CS 375 – Analysis of Algorithms

Professor Eric Aaron

Lecture – M W 1:00pm

Lecture Meeting Location: Davis 117

Business

Please read the emailed Classwide Comments

- Smaller Assignment 0 returned already
  - Let me know if there are problems accessing it
- Smaller Assignment 1, due already
- Problem Set 0 out, due Sept. 21

- Project 1 due Sept. 28
  - Please direct project-specific questions to me, rather than to TAs
    - Questions about general concepts that show up on the project (e.g., Theta notation), though, rather than specifics, can go to TAs
  - Everyone was on a team as of yesterday
  - Let me know if there are problems / concerns with team assignments

Business, pt. 2

- Class will be cancelled Monday, Sept. 26
  - Will be an optional make-up class later in the semester
- Let's go over SA0 Exercise 1.f
  - If A={x,y,z} and B={x,y}, what is AxB?
  - AxB is, by definition, a *set of ordered pairs*—please be sure to use the correct notation and concepts (notation and semantics matter to CS! Just ask your compiler!)

Asymptotic Analysis / Big-O Notation

- With insertion sort, if we gloss over minor details, we can see the number of operations (worst case) is *on the order of* n^2

$$\left(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2}\right)n^2+\left(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8\right)n - (c_2+c_4+c_5+c_8)\ .$$

  - i.e., it is $c*n^2$ + (lower order terms)
  - … for some constant c
  - … where n is the size of the input
- Definition: An algorithm runs in time $O(f(n))$ (read: "order of f(n)") means:
  - There exist $c > 0$, $n_0 > 0$ s.t. …
  - …for all $n \geq n_0$, the running time of the algorithm is less than $c*f(n)$
  - (Basically, that means that for every input "big enough," the running time is less than a constant times $f(n)$)

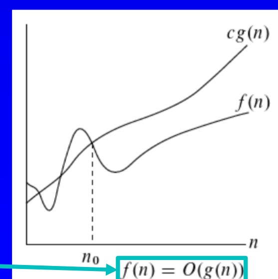
So, we'd say Insertion sort is $O(n^2)$

Asymptotic Analysis / Big-O Notation

- Definition: An algorithm runs in time *O(f(n))* (read: "order of f(n)") means:
  - There exist $c > 0$, $n_0 > 0$ s.t. …
  - …for all $n \geq n_0$, the running time of the algorithm is less than $c*f(n)$
  - (Basically, that means that for every input "big enough," the running time is less than a constant times *f(n)*)

Defn. repeated from prev. slide

- Informal Intuition: Big-O is about *upper bounds*
  - If a runtime T(n) is O(f(n)), then for "big enough" n, *T(n)* is upper bounded by *c*f(n)* for some *leading constant* c



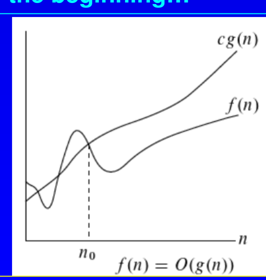
Note: This figure from your textbook uses f(n) for runtime and g(n) for the bounding function, but it's the same idea—f(n) is O(g(n)), upper bounded by c*g(n) for all $n \geq n_0$

Breaking Down the Phrase "Big-O Asymptotic Complexity"

- Major takeaways about *Big-O Asymptotic Complexity*

> - **In fact, there's one major takeaway for each of the three words in the phrase "*Big-O Asymptotic Complexity*", based on their meaning.**
> - **It's best to work from the end of that phrase to the beginning…**

  - ***Complexity***: It's about describing the *resource usage* of an algorithm
  - ***Asymptotic***: It describes complexity based on behavior on *large input sizes* n—small inputs aren't really the point
  - ***Big-O***: It's an *upper bound* on complexity on large inputs



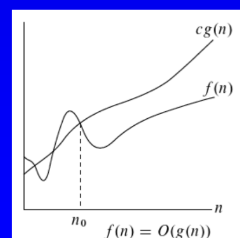
Big-O: In this picture, for large enough *n* (that is, $n \geq n_0$), *f(n)* is upper bounded by a leading constant *c* times *g(n)*

Asymptotic Analysis / Big-O Notation

- Definition: An algorithm runs in time $O(f(n))$ (read: "order of f(n)") means:
  - There exist $c > 0$, $n_0 > 0$ s.t. …
  - …for all $n \geq n_0$, the running time of the algorithm is less than $c * f(n)$
  - (Basically, that means that for every input "big enough," the running time is less than a constant times $f(n)$)

