Week 2 Tuesday Lecture Notes

**** Picture are now where they are referenced in the notes rather than at the end. ****

Prep for Class: Watch Tim Roughgarden Lectures on **Merge Sort**

(Can be found Under Canvas-> Home -> Lecture Schedule -> Week 1 Thursday Prep Lectures 1.5, 1.6 and 1.7)

Explanation of Tim Roughgarden Lectures: Merge Sort splits an array into smaller and smaller arrays by splitting the arrays in half. After these recursive calls the merge then compares each new smaller array to each other to figure out the correct order of the merge. Because of being able to divide the array in half over and over the runtime of Merge Sort is at most $6n*\log_2 n + 6n$, which will have a Big-Oh of n*log(n) vs the relatively slow Insertion Sort or Selection sort that has a Big-Oh runtime of n^2. This is why Merge sort is much faster especially as N increases in size.

Summary: Merge sort is much faster at sorting a large array n because of how it recursively divides the size of the array in half over and over rather than going through the array n^2 times as is the case for other slower algorithms.

Lecture Starts on PowerPoint: Lecutres-1 Slide 18. (Big-Omega Ω Notation)

- Restate information about office hours and email instructor if you need help.

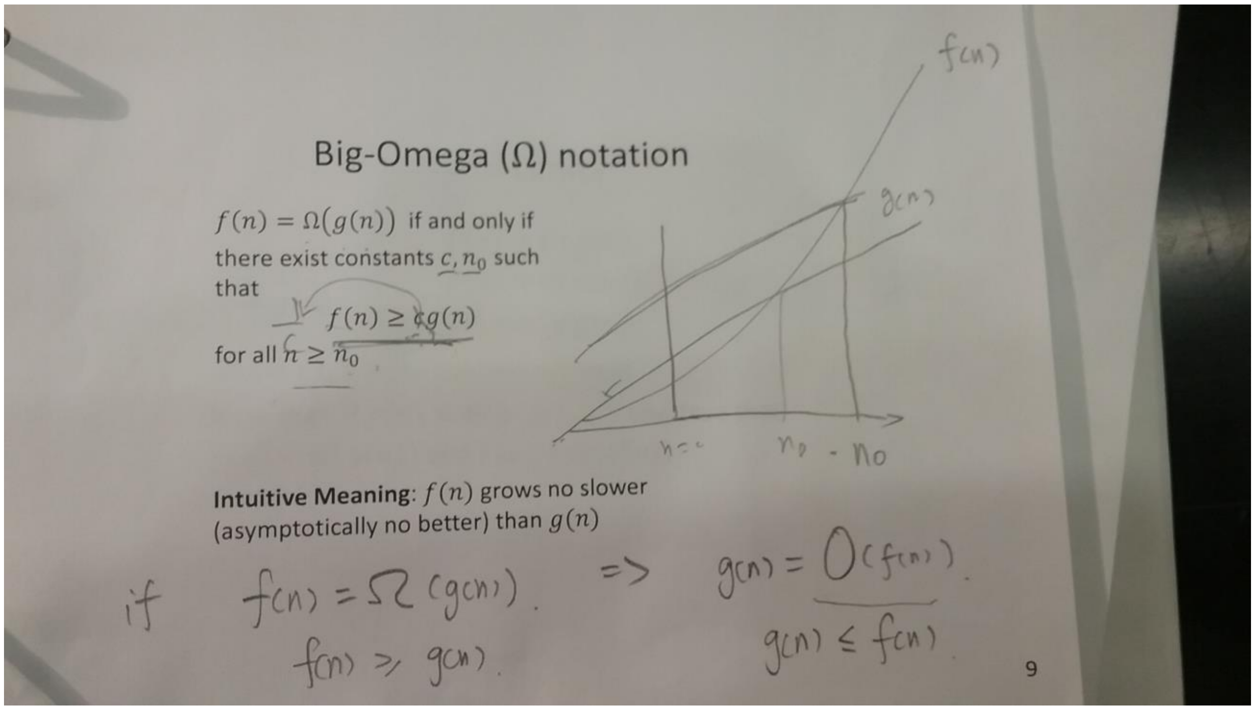
Previous Lecture Review:

- Last lecture we talked about algorithm and why we care about algorithm. Runtime we care about the wort case runtime analysis.

- Asymptotic Analysis: ignore lower order terms (Big-Oh Notation) This is because we care about really large size input.
- Asymptotic Complexity: input size is very very large.
- f(n) = 0(g(n))   The runtime of f is as fast or faster than g.
- There exists constants c, n_0. Such that
- F(n) <= cg(n) for all n>=n_0.

