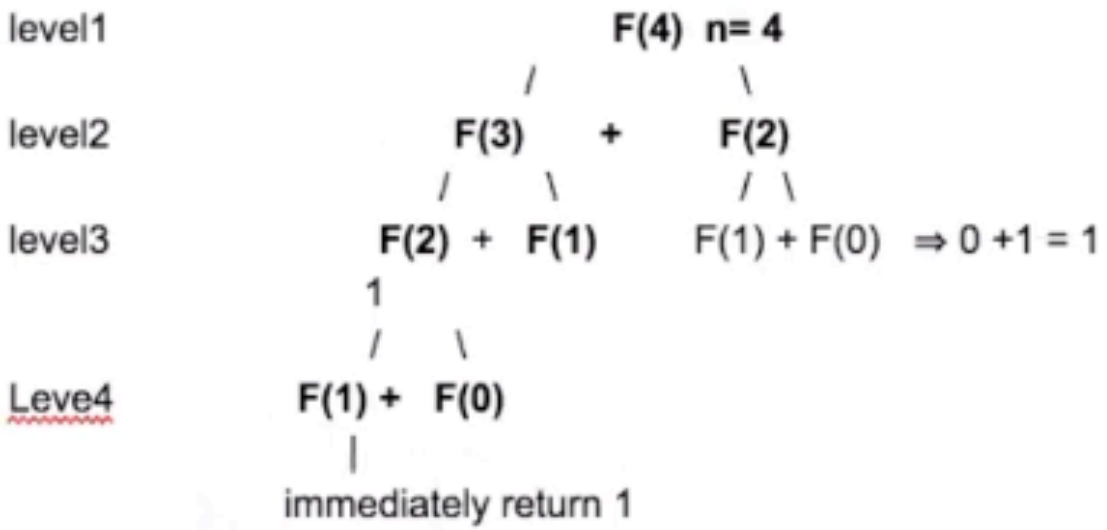
Class 2 Recursion I and Binary Search

Recursion 需要掌握的知识点：

1) 表象上：function calls itself
2) 实质上：Boil down a big problem to smaller ones (size n depends on size n-1, or n-2 or ...n/2)
3) **Implementation**上：
    a) **1. Base case:** smallest problem to solve
    b) **2. Recursive rule.** how to make the problem smaller (if we can resolve the same problem but with a smaller size, then what is left to do for the current problem size n)
4) will be introduced...

Call Stack (Terminology): was designed to record all **local variables** that are allocated in the stack.

content: Level1: n = 4
Level2: n = 3
Level3: n = 2
Level4: n = 1



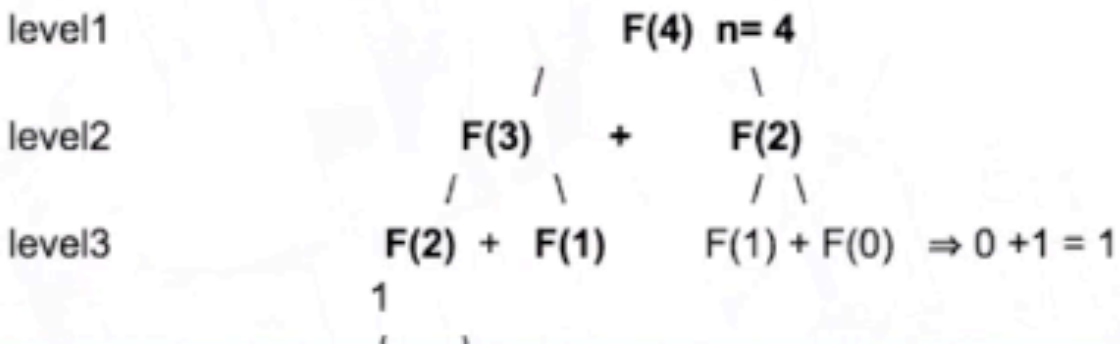
if the

```
// Calculating Fibonacci value          n = 4
00 int fibo (int n) {
01   // Base case. (进入function之后首先check是否要停下)
02   if (n == 0) {
03     return 0;
04   } else if (n == 1) {
05     return 1;
06   }
07   return fibo(n-1) break point...+ fibo(n-2);   // Recursive rule
08}
```

Generally, how to analyze a time complexity of the function???

