Karen Eunice Oviedo Garza

Numero primo y Fibonacci

Un número primo es un número entero mayor que cero, que tiene exactamente dos divisores positivos, el mismo número y el 1.

Cómo averiguar si un número es primo.

El algoritmo más sencillo que puede utilizarse para saber si un número n es primo es el de la división. Se trata de ir probando para ver si tiene algún divisor propio. Para ello vamos dividiendo el número n entre 2, 3, 4, 5, ... , n-1. Si alguna de las divisiones es exacta (da resto cero) podemos asegurar que el número n es compuesto. Si ninguna de estas divisiones es exacta, el número n es primo. Este método puede hacerse más eficiente observando simplemente, que si un número es compuesto alguno de sus factores (sin contar el 1) debe ser menor o igual que √ n. Por lo tanto, el número de divisiones a realizar es mucho menor. Sólo hay que dividir entre 2, 3, 4, 5, ... , [√ n]. En realidad, bastaría hacer las divisiones entre los números primos menores o iguales que √ n.

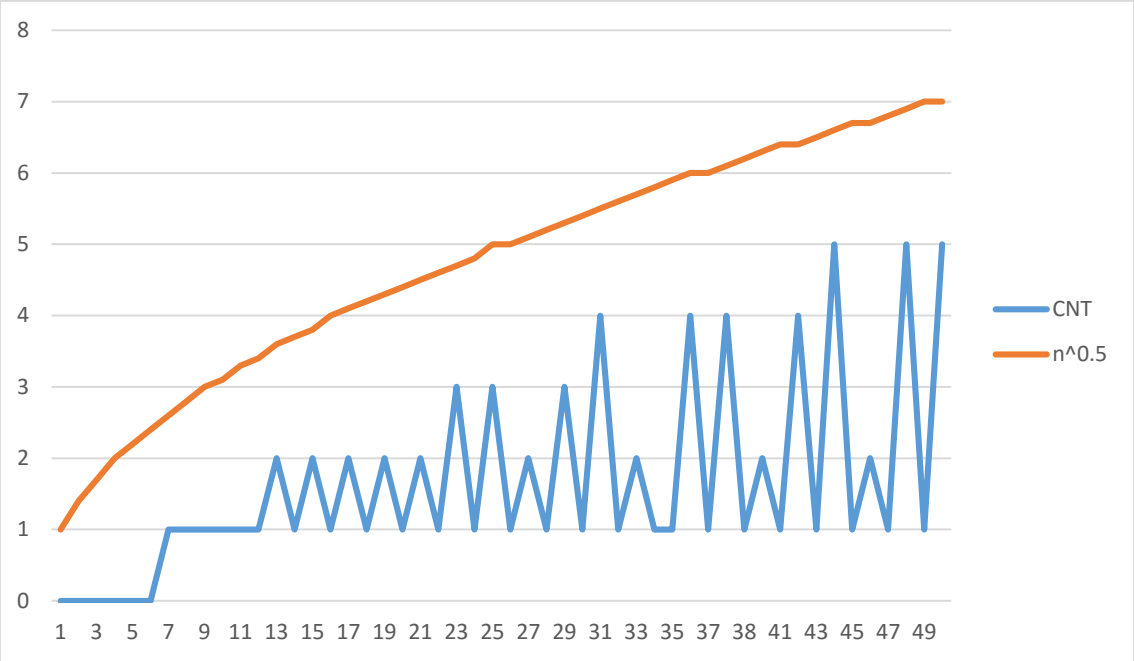
```
>>> def primo(n):
        for i in range(2,round(n**(0.5))):
                if(n%i)==0:
                        return("no es primo")
                return("Es primo")


>>> primo(7)
'Es primo'
>>> primo(17)
'Es primo'
>>> primo(8)
'no es primo'
```

Ahora graficaremos el número de operaciones que hace el código para saber si un número es primo o no.

```
>>> def primo(n):
        cnt=0
        for i in range(2,round(n**(0.5))):
                cnt=cnt+1
                if((n%i)==0):
                        break
        return cnt

>>> for i in range(1,50):
        print(primo(i))
```



Tiene una complejidad de n^0.5

Fibonacci

La serie de Fibonacci es una sucesión infinita de números naturales: 0, 1, 1, 2, 3, 5, 8, 13, 21,… Los dos primeros números de la sucesión son 0 y 1. Los otros términos son la suma de los 2 términos anteriores en la sucesión: 0+1=1, 1+1=2, 1+2=3, 2+3=5, 3+5=8, 5+8=13…

Esta sucesión se la debemos a Leonardo de Pisa, también conocido como Fibonacci, matemático Italiano del siglo XIII. La sucesión descubierta por este matemático la podemos encontrar en la computación, matemáticas, juegos, en el arreglo de un cono, en la música

