COMP 3711 Design and Analysis of Algorithms

Course Mechanics
Review of Asymptotic Notations

Topics Covered (tentative)

- Techniques (with multiple examples)
  - Divide & Conquer
  - Greedy Algorithm Design
  - Dynamic Programming
- Sorting
- Basic Randomized Algorithms
- Graph Algorithms
  - Breadth & Depth First Search
  - Shortest Paths
  - Spanning Trees
  - Maximum Flow and Bipartite Matchings
- Extra topics (if time available)
  - AVL Trees
  - Basic String Matching
  - Hashing

About the Class

- What this class is
  - Learning algorithmic techniques to solve problems.
  - Learning how to prove correctness of algorithms.
  - Learning how to model problems.
- What this class isn't
  - Coding.
  - Language specific programming hacks.

Prerequisites

- COMP2711 – Discrete Math
  - In particular
    - $O(\,)$ notation
    - Basic Formulas, e.g., formulas for $\sum i$, $\sum i^2$, geometric series
    - Basic Combinatorics, e.g, $\binom{n}{2} = \Theta(n^2)$

- COMP 2011 – Basic Data Structures
  - In particular
    - Linked Lists (singly and doubly)
    - Stacks and Queues
    - Binary (Search) Trees

Accessing Course Material

- There are separate canvas pages for COMP3711 L1 and L2.
- Assignments will be given out via canvas. All assignments must be submitted online via canvas. Hard copies will not be accepted. Email submissions will not be accepted. Late submissions will not accepted without prior approval.
- All classes and tutorials will be conducted face to face.
  - Links to lecture videos will be posted on the canvas course page.

Tentative Grading Scheme

- 30% Written Assignments
  - 4 written assignments
    - Assignments are a combination of problems that require you to work through taught algorithms on given input and problems that require you to design new algorithms
  - Submission will be online via Canvas. You can typeset your solution, or scan your handwritten solutions, take pictures of your handwritten solutions.
- 30% Midterm
- 40% Final Exam
  - Will cover the entire semester's material.

Office Hours

- You are welcome to ask your question in class.
- You can also email your question to the instructor at scheng@cse.ust.hk.
- You are also welcome to set up a meeting with the instructor via email (physical or zoom meeting).

Course Policies

- Making up Missed Midterm or Final Exam
  - Only for medical reasons (with Doctor's note)
  - OR prior approval of instructor, e.g., need to be away for academic competition
- Assignments
  - Plagiarism: While collaboration **is** allowed,
    - Assignment solutions must be written in your own words (not copied or slightly modified from some other sources).
    - You must understand your solution and its derivation. (We may ask you to explain it.)
    - You must explicitly acknowledge in writing in the assignment your collaborators (whether or not they are classmates) or any other outside sources on each assignment.
    - Failing to do any of these will be considered plagiarism, and may result in a failing grade in the course and notification for appropriate disciplinary action.

Review Material - Input Size of Problems

The input size indicates how large the input is. Since we assume that any number can be stored in a computer word and each arithmetic operation takes constant time, we take the number of items in the input as the input size (instead of bits).

By an item, we mean something that can be represented by a number and so it can be stored using one computer word.

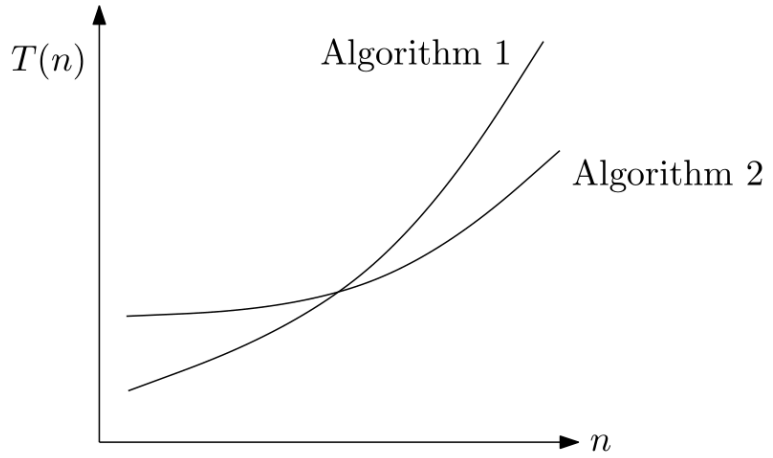
Examples:

Sorting	Size of the list or array
Graph problems	Numbers of vertices and edges
Searching	Number of input keys

Measuring Running Time using Asymptotic Notation

Throughout the class we will measure the running time/cost of algorithms

- as a function of input size: $T(n)$
- using the number of instructions/steps/operations (e.g., comparisons between two numbers)
- using asymptotic notation, which ignores constants and non-dominant growth terms.



