CSE 12 – Basic Data Structures
Prof. Christine Alvarado
[Slides borrowed/adapted from slides by Cynthia Lee]
Announcements

1. HW3 posted, code available tomorrow morning, but there’s plenty to do in the meantime
Faculty Coffee Hour
Wed April 16th 1:30pm - 3pm
Jacobs Hall 4th Floor, Room 4309

Get a chance to network with your CS professors over coffee provided by WIC. Talk about classes, ask for industry advice, or just have a quick chat over a cup of joe!
$f_1$ is $O(f_2)$

A. TRUE
B. FALSE

Why or why not?

In other words, for large $n$, can you multiply $f_2$ by a constant and have it always be bigger than $f_1$
f(n) = O(g(n)), if there are positive constants c and n₀ such that \( f(n) \leq c \cdot g(n) \) for all \( n \geq n₀ \).

- Obviously \( f_2 = O(f_1) \) because \( f_1 > f_2 \) (after about \( n=10 \), so we set \( n₀ = 10 \))
  - \( f_1 \) is clearly an *upper bound* on \( f_2 \) and that’s what big-O is all about
- But \( f_1 = O(f_2) \) as well!
  - We just have to use the “c” to adjust so \( f_2 \) that it moves above \( f_1 \)
f(n) = O(g(n)), if there are positive constants c and \( n_0 \) such that \( f(n) \leq c \times g(n) \) for all \( n \geq n_0 \).

- Obviously \( f_2 = O(f_1) \) because \( f_1 > f_2 \) (after about \( n=10 \), so we set \( n_0 = 10 \))
  - \( f_1 \) is clearly an upper bound on \( f_2 \) and that’s what big-O is all about
- But \( f_1 = O(f_2) \) as well!
  - We just have to use the “\( c \)” to adjust so \( f_2 \) that it moves above \( f_1 \)
f(n) = O(g(n)), if there are positive constants c and \( n_0 \) such that
\[
f(n) \leq c \cdot g(n)
\]
for all \( n \geq n_0 \).

- Obviously \( f_2 = O(f_1) \) because \( f_1 > f_2 \) (after about \( n=10 \), so we set \( n_0 = 10 \))
  - \( f_1 \) is clearly an upper bound on \( f_2 \) and that’s what big-O is all about
Common Big-O confusions when trying to argue that $f_2$ is $O(f_1)$:

- What if we multiply $f_2$ by a large constant, so that $c \cdot f_2$ is larger than $f_1$? Doesn’t that mean that $f_2$ is not $O(f_1)$?
  No, because we get to control the constants to our advantage, and only on $f_1$.

- What about when $n$ is less than 10? Isn’t $f_2$ larger than $f_1$?
  Remember, we get to pick our $n_0$, and only consider $n$ larger than $n_0$. 

![Graph showing f1 and f2 with c=1]
\( f_1 \) is \( \Omega(f_2) \)

A. TRUE  
B. FALSE

Why or why not?

Is \( f_2 \) always smaller than \( f_1 \) when \( f_2 \) is multiplied by some \( c \)?

\[ n = O(1) \quad 2 = O(n) \]
We say a function \( f(n) \) is "big-omega" of another function \( g(n) \), and write \( f(n) = \Omega(g(n)) \), if there are positive constants \( c \) and \( n_0 \) such that:

\[ f(n) \geq c \cdot g(n) \text{ for all } n \geq n_0. \]

In other words, for large \( n \), can you multiply \( g(n) \) by a constant and have it always be smaller than or equal to \( f(n) \).
\[ f_2 \text{ is } \Omega(f_1) \]

A. TRUE
B. FALSE

Why or why not?

In other words, for large \( n \), can you multiply \( f_1 \) by a positive constant and have it always be smaller than \( f_2 \)
\( f(n) = \mathcal{O}(g(n)) \), if there are positive constants \( c \) and \( n_0 \) such that \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \).

\( f(n) = \Omega(g(n)) \), if there are positive constants \( c \) and \( n_0 \) such that \( f(n) \geq c \cdot g(n) \) for all \( n \geq n_0 \).

- Obviously \( f_1 = \Omega(f_2) \) because \( f_1 > f_2 \) (after about \( n=10 \), so we set \( n_0 = 10 \))
  - \( f_2 \) is clearly a **lower bound** on \( f_1 \) and that’s what big-\( \Omega \) is all about
- But \( f_2 = \Omega(f_1) \) as well!
  - We just have to use the “\( c \)” to adjust so \( f_1 \) that it moves below \( f_2 \)
\[ f(n) = O(g(n)), \] if there are positive constants \( c \) and \( n_0 \) such that \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \).

\[ f(n) = \Omega(g(n)), \] if there are positive constants \( c \) and \( n_0 \) such that \( f(n) \geq c \cdot g(n) \) for all \( n \geq n_0 \).

- Obviously \( f_1 = \Omega(f_2) \) because \( f_1 > f_2 \) (after about \( n=10 \), so we set \( n_0 = 10 \))
  - \( f_2 \) is clearly a lower bound on \( f_1 \) and that’s what big-\( \Omega \) is all about
- But \( f_2 = \Omega(f_1) \) as well!
  - We just have to use the “\( c \)” to adjust so \( f_1 \) that it moves below \( f_2 \)
\[ f(n) = \mathcal{O}(g(n)), \text{ if there are positive constants } c \text{ and } n_0 \text{ such that } f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0. \]

\[ f(n) = \Omega(g(n)), \text{ if there are positive constants } c \text{ and } n_0 \text{ such that } f(n) \geq c \cdot g(n) \text{ for all } n \geq n_0. \]

\( f_1 \text{ is } \mathcal{O}(f_3) \)

A. TRUE

B. FALSE

Why or why not?

