DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120
GROWTH FUNCTIONS

AGENDA

- Growth functions
- Asymptotic notations

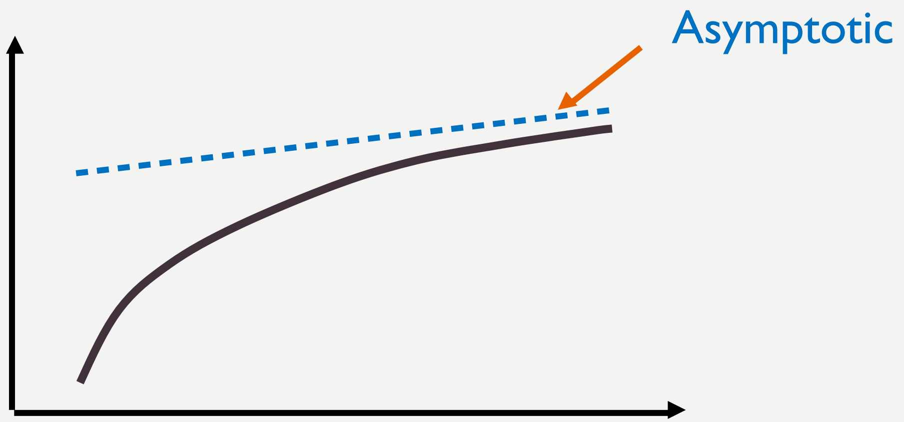
ORDER OF GROWTH

- Recall
  - Order of growth describes the **rate of growth** in the running time as the input size increases.

- Some running time functions do not grow indefinitely.
- Use **growth functions** to express the bounds (upper and lower) of the running time function.
  - Used in conjunction with an ***asymptotic notation***.

ASYMPTOTIC NOTATIONS

- ***Asymptotic*** is a line that approaches a curve but never touches.



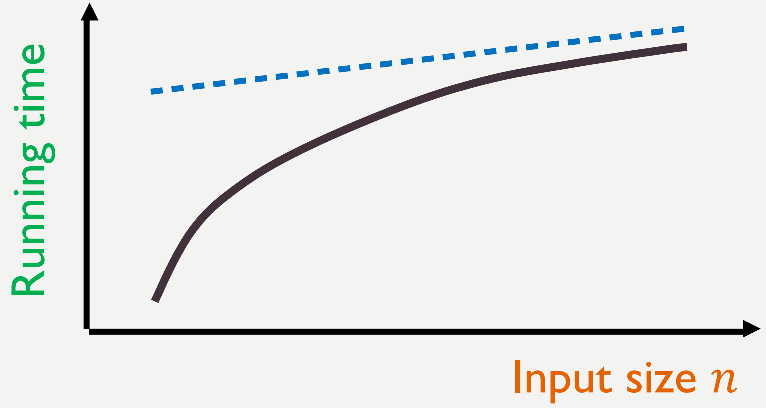
  - In the world of algorithms, an asymptotic can be another curve.

ASYMPTOTIC EFFICIENCY

- How the running time increases with the size of the input *in the limit* (***asymptotic***), as the size of input increases

- Usually, an algorithm that is asymptotically more efficient will be the best choice for all *but very small* inputs.

- **Asymptotic notations** help us express that efficiency.



ASYMPTOTIC NOTATIONS

- Commonly used asymptotic notations
  - The Θ-notation (the big-theta notation)
  - The O-notation (the big-oh notation)
  - The Ω-notation (the big-omega notation)