Which algorithm is better for large n?

- For Algorithm 1, $T(n) = 3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$
- For Algorithm 2, $T(n) = 7n^2 - 8n + 20 = \Theta(n^2)$
- Clearly, Algorithm 2 is better for large inputs

Behavior of some Commonly Encountered Functions

n	$\log_2 n$
2	1
32	5
1,024	10
1,048,576	20
1,073,741,824	30

n	$n \log_2 n$
32	160
1,024	10,240
32,768	491,520
1,048,576	20,971,520

n	n^2
10	100
100	10,000
1,000	1,000,000
1,000,000	1,000,000,000,000

Sorting Example: 2Hz CPU

Population	Insertion Sort n^2	Merge Sort $n \log_2 n$
Macau (0.5M)	~2 min	0.0048s
HK (8M)	~9 hr	0.1s
China (1357M)	~29 yr	~21s

Asymptotic Notation: Quick Revision

Upper bounds. $T(n) = O(f(n))$

if exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, $T(n) \leq c \cdot f(n)$.

Lower bounds. $T(n) = \Omega(f(n))$

if exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, $T(n) \geq c \cdot f(n)$.

Tight bounds. $T(n) = \Theta(f(n))$

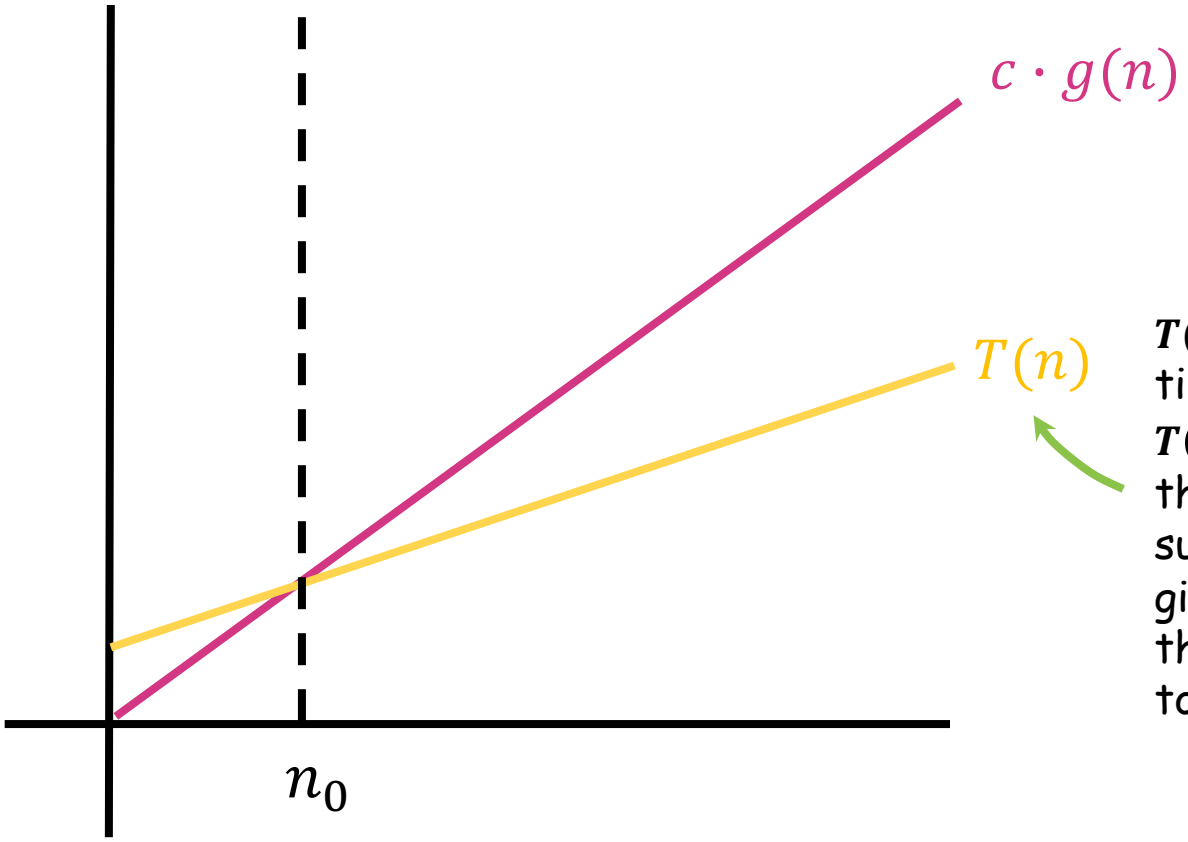
if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$.

