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

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

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

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

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- $O(n\log n)$
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 \texttt{int} is 32 bits and signed, so “max int” is “about 2 billion”
\texttt{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 \( a^y = x \)

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

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

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

\[ \log_A x = \frac{\log_B(x)}{\log_B(A)} \]

so

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

Floor function: the largest integer \( \leq X \)

\[
\left\lfloor X \right\rfloor = 2 \quad \left\lceil -2.7 \right\rceil = -3 \quad \left\lfloor 2 \right\rfloor = 2
\]

Ceiling function: the smallest integer \( \geq X \)

\[
\left\lceil X \right\rceil = 3 \quad \left\lfloor -2.3 \right\rfloor = -2 \quad \left\lceil 2 \right\rceil = 2
\]
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. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$ if $n$ is an integer
Back to Big-O
What’s the asymptotic runtime of this (semi-)pseudocode?

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

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

(Some textbooks format algorithms in this style of semi-pseudocode)
What’s the asymptotic runtime of this (semi-)pseudocode?

```plaintext
x := 0;
for i=1 to N do
    for j=1 to i do
        x := x + 3;
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 \( n = k+1 \)
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 \( \frac{n \times (n + 1)}{2} \)

\[
x := 0;
for i=1 to N do
  for j=1 to i do
    x := x + 3;
return x;
\]
(Extra room for notes)
Example #2:
Prove that $1 + 2 + 4 + 8 + \ldots + 2^n = 2^{n+1} - 1$
(Extra room for notes)
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.
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

```java
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: Solving the Recurrence Relation

Recurrence Relation:
Example #2: Binary Search

| 2 | 3 | 5 | 16 | 37 | 50 | 73 | 75 | 126 |

Find an integer in a *sorted* array

(Can also be done non-recursively)

// Requires the array to be sorted.
// Returns whether k is in array.
public boolean find(int[] arr, int k) {
    return help(arr, k, 0, arr.length);
}

private boolean help(int[] arr, int k, int lo, int hi) {
    int mid = (hi+lo)/2; // i.e., lo+(hi-lo)/2
    if (lo==hi) return false;
    if (arr[mid]==k) return true;
    if (arr[mid]<k) return help(arr, k, mid+1, hi);
    else return help(arr, k, lo, mid);
What is the recurrence relation?

// Requires the array to be sorted.
// Returns whether k is in array.
public boolean find(int[] arr, int k) {
    return help(arr, k, 0, arr.length);
}

private boolean help(int[] arr, int k, int lo, int hi) {
    int mid = (hi + lo) / 2; // i.e., lo + (hi - lo) / 2
    if (lo == hi) return false;
    if (arr[mid] == k) return true;
    if (arr[mid] < k) return help(arr, k, mid + 1, hi);
    else return help(arr, k, lo, mid);
}

A. 2T(n-1) + 3
B. T(n-1)*T(n-1) + 3
C. T(n/2) + 3
D. T(n/2) * T(n/2) + 3