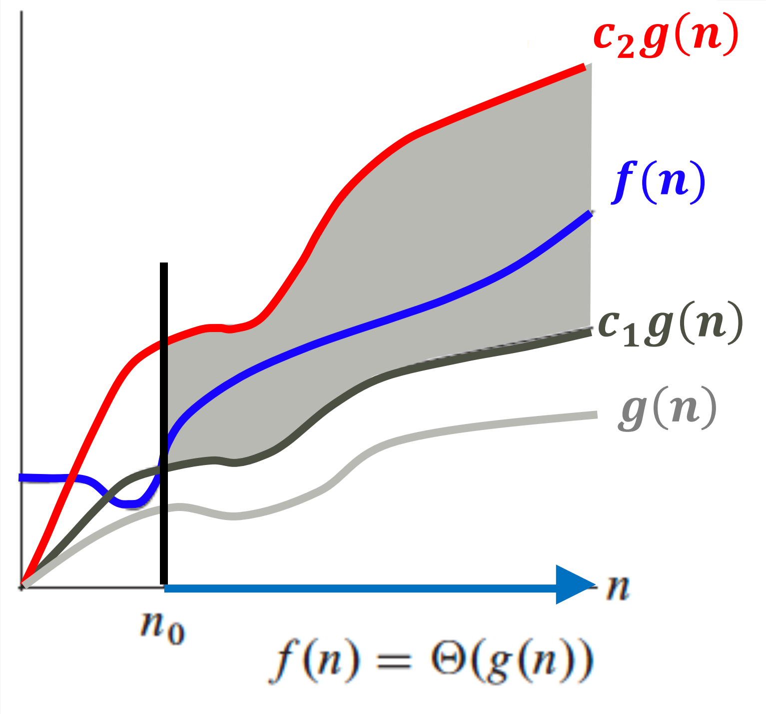
THE Θ-NOTATION
THE BIG-THETA NOTATION

- **Formal definition**
  - For a given function $g(n)$, we denote by $\Theta(g(n))$ the *set of functions* $\Theta(g(n)) = \{f(n)$: there exist **positive constants c_1, c_2, and n_0** such that $\mathbf{0 \le c_1 g(n) \le f(n) \le c_2 g(n)}$ **for all $n \ge n_0$**$\}$
    - n_0 is the minimum possible value that makes the inequality hold.

- The function $g(n)$ is an asymptotic **tight bound** for $f(n)$.

THE Θ-NOTATION
GRAPHICAL INTERPRETATION

- Graphic example of $f(n) = \Theta(g(n))$
  - $f(n)$ is "sandwiched" by $c_1(g(n))$ and $c_2(g(n))$
  - n_0 is the minimum possible value that makes the inequality hold.
    - Any greater value would also work.

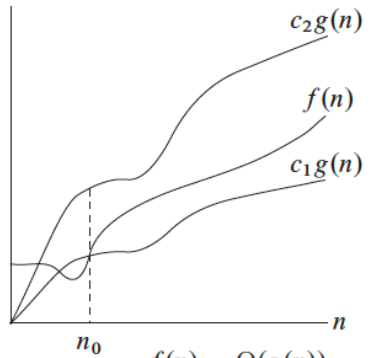


THE Θ-NOTATION PRACTICE

- Complete the defitinion of Θ-notation.
  - For a given function $g(n)$, we denote by ____________ the *set of functions*

    ____________ = { _______: there exist ______________________________ c_1, c_2, and ________

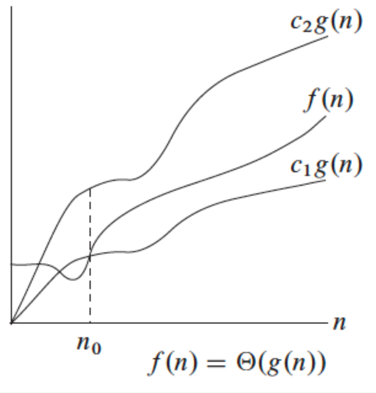
    such that ____ for all ______________}



$f(n) = \Theta(g(n))$

THE Θ-NOTATION
PRACTICE

- Consider the following asymptotic notations $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. Identify the $f(n)$ and $g(n)$ in the definition.
  - $f(n) =$ ______________________________
  - $g(n) =$ ______________________________



