2 - 1 时间复杂度和空间复杂度分析

Big O notation

O(1) : Constant Complexity 常数复杂度

O(log n) : 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!) : Factorial 阶乘

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);
}
```

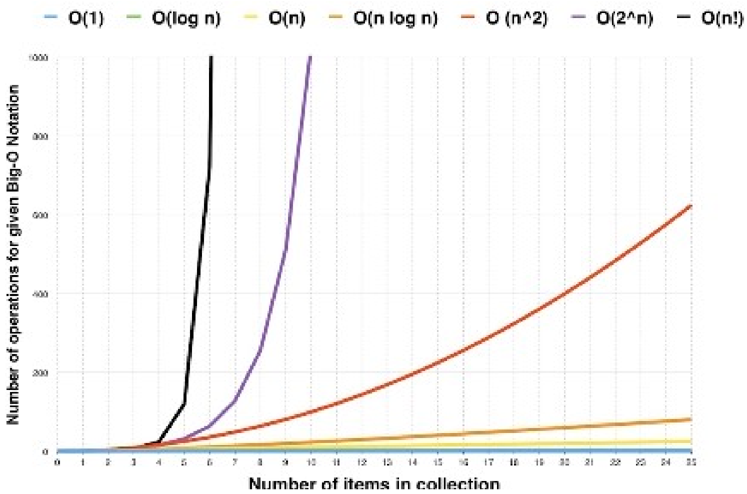
O(N^2)

```
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);
   }
}
```

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

时间复杂度曲线图



Note：

> 1. 对自己所写程序下意识地分析时间和空间复杂度；
> 2. 用最简洁的时间复杂度和空间复杂度完成代码。

Example：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)

递归：更复杂的情况分析

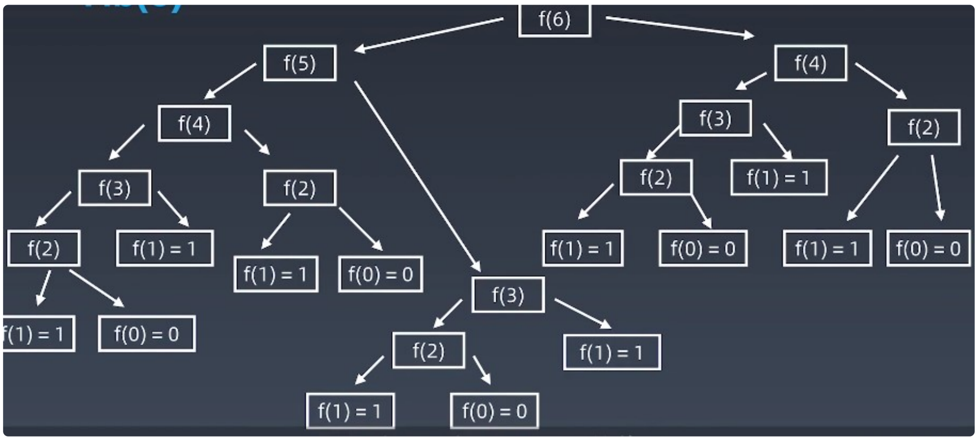
Example :

- Fib : 0,1,1,2,3,5,8,13,21,...
- F(n) = F(n-1) + F(n-2)

面试（直接递归）：

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

- f(6) :
  - 每一层节点数，即执行次数是按指数级递增的 ： 2^n ;
  - 有重复的节点出现在状态树中，所以时间复杂度比较高，面试时不建议使用，可以用 循环 代替。



Master Theorem 主定律

解决所有递归函数如何计算它的时间复杂度。

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 $a = 2, b = 2, c = 1, k = 1$

空间复杂度

1. 数组的长度
2. 递归的深度