CSE 373: Data Structures and Algorithms

Lecture 3: Asymptotic Analysis part 2
Math Review, Inductive Proofs, Recursive Functions

Instructor: Lilian de Greef
Quarter: Summer 2017
Today:

• Brief Math Review (review mostly on your own)
• Continue asymptotic analysis with Big-O
• Proof by Induction
• Recursive Functions
Common Big-O Names

\[ O(1) \text{ constant (same as } O(k) \text{ for constant } k) \]
\[ O(\log n) \text{ logarithmic} \]
\[ O(n) \text{ linear} \]
\[ O(n \log n) \text{ “n log } n” \]
\[ O(n^2) \text{ quadratic} \]
\[ O(n^3) \text{ cubic} \]
\[ O(n^k) \text{ polynomial (where } k \text{ is any constant)} \]
\[ O(k^n) \text{ exponential (where } k \text{ is any constant } > 1) \]
A Few Common Big-O's

- O(1)
- O(n)
- O(n^2)
- O(logn)
- O(2^n)
- O(nlogn)
A Few Common Big-O's

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- $O(n\log n)$
A Few Common Big-O's

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- $O(n\log n)$

Graph showing the comparison of growth rates for different Big-O notations for values of $x$ from 1 to 91.
Powers of 2: Fun Facts

• A bit is 0 or 1 (just two different “letters” or “symbols”)
• A sequence of $n$ bits can represent $2^n$ distinct things
  (For example, the numbers 1 through $2^n$)
• $2^{10}$ is 1024 (“about a thousand”, kilo in CSE speak)
• $2^{20}$ is “about a million”, mega in CSE speak
• $2^{30}$ is “about a billion”, giga in CSE speak

Java: an `int` is 32 bits and signed, so “max int” is “about 2 billion”
  a `long` is 64 bits and signed, so “max long” is $2^{63}-1$
Which means...

You could give a unique id to...

• Every person in the U.S. with 29 bits
• Every person in the world with 33 bits
• Every person to have ever lived with 38 bits (estimate)
• Every atom in the universe with 250-300 bits

So if a password is 128 bits long and randomly generated, do you think you could guess it?
Math Review: Logs & Exponents

(Interlude #2 from Big-O)
Logs & Exponents

Definition: \( \log_a x = y \) if \( a^y = x \)

- \( \log_2 32 = 5 \)
- \( \log_{10} 10,000 = 4 \)
A Few Common Big-O's

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- $O(n\log n)$
$O(\log n)$
Logs & Exponents

Definition: \( \log_a x = y \) if \( a^y = x \)

- \( \log_2 32 = \)
- \( \log_{10} 10,000 = \)

Outside of CSE, \( \log(x) \) is often short-hand for \( \log_{10} x \)

In CSE, \( \log(x) \) is often short-hand for \( \log_2 x \)

...but, does it matter?
Can Make a $\log_2$ Out of Any $\log$!

$$\log_A x = \frac{\log_B(x)}{\log_B(A)}$$

so

$$\log_2 x = \frac{\log_{\text{whatever}}(x)}{\log_{\text{whatever}}(2)}$$

$$= \log_{\text{whatever}} x \cdot \left(\frac{1}{\log_{\text{whatever}}(2)}\right)$$
Other Properties of Logarithms
(to review on your own time)

• $\log(A \times B) = \log A + \log B$
  - So $\log(N^k) = k \times \log N$

• $\log(A/B) = \log A - \log B$

• $\log(\log x) = \log \log x$
  - Grows as slowly as $2^x$ grows quickly

• $\log(x)\log(x)$ is written $\log^2(x)$
  - It is greater than $\log(x)$ for all $x > 2$
  - It is not the same as $\log \log x$
Floor and Ceiling
(to review on your own time)

$\lfloor X \rfloor$  Floor function: the largest integer $\leq X$

$\lfloor 2.7 \rfloor = 2 \quad \lfloor -2.7 \rfloor = -3 \quad \lfloor 2 \rfloor = 2$