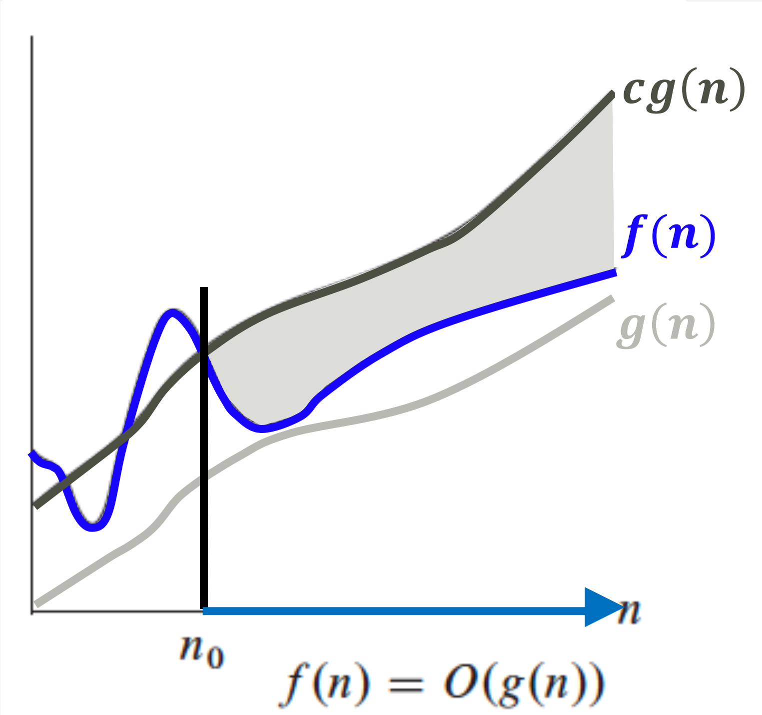
THE O-NOTATION
THE BIG-OH NOTATION

- **Formal definition**
  - For a given function $g(n)$, we denote by $O(g(n))$ the *set of functions* $O\big(g(n)\big) = \{f(n)$: there exist **positive constants c and n_0** such that $\mathbf{0 \le f(n) \le cg(n)}$ for all $\boldsymbol{n \ge n_0}\}$.
    - n_0 is the minimum possible value that makes the inequality hold.

- We say that $g(n)$ is an asymptotic **upper bound** for $f(n)$.

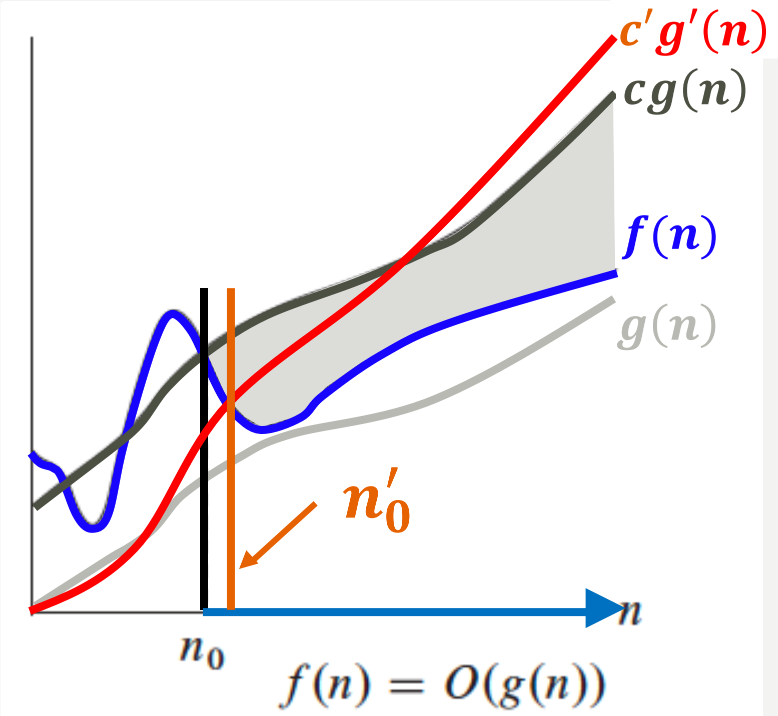
THE O-NOTATION GRAPHICAL INTERPRETATION

- Graphic example of $f(n) = O(g(n))$
  - $f(n)$ is **upper bounded** by $g(n)$ for a sufficiently large n.
    - The value of function $f(n)$ is **on or below** $cg(n)$.
    - Any greater value would also work.