Note: Here "=" means "is", not equal.

Big-O Notation

Big-O notation is a mathematical notation for upper-bounding a function's rate of growth.

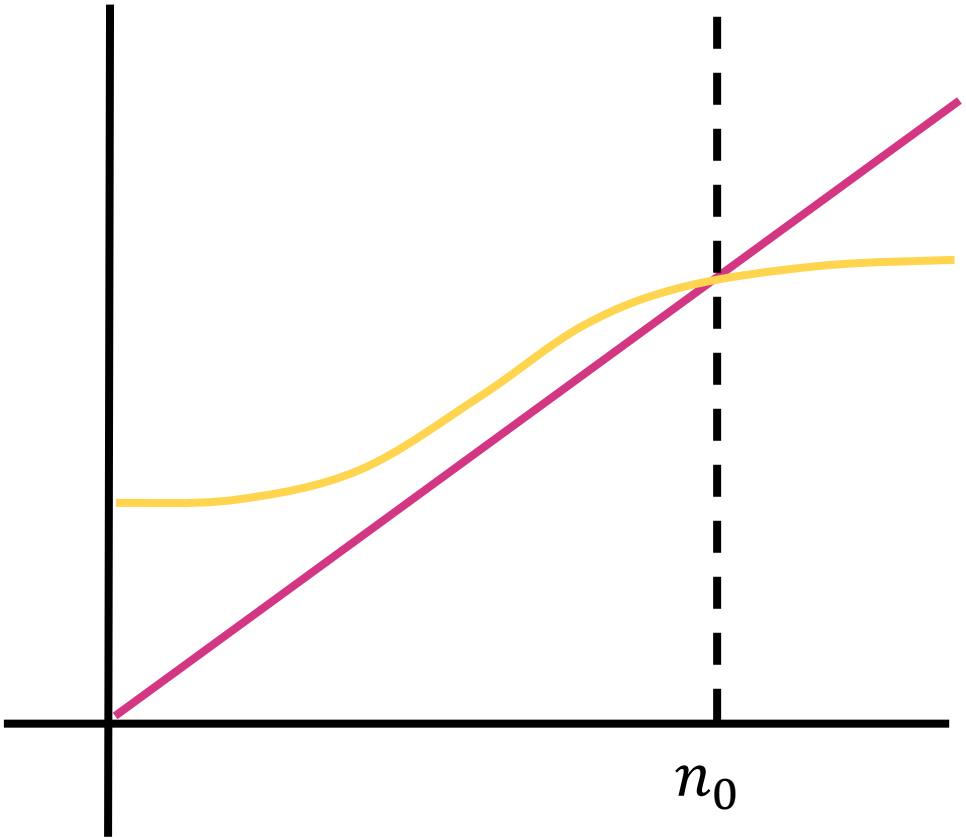
$$T(n) = O(g(n))$$
$$\text{iff}$$
$$\exists\, c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



$T(n)$ describes the running time of an algorithm. $T(n) = O(g(n))$ means that there exists some c and n_0 such that the pink line given by $c \cdot g(n)$ is **above** the yellow line for all values to the right of n_0.