Defn. repeated from prev. slide

Recall: Big-O is about *upper bounds*



- This runtime measure captures some essential characteristic of an algorithm
  - $O(n^2)$ algorithms differ from $O(n^3)$, from $O(n \log n)$, etc.
- Can talk about asymptotic *complexity classes*
  - We say Insertion sort is in complexity class $O(n^2)$

Conventional Wisdom about Big-O Classes

- If two algorithms are in different big-O classes, then there seems to be something substantially different about their speeds
  - Even though, for some small values of n, an $O(2^n)$ algorithm could be faster than an $O(n^2)$ algorithm…
  - It is nonetheless true that 2^n *grows* faster than n^2…
  - Thus, an $O(2^n)$ algorithm is, in a relevant sense, *inherently* slower than an $O(n^2)$ algorithm

Important Vocab (see CLRS, pg. 28): These functions of *n* have very different *orders of growth*—i.e., how fast they grow as *n* gets larger

- For an $O(n)$ algorithm (called "linear")
  - Doubling the input size does what to the running time?
  - Increasing input size by factor of 100 does what to running time?
- For an $O(n^2)$ algorithm ("quadratic")
  - Doubling the input size does what to the running time?
  - Increasing input size by factor of 100 does what to running time?
- For an $O(2^n)$ algorithm ("exponential")
  - Doubling the input size does what to the running time?

Common complexity measures and how they relate to input sizes

- Algorithms are sometimes described by their time complexity. There are
  - Logarithmic algorithms
  - Quadratic algorithms
  - Exponential algorithms
  - Factorial algorithms
  - etc.
- To see which kind is fastest, see how these functions grow with increases in the input size:

n	$\log_{10} n$	n^2	2^n	$n!$
1	0	1	2	1
10	1	100	1024	3628800
50	1.70	2500	1.13e15	3.04e64
100	2	10000	1.27e30	9.44e157

Using the Big-O Definition

- Definition: $O(g(n)) = \{f(n) \mid$ \exists $c, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq f(n) \leq c*g(n)\}$
- Is each of the below statements true? Explain your answers!

1. $100n + 5 = O(n^2)$
2. $n^2/2 - 3n = O(n^2)$
3. $100n^2 = O(n^2)$
4. $100n^2 = O(n^3)$
5. $0.01n^3 = O(n^2)$
6. $n \lg n = O(\lg^2 n)$
7. $2^{n+1} = O(2^n)$
8. $2^{2n} = O(2^n)$

Using the Big-O Definition

- Definition: $O(g(n)) = \{f(n) \mid$ \exists $c, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq f(n) \leq c*g(n)\}$
- Is each of the below statements true? Explain your answers!

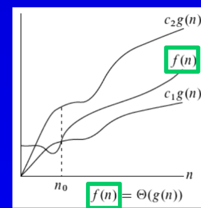
1. $100n + 5 = O(n^2)$
2. $n^2/2 - 3n = O(n^2)$
3. $100n^2 = O(n^2)$
4. $100n^2 = O(n^3)$
5. $0.01n^3 = O(n^2)$
6. $n \lg n = O(\lg^2 n)$
7. $2^{n+1} = O(2^n)$
8. $2^{2n} = O(2^n)$

Pro Tip on how to explain these:
In general, when explaining why an existential ("\exists") statement is true, explicitly give some *witness* value(s) that make it true as part of the explanation.

Here, if a statement is true, can you give specific values for *c, n_0* that make it true?

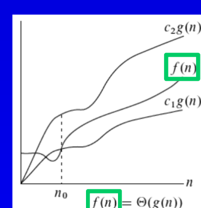
Big "Oh… there's more?" Notation

- Theta notation: *Asymptotically tight bound*
  - Definition: $\theta(g(n))$ = {$f(n)$ | \exists $c1$, $c2$, $n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c1*g(n) \leq f(n) \leq c2*g(n)$}

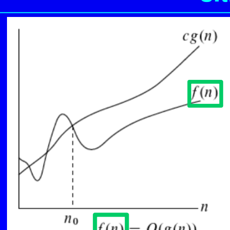


Big "Oh… there's more?" Notation

- Theta notation: *Asymptotically tight bound*
  - Definition: $\theta(g(n))$ = {$f(n)$ | \exists $c1$, $c2$, $n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c1*g(n) \leq f(n) \leq c2*g(n)$}

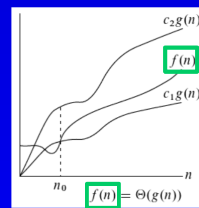


Reminder--defn of Big-O: $O(g(n))$ = {$f(n)$ | \exists c, $n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq f(n) \leq c*g(n)$}



