Notes on Day 2 slides:

Slide 10: Analysis of the recursive algorithm

Where does the 3 come from? Two comparisons plus a sum.

Conclusion: T(N) >= F(N) Q4

F(N) = (1 / sqrt(5)) (((1 + sqrt(5))/2)^N - (1 - sqrt(5))/2)^N)

(1 + sqrt(5))/2 ≈ 1.618, so it is exponential. 1.618^N = 2^(0.694*N)

How long would it take to compute F(200)? T(200) >= F(200) >= 2^138.

Last question 2^(138-18) = 2^120 seconds.

How many seconds in a year? (less than 2^22).

So time for F(200) is at least 2^98 years!

What is this document? I have simply copied the instructor notes from the slides that have them, and pasted them in here.

These are some of the things I would say and/or write on the board during class time

Slide 11: A Polynomial-time algorithm

This algorithm is clearly linear.

Or is it?

Addition is NOT a constant-time operation unless there is a limit on the size of the numbers involved.

Slide 12: A more efficient Algorithm

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

Solution 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 12: Why so complicated?

Because we'd have to use decimal approximation of the square roots, and that would be very messy. More so than the matrix version.

Slide 17: Multiplication: The running time of the algorithm is O(n^2)