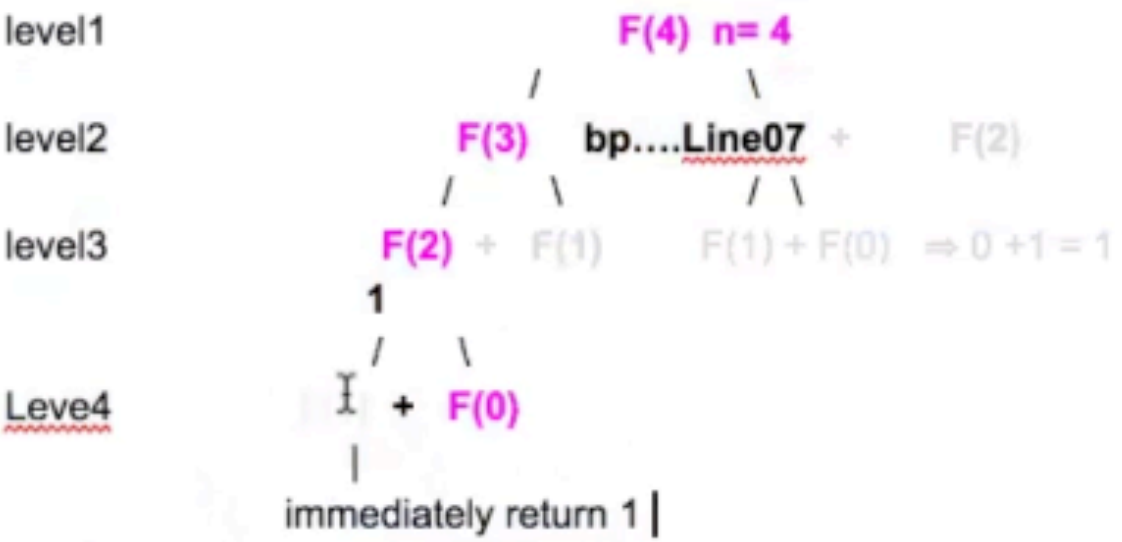
Call Stack (Terminology): was designed to record all **local variables** that are allocated in the stack.

content: Level1: n = 4



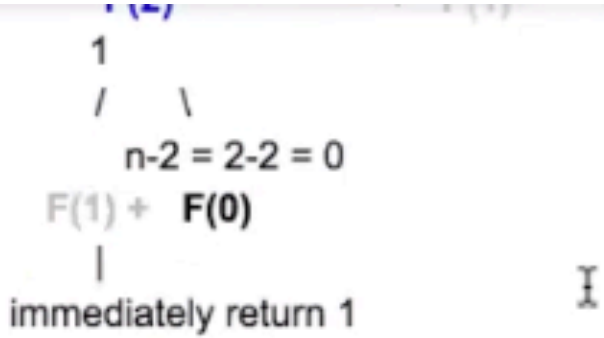
Call Stack (Terminology): was designed to record all **local variables** that are allocated in the stack.

content: Level1: n = 4



if input = n, how many levels are there in the recursion tree????

Time complexity = 1+ 2^1 +2^2 +2^3 .... + 2^n => 2^n



There are n levels in the recursion tree, and this recursion tree is a binary tree. Thus, there are totally at most O(2^n) nodes in the tree.
Time = O(2^n)

Trick: 所有前面的node的个数的总和，都没有最后一层node的个数多，因此我们分析tree的time complexity，往往只看最后一层node的个数。
1 + 2^1 + 2^2 + 2^3 + 2^4 .... + 2^n-1 = 2^n

(Da Xing): **Space = O(n)** because there are n levels of recursion function call, and thus there are n levels of call_stack. In call_stack, each level only stores 1 local variable, that is, int n.

Classical Problem 2:
Example question: how to calculate a^b (where a is an integer and b is also an integer, we do not care about the corner case where a = 0 or b < 0 for now)
2^1000
a = 2

size = n

xxxxxxxxxxxxxxx.....xxxxxTargetxxx x| null null null null null .... null

___

_____

_____

_____|

...

_____ i-1 times i-th time

_____| (< 2n)

x 10 times

_____...___(<10n)

	Jump Out	Jump In
2 times	O(log_2(n))	O(log_2(2n))
10 times	O(log_10(n))	O(log_2(10n))

Further discussion about binary search:

Why not jump_step = jump_step * 10, instead of jump_step = jump_step *2

```
10Times:  log_10 (n) + log_2(10n)
2 Times:  log_2(n)  + log_2(2n)
```

10 times - 2times <

$$(\log_{10}(n) + \log_2(10n)) - (\log_2(n) + \log_2(2n)) < 0$$

(1) (2) (3) (4)

$$\log_{10}(n) - \log_2(n) + (\log_2(10n) - \log_2(2n))$$

(1) (3) (2) (4)

$$-10000$$

$$1-3$$

$$\log_2(5) \sim 2.5$$

$$2-4$$

Conclusion: So jump 10 steps is faster than 2 steps!