读程序 判定→ 时间复杂度

时间复杂度 表达

Big O notation

O(1) : Constant Complexity 常数复杂度 只要是常数 O(1), 2, 3, 4…) 都是 O(1)

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

$O(n^3)$: N cubic Complexity 立方

$O(2^n)$: Exponential Growth 指数

O(n!) : Factional 阶乘

只看最高复杂度的运算

根据 n 的情况运行了多少次

O(1)

```
int n = 1000;
System.out.println("Hey - your input is: " + n);
```

O(?)
O(1)

```
int n = 1000;
System.out.println("Hey - your input is: " + n);
System.out.println("Hmm.. I'm doing more stuff with: " + n);
System.out.println("And more: " + n);
```

O(N)

```
for (int i = 1; i <= n; i++) {
    System.out.println("Hey - I'm busy looking at: " + i);
}
```

O(N^2)
$O(N^2)$

```
for (int i = 1; i <= n; i++) {
  for (int j = 1; j <=n; j++) {
    System.out.println("Hey - I'm busy looking at: " + i + " and " + j);
  }
}
```

如果是并列执行，
还是 O(N) 复杂度

如果是嵌套执行，
则可能是 $O(n^2)$
$O(n^3)$ …

O(log(n))

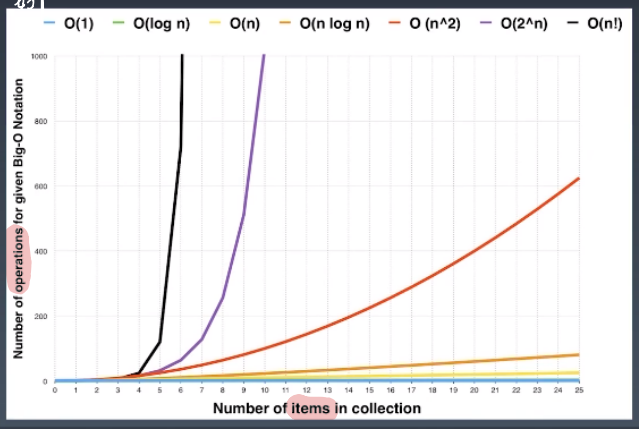
```
for (int i = 1; i < n; i = i * 2) {
    System.out.println("Hey - I'm busy looking at: " + i);
}
```

O(k^n)

O(k^n)

```
int fib(int n) {
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}
```

时间复杂度曲线



计算：1 + 2 + 3 + ... + n

- 方法一：从1到n的循环累加

```
y = 0
for i = 1 to n:
    y += i
```

O(n)

- 方法二：求和公式 sum = n(n+1)/2

```
y = n * (n + 1) / 2
```

O(1)
∵只执行一次

因为系统化的思考题目的方式　　切题四件套

- Clarification：确保自己对这个题目的理解是正确的
- Possible Solutions：想所有可能的解法来解这个题目
  - compare (time / space)　比较不同的想法，它们的时间和空间复杂度
  - optimal (加强)
- Coding (写)
- Test cases：测试样例，列举几个

更复杂的情况：递归

Solution：递归状态的递归树 / 状态树

example：

Fib: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

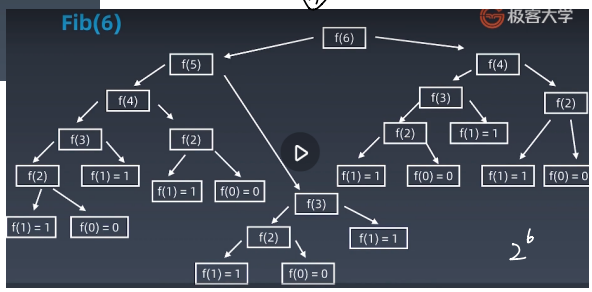
- F(n) = F(n - 1) + F(n - 2)
- 面试 （直接用递归）

```
int fib(int n) {
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}
```

假设计算 F(6)

∵ n=6　　n＞2

∴ 不return n



现象：① 每次展开一层运行的节点数

就是上面一层的两倍

② 有重复节点

加一个缓存 / 用一个循环

Master Theorem

主定理：用来解决所有递归的函数，怎么计算时间复杂度

Binary search：一般发生在数列本身有序，在有序的数列里找到你要的目标数

Binary Tree traversal（二叉树）：会每个节点都访问一次，且仅访问一次

Application to common algorithms [edit]

Algorithm	Recurrence relationship	Run time	Comment
Binary search 二分查找	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$	Apply Master theorem case $c = \log_b a$, where $a = 1, b = 2, c = 0, k = 0$[5]
Binary tree traversal 二叉树	$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$	$O(n)$	Apply Master theorem case $c < \log_b a$ where $a = 2, b = 2, c = 0$[5]
Optimal sorted matrix search 排好序的二维矩阵	$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$	$O(n)$	Apply the Akra–Bazzi theorem for $p = 1$ and $g(u) = \log(u)$ to get $\Theta(2n - \log n)$
Merge sort 归并排序	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$	Apply Master theorem case $c = \log_b a$, where $a = 2, b = 2, c = 1, k = 0$

思考题

二叉树遍历 - 前序、中序、后序：时间复杂度是多少？

O(n)

↳ 二叉树里面树的节点总数

为什么 O(n)：根据定理得到

or

二叉树每个节点会访问一次且仅访问一次

小、时间复杂度 线性于 二叉树的节点总数

二叉树遍历 - 前序、中序、后序：时间复杂度是多少？ O(n)

图的遍历：时间复杂度是多少？ O(n)

搜索算法：DFS、BFS 时间复杂度是多少？ O(n)

二分查找：时间复杂度是多少？ O(logn)

空间复杂度

小、数组的长度

2、递归的深度 (特殊情况)