61A Lecture 7
-----------------------------
Wednesday, February 4

Announcements
---------------------------------------------------
- Project 1 is due Thursday 2/5 @ 11:59pm; Early bonus point for submitting on Wednesday!
- Extra tutor office hours on Wednesday 2/4 (See Piazza for details)
- Midterm 1 is on Monday 2/9 from 7pm to 9pm!
  - Review session on Saturday 2/7
  - HKN review session on Sunday 2/8
  - Includes topics up to and including this lecture
  - Closed book/note exam, except for one page (2 sides) of hand-written notes & study guide
  - Cannot attend? Fill out the conflict form by Wednesday 2/4! http://goo.gl/2P5fKq
- Optional Hog strategy contest ends Wednesday 2/18 @ 11:59pm

Hog Contest Rules
-----------------------------------
- Up to two people submit one entry; Max of one entry per person
- Your score is the number of entries against which you win more than 50% of the time
- All strategies must be deterministic, pure functions of the current player's scores
- All winning entries will receive 2 points of extra credit
- The real prize: honor and glory

Spring 2015 Winners
----------------------
YOUR NAME COULD BE HERE... FOREVER!

Order of Recursive Calls
------------------------------
The Cascade Function
------------------------
- Each cascade frame is from a different call to cascade.
- Until the Return value appears, that call has not completed.
- Any statement can appear before or after the recursive call.

Interactive Diagram

Example: Inverse Cascade
--------------------------
Write a function that prints an inverse cascade:

```python
def inverse_cascade(n):
    grow(n)
    shrink(n)
```

Two Definitions of Cascade
----------------------------
- If two implementations are equally clear, then shorter is usually better
- In this case, the longer implementation is more clear (at least to me)
- When learning to write recursive functions, put the base cases first
- Both are recursive functions, even though only the first has typical structure

Inverse Cascade
-----------------
Write a function that prints an inverse cascade:
Tree Recursion

Tree-shaped processes arise whenever executing the body of a recursive function makes more than one recursive call.

\[
\begin{align*}
\text{fib}(n) &\quad n: 0, 1, 2, 3, 4, 5, 6, 7, 8, \ldots \quad 35 \\
\end{align*}
\]

A Tree-Recursive Process

The computational process of fib evolves into a tree structure.

Example: Counting Partitions

The number of partitions of a positive integer \( n \), using parts up to size \( m \), is the number of ways in which \( n \) can be expressed as the sum of positive integer parts up to \( m \) in increasing order.

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don’t use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.

Repetition in Tree-Recursive Computation

This process is highly repetitive; fib is called on the same argument multiple times.

Counting Partitions

The number of partitions of a positive integer \( n \), using parts up to size \( m \), is the number of ways in which \( n \) can be expressed as the sum of positive integer parts up to \( m \) in increasing order.

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don’t use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.

Interactive Diagram

Interactive Diagram