THE O-NOTATION
SIDE NOTE

- The O-notation does NOT necessarily mean **an asymptotic tight upper bound** of a given function.
  - Graphic example
    - $f(n) = O(g'(n)), f(n) = O(g(n))$,
    - Compared with $g(n)$, $g'(n)$ is a relatively ***loose upper bound*** of $f(n)$ with a different choice of constant c' and n_0'.



THE O-NOTATION PRACTICE

- Explain why the statement, "The running time of algorithm A is at least $O(n^2)$," is meaningless.

THE O-NOTATION PRACTICE

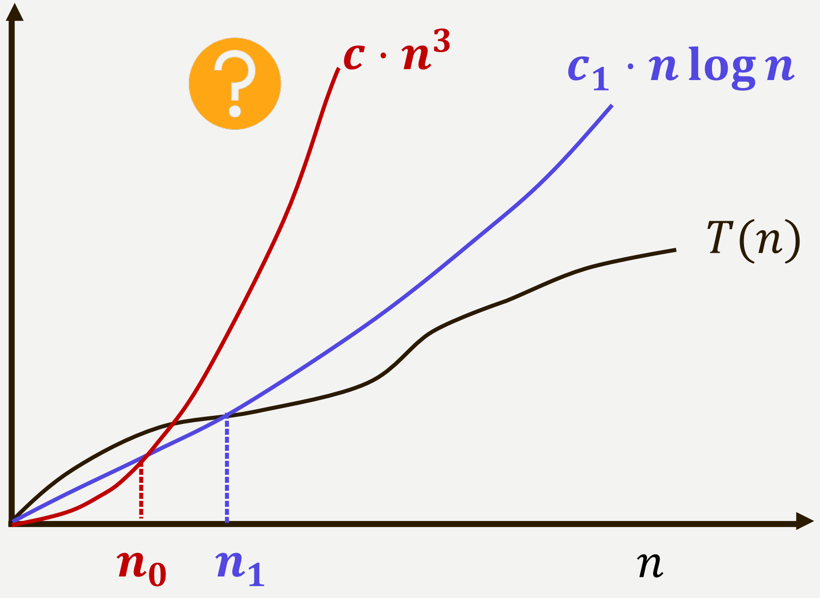
- Explain why the statement, "The running time of algorithm A is at least $O(n^2)$," is meaningless.

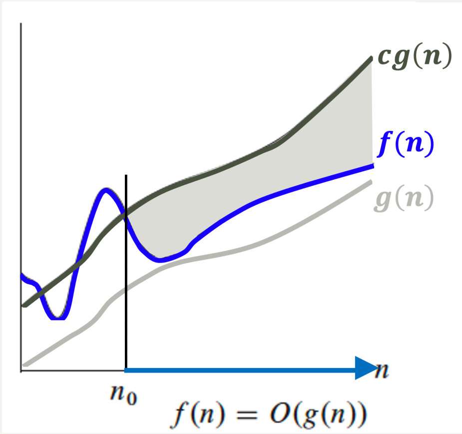
  The O-notation describes a set of functions $O\big(g(n)\big)$.
  Let the running time of algorithm A be denoted by $T(n)$.
  We can rephrase the statement as follows: "$T(n) \geq O(n^2)$."
  Let $f(n) = O(n^2)$, meaning that there exist positive constants c and n_0 such that
  ______________________________ for all _________________.
  By choosing $f(n) = c^*$, where c^* is a positive constant, we can further rephrase the statement as follows: "________________________________," which is meaningless, because any algorithm runs in at least_______________________________________.

THE O-NOTATION PRACTICE

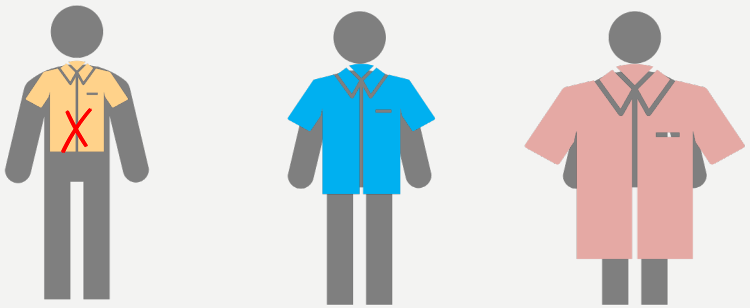
- If the running time of algorithm A, denoted by $T(n)$, is $O(n \lg n)$, is $T(n) = O(n^3)$?
  - Understand the problem from a graphical view.

