COMP 3170 - Analysis of Algorithms & Data Structures

Shahin Kamali

Lecture 3 - Jan. 11, 2019

CLRS 1.1, 1.2, 2.2, 3.1, 4.3, 4.5

University of Manitoba

Picture is from the cover of the textbook CLRS.
Review

- To analyze running time of an algorithm (under RAM model) we sum the number of primitive operations and memory accesses of the algorithm.

- The cost (running time) of algorithm $A$ for a problem of size $n$ would be a function $T_A(n)$.

- How do we compare two different algorithms? say $T_A(n) = \frac{1}{1000} n^3$ and $T_B(n) = 1000n^2 + 500n + 200$.

- Summarize the time complexity using asymptotic notations!

- Idea: assume the size of input grows to infinity; identify which component of $T_A(n)$ contributes most to the grow of $T_A(n)$.

- As $n$ grows:
  - constants don’t matter (e.g., $T_A(n)$)
  - low-order terms don’t matter (e.g., $T_B(n)$)
Informally $f(n) = O(g(n))$ means $f$ is asymptotically smaller than or equal to $g$.

**Definition**

$$f(n) \in O(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M \cdot g(n)$$

- Ignore low-order terms
- Ignore constants
Big O Notations

- E.g., \( f(n) = 2n, \ g(n) = n \). Is it that \( f(n) \in O(g(n)) \)?
  - Yes, \( f(n) \) is asymptotically smaller than or equal (equal) to \( g(n) \).
  - To prove, we should show
    \[ \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M \cdot g(n) \]
  - It suffices to define \( n_0 = 1 \) and \( M = 3 \), we have \( \forall n > 1, 2n \leq 3n \).
  - \( M \) could be any number larger than or equal to 2, and \( n_0 \) could be any number.

- We require specific values of \( M \) (not all choices for \( M \) work)
Big O Notations

E.g., $f(n) = 2n + 100/n$, $g(n) = n$. Is it that $f(n) \in O(g(n))$?

- Yes, again, $f(n)$ is asymptotically smaller than or equal (equal) to $g(n)$.
- To prove, we should show
  \[ \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M \cdot g(n) \]
- It suffices to define $n_0 = 10$ and $M = 3$, we have $\forall n > 10, 2n + 100/n \leq 3n$.

- We require specific values of $M$ and $n_0$ (not all choices work)

![Graph showing functions f(n), g(n), and 3g(n)]
Big O Notation

Let \( f(n) = 2019n^2 + 1397n \) and \( g(n) = n^3 \). Prove \( f(n) \in O(g(n)) \)

We should define \( M \) and \( n_0 \) s.t. \( \forall n > n_0 \) we have 
\( 2019n^2 + 1397n \leq Mn^3 \). This is equivalent to \( 2019n + 1397 \leq Mn^2 \).

We have \( 2019n + 1397 \leq 2019n + 1397n = 3416n \). So, to prove 
\( 2019n + 1397 \leq Mn^2 \), it suffices to prove \( 3416n \leq Mn^2 \), i.e., 
\( 3416 \leq Mn \). This is always true assuming \( M = 1 \) and \( n \geq 3416 \) 
(\( n_0 = 3416 \)).

Setting \( M = 3416 \) and \( n_0 = 1 \) also work!
Little $o$ Notations

- Informally $f(n) = o(g(n))$ means $f$ is **asymptotically smaller than** $g$.

**Definition**

$$f(n) \in o(g(n)) \iff \forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) < M \cdot g(n)$$

ignore low-order terms
Little o Notations

- E.g., $f(n) = 2n$, $g(n) = n$. Is it that $f(n) \in o(g(n))$?
  - No because for $M = 1$, it is not true that $f(n) < Mg(n)$ (i.e., $2n < n$) for large values of $n$. 
Little o Notation

Prove that \( n^2 \sin(n) + 1984n + 2016 \in o(n^3) \).