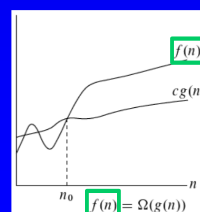
Big "Oh… there's more?" Notation

- Theta notation: *Asymptotically tight bound*
  - Definition: $\theta(g(n)) = \{f(n) \mid$ \exists $c1, c2, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c1*g(n) \leq f(n) \leq c2*g(n)\}$



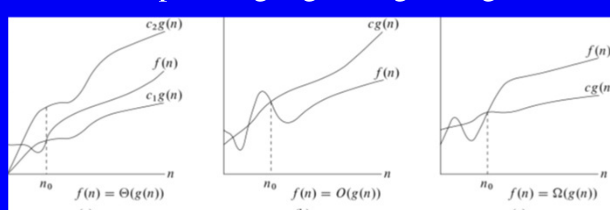
Reminder--defn of Big-O: $O(g(n)) = \{f(n) \mid$ \exists $c, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq f(n) \leq c*g(n)\}$

- Big-Omega notation: *Asymptotic lower bound*
  - Definition: $\Omega(g(n)) = \{f(n) \mid$ \exists $c, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c*g(n) \leq f(n)\}$



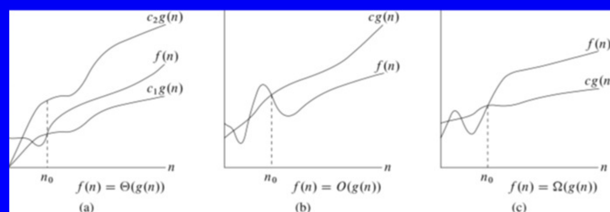
Big "Oh… there's more?" Notation

- Theta notation: *Asymptotically tight* bound
  - Definition: $\theta(g(n)) = \{f(n) \mid$ \exists $c1, c2, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c1*g(n) \leq f(n) \leq c2*g(n)\}$
- Big-Omega notation: Asymptotic lower bound
  - Definition: $\Omega(g(n)) = \{f(n) \mid$ \exists $c, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c*g(n) \leq f(n)\}$
- What is the relationship among big-O, big-Omega, and Theta classes?



A Big-Symbols Theorem

- Definition: $\theta(g(n)) = \{f(n)$ | \exists $c1, c2, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c1*g(n) \leq f(n) \leq c2*g(n)\}$
- Definition: $\Omega(g(n)) = \{f(n)$ | \exists $c, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c*g(n) \leq f(n)\}$
- *Theorem*: For any two functions $f(n)$ and $g(n)$, $f(n) = \theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.



Using the θ, Ω Definitions

- Definition: $\theta(g(n)) = \{f(n)$ | \exists $c1, c2, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c1*g(n) \leq f(n) \leq c2*g(n)\}$
- Definition: $\Omega(g(n)) = \{f(n)$ | \exists $c, n_0 > 0$ s.t. \forall $n \geq n_0$, $0 \leq c*g(n) \leq f(n)\}$
- Is each of the below statements true?

1. $100n + 5 = \theta(n^2)$
2. $100n + 5 = \Omega(n^2)$
3. $n^2/2 - 3n = \theta(n^2)$
4. $n^2/2 - 3n = \Omega(n^2)$
5. $100n^2 = \theta(n^3)$
6. $0.01n^3 = \Omega(n^2)$
7. $2^{n+1} = \theta(2^n)$
8. $2^{2n} = \Omega(2^n)$

Conventions: Order of Growth (to within a constant multiple)

- Two different levels of detail can be useful with asymptotic complexity:
  - Formal definitions and detailed explanations
  - Informal, high-level understanding and explanations
- When informally talking about asymptotic complexity, we often talk about the *order of growth* of runtime functions, to *within a (leading) constant multiple*
  - We don't say exactly what the leading constant c or n_0 threshold is
  - Order of growth of the highest order / dominant term is most important

In CS375, unless specified otherwise, feel free to use the informal, high-level approach

Log It:
Questions about exponents

- When solving equations, we may want to know the value of an exponent
  - E.g., in equation $2^x=375$, we might want to ask what value of x makes that true
  - *How could we even phrase that question?*
- The *logarithm* function lets us ask the question
  - So, for $2^x = 375$, we'd say $x = log_2\, 375$ (read as "log base 2 of 375")
  - Examples: $log_3\, 81 = 4$; $log_4\, 16 = 2$; $log_2\, 1024 = 10$
- Logarithms *are* exponents, so rules of exponentiation apply
  - E.g., $log_b\, (m*n) = log_b\, m + log_b\, n$

If $b^x = m$ and $b^y = n$, then $b^x*b^y = b^{x+y} = m*n$