What do c and n_0 look like?





THE O-NOTATION PRACTICE SUMMARY



- If the running time of algorithm A, denoted by $T(n)$, is $O(n \lg n)$, is $T(n) = O(n^3)$?

By the definition of O-notation, $T(n) = O(n \lg n)$ means that there exist __________ c_1 and n_1 such that ____________________, for all __________ (Inequality #1). Obviously, $n \lg n = O(n^3)$. There exist __________ c_2 and n_2 such that ________________________, for all __________ (Inequality #2) .

Multiply the inequality #2 by ___. We have __________________ under the constraints that __________ (Inequality #3). Combine inequality #3 and #1. We have _________________, under the constraints that _____________ (Inequality #4). We can optimize the range of n as ________________.

Let us get rid of the middle-man in inequality #4 and simplify the inequality as ___________________. Let c be _____, n_0 be ______. Both c and n_0 are ___________. We can re-write the inequality as ________________ for all _________.
Therefore, based on the definition of the Big-O notation $T(n) = O(n^3)$.

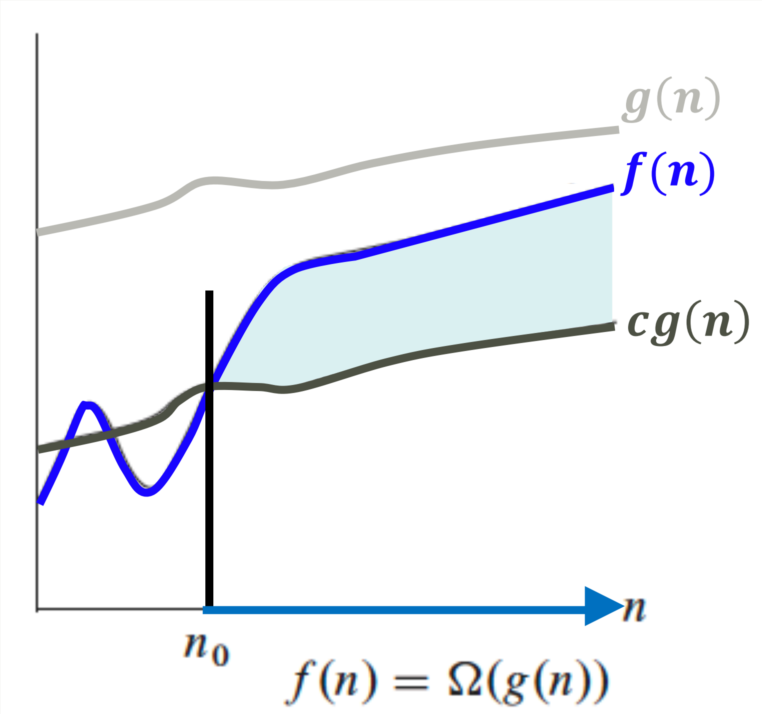
THE Ω-NOTATION
THE BIG-OMEGA NOTATION

- **Formal definition**
  - For a given function $g(n)$, we denote by $\Omega(g(n))$ the *set of functions* $\Omega\big(g(n)\big) = \{f(n)$: there exist **positive constants c and n_0** such that $\mathbf{0 \le cg(n) \le f(n)}$ for all $\boldsymbol{n \ge n_0}\}$

- We say that $g(n)$ is an asymptotic **lower bound** for $f(n)$.

THE Ω-NOTATION GRAPHICAL INTERPRETATION

- Graphic example of $f(n) = \Omega(g(n))$
  - $f(n)$ is **lower bounded** by $g(n)$ for a sufficiently large n.
    - The value of function $f(n)$ is **on or above** $cg(n)$.
  - n_0 is the minimum possible value that makes the inequality hold.
    - Any greater value would also work.