Lecture 1 Continued: Omega, Theta, Constants and Proof by Induction.

Big-Omega (Ω) notation.



Big-Omega (Ω) notation

$f(n) = \Omega(g(n))$ if and only if there exist constants $\underset{\smile}{c}, \underset{\smile}{n_0}$ such that

$\underline{f(n) \geq cg(n)}$

for all $n \geq n_0$

Intuitive Meaning: $f(n)$ grows no slower (asymptotically no better) than $g(n)$

if $f(n) = \Omega(g(n))$. => $g(n) = O(f(n))$.

$f(n) \geq g(n)$. $g(n) \leq f(n)$.

9

f(n)
g(n)
n=c
n_0
- n_0

f(n) = Ω(g(n)) if and only if there exists constants c, n_0 such that:

f(n) >= cg(n) for all n>=n_0

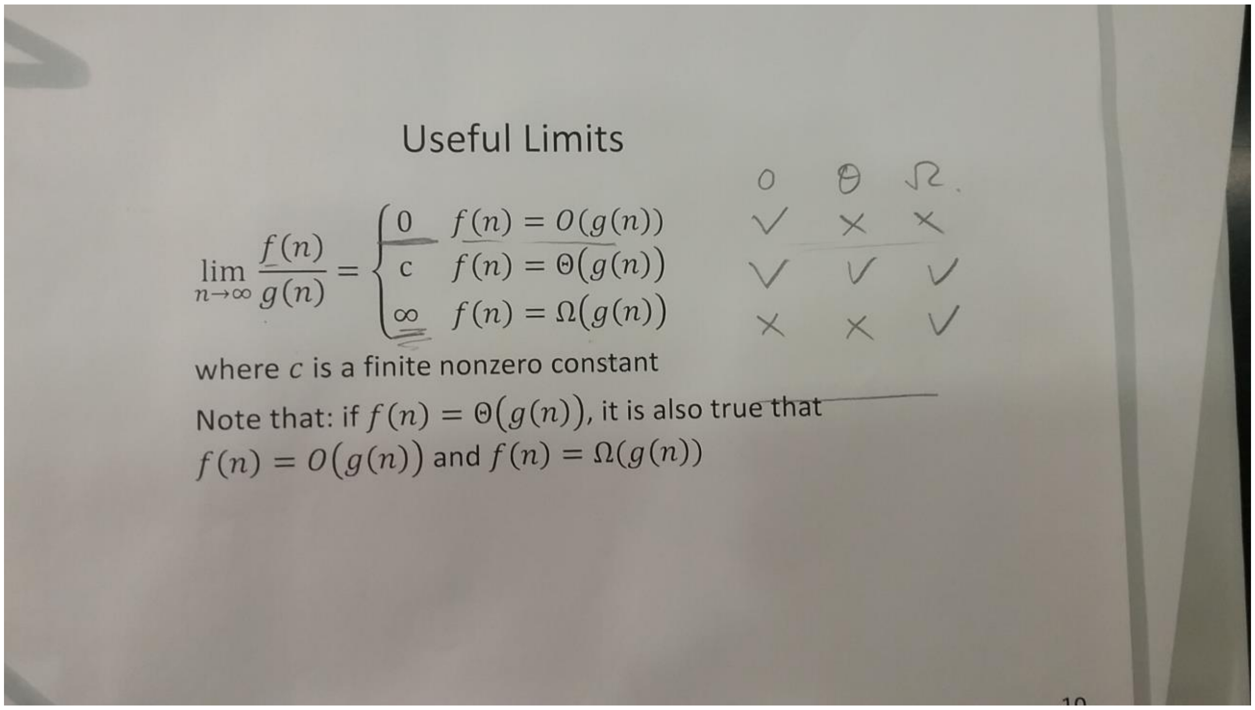
Meaning: f(n) runs as fast or faster than g(n)

See Big-Omega Graph Picture Example.

Big-Theta(Θ) notation

Meaning: f(n) grows the same rate as g(n) f(n) = **Θ(g(n))**

Finding how Limits can be found using 0, Θ and Ω:



Useful Limits

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = O(g(n)) \\ c & f(n) = \Theta(g(n)) \\ \infty & f(n) = \Omega(g(n)) \end{cases}$$

O	Θ	Ω
✓	✗	✗
✓	✓	✓
✗	✗	✓

where c is a finite nonzero constant

Note that: if $f(n) = \Theta(g(n))$, it is also true that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Example:

- It is easier to use lim as n-> infinity. Cancel out terms in order to see what will go to infinity vs going to 0.
- How do we know based on lim as n → infinity?