- We have to prove that for all values of \( M \) there is an \( n_0 \) so that for \( n > n_0 \) we have \( n^2 \sin(n) + 1984n + 2016 < Mn^3 \).
- We know \( n^2 \sin(n) \leq n^2 \), \( 1984n \leq 1984n^2 \) and \( 2016 \leq 2016n^2 \). So, \( n^2 \sin(n) + 1984n + 2016 \leq (1 + 1984 + 2016)n^2 = 4001n^2 \).
- So, to prove \( n^2 \sin(n) + 1984n + 2016 < Mn^3 \) it suffices to prove \( 4001n^2 < Mn^3 \), i.e., \( 4001/M < n \), so, we can define \( n_0 \) to be any value larger than \( 4001/M \).

- For little \( o \), \( n_0 \) is often defined as a function of \( M \).
**Big \( \Omega \) Notation**

- \( f(n) = o(g(n)) \) means \( f \) is **asymptotically larger than or equal to** \( g \).

**Definition**

\[
f(n) \in \Omega(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \geq M \cdot g(n)
\]

- Let \( f(n) = n/2020 \) and \( g(n) = \log(n) \). Prove \( f(n) \in \Omega(g(n)) \).
  - We need to provide \( M \) and \( n_0 \) so that for all \( n \geq n_0 \) we have \( n/2020 \geq M \log(n) \), i.e., \( n \geq 2020M \log(n) \).
  - We know \( \log(n) < n \) (assuming \( n > 1 \)). So, in order to show \( 2020M \log(n) \leq n \), it suffices to have \( 2020M \leq 1 \), i.e., \( M \) can be any value smaller than \( 1/2020 \) (and \( n_0 \) can be 1 or any other positive integer).
Little $\omega$ Notation

- $f(n) = \omega(g(n))$ means $f$ is asymptotically larger than $g$.

**Definition**

$$f(n) \in \omega(g(n)) \iff \forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) > M \cdot g(n)$$

- Let $f(n) = n/2020$ and $g(n) = \log(n)$. Prove $f(n) \in \omega(g(n))$.
  
  - For any constant $M$ we need to provide $n_0$ so that for all $n \geq n_0$ we have $n/2020 > M \log(n)$, i.e., $n > 2020M \log(n)$.
  
  - We know $\log(n) < \sqrt{n}$ (assuming $n > 16$). So, in order to show $2020M \log(n) < n$, it suffices to have $2020M \sqrt{n} < n$, i.e., $2020M < \sqrt{n}$. For that, it suffices to have $(2020M)^2 < n$, i.e., $n_0$ can be defined as $\max\{16, (2020M)^2\}$.

- Similarly to little $o$, for $\omega$, we often need to define $n_0$ as a function of $M$. 
Θ Notation

- Informally $f(n) = \Theta(g(n))$ means $f$ is asymptotically equal to $g$.

**Definition**

$f(n) \in \Theta(g(n)) \iff \\
\exists M_1, M_2 > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, M_1 \cdot g(n) \leq f(n) \leq M_2 \cdot g(n)$

- Let $f(n) = n$ and $g(n) = n/2020$. Prove $f(n) \in \Theta(g(n))$.
  
  - We need to provide $M_1, M_2, n_0$ so that for all $n \geq n_0$ we have $M_1 \cdot n/2020 \leq n \leq M_2 \cdot n/2020$.
  - For the first inequality, we can have $M_1 = 1$ and for all $n$ we have $n/2020 \leq n$.
  - For the second inequality, we let $M_2$ to be any constant larger than 2020 which gives $M_2/2020 \geq 1$.
  - $n_0$ can be any value, e.g., $n_0 = 1$. 
Common Growth Rates

- \( \Theta(1) \) → constant complexity
  - e.g., an algorithms that only samples a constant number of inputs
- \( \Theta(\log n) \) → logarithmic complexity
  - Binary search
- \( \Theta(n) \) → linear complexity
  - Most practical algorithms :)
- \( \Theta(n \log n) \) → pseudo-linear complexity
  - Optimal comparison based sorting algorithms, e.g., merge-sort
- \( \Theta(n^2) \) → Quadratic complexity
  - naive sorting algorithms (Bubble sort, insertion sort)
- \( \Theta(n^3) \) → Cubic Complexity
  - naive matrix multiplication
- \( \Theta(2^n) \) → Exponential Complexity
  - The ‘algorithm’ terminates but the universe is likely to end much earlier even if \( n \approx 1000 \).