ASYMPTOTIC NOTATIONS

- Asymptotic **tight** bound (Θ), **upper** bound (O), and **lower** bound (Ω).

- Abusing the equal ($=$) sign
  - When we say $f(n) = O\big(g(n)\big)$, we are merely claiming that some constant multiple of $g(n)$ is **an** asymptotic upper bound of $f(n)$, with no claim about how tight an upper bound it is.
  - Similarly, $f(n) = \Omega(g(n))$ means that $g(n)$ is **an** asymptotic lower bound of $f(n)$,

STANDARD FUNCTIONS EXPONENTIALS

- For all real $a > 0, m,$ and $n,$ we have the following identities
  - $a^0 = 1$
  - $a^1 = a$
  - $a^{-1} = 1/a$
  - $(a^m)^n = a^{mn}$
  - $(a^n)^m = a^{nm}$
  - $a^m a^n = a^{m+n}$

STANDARD FUNCTIONS
LOGARITHMS

- We shall use the following notations
  - $\lg n = \log_2 n$ (binary logarithm)
  - $\ln n = \log_e n$ (natural logarithm)
  - $\lg^k n = (\lg n)^k$ (exponentiation)
  - $\lg \lg n = \lg(\lg n)$ (composition)
- If we hold $b > 1$ constant, then for $n > 0$, the function $\log_b n$ is strictly increasing.
- **In CS, $\lg n$ is equivalent to $\log_2 n$.**

- For all real $a > 0, b > 0, c > 0$ and n,
  - $a = b^{\log_b a}$
  - $\log_c(ab) = \log_c a + \log_c b$
  - $\log_b a^n = n \log_b a$
  - $\log_b(1/a) = -\log_b a$
  - $\log_b a = \frac{\log_c a}{\log_c b}, \log_b a = \frac{1}{\log_a b}$
  - $a^{\log_b c} = c^{\log_b a}$

STANDARD FUNCTIONS
POLYNOMIALS

- Given a nonnegative integer d, a **polynomial in n of degree d** is a function $p(n)$ of the form

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

  where the constants $a_0, a_1, \ldots, a_d$ are the **coefficients** of the polynomial and $a_d \neq 0$.

- A polynomial is ***asymptotically positive*** if and only if $a_d > 0$.
- For an asymptotically positive polynomial $p(n)$ of degree d, we have $\boldsymbol{p(n) = \Theta(n^d)}$.

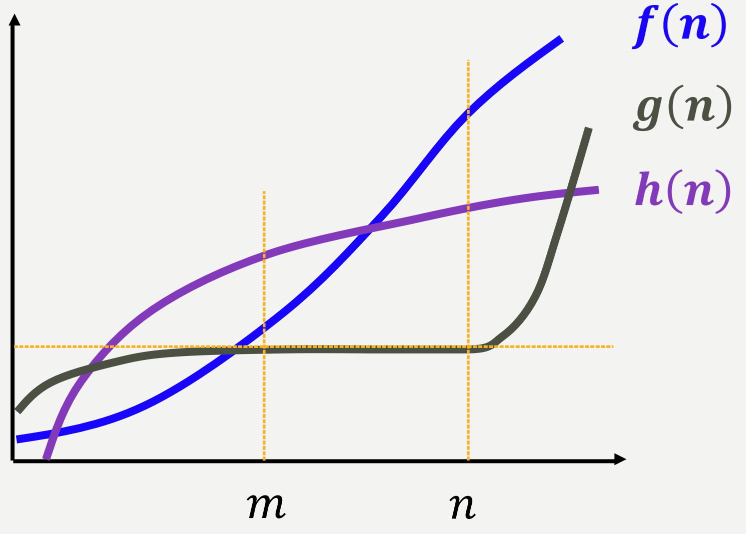
STANDARD FUNCTIONS
FLOORS AND CEILINGS

- Floors
  - For any real number x, we denote the greatest integer less than or equal to x by $\lfloor x \rfloor$.
    - $\lfloor 3.5 \rfloor = 3, \lfloor 3 \rfloor = 3.$
- Ceilings
  - For any real number x, we denote the least integer greater than or equal to x by $\lceil x \rceil$.
    - $\lceil 3.5 \rceil = 4, \lceil 3 \rceil = 3.$

- For all real x, $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1.$
- For any integer n, $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n.$
- For any real number $x \geq 0$ and integers $a, b > 0$,
  - $\left\lceil \frac{\lceil x/a \rceil}{b} \right\rceil = \left\lceil \frac{x}{ab} \right\rceil, \left\lfloor \frac{\lfloor x/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{x}{ab} \right\rfloor.$
- Both the floor function $f(x) = \lfloor x \rfloor$ and the ceiling function $f(x) = \lceil x \rceil$ are increasing.