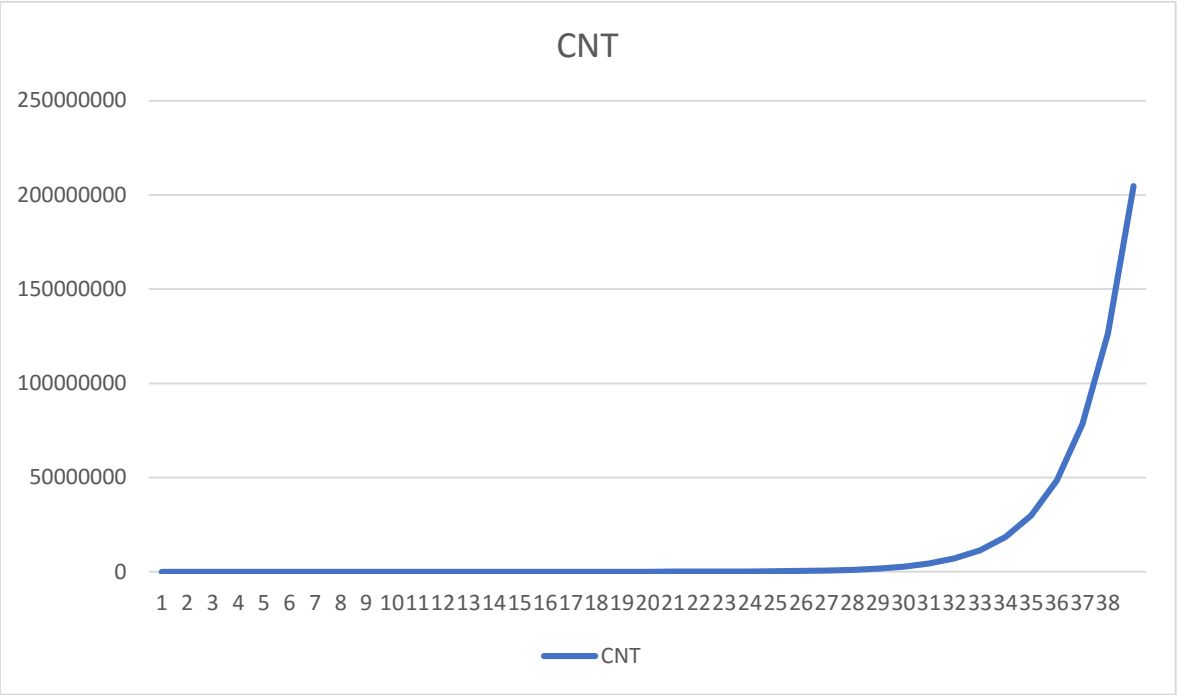
Fibonacci recursivo

```
>>> cnt=0
>>> def fibonacci(n):
        global cnt
        cnt=cnt+1
        if n==0 or n==1:
                return(1)
        return fibonacci(n-2) + fibonacci(n-1)

>>> for i in range(1,50):
        cnt=0
        fibonacci(i)
        print(i,",",cnt)
```

```
1 , 1
2
2 , 3
3
3 , 5
5
4 , 9
8
5 , 15
13
6 , 25
21
7 , 41
34
8 , 67
55
9 , 109
89
10 , 177
144
11 , 287
233
12 , 465
377
13 , 753
610
14 , 1219
987
15 , 1973
1597
16 , 3193
2584
17 , 5167
4181
18 , 8361
6765
19 , 13529
10946
20 , 21891
```

Ahora graficaremos el número de operaciones que hace el código.

