Announcements
- Project 3 due Thursday 3/12 @ 11:59pm
- Project party on Tuesday 3/10 5pm-6:30pm in 2050 VLSB
- Bonus point for early submission by Wednesday 3/11
- Guerrilla section this weekend on recursive data (linked lists and trees)
- Homework 6 due Monday 3/16 @ 11:59pm
- Midterm 2 is on Thursday 3/19 7pm-9pm
- Fill out conflict form if you cannot attend due to a course conflict

Time

The Consumption of Time
Implementations of the same functional abstraction can require different amounts of time.

Problem: How many factors does a positive integer n have?
A factor k of n is a positive integer that evenly divides n.

\[
def \text{factors}(n):
\]

```
Slow: Test each k from 1 through n

Fast: Test each k from 1 to square root n
For every k, n/k is also a factor!
Greatest integer less than \sqrt{n}
```

Space

The Consumption of Space
Which environment frames do we need to keep during evaluation?
At any moment there is a set of active environments:
Values and frames in active environments consume memory.
Memory that is used for other values and frames can be recycled.

Active environments:
- Environments for any function calls currently being evaluated
- Parent environments of functions named in active environments

Fibonacci Space Consumption

Interactive Diagram
Order of Growth

A method for bounding the resources used by a function by the "size" of a problem

$m$: size of the problem

$R(n)$: measurement of some resource used (time or space)

$R(n) = \Theta(f(n))$

means that there are positive constants $k_1$ and $k_2$ such that

$$k_1 f(n) \leq R(n) \leq k_2 f(n)$$

for all $n$ larger than some minimum $m$

Counting Factors

Number of operations required to count the factors of $n$ using factors_fast is $\Theta(\sqrt{n})$

To check the lower bound, we choose $k_1 = 1$:

- Statements outside the while: 4 or 5
- Statements within the while (including header): 3 or 4
- while statement iterations: between $\sqrt{n}$ and $\sqrt{n} - 1$

To check the upper bound:

- Maximum statements executed: $5 + 4 \sqrt{n}$
- Maximum operations required per statement: some $p$

Assumption: every statement, such as addition-then-assignment using the += operator, takes some fixed number of operations to execute

Exponentiation

Goal: one more multiplication lets us double the problem size

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)
def square(x):
    return x * x

def exp_fast(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(exp_fast(b, n/2))
    else:
        return b * exp_fast(b, n-1)
```

Comparing Orders of Growth
**Properties of Orders of Growth**

**Constants:** Constant terms do not affect the order of growth of a process

\[ \Theta(n) \quad \Theta(0) \quad \Theta(\log n) \]

**Logarithms:** The base of a logarithm does not affect the order of growth of a process

\[ \Theta(\log_2 n) \quad \Theta(\log_{10} n) \quad \Theta(\ln n) \]

**Nesting:** When an inner process is repeated for each step in an outer process, multiply the steps in the outer and inner processes to find the total number of steps

\[
\text{def overlap}(a, b):
\quad \text{count} = 0
\quad \text{for item in a:
\quad \quad \text{if item in b:
\quad \quad \quad count += 1
\quad \quad }
\quad \text{return count}
\]

\[
\text{Outer: length of a}
\quad \text{Inner: length of b}
\]

If a and b are both length \( n \), then overlap takes \( \Theta(n^2) \) steps.

**Lower-order terms:** The fastest-growing part of the computation dominates the total

\[ \Theta(n^2) \quad \Theta(n^2 + n) \quad \Theta(n^2 + 500 \cdot n + \log_2 n + 1000) \]

**Comparing orders of growth (n is the problem size)**

\[ \Theta(n^3) \quad \text{Exponential growth. Recursive fib takes} \]
\[ \Theta(n^2) \quad \text{steps, where } \phi = 1 + \sqrt{5} \approx 1.61828 \]
\[ \text{Incrementing the problem scales R(n) by a factor} \]
\[ \Theta(n^2) \quad \text{Quadratic growth. E.g., overlap} \]
\[ \text{Incrementing n increases R(n) by the problem size n} \]
\[ \Theta(n) \quad \text{Linear growth. E.g., slow factors or } \exp \]
\[ \Theta(\sqrt{n}) \quad \text{Square root growth. E.g., factors_fast} \]
\[ \Theta(\log n) \quad \text{Logarithmic growth. E.g., exp_fast} \]
\[ \text{Doubling the problem only increments R(n).} \]
\[ \Theta(1) \quad \text{Constant. The problem size doesn't matter} \]