



算法复杂度



算法效率

- 时间复杂度和空间复杂度

衡量算法效率的重要标准。

- 乐学反馈

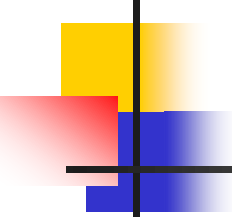
- **TLE**: 时间超时（死循环或效率太低）
- **MLE**: 内存超限（数组越界或者栈溢出）



时间复杂度

■ 基本操作数

- 同一个算法在不同的计算机上运行的速度会有差别，并且实际运行速度难以在理论上进行计算，实际去测量又比较麻烦，所以**不考虑算法运行的实际用时**，而是**计算**算法运行所需要进行的**基本操作的数量**。
- **加减乘除、访问变量、给变量赋值**等，都可以看作基本操作。
- **将基本操作的数量作为算法用时指标**



■ 数据规模

- 衡量一个算法的快慢，一定要考虑数据规模的大小。
- 数据规模，一般指输入数据的数量，例如数字个数、输入中给出的图的点数与边数等等。
- 一般来说，数据规模越大，算法的用时就越长。
- 衡量一个算法的效率时，最重要的不是看它在某个数据规模下的用时，而是看它的用时随数据规模而增长的趋势，即“时间复杂度”。



■ 时间复杂度

- **最坏**时间复杂度，即每个输入规模下用时最长的输入对应的复杂度。
- **平均**（期望）时间复杂度，即每个输入规模下所有可能输入对应用时的平均值的复杂度（随机输入下期望用时的复杂度）
- **最理想**情况下的时间复杂度
- 评估最坏情况可以避免后顾之忧，因此一般情况下，都要**估算最坏情况的时间复杂度**。



■ 时间复杂度的表示

- 时间复杂度可以看作输入规模的函数，但一般不用函数表达式来表示，而是用渐进符号来表示。
- 渐进符号忽略一个函数中增长较慢的部分以及各项的系数，仅保留可以用来表明该函数增长趋势的重要部分。
- 大O表示法： $O(\cdot)$ ，含义：小于等于，表示上界；指的是一个算法在最糟糕情况下的运行时间。
只保留函数的最高阶项（不含系数）。



■ 时间复杂度分析

```
1 void Demo(int n) {  
2     for(int i = 0; i < n; i++) {  
3         for(int j = 0; j < n; j++) {  
4             printf("Hello, World!\n"); // 第一部分时间复杂度为  $O(n^2)$   
5         }  
6     }  
7     for(int j = 0; j < n; j++) {  
8         printf("Hello, World!\n"); // 第二部分时间复杂度为  $O(n)$   
9     }  
10 }  
11 //时间复杂度为  $\max(O(n^2), O(n))$ , 即  $O(n^2)$ 。
```



■ 时间复杂度示例

`for (int i = 1; i <= n; i++) {}` **$O(n)$**

`for (int i = 1; i <= n; i *= 2) {}` **$O(\log n)$**

`for (int i = 1; i <= n; i++)
 for (int j = 1; j <= m; j++) {}` **$O(n*m)$**

`for (int i = 1; i <= n; i++)
 for (int j = 1; j <= n; j += i) {}` **$O(n \log n)$**

