Announcements

- Project 1 is due Thursday 9/18 @ 11:59pm; Early bonus point for submitting on Wednesday!
- Project Party in Stern Main Lounge (Stern Hall in Unit 4) 8pm-10pm on Wednesday 9/17
- Midterm 1 is on Monday 9/22 from 7pm to 9pm
- 2 review sessions on Saturday 9/20 3pm-4:30pm and 4:30pm-6pm in 1 Pimentel
- HKN review session moved to Sunday 9/21, time/location TBD
- Includes topics up to and including this lecture
- Closed book/note exam, except for one page of hand-written notes and a study guide
- Cannot attend? Fill out the conflict form by Wednesday 9/17 @ 5pm!
- Optional Hog strategy contest ends Wednesday 10/1 @ 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 scores
- All winning entries will receive 2 points of extra credit
- The real prize: honor and glory

Order of Recursive Calls

- 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

The Cascade Function

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

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

Interactive Diagram

Two Definitions of Cascade

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

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

Example: Inverse Cascade

```
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)
```
Tree Recursion

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

\[
\begin{align*}
\text{fib}(n): & \quad 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots \\
\text{fib}(n): & \quad 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots , 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465 \\
\end{align*}
\]

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

A Tree-Recursive Process

The computational process of \text{fib} evolves into a tree structure.

Repetition in Tree-Recursive Computation

This process is highly repetitive; \text{fib} is called on the same argument multiple times.

We can speed up this computation dramatically in a few weeks by remembering results.

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.

\[
\begin{align*}
\text{count_partitions}(6, 4): & \quad 2 + 4 = 6 \\
& \quad 1 + 1 + 4 = 6 \\
& \quad 3 + 3 = 6 \\
& \quad 1 + 2 + 3 = 6 \\
& \quad 1 + 1 + 1 + 3 = 6 \\
& \quad 2 + 2 + 2 = 6 \\
& \quad 1 + 1 + 2 + 2 = 6 \\
& \quad 1 + 1 + 1 + 1 + 2 = 6 \\
& \quad 1 + 1 + 1 + 1 + 1 + 1 = 6
\end{align*}
\]

Example: Counting Partitions

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.

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
def count_partitions(n, m):
    if n == 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
```

Interactive Diagram