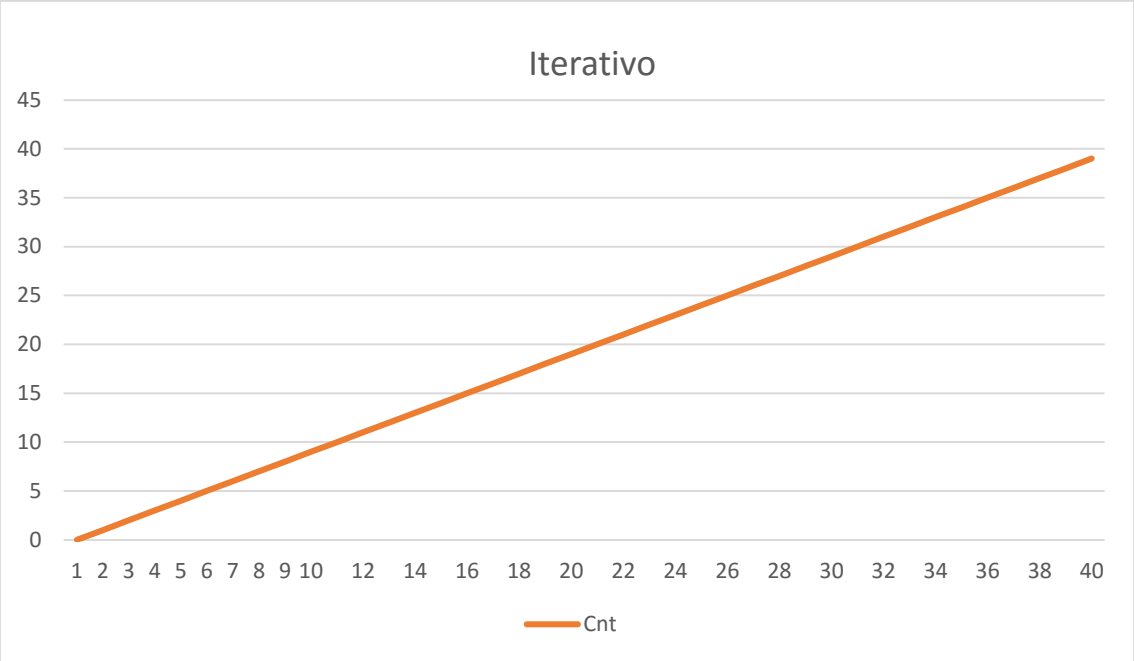


Fibonacci iterativo

```
>>> def fibonacci(n):
        cnt=0
        if n==0 or n==1:
                return (1)
        r,r1,r2=0,1,1
        for i in range(2,n):
                cnt=cnt+1
                r=r1+r2
                r2=r1
                r1=r
        return r,cnt

>>> for i in range(1,50):
        cnt=0
        fibonacci(i)
        print(i,",",cnt)
```

Karen Eunice Oviedo Garza



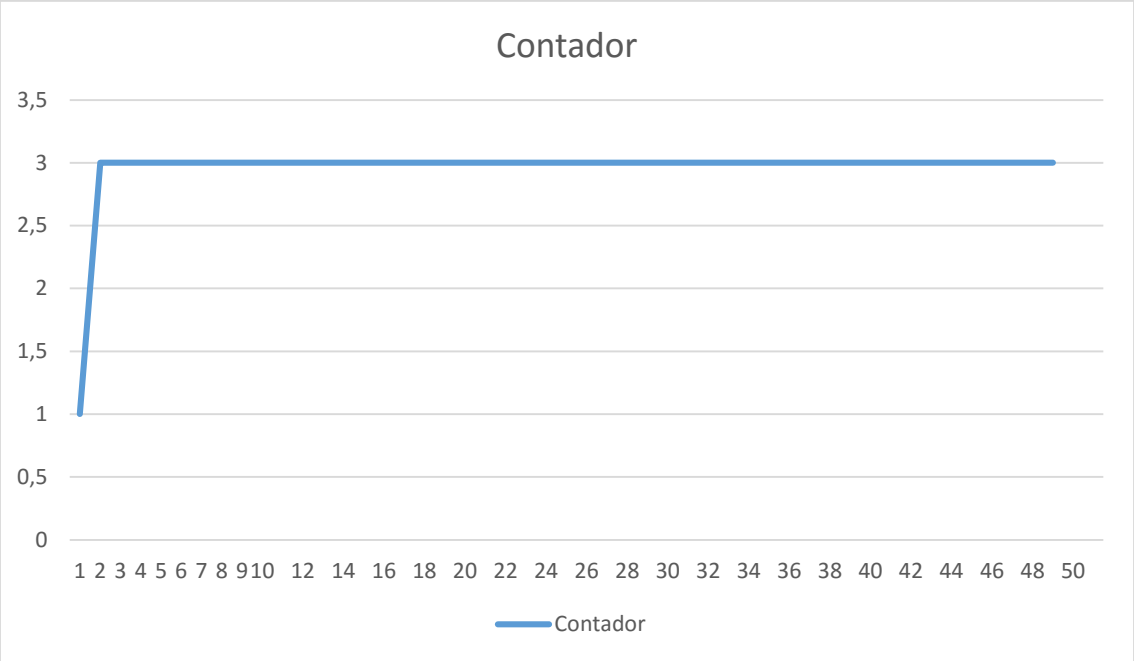
Fibonacci con memoria

```
>>> memo={}
>>> cnt=0
>>> def fibonacci(n):
        global memo,cnt
        cnt=cnt+1
        if n==0 or n==1:
                return (1)
        if n in memo:
                return memo[n]
        else:
                val= fibonacci(n-2)+fibonacci(n-1)
        memo[n]=val
        return val

>>> for i in range(1,50):
        cnt=0
        fibonacci(i)
        print(i,",",cnt)
```

```
1
1 , 1
2
2 , 3
3
3 , 3
5
4 , 3
8
5 , 3
13
6 , 3
21
7 , 3
34
8 , 3
55
9 , 3
89
10 , 3
144
11 , 3
233
12 , 3
377
13 , 3
610
14 , 3
987
15 , 3
1597
16 , 3
2584
17 , 3
4181
18 , 3
6765
19 , 3
10946
```

Karen Eunice Oviedo Garza



En conclusión pienso que el más eficiente ya que es el que hace menos operaciones es Fibonacci con memoria, hace el mismo número de operaciones porque los números ya se encuentran guardados en la memoria.