$\lceil X \rceil$  Ceiling function: the smallest integer $\geq X$

$\lceil 2.3 \rceil = 3 \quad \lceil -2.3 \rceil = -2 \quad \lceil 2 \rceil = 2$
Floor and Ceiling Properties
(to review on your own time)

1. $X - 1 < \lfloor X \rfloor \leq X$
2. $X \leq \lceil X \rceil < X + 1$
3. $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$ if $n$ is an integer
Back to Big-O
What's the asymptotic runtime of this (semi-)pseudocode?

\texttt{x := 0;}
\texttt{for i = 1 to N do}
  \texttt{for j = 1 to i do}
    \texttt{x := x + 3;}
\texttt{return x;}

A. O(n)
B. O(n^2)
C. O(n + n/2)
D. None of the above

How do we prove the right answer?
Proof by Induction!
Inductive Proofs

(Interlude from Asymptotic Analysis)
Steps to Inductive Proof

1. If not given, define \( n \) (or “x” or “t” or whatever letter you use)

2. Base Case

3. Inductive Hypothesis (IHOP):
   Assume what you want to prove is true for some arbitrary value \( k \) (or “p” or “d” or whatever letter you choose)

4. Inductive Step:
   Use the IHOP (and maybe base case) to prove it’s true for \( n = k+1 \)

\[ \frac{\text{prime}}{\text{conclusion}} = \frac{3}{3} \]
Example #0: Proof that I can climb any length ladder

1. Let \( n \) = number of rungs on a ladder.
2. **Base Case:** for \( n = 1 \) ✓
3. **Inductive Hypothesis (IHOP):** Assume true for some arbitrary integer \( n = k \).
4. **Inductive Step:** (aiming to prove it's true for \( n = k + 1 \))
   - By IHOP, I can climb \( k \) steps of the ladder.
   - If I’ve climbed that far, I can always climb one more.
   - So I can climb \( k + 1 \) steps.
   - I can climb forever!
Example #1

Prove that the number of loop iterations is \( n \times (n + 1) \frac{n}{2} \).

\[
\begin{align*}
\text{x := 0;} \\
\text{for } i=1 \text{ to } N \text{ do} \\
\quad \text{for } j=1 \text{ to } i \text{ do} \\
\quad \quad \text{x := x + 3;} \\
\text{return x;}
\end{align*}
\]

\[n = N\]

\[\begin{align*}
\text{Base Case: } & \text{ true for } n=1 \\
n=1 & \quad \frac{1(1+1)}{2} = \frac{2}{2} = 1
\end{align*}\]

\[1-10P: \text{ assume true } n=k\]

where \( k \) is an arbitrary integer \( > 2 \)

\[\begin{align*}
\text{Inductive step} & \\
\text{add last step } i=k+1
\end{align*}\]
Inductive step: (Goal: show it's true for $n = k+1$)

For $N = k+1 = (\text{iterations for } N = k) + (k+1)$

by IHOP

$$p(n+1) = p(n) + n + 1$$ (true for this problem)

$$p(n) = \frac{k(k+1)}{2} + k + 1$$

$$= \frac{k^2 + k + 2(k+1)}{2}$$

$$= \frac{k^2 + k + 2k + 2}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

QED
Example #2:
Prove that $1 + 2 + 4 + 8 + \ldots + 2^n = 2^{n+1} - 1$

1. $P(n) = n = n$

2. Base case: $n = 0$
   $2^0 = 1 = 2^{0+1} - 1 = 2 - 1 = 1$
   True for $n = 0$

3. Inductive step: Assume true for $n = k$
   $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

4. Inductive step:
   \begin{align*}
   \text{Goal: show it's true for } n = k+1 \\
   w+1 & : \sum_{i=0}^{k+1} 2^i = 2^{k+2} - 1
   \end{align*}

