Hi! Welcome to 61A Discussion :)

We will begin at **5:10**!

Attendance: **go.cs61a.org/ben-disc**

Slides + Info: **cs61a.bencuan.me**

Announcements

- Hog due **tonight!**
- Midterm regrade requests due next Weds.
- General questions/concerns? Join advising OH or email me!

Agenda

- Attendance
- Midterm debrief
- Recursion
  - Recursion
    - Recursion
      - Recursion
        - Recursion
          - Recursion

About the midterm

General thoughts

- Congrats on finishing a really tough exam!
- Everyone learns CS at their own pace, **don’t compare yourself to others!** enjoy the process :)
- Low exam averages can be expected in most CS classes! It’s totally OK and expected to get a low score

61A by the numbers

		A	≥ 285	A-	≥ 270
B+	≥ 250	B	≥ 225	B-	≥ 205
C+	≥ 190	C	≥ 180	C-	≥ 175
D+	≥ 170	D	≥ 165	D-	≥ 160

- 300 points total, 135 of which are non-exam.
- (Points desired - 135 - 40 + MT1 score - 7 - exam recovery pts) = points needed for MT2 and Final
- Example: 15/40 on MT1, want a B+?
  - 250 - 135 - 40 + 15 - 7 - 4 = 79
    - 25/50 on MT2, 54/75 on Final = B+!

Some reassurances

- MT1 is a small part of 61A! Much more yet to come
- Many people do better in 61B by >= 1 grade bin!
- 61A or CS grades in general are not super important for life/career/research purposes
- CSM is here to help :)
- Many past students (such as myself) find the final easier than midterms since there is less time crunch

Recursion

What is this recursion thing?

Recursion is **when a function calls itself**

```
def factorial(n):
    """Return the factorial of N, a positive integer."""
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

This is recursion!

```
def print_all(x):
    print(x)
    return print_all
```

This is NOT recursion (no call)!

Ways to conceptualize recursion

- HOF's? (discussed last slide)
- Iteration? (they do the same thing)
- Induction? (for math people)
- Recursion :) (google it!)

The 3 parts of a recursive function

1. **Base Case** (what's the simplest possible input?)

2. **Recursive Case** (how do we make the problem even simpler?

3. **Recursive Leap of Faith** (assume simpler problems are solved already)

Example: Factorial

```
def factorial(n):
    """Return the factorial of N, a positive integer."""
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

Goal: compute `n! = n * (n-1) * (n-2) … * 1`

1. **Base Case:** simplest input to n?
   a. `n == 1` or `n == 0`

2. **Recursive Case:** smaller problem?
   a. n! = n * (n-1)!

3. **Leap of Faith:** what do we assume?
   a. that `factorial(n-1)` actually gives `(n-1)!`

A warning: arm's length recursion

Recursive cases should be as simple as possible!

```
n * factorial(n-1): good! :)
```

```
n * (n-1) * factorial(n-2): not good :(
```

Q1: Recursive Multiplication

Goal: do multiplication recursively by adding a bunch of times

Base Case: what's the simplest input?

Recursive Case: what does multiply(m-1, n) do? What about multiply(n, m-1)? Does it matter?

- Hint: leap of faith

Q2: Recursive ED

```
def rec(x, y):
    if y > 0:
        return x * rec(x, y - 1)
    return 1

rec(3, 2)
```

What does this function do?

How many frames are created?

Q3: Bug Finding

```
def skip_mul(n):
    """Return the product of n * (n - 2) * (n - 4) * ...

    >>> skip_mul(5) # 5 * 3 * 1
    15
    >>> skip_mul(8) # 8 * 6 * 4 * 2
    384
    """
    if n == 2:
        return 2
    else:
        return n * skip_mul(n - 2)
```

Hint: try some inputs. Which values of n break the function?

Q4: Recursive Hailstone

You've done it iteratively... now do it recursively!

Hints:

1. Are there any side effects from making the recursive call?

2. How many base cases do you need? How many recursive cases do you need?

Q5: Merge Numbers (hard mode)

Write a procedure `merge(n1, n2)` which takes numbers with digits in decreasing order and returns a single number with all of the digits of the two, in decreasing order. Any number merged with 0 will be that number (treat 0 as having no digits). Use recursion.

> Hint: If you can figure out which number has the smallest digit out of both, then we know that the resulting number will have that smallest digit, followed by the merge of the two numbers with the smallest digit removed.

```
def merge(n1, n2):
    """ Merges two numbers by digit in decreasing order
    >>> merge(31, 42)
    4321
    >>> merge(21, 0)
    21
    >>> merge (21, 31)
    3211
    """
```

Is Prime

```
if n == 1:
    return False
k = 2
while k < n:
    if n % k == 0:
        return False
    k += 1
return True
```

You've done it iteratively... now do it recursively!