Big-O Notation

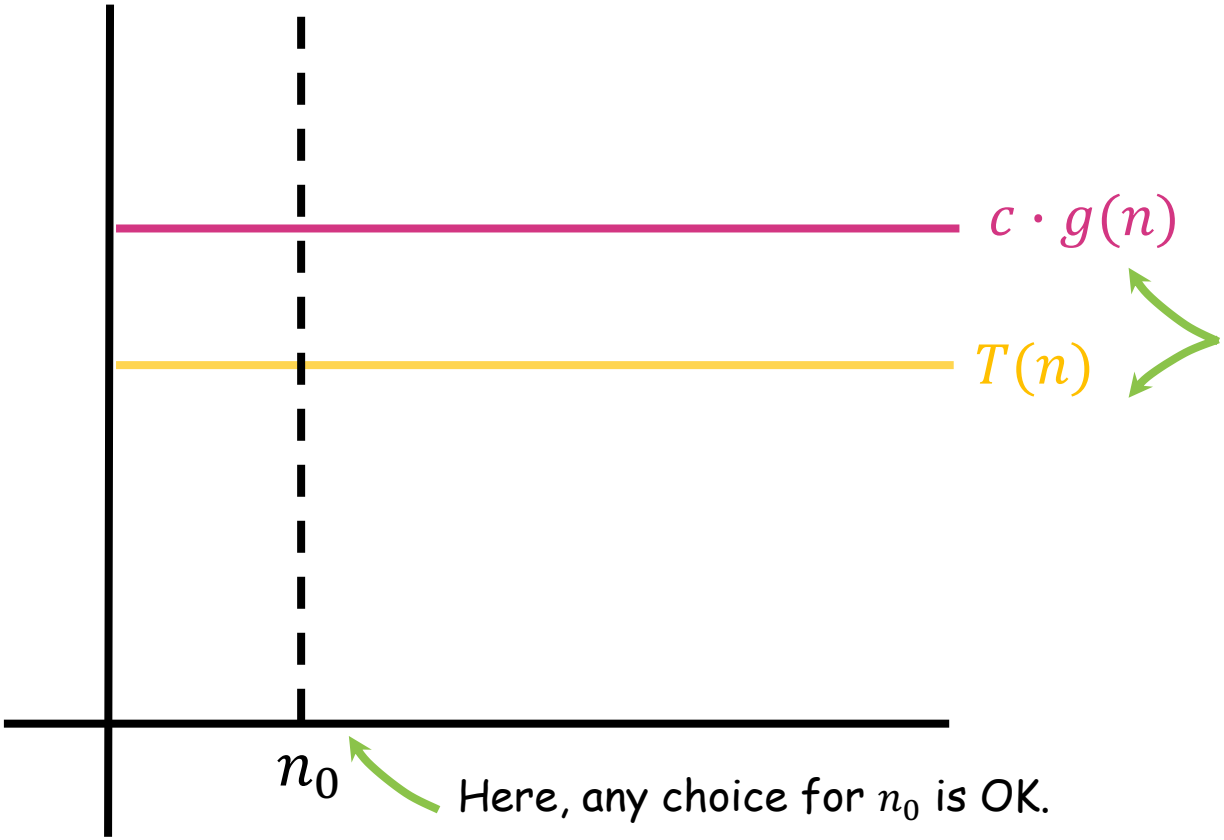
$$T(n) = O(g(n))$$
iff
$$\exists\, c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



$T(n)$ doesn't always have to be linear.

Big-O Notation

$$T(n) = O(g(n))$$
iff
$$\exists\, c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



$T(n)$ and $g(n)$ do not always have to be increasing.

Here, any choice for n_0 is OK.

Big-O Notation Proofs

$$T(n) = O(g(n))$$
iff
$$\exists\, c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$

- Big-O is the most common notation because it is used to provide an upper bound for the cost of algorithms in the worst case
- To prove $T(n) = O(g(n))$, show that there exists **c** and $\boldsymbol{n_0}$ that satisfies the definition.
  - **For example,**

    Suppose $T(n) = n$ and $\mathrm{g(n)} = n \log_2 n$. We prove that $T(n) = O(g(n))$.

    Select the values $c = 1$ and $n_0 = 2$. We have $n \leq c \cdot n \log_2 n$ for $n \geq n_0$ since n is positive and $1 \leq \log_2 n$ for $n \geq 2$.

Big-O Notation Proofs

$$T(n) = O(g(n))$$
iff
$$\exists\ c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$

- To prove $T(n) \neq O(g(n))$, proceed by contradiction.
  - **For example,**

    Suppose $T(n) = n^2$ and $g(n) = n$. We prove that $T(n) \neq O(g(n))$.

