Asymptotic Notation, Review of Functions & Summations

*

Asymptotic Complexity

- Running time of an algorithm as a function of input size n **for large *n***.
- Expressed using only the **highest-order term** in the expression for the exact running time.
  - Instead of exact running time, say $\Theta(n^2)$.
- Describes behavior of function in the limit.
- Written using ***Asymptotic Notation***.

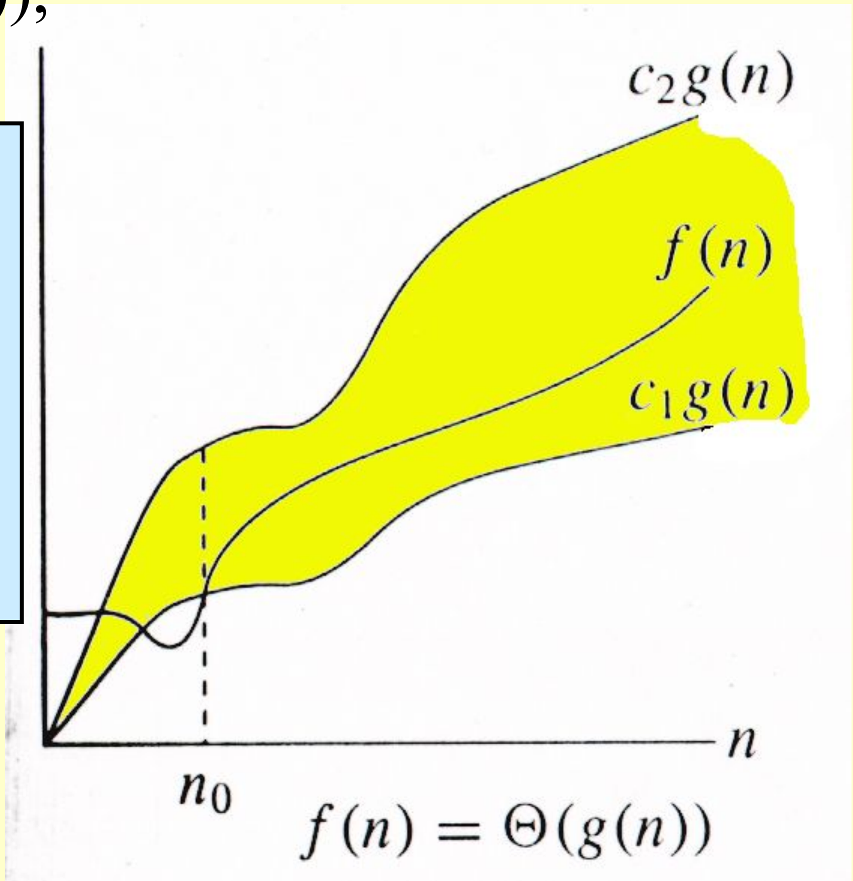
Asymptotic Notation

- **Θ, O, Ω, o, ω**
- Defined for functions over the natural numbers.
  - **Ex:** $f(n) = \Theta(n^2)$.
  - Describes how $f(n)$ grows in comparison to n^2.
- Define a ***set*** of functions; in practice used to compare two function sizes.
- The notations describe different rate-of-growth relations between the defining function and the defined set of functions.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n)$:
$\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$



