61A Lecture 21

Monday, October 20

Announcements

• Homework 6 is due Monday 10/20 @ 11:59pm
• Project 3 is due Thursday 10/23 @ 11:59pm
• Project/Homework party on Monday 10/20 6pm-8pm in 155 Dwinelle
• Midterm 2 is on Monday 10/27 7pm-9pm
• Class Conflict? Fill out the conflict form at the top of http://cs61a.org
• Review session on Saturday 10/25 3pm-6pm in 2050 VLSB

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 factors(n):
    greatest = int(n ** 0.5)
    slow = [i for i in range(1, 1 + greatest) if n % i == 0]
    fast = [i for i in range(1, 1 + greatest) if n % i == 0 for i in range(1, greatest + 1) if n / i in slow]
    return slow + fast
```

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

Interactive Diagram

Fibonacci Space Consumption

Interactive Diagram

Fibonacci Space Consumption
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

\[
k_1 \cdot f(n) \leq R(n) \leq k_2 \cdot f(n)
\]

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 \( \sqrt{n} - 1 \) and \( \sqrt{n} \)

To check the upper bound:
- Maximum statements executed: \( 5 \cdot \sqrt{n} \)
- Maximum operations required per statement: some \( p \)

We choose \( k_2 = 5p \)

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

\[
exponentiation(b, n):
\begin{align*}
\text{if } n = 0 & : \quad 1 \\
\text{else } & : \quad b \cdot exponentiation(b, n-1)
\end{align*}
\]

\[
square(x): \quad x \cdot x
\]

\[
exponentiation_fast(b, n):
\begin{align*}
\text{if } n = 0 & : \quad 1 \\
\text{else if } n \equiv 0 \mod 2 & : \quad square(exponentiation_fast(b, n/2)) \\
\text{else } & : \quad b \cdot exponentiation_fast(b, n-1)
\end{align*}
\]

Comparing Orders of Growth
Properties of Orders of Growth

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

\[ \Theta(1) \quad \Theta(500 \cdot n) \quad \Theta(\frac{1}{500} \cdot n) \]

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{outer length \( n \)} \times \text{inner length \( 500 \cdot n \)} \times \text{inner length \( 1500 \cdot n \)} \times \Theta(\log_2 n) \times \Theta(\log_{10} n) \times \Theta(\ln n) \]

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Comparing orders of growth (n is the problem size)

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