**473: Notes on Day 6 slides:**

Slide 3 Asymptotic Analysis Example:

C < 1. The infinite series converges to 1/(1-c), so Theta(1)

C = 1 f(n) = n, so it is Theta(n)

C > 1 f(n) = (c^(n+1) – 1) / (c – 1). The limit of f(n) divided by c^n as n→infinity is c/(c-1),

so f(n) is Theta(c^n).

Slide 6: A Creative O(log N) Algorithm?

The file **fib3.py** has the solution, and **fib3-incomplete.py** has it partially completed.

```
# Code for matrix_power:
def matrix_power(m, n):
   result = identity_matrix
   power = x
   while n > 0:
      if n % 2 == 1:
         result = matrix_multiply(result, power)
      power = matrix_multiply(power, power)
#     print "In loop:", n, power, result
      n = n /2
   return result
```

Slide 8: Reconsider our Fibonacci algorithms

<See the disclaimer on slide 7>

Note on recursive: The length of F(n) is .694n, which is O(n).

Note on last one: The # of bits in the numbers in the matrix at most doubles with each matrix multiplication.

Thus we get AN UPPER BOUND OF M(1) + M(2) + M(4) + M(8) + … + M(F(n))

If a = 2, we get (Maple notation):

> **sum((2^i)^2), i=0..log[2](n));**

> **simplify(sum((2^i)^log[2](3), i=0..log[2](n)));**

Slide 10: Algorithm for Integer Division

```
divide(19, 4):
  q, r = divide(9, 4)
     q, r = divide (4, 4)
       q, r = divide(2, 4)
            q, r = divide(1, 4)
              q, r = divide(0, 4) = 0, 0
              x is odd, so
              r = 1
              return 0, 1
            q, r, = 0, 2
            return 0, 2
       q, r = 0, 4
       q, r = 1,  0
       return 1, 0
     q, r, = 2, 0
     x is odd, so
     q, r = 2, 1
     return 2, 1
  q, r = 4, 2
  x is odd, so
  q, r, = 4, 3
  return 4, 3
```

Analysis: If n is max number of bits in x, y, each recursive vcall is O(n), and at most n calls, so n^2.

Slide 14: Modular Addition and Multiplication

0 to N – 1, 0 to 2(N-1), Theta(n).
0 to $(N-1)^2$ , 2n Theta($n^2$)

Slide 15: Modular Exponentiation

To do better, we can use an algorithm like our previous recursive exponentiation algorithm

Slide 16: Modular Exponentiation Algorithm

Answers: n, n2, n3