\( f_3 \text{ is bigger than } f_1 \).
\[ f(n) = \mathcal{O}(g(n)), \text{ if there are positive constants } c \text{ and } n_0 \text{ such that } f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0. \]

\[ f(n) = \Omega(g(n)), \text{ if there are positive constants } c \text{ and } n_0 \text{ such that } f(n) \geq c \cdot g(n) \text{ for all } n \geq n_0. \]

\[ f_3 \text{ is } \mathcal{O}(f_1) \]

A. TRUE
B. FALSE

Why or why not?
\[ f_1 = O(f_3) \text{ but } f_3 \neq O(f_1) \]

- There is no way to pick a \( c \) that would make an \( O(n) \) function \((f_1)\) stay above an \( O(n^2) \) function \((f_3)\).

\[ n = O(n^{100}) \ ? \text{ yes} \]
\[ n^{100} = O(n) \ ? \text{ no} \]
f(n) = \( \mathcal{O}(g(n)) \), if there are positive constants \( c \) and \( n_0 \) such that \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \).

f(n) = \( \Omega(g(n)) \), if there are positive constants \( c \) and \( n_0 \) such that \( f(n) \geq c \cdot g(n) \) for all \( n \geq n_0 \).

\( f_3 \) is \( \Omega(f_1) \)

A. TRUE
B. FALSE

Why or why not?
\( f(n) = \Omega(g(n)) \), if there are positive constants \( c \) and \( n_0 \) such that \( f(n) \geq c \cdot g(n) \) for all \( n \geq n_0 \).

\[ f(n) = \Omega(f_3) \]

A. TRUE
B. FALSE

Why or why not?
\[ f_3 = \Omega(f_1) \text{ but } f_1 \neq \Omega(f_3) \]

- There is no way to pick a \( c \) that would make an \( O(n^2) \) function (\( f_3 \)) stay below an \( O(n) \) function (\( f_1 \)).
Summary

**Big-O**

- **Upper bound** on a function
  
  $f(n) = O(g(n))$ means that we can expect $f(n)$ will always be **under** the bound $g(n)$
  
  But we don’t count $n$ up to some starting point $n_0$

  And we can “cheat” a little bit by moving $g(n)$ up by multiplying by some constant $c$

**Big-Ω**

- **Lower bound** on a function
  
  $f(n) = \Omega(g(n))$ means that we can expect $f(n)$ will always be **over** the bound $g(n)$
  
  But we don’t count $n$ up to some starting point $n_0$

  And we can “cheat” a little bit by moving $g(n)$ down by multiplying by some constant $c$
Tight bound on a function.

If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then $f(n) = \theta(g(n))$.

Basically it means that $f(n)$ and $g(n)$ are interchangeable.

Examples:

- $3n + 20 = \theta(10n + 7)$

- $5n^2 + 50n + 3 = \theta(5n^2 + 100)$
$f_1$ is $\Theta(f_2)$

A. TRUE
B. FALSE

Why or why not?
\( f_1 \) is \( \Theta(f_2) \)

A. TRUE  
B. FALSE

Why or why not?

Since \( f_1 \) is \( O(f_2) \) and \( \Omega(f_2) \), it is also \( \Theta(f_2) \) (this is the definition of big-Theta)
\( f_1 \) is \( \Theta(f_3) \)

A. TRUE

B. FALSE

Why or why not?
Big-\(\Theta\) and sloppy usage

- Sometimes people say, “This algorithm is \(O(n^2)\)” when it would be more precise to say that it is \(\Theta(n^2)\)
  - They are intending to give a tight bound, but use the looser “big-O” term instead of the “big-\(\Theta\)” term that actually means tight bound
  - Not wrong, but not as precise
- I don’t know why, this is just a cultural thing you will encounter among computer scientists

\[\text{May alg runs in } n \text{ steps which is } O(n)\]
Shortcuts for calculating

Big-O analysis starting with a function characterizing the growth in cost of the algorithm
Let \( f(n) = 3 \log_2 n + 4n \log_2 n + n \)

Shortcut: find the highest "power" of \( n \).

Which of the following is true?

- [X] \( f(n) = O(\log_2 n) \)
- [B] \( f(n) = O(n \log_2 n) \)
- [C] \( f(n) = O(n^2) \)
- [X] \( f(n) = O(n) \)
- [E] Other/none/more

\[ \log_2 n < n < \log_2 n \times n < n \times n \times n^2 \]
Let $f(n) = 546 + 34n + 2n^2$

Which of the following is true?

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^3)$
E. Other/none/more
Let \( f(n) = 2^n + 14n^2 + 4n^3 \)

Which of the following is true?

- A. \( f(n) = O(2^n) \)
- B. \( f(n) = O(n^2) \)
- C. \( f(n) = O(n) \)
- D. \( f(n) = O(n^3) \)
- E. Other/none/more
Let $f(n) = 100$

Which of the following is true?

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^{100})$
E. Other/none/more
Extracting time cost from example code

Algorithm analysis starting with the algorithm
A student has counted how many times we perform each line of code

Is the count $3n+5$:

A. the best case?
B. the worst case?
C. the average case?
D. Other/none/more

$f(n) = 3n+5 = \mathcal{O}(n)$