复杂度

习惯

写完程序下意识分析时间空间复杂度

用最简洁的时间和空间复杂度完成这段程序

时间复杂度

O(1): Constant Complexity 常数复杂度

O(logn): Logarithmic Complexity 对数复杂度

O(n): Linear Complexity 线性时间复杂度

O(n^2): N square Complexity 平方 ← for循环嵌套几层, 就是 O(n^几层)

O(n^3): N square Complexity 立方

O(2^n): Exponential Growth 指数 ← 斐波那契 F(n) = F(n - 1) + F(n - 2), 常规递归复杂度 O(2^n)

O(n!): Factorial 阶乘

注意: 只看最高复杂度的运算，不考虑系数

应用

两数求和 暴力 for循环是 O(n)，求和公式 sum = n(n+1)/2 是 O(1)

递归: 算递归了多少次，画出递归树

斐波那契可以用循环写, 如果用递归的话, 使用缓存来保存中间结果, 避免指数级复杂度(递归里面太多重复计算)

主定理: 用于算出递归执行次数

以下是四种常用递归复杂度

Application to common algorithms [edit]

Algorithm	Recurrence relationship	Run time	Comment
Binary search	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$	Apply Master theorem case $c = \log_b a$, where $a = 1, b = 2, c = 0, k = 0$[5]
Binary tree traversal	$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$	$O(n)$	Apply Master theorem case $c < \log_b a$ where $a = 2, b = 2, c = 0$[5]
Optimal sorted matrix search	$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$	$O(n)$	Apply the Akra–Bazzi theorem for $p = 1$ and $g(u) = \log(u)$ to get $\Theta(2n - \log n)$
Merge sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$	Apply Master theorem case $c = \log_b a$, where $a = 2, b = 2, c = 1, k = 0$

二分查找每次缩小查找范围二分之一

归并排序是 O(nlogn)

图的遍历 O(n), 访问且仅访问结点一次

二叉树遍历-前序、中序、后序时间复杂度O(n): 每个节点都会访问一次, 且仅访问一次, O(n), n线性于这棵树的结点总数

搜索算法: DFS、BFS时间复杂度、访问且仅访问结点一次
O(n), n指的是搜索空间里面的结点总数

已排序的二维查找