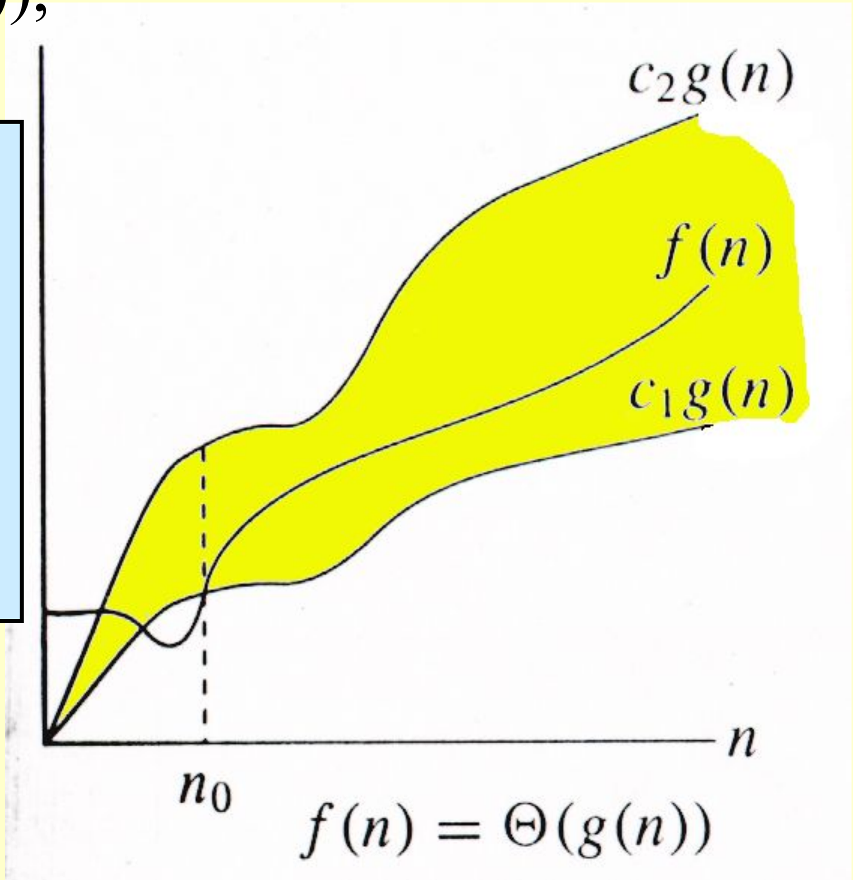
Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.

$g(n)$ is an *asymptotically tight bound* for $f(n)$.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n)$:
$\exists$ **positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,**
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$



Technically, $f(n) \in \Theta(g(n))$.
Older usage, $f(n) = \Theta(g(n))$.
I'll accept either…

***f*(*n*) and *g*(*n*) are nonnegative, for large *n*.**

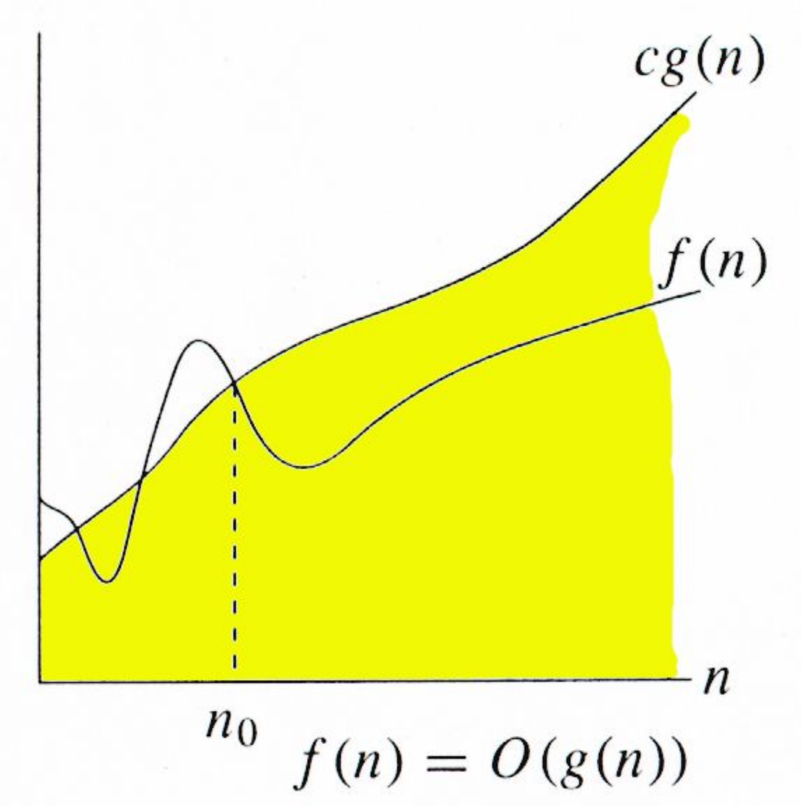
Example

$\Theta(g(n)) = \{f(n) :$ $\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

