Fibonacci sequence in Python

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

In [1]:
```
# Recursive implementation of Fibonacci sequence - O(2^n)
# Very bad performance
def fibonacci_recursive(n):
    if n == 0: return 0
    elif n == 1: return 1
    else: return fibonacci_recursive(n-1) + fibonacci_recursive(n-2
)
```

In [2]:
```
# Iterative implementation of Fibonacci sequence - O(n)
def fibonacci_iterative(n):
    a, b = 0, 1
    for i in range(0, n):
        a, b = b, a + b
    return a
```

In [3]:
```
# Another version of iterative implementation of Fibonacci sequence
# It is a bit slower and takes much more memory (keep full sequence
)
def fibonacci_iterative_v2(n):
    fib = [0, 1, 1]
    for f in range(2, n):
        fib.append(fib[-1] + fib[-2])
    return fib[n]
```

In [4]:
```
# Arithmetic implementation of Fibonacci sequence - O(log n)
def fibonacci_arithmentic(n):
    return pow(2 << n, n + 1, (4 << 2*n) - (2 << n) - 1) % (2 << n)
```

In [5]:
```
# compare results
for i in range(0,18):
    print("# {:>3}  |  ".format(i+1)+
          "Rec.: {:>4}\tIter.: {:>4}\tIter2.: {:>4}\tArit.: {:>4}"\
          .format(fibonacci_recursive(i),
                  fibonacci_iterative(i),
                  fibonacci_iterative_v2(i),
                  fibonacci_arithmentic(i)))
```

```
#   1  |  Rec.:    0    Iter.:    0     Iter2.:    0    Arit.:
0
#   2  |  Rec.:    1    Iter.:    1     Iter2.:    1    Arit.:
1
#   3  |  Rec.:    1    Iter.:    1     Iter2.:    1    Arit.:
1
#   4  |  Rec.:    2    Iter.:    2     Iter2.:    2    Arit.:
2
#   5  |  Rec.:    3    Iter.:    3     Iter2.:    3    Arit.:
3
#   6  |  Rec.:    5    Iter.:    5     Iter2.:    5    Arit.:
5
#   7  |  Rec.:    8    Iter.:    8     Iter2.:    8    Arit.:
8
#   8  |  Rec.:   13    Iter.:   13     Iter2.:   13    Arit.:   1
3
#   9  |  Rec.:   21    Iter.:   21     Iter2.:   21    Arit.:   2
1
#  10  |  Rec.:   34    Iter.:   34     Iter2.:   34    Arit.:   3
4
#  11  |  Rec.:   55    Iter.:   55     Iter2.:   55    Arit.:   5
5
#  12  |  Rec.:   89    Iter.:   89     Iter2.:   89    Arit.:   8
9
#  13  |  Rec.:  144    Iter.:  144     Iter2.:  144    Arit.:  14
4
#  14  |  Rec.:  233    Iter.:  233     Iter2.:  233    Arit.:  23
3
#  15  |  Rec.:  377    Iter.:  377     Iter2.:  377    Arit.:  37
7
#  16  |  Rec.:  610    Iter.:  610     Iter2.:  610    Arit.:  61
0
#  17  |  Rec.:  987    Iter.:  987     Iter2.:  987    Arit.:  98
7
#  18  |  Rec.: 1597    Iter.: 1597     Iter2.: 1597    Arit.: 159
7
```

Compare performance

In [6]:
```
%timeit fibonacci_recursive(20)
```

```
3.46 ms ± 202 µs per loop (mean ± std. dev. of 7 runs, 100 loops e
ach)
```

In [7]:
```
%timeit fibonacci_iterative(20)
```

```
1.45 µs ± 47.4 ns per loop (mean ± std. dev. of 7 runs, 1000000 lo
ops each)
```

In [8]:
```
%timeit fibonacci_iterative_v2(20)
```

```
3.14 µs ± 84.6 ns per loop (mean ± std. dev. of 7 runs, 100000 loo
ps each)
```

In [9]:
```
%timeit fibonacci_arithmentic(20)
```

```
1.96 µs ± 10.5 ns per loop (mean ± std. dev. of 7 runs, 1000000 lo
ops each)
```

Other excercises

Show first N numbers greater than X

In [10]:
```
# based on the iterative implementation
def fib_n_bigger_than_x(n=10, x=1000):
    a, b = 0, 1
    res = []
    while(len(res)<n):
        a, b = b, a + b
        if(a>x): res.append(a)
    return res

fib_n_bigger_than_x()
```

Out[10]:
```
[1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393
]
```

Sum of the first N numbers bigger than X

In [11]:
```
# based on the iterative implementation
def sum_n_bigger_than_x(n=10, x=1000):
    a, b = 0, 1
    res = []
    while(len(res)<n):
        a, b = b, a + b
        if(a>x): res.append(a)
    return sum(res)

sum_n_bigger_than_x(n=2, x=10)
```

Out[11]: 34