    Suppose there exists some c and n_0 such that for all $n \geq n_0$, $n^2 \leq c \cdot n$. Then, $n \leq c$, $\forall n \geq n_0$, which is not possible because c is a constant and n can be arbitrarily large.

Big-Ω Notation

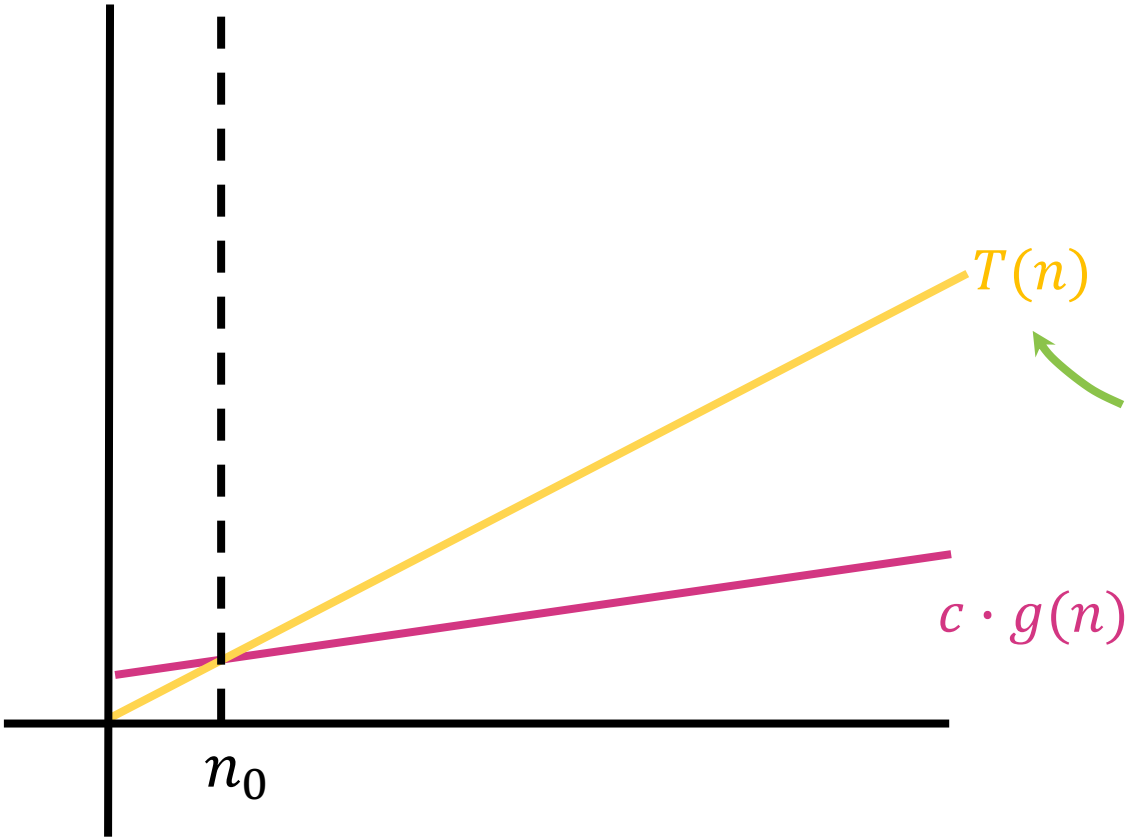
- Big-Ω notation is a mathematical notation for **lower**-bounding a function's rate of growth.

- Let $T(n), g(n)$ be functions of positive integers.
  - You can think of $T(n)$ as being a runtime: positive and increasing as a function of n.
- We say "$\boldsymbol{T(n) = \Omega(g(n))}$" if $g(n)$ grows at most as fast as $T(n)$ as n gets large. Formally,

$$\boldsymbol{T(n) = \Omega\big(g(n)\big)}$$
$$\textbf{iff}$$
$$\boldsymbol{\exists\, c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,}$$
$$\boldsymbol{0 \leq c \cdot g(n) \leq T(n)}$$