- $10n^2 - 3n = \Theta(n^2)$
- What constants for n_0, c_1, and c_2 will work?
- Make c_1 a little smaller than the leading coefficient, and c_2 a little bigger.
- ***To compare orders of growth, look at the leading term.***
- Exercise: Prove that $n^2/2-3n= \Theta(n^2)$

O-notation

For function $g(n)$, we define $O(g(n))$, big-O of n, as the set:



$O(g(n)) = \{f(n) :$
$\exists$ positive constants c and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq f(n) \leq cg(n)$ $\}$

Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.

$g(n)$ is an *asymptotic upper bound* for $f(n)$.

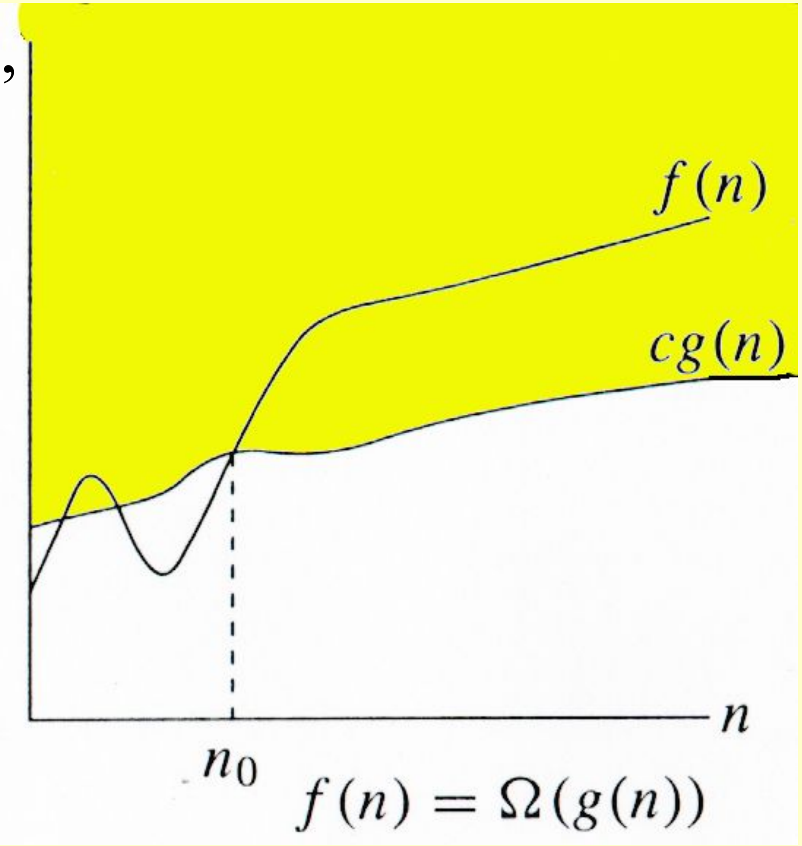
$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.

$\Theta(g(n)) \subset O(g(n))$.

Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
$\exists$ positive constants c and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq cg(n) \leq f(n)\}$

