I'm Back!

Reminders:

HW3 due today
HW4 posted tonight (due Mar 22)
Midterm 2 in 2 weeks (Mar 23/24)

Office hrs this week:
Friday in BA5287 (Noah)
Thurs CANCELLED

See Handouts from Week 7 on 0, π, θ
Example 1  Simple Nested Loop

```python
def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print (i+j)
            j = j+2
        i = i+1
```

When \( i = 0 \)
inner loop executes \( \frac{n^2}{2} \) times

We will estimate the number of times
the inner loop block * executes
Work from inside out
Argue the # of times * executes is
\( \Theta \left( \text{total # of steps in algorithm} \right) \)
Example 1: Simple Nested Loop

```python
def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print(i + j)
            j = j + 2
        i = i + 1

When i = 0

(*) executes \( \lceil \frac{n}{2} \rceil \) times

\( c = 1 \)

(*) executes \( \lceil \frac{n}{2} \rceil \) times

\( i = n - 1 \)

(*) executes \( \lceil \frac{n}{2} \rceil \) time

then (*) doesn't execute anymore

So total # of executions of (*)

is \( n \cdot \lceil \frac{n}{2} \rceil \)
Example 1: Simple Nested Loop

def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print(i+j)
            j = j + 2
        i = i + 1

Summary:
1. Every time outer loop executes, inner loop executes \( \frac{n}{2} \) time
   (Hence executions of inner loop don't depend on \( i \))
2. So overall \# executions of inner loop = \((\#\text{executions of inner loop})\cdot (\#\text{executions of outer loop})\)
Example 1  Simple Nested Loop

```python
def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print(i+j)
            j = j + 2
        i = i + 1
```

$\Theta(n^2)$

We could have done a perfect analysis and gotten an exact expression for steps, but it would still be $\Theta(n^2)$.
So now we want to give a theta expression

\[ n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor = \Theta(n^2) \]

To prove \( n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor = \Theta(n^2) \), need to show

1. \( n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \in O(n^2) \)

   \[ \exists n_0 \in \mathbb{N} \exists c \in \mathbb{R}^+ \quad (\forall n \geq n_0) \quad n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \leq c \cdot n^2 \]

   Let \( n_0 = 1 \), \( c_0 = 1 \). Check \( \forall n \in \mathbb{N} \) \( (n \geq 1) \quad n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \leq n \cdot n = n^2 \)

2. \( n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \in \Omega(n^2) \)

   Show \( \exists n_0 \in \mathbb{N} \exists c \in \mathbb{R}^+ \forall n = n_0 \quad n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \geq c \cdot n^2 \)

   Pick \( c = \frac{1}{2} \), \( n_0 = 1 \)

   \[ n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \geq n \cdot \frac{n}{2} = \frac{1}{2} \cdot n^2 \]
Example 2 Nested loop where loop cost changes

```python
def Nested2(n):
    i = 0
    while i < n:
        j = 0
        while j < i:
            print((i + j))
            j = j + 1
            print(\((*)\))
        i = i + 1
```

We will analyze in the same way, but now # q times \((*)\) executes every time outer loop executes depends on i
Example 2  Nested loop where loop cost changes

def Nested2(n):
    i=0
    while i<n:
        j=0
        while j<i:
            print (i+j)
            j = j+1
        i = i+1

When outer loop has i set to i, inner loop (*) executes i times.
So overall # of times (*) executes is
\[ 0 + 1 + 2 + 3 + \ldots + n-1 = \sum_{i=0}^{n-1} i \]
\[ i=0 \quad i=1 \quad i=2 \quad i=3 \quad i=n-1 \quad i=0 \]
Example 2  Nested Loop where Loop cost changes

def Nested2(n):
    i=0
    while i<n:
        j=0
        while j<i:
            print (i+j)
            j=j+1
        i=i+1

Want to write this in \( \Theta \) form:

\[
\sum_{i=0}^{n-1} i = \frac{n \cdot (n-1)}{2} = \Theta(n^2)
\]

\( \Theta(n^2 - n) = \Theta(n^2) \)
Example 3: Factoring – find a nontrivial factor of $n$

```python
def factor(n):  
    (n ≥ 2)
    d = 2
    while d < n
        if n % d == 0:
            return d
        d = d + 1
    return -1
```

If $n$ is even $⇒$ runtime is constant $O(1)$
other extreme $n$ is prime $⇒$ runtime is $O(n)$
in between $⇒$ runtime depends on the smallest prime divides $n$
so Runtime is $Ω(1)$, and $O(n)$
It turns out there is no elementary function \( f \) s.t. the runtime of \( \text{factor} \) is \( \Theta(g) \)

\[ Q = \] But factoring is supposed to be really hard!

the input size is \( \log n \) not \( n \)

so to factor a number with 100 digits takes time \( 2^{100} \) in worst case

\[ \uparrow \text{terrible} \]