Fibonacci Sequence

If you have even a passing interest in mathematics, you've heard of the Fibonacci sequence.

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

0th 1st 2nd 3rd 4th 5th 6th 7th 8th

where:

$$f_0 = 0$$
$$f_1 = 1$$
$$f_2 = f_1 + f_0 = 2$$
$$f_3 = f_2 + f_1 = 3$$
$$\vdots$$

etc.

or $\boxed{f_n = f_{n-1} + f_{n-2}}$

The nature of this sequence makes it a great candidate for recursion.

```
def fibonacci(n):
    if (n==0):
        return 0
    elif (n==1):
        return 1
    else:
        return fibonacci(n-1)
               + fibonacci(n-2)
```

Test:

n	fibonacci(n)
0	0
1	1
2	fibonacci(1) +

fibonacci(0)

$$\downarrow \qquad \downarrow$$

$$1 + \phi$$

$$= 1$$

3

fibonacci (2) + fibonacci (1)

$$\downarrow \qquad \downarrow$$

$$1 + 1$$

$$= 2$$

→ seems like it should work :)

→ and it does!! see fibonacci.py

We are so clever. Recursion is so cool. We are f*&!*ng smart.

Or, maybe we're not.

What is the computational complexity of this algorithm???

n	# of calls
ϕ	1
1	1
2	3 $(= 1+1+1)$
3	$3+1 = 4$
4	$4+3 = 7$
5	$7+4 = 11$
6	$11+7 = 18$
7	$18+11 = 29$
8	$29+18 = 47$
9	$47+29 = 76$

I'm not sure .... let's write a python program to calculate the # of calls, and plot it, to see what it looks like....

See fibonacci_complexity.py.

Result: for $n = 50$,

calls ~ 800,000 !!!

in general, # calls ~ e^n

$\therefore$ complexity = $O(e^n)$

It is difficult for me to emphasize how horribly bad this is.

Is there anything we can do?

Please, Dr. Brosh, save us!!!

Okay, let's calm down....

Whenever we have an algorithm that is $O(e^n)$, the thought is that we probably could find something that is $O(n)$ for example.

better ... like, O(,)

Non-recursive Fibonacci :

```
def fibonacci(n) :
    if n == 0 :
        return Φ
    elif n == 1 :
        return 1

    else:
        t1 = Φ
        t2 = 1
        for i in range(2, n+1):      # one more than n!
            t3 = t2 + t1             # ← sum of two terms
            t1 = t2                  # } update
            t2 = t3                  #   terms
        return t3
```

→ This works beautifully, and is O(n)

→ Conclusion: be very careful using recursive functions !

But wait! There's more. ☺

As you may know, the ratio of successive terms in the Fibonacci sequence is interesting, as well.

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ \ldots$$

r = 1   r = 2   r = 1.5   r = 1.666   r = 1.6   r = 1.625   r = 1.6154

ratio seems to be converging ... $\boxed{1.62\ldots}$

golden ratio $= \phi$

1.61803988...

Let's use some basic maths:

$$f_n = f_{n-1} + f_{n-2}$$

$$\underbrace{\frac{f_n}{f_{n-1}}}_{=\phi} = \underbrace{\frac{f_{n-1}}{f_{n-1}}}_{=1} + \underbrace{\frac{f_{n-2}}{f_{n-1}}}_{\frac{1}{\phi}}$$

$$\therefore \quad \phi = 1 + \frac{1}{\phi}$$

$$\phi^2 = \phi + 1$$

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Quadratic in ϕ → $$\phi^2 - \phi - 1 = 0$$

$$\phi = \frac{1 \pm \sqrt{1 - 4(1)(-1)}}{2}$$

$$\boxed{\phi = \frac{1 + \sqrt{5}}{2}}$$ (= 1.618033...)

Okay, so not so magical, perhaps. ☺

After a bit more math (Binet 1843), we find that:

$$\boxed{f_n = \frac{(\phi)^n - \left(-\frac{1}{\phi}\right)^n}{\sqrt{5}}}$$

New fibonacci function.

```
import math
phi = 1.0 + math.sqrt(5.0)
```

$$phi \quad \overline{\qquad\qquad} \quad 2.0$$

```
def fibonacci(n)
    global phi
    return int((phi**n - (-1.0/phi)**n)
               / math.sqrt(5.0))
```

Let's check:

n	$f(n)$ formula	$f(n)$ actual
0	0	0
6	8	8
20	6765	6765
40	102334155	102334155
60	1548008755920	1548008755920
75	2111485077978055	2111485077978050

oops!

Why? It's coming from the fact that our calculation of $\phi = \frac{1+\sqrt{5}}{2}$

is being ~~Truncated~~ at worst it

decimal places. This is the maximum accuracy that one can achieve for storing floating point #'s using 64-bit binary floating point representation!!

So, what to do? ....

One of the really useful modules in python is

$\boxed{\text{decimal}}$

```
import decimal as dec
```

The decimal module allows us to specify the number of decimal places, up to relatively arbitrary precision!

dec.getcontext().prec = 50

onehalf = dec.Decimal(0.5)

This will actually store

0. 5 0 0 0 0 0 0 0 0 . . . . 0 0 0

1 2 3 4 5 6 7 8 9 . . . . 48 49 50

one = dec.Decimal(1)
two = dec.Decimal(2)
five = dec.Decimal(5)

phi = (one + five ** onehalf)
/ two

def fibonacci(n):
global phi
global one, two, five, onehalf

return round((phi ** n
- (- one/phi) ** n) /
five ** onehalf)

See fibonacci_formula.py

Summary:

1. recursive Fibonnacci $\rightarrow O(e^n)$
2. non-recursive Fibonacci $\rightarrow O(n)$
3. formula Fibonnacci* $\rightarrow O(1)$

→ requires higher level floating pt. precision library.

→ so, might be slower !!