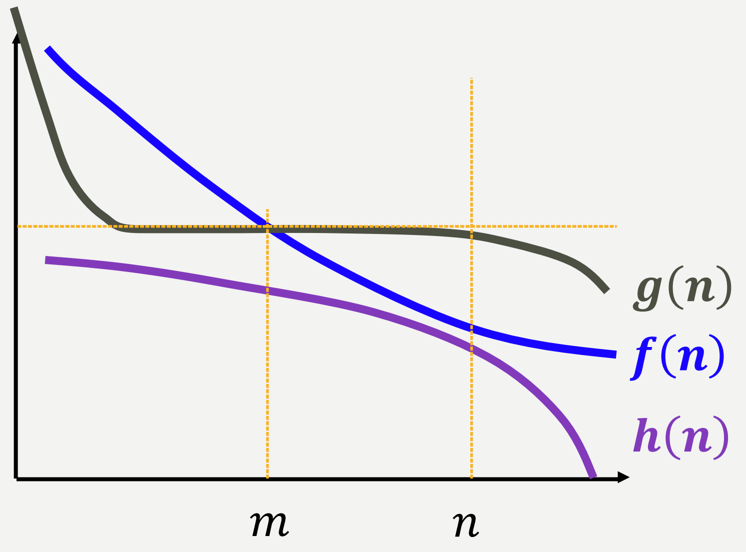
STANDARD FUNCTIONS
MONOTONICITY

- A function $f(n)$ is **monotonically increasing** if $m \leq n$ implies $f(m) \leq f(n)$.
  - A function $f(n)$ is **strictly increasing** if $m \leq n$ implies $f(m) < f(n)$.



STANDARD FUNCTIONS
MONOTONICITY

- A function $f(n)$ is **monotonically decreasing** if $m \le n$ implies $f(m) \ge f(n)$.
  - A function $f(n)$ is **strictly decreasing** if $m \le n$ implies $f(m) > f(n)$.



BOUNDING FUNCTIONS

- When bounding a given function, we focus only on the leading term.
- Any **exponential function** with a base strictly greater than 1 **grows faster than any polynomial function**.
- For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta\big(g(n)\big)$ ***iff*** (if and only if) $f(n) = O\big(g(n)\big)$ and $f(n) = \Omega\big(g(n)\big)$.
- 2^{500} (or any constant number) $< \lg(\lg n)^2 < \lg n < \log_4 n < \log^3 n < \sqrt{n} < 2^{\lg n} < n \cdot \log n \ < n^2 \lg n < n^2 \lg^5 n < n^3 < 2^n < n!$

STANDARD FUNCTIONS
PRACTICE

A. Rewrite the following expression in the form of one number, or a single exponentiation/logarithm with a possible coefficient.

- $8^3 \cdot (2^8)^{1/2}$
  - Hint: $(a^m)^n = a^{mn}, a^m a^n = a^{m+n}$
- $\log_4 27 \cdot \log_3 4$
  - Hint: $\log_a b = \frac{1}{\log_b a}, log_b a = \frac{log_c a}{log_c b}$
- $n^{\frac{1}{\log_m n}}$
  - Hint: $log_b a = \frac{1}{log_a b}, log_b a^n = n \, log_b a$

NEXT UP
DESIGN TECHNIQUE

- Divide and Conquer

REFERENCE

- https://www.youtube.com/watch?v=SEbzTe0CzT8
- https://www.yourdictionary.com/asymptotic#:~:text=adjective,are%20asymptotic%20to%20each%20other.