Hog Contest Rules

- Up to two people submit one entry;
  Max of one entry per person
- Slight rule changes
- Your score is the number of entries against which you win more than 50.00001% of the time
- Strategies are time-limited
- All strategies must be deterministic, pure functions of the players' scores
- All winning entries will receive extra credit
- The real prize: honor and glory
- See website for detailed rules

Hog Contest Winners

Fall 2011 Winners
Kaylee Mann
Yan Du and Ziming Li
Brian Price and Zhengqian Qian
Parker Schuh and Robert Chatham
Fall 2012 Winners
Chenyang Yuan
Joseph Hu
Fall 2013 Winners
Paul Brennan
Sam Rumpf and Kangshik Lee
Kevin Chen
Fall 2014 Winners
Alan Tong and Elaine Zhao
Zhengyang Cheng
Adam Robert Villador and Jonny Gao
Zhao Qin and Dian Chen
Zizheng Tai and Yunfei Lin

Two Definitions of Cascade

```python
def cascade(n):
    if n < 10:
        print(n)
    else:
        print(n)
        cascade(n//10)

def cascade(n):
    print(n)
    if n >= 10:
        cascade(n//10)
```

- 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:

```python
1
def inverse_cascade(n):
    grow(n)
    print(n)
    shrink(n)

def f_then_g(f, g, n):
    if n:
        f(n)
        g(n)

grow = lambda n: f_then_g(grow, print, n // 10)
shrink = lambda n: f_then_g(print, shrink, n // 10)

inverse_cascade(6)
```

Tree Recursion

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

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 2) + fib(n - 1)
```

Repetition in Tree-Recursive Computation

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

```python
fib(5)
```

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.

```python
count_partitions(6, 4)
```

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.

```python
count_partitions(6, 4)
```

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)
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:
  - \( \text{count_partitions}(2, 4) \)
  - \( \text{count_partitions}(6, 3) \)
- Tree recursion often involves exploring different choices.

```python
def count_partitions(n, m):
    if m == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m
```

(Demo)