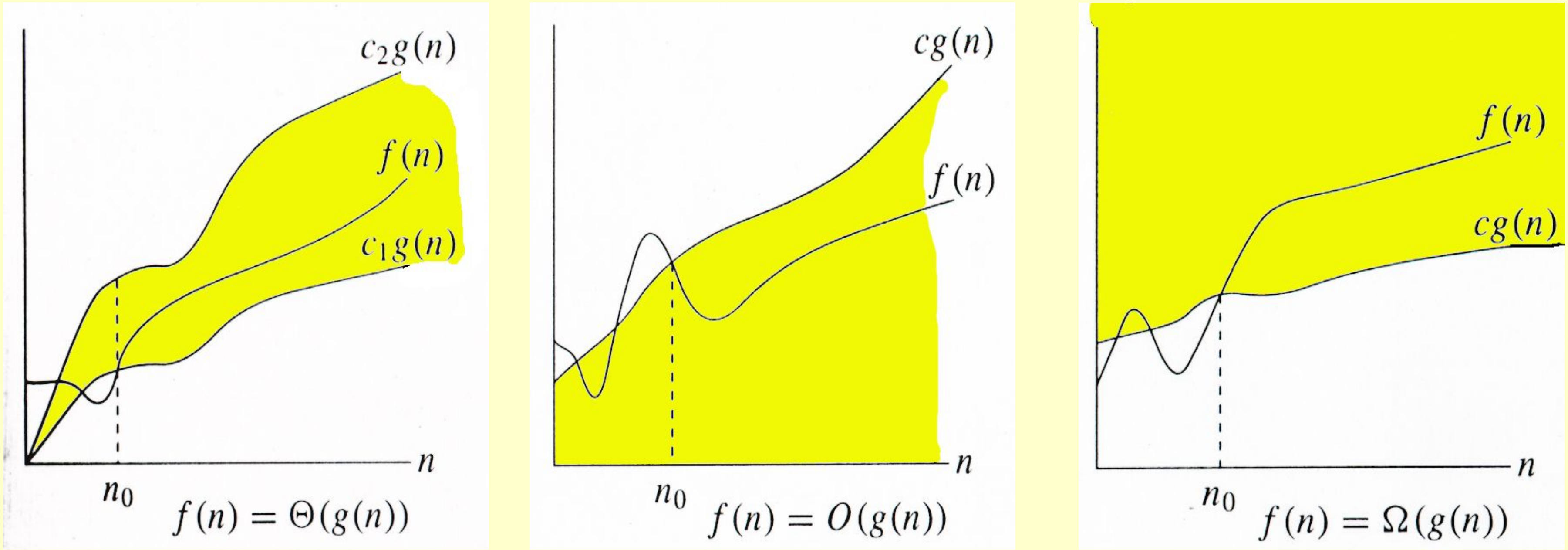


Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.

$g(n)$ is an *asymptotic lower bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$.
$\Theta(g(n)) \subset \Omega(g(n))$.

Relations Between Θ, O, Ω



Let us consider the problem of preparing an omelet. For preparing omelet, general steps we follow are:

1) Get the frying pan.
2) Get the oil.
    a. Do we have oil?
        i. If yes, put it in the pan.
        ii. If no, do we want to buy oil?
            1. If yes, then go out and buy.
            2. If no, we can terminate.
3) Turn on the stove, etc...

What we are doing is, for a given problem (preparing an omelet), giving step by step procedure for solving it. Formal definition of an algorithm can be given as:

An algorithm is the step-by-step instructions to solve a given problem.

Note: we do not have to prove each step of the algorithm.

Definition

An algorithm is any **well-defined computational procedure** that takes some values or set of values as **input** and produces some values or set of values as **output**

Definition

A sequence of **computational steps** that transforms the **input** into **output**

Algorithms

Properties of algorithms:

- **Input** An algorithm has zero or more inputs,
- **Output** An algorithm produces at least one output.
- **Clear and Unambiguous**: The algorithm should be clear and unambiguous
- **Definiteness** of every step in the computation, Every fundamental operator in instruction must be defined without any ambiguity.
- **Language Independent:** The Algorithm designed must be language-independent

Algorithms

Properties of algorithms:

- **Definiteness** of every step in the computation, Every fundamental operator in instruction must be defined without any ambiguity.
- **Correctness** of output for every possible input,
- **Finiteness** An algorithm must terminate after a finite number of steps in all test cases. Every instruction which contains a fundamental operator must be terminated within a finite amount of time.
- **Effectiveness** An algorithm must be developed by using very basic, simple, and feasible operations.

The Growth of Functions

"Popular" functions $g(n)$ are
$n.\log n$, 1, 2^n, n^2, $n!$, n, n^3, $\log n$

Increasing rates of growth:

- 1
- $\log n$
- n
- $n \log n$
- n^2
- n^3
- 2^n
- $n!$