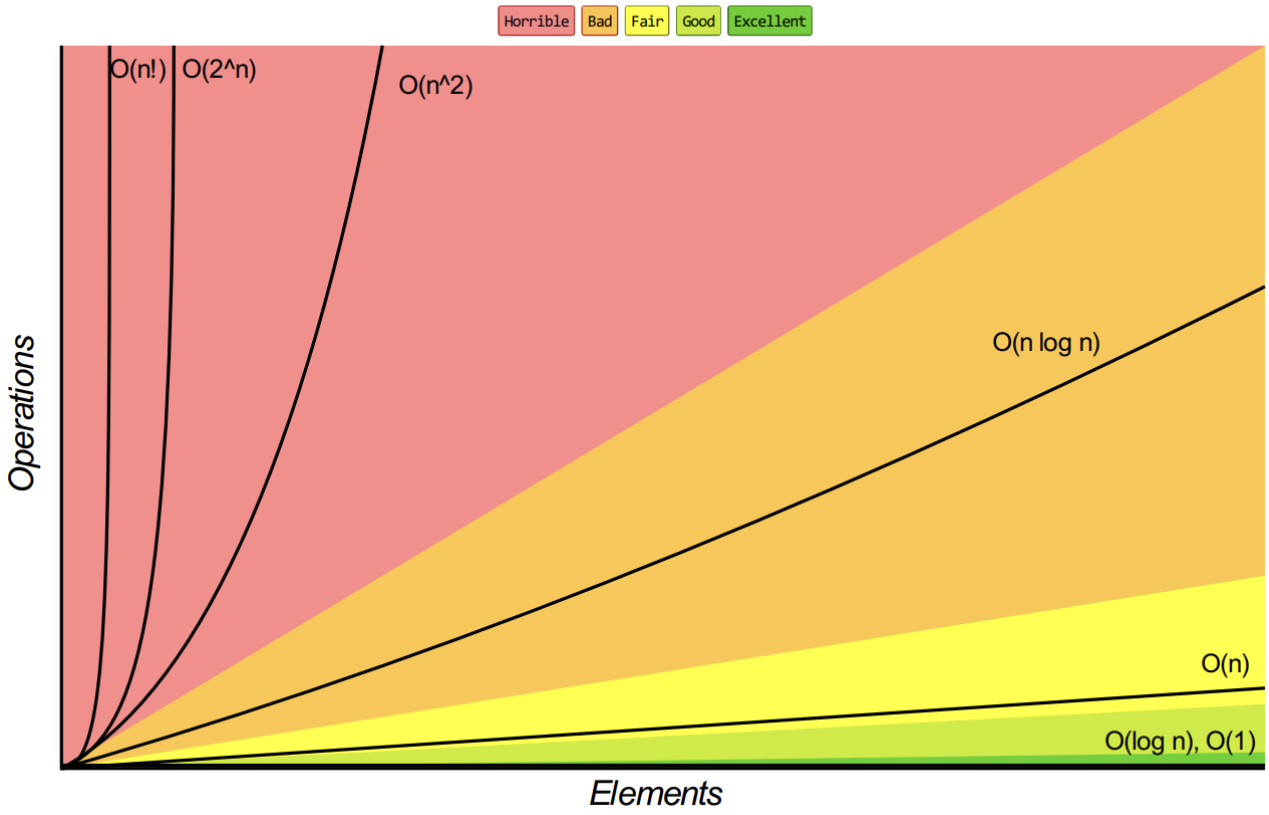
```
def ONlogN(num1,num2):
    total = 0
    j = 1
    for i in range(num1):  //M
        while(j < num2):   //N
            total += i + j
            j = j * 2
    return total
```

内层while循环的时间复杂度是O(logN),又因为外层还有个for循环，时间复杂度是O(M), 所以这段代码的时间复杂度是O(MlogN), 通常写为 O(NlogN)

6. O(N^2)：两层**for**循环嵌套

```
def ON2(num)
    total = 0;
    for i in range(num):       //N
        for j in range(num):   //N
            total += i + j
    return total
```

7.总结：



$O(1) < O(\log N) < O(N) < O(N \log N) < O(N^2) < O(2^n) < O(n!)$