4. $P = n = k+1$
\[
\begin{align*}
(\text{w+5}) \sum_{i=0}^{k+1} 2^i &= 2^{k+2} - 1 \\
\sum_{i=0}^{k+1} 2^i &= \left(\sum_{i=0}^{k} 2^i\right) + 2 \\
2(2^x) &= 2^{x+1} = 2^{k+1} - 1 + 2 \\
2(2^x) &= (2 \cdot 2 \cdot 2 \cdots \cdot 2) \times \text{times} \\
2^{k+1} &= 2 \cdot 2 \cdot 2 \cdot \cdots \cdot 2 \times \text{times} \\
2^{k+1} &= 2 \cdot 2^{k+1} - 1 \\
2 = 2 \cdot 2 \cdot 2 \cdot \cdots \cdot 2 \times \text{times}
\end{align*}
\]
Useful Mathematical Property!

\[ \sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \]

You’ll use it or see it again before the end of CSE 373.
Example #3: (Parody) Reverse Induction!

Proof by Reverse Induction That You Can Always Cage a Lion:

Let \( n \) = number of lions

**Base Case:** There exists some countable, arbitrarily large value of \( M \) such that when \( n = M \), the lions are so packed together that it's trivial to cage one.

**IHOP:** Assume this is also true for \( n = k \) for some arbitrary value \( k \).

**Inductive Step:** Then for \( n = k - 1 \), release a lion to reduce the problem to the case of \( n = k \), which by the IHOP is true.

QED :)

Fun fact: Reverse induction is a thing! The math part of the above is actually correct.
Recursive fn: fn that calls itself

Big-O: Recursive Functions

How do we asymptotically analyze recursive functions?
Example #1: Towers of Hanoi
Example #1: Towers of Hanoi

// Prints instructions for moving disks from one pole to another, where the three poles are labeled with integers "from", "to", and "other".
// Code from rosettacode.org

public void move(int n, int from, int to, int other) {
    if (n == 1) {
        System.out.println("Move disk from pole " + from + " to pole " + to);
    } else {
        move(n - 1, from, other, to);
        move(1, from, to, other);
        move(n - 1, other, to, from);
    }
}
Example #1: Towers of Hanoi

Base Case: \( H(1) = 1 \)

if \( n == 1 \) {
    System.out.println("Move disk from pole " + from + " to pole " + to);
}

Recursive Step:
else {
    move(n - 1, from, other, to);
    move(1, from, to, other);
    move(n - 1, other, to, from);
}

All together:
\( H(n) = H(n-1) + 1 + H(n-1) \)
\( H(n) = 1 + 2H(n-1) \)
Example #1: Solving the Recurrence Relation

Recurrence Relation:

\[ H(n) = 1 + 2H(n-1) \]

Expanding (plug in for \( H(n) \)).

1st

\[ H(n) = 1 + 2H(n-1) \]

2nd

\[ = 1 + 2(1 + 2H(n-1)) \]

\[ = 1 + 2 + 4H(n-1) \]

3rd

\[ H(n) = 1 + 2 + 4 + 8H(n-2) \]

\[ = 1 + 2 + 4 + 8 + 16H(n-3) \]

\[ \vdots \]

\[ = 2^0 + 2^1 + 2^2 + \ldots + 2^{k-1} + 2^k H(n-k) \]
(continued)

\[ H(n) = 2^k - 1 + 2^k H(n-k) \]

Base case is at \( H(1) \), so let's solve \( n-k = 1 \)

\[ \Rightarrow k = n-1 \]

Plug it in:

\[ H(n) = 2^{n-1} - 1 + 2^{n-1} H(n-(n-1)) \]

\[ = 2^{n-1} - 1 + 2^{n-1} H(1) \]

\[ \Rightarrow \quad H(1) = 1 \]

\[ H(n) = 2^{n-1} \]

\[ = 2 \cdot (2^{n-1}) - 1 = 2^n - 1 \]

\[ O(2^n) \]