Big-Ω Notation

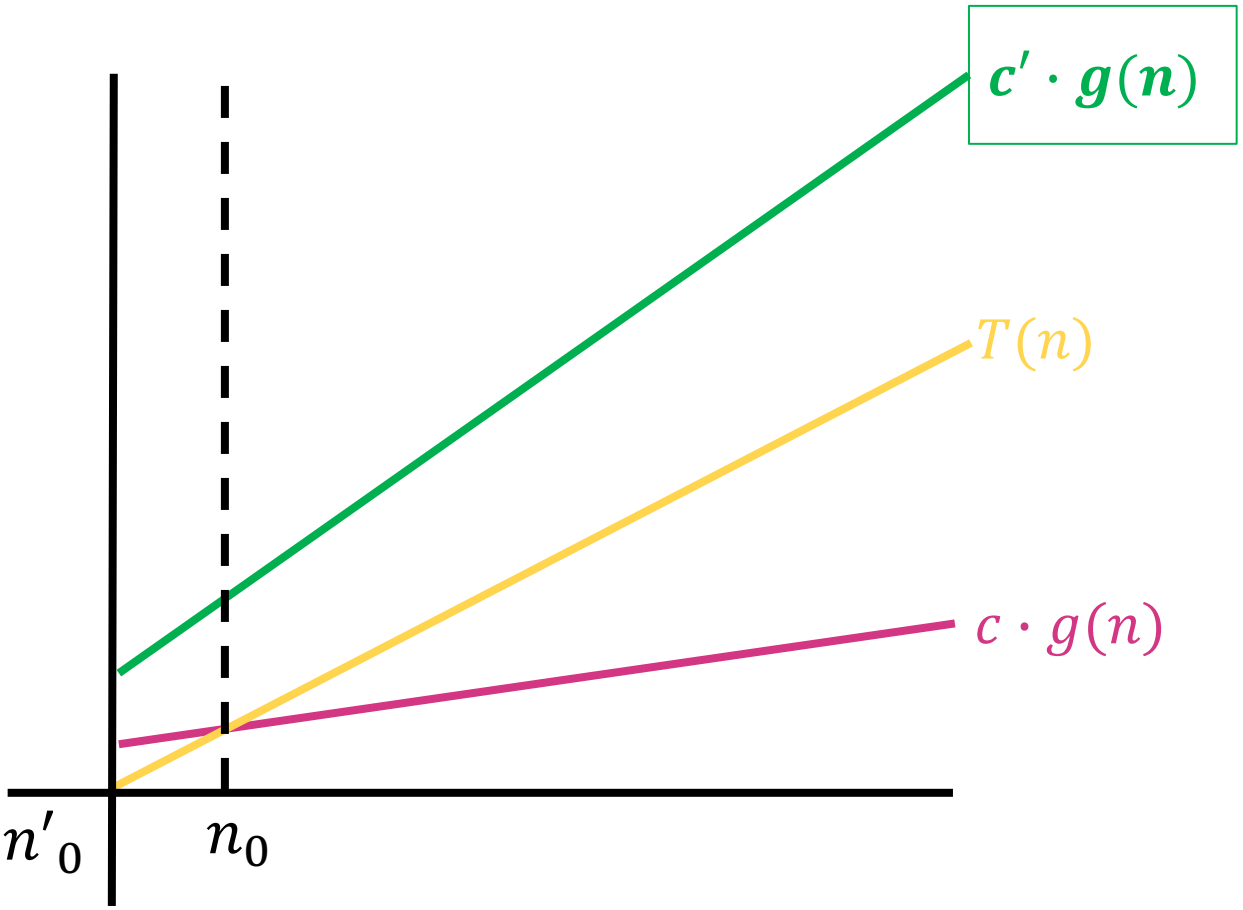
$$T(n) = \Omega(g(n))$$
iff
$$\exists\, c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,$$
$$0 \leq c \cdot g(n) \leq T(n)$$



Big-Ω defines "$T(n) = \Omega(g(n))$" to mean there exists some c and n_0 such that the pink line given by $c \cdot g(n)$ is **below** the yellow line for all values to the right of n_0.

Big-Θ Notation

We say "$T(n) = \Theta(g(n))$" iff $T(n) = O(g(n))$ AND $T(n) = \Omega(g(n))$.



Discussion about constants and non-dominant growth terms

Asymptotic notations ignore