Quick in Class example: Which are True/False?

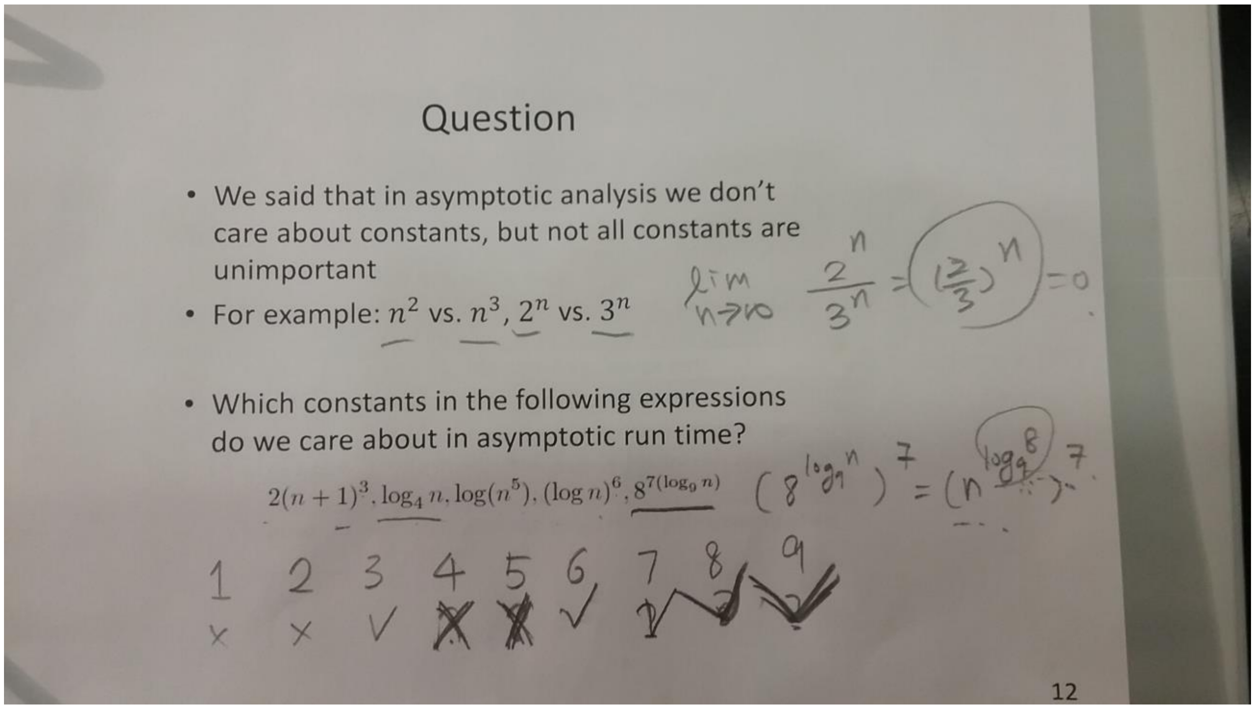
1. False because n grows no faster than our function.
2. True because n^3 grows no slower than our function.
3. True because n will always grow no faster than our function.

4. True because n^2 grows no slower or faster than our function.

Quick in Class example 2: Which of the following are correct?

All true because the n power are the same between f(n) and g(n) 2^n vs 2^{2+10} you can ignore the constant of 10. If Big-Theta Θ is true than Big-Omega and Big-O are also true.

What constants do we care about?



Question

- We said that in asymptotic analysis we don't care about constants, but not all constants are unimportant
- For example: n^2 vs. n^3, 2^n vs. 3^n

$$\lim_{n \to \infty} \frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n = 0$$

- Which constants in the following expressions do we care about in asymptotic run time?

$2(n+1)^3, \log_4 n, \log(n^5), (\log n)^6, 8^{7(\log_9 n)}$

$$\left(8^{\log_9 n}\right)^7 = \left(n^{\log_9 8}\right)^7$$

1	2	3	4	5	6	7	8	9
×	×	✓	×	×	✓	✓	✓	✓

12

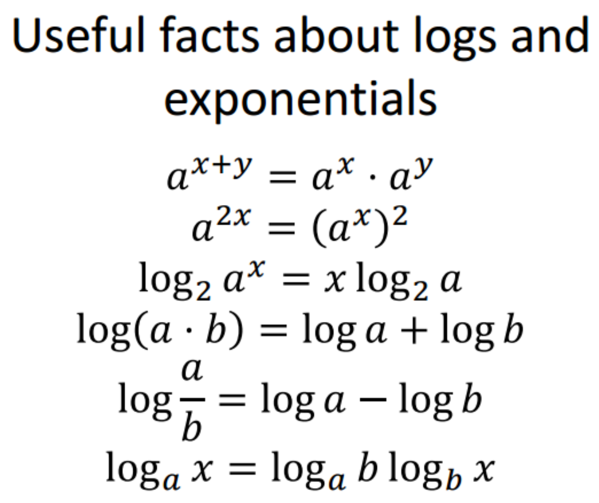
From the picture above:

- At the bottom there are 9 numbers that correlate to 9 constants of the expression. We are saying if they care about for asymptotic run time speed.
- 1: Does **not** matter because it's a constant being added to n.
- 2: Does **not** matter because it's a product.

- 3: Does matter because it’s an exponent to an n.
- 4: Does **not** matter, log of x does not matter when x is a constant.
- 5: Does **not** matter. Exponents inside of a log can be rewritten outside as a product.
- 6. Does matter because the exponent is outside of the log and correlates to an n.
- 7: Does matter because it’s in an exponent and the exponent contains an n.
- 8: Does matter because any constant to the power of something that contains an n will change runtime.
- 9: Does matter only because the log value is in an exponent.

There are many edge cases when finding out what information is and is not important for runtime.

Useful Facts about Logs and Exponentials. From lecture

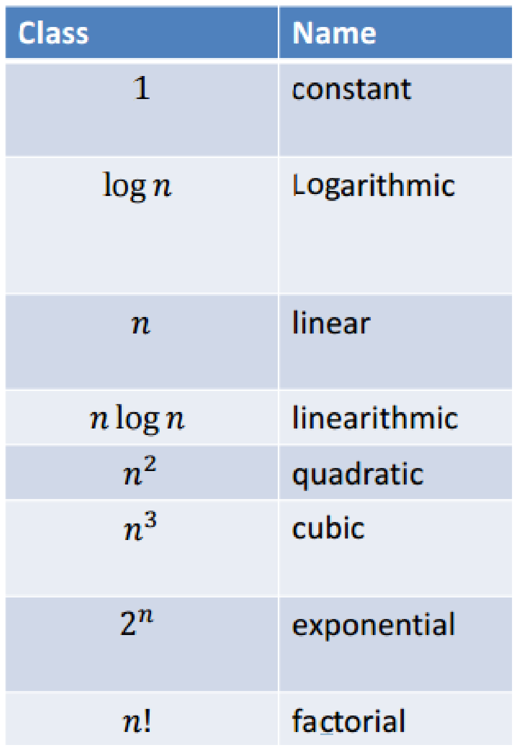


Useful facts about logs and exponentials

$$a^{x+y} = a^x \cdot a^y$$
$$a^{2x} = (a^x)^2$$
$$\log_2 a^x = x \log_2 a$$
$$\log(a \cdot b) = \log a + \log b$$
$$\log \frac{a}{b} = \log a - \log b$$
$$\log_a x = \log_a b \log_b x$$

These Rules can be useful in determining what is important when finding runtime speed of algorithms.

Common Efficiency Class: Order of Efficiency from faster to slowest.



Class	Name
1	constant
$\log n$	Logarithmic
n	linear
$n \log n$	linearithmic
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

Start of Lecture 2 Material:

Review: Merge Sort video (at top of these notes there is a summary of merge sort videos)

Recursive Merge sort has a runtime of: **nlogn**

Proof By Induction: Assume that the smallest size holds the base case: array of size 0 or 1 is already sorted.

Inductive assumption: size is 1←→k

Inductive step: For array A size k + 1.

Postcard Example: Using an Inductive Proof

Summary of Postcard Example: There are 3 parts to induction:

1. Setting up a Base Case
2. Inductive Assumption
3. Inductive Step to proof that any k+1 can be solved.

The PowerPoint Slides do a very good job at explaining the Inductive Proof**. (See Page 5 in Lecture 2 Slides)**

Ending Notes:

- **Review:** Recursive Merge Sort Video.
- This class can have a focus on abstraction with regard to proofs.
- Questions for Quiz Review have been posted.
  (Review questions are not graded but serve as an important tool in understanding proofs)
- Will Discuss Recurrence as it pertains to algorithms further during Tuesdays Lecture.

Ended on Page 7 of Lecture 2 Notes:

<u>End of Week 2 Thursday Lecture Notes</u>

~Information composed by Notetaker Scott Russell for CS 325 **DAS** student