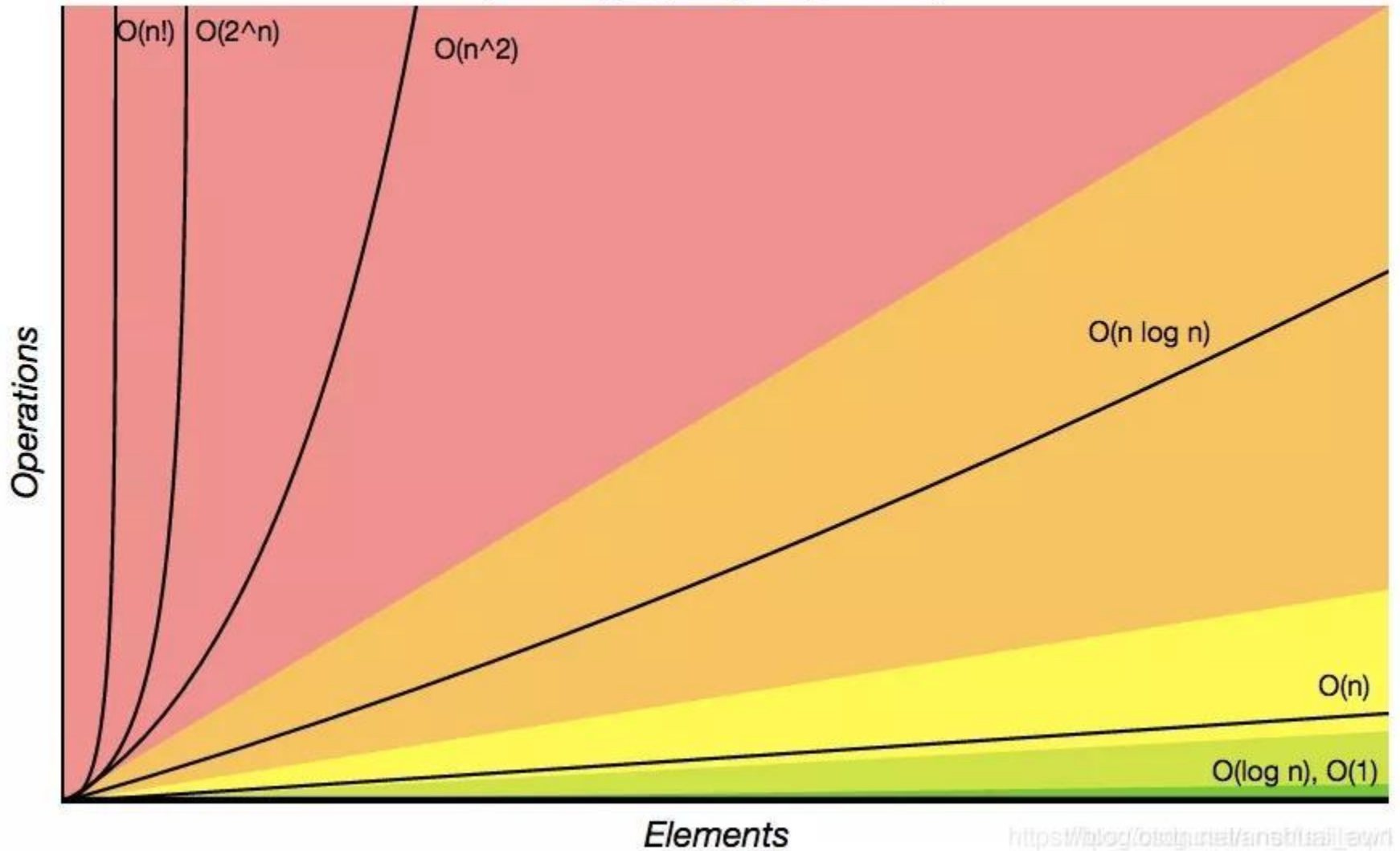


空间复杂度

- 算法所使用的空间随输入规模变化的趋势
- 与时间复杂度类似
- 空间与时间，互相制约的两个因素
可以空间换时间，或是时间换空间
- 常用的复杂度量级
常数阶 $O(1)$ 、对数阶 $O(\log n)$ 、线性阶 $O(n)$ 、
线性对数阶 $O(n \log n)$ 、平方阶 $O(n^2)$ 、
立方阶 $O(n^3)$ 、K次方阶 $O(n^k)$ 、
指数阶 $O(2^n)$ 、阶乘阶 $(n!)$

Big-O Complexity Chart

Horrible Bad Fair Good Excellent





参考资料

- https://blog.csdn.net/Yusean_L/article/details/121628410
- <https://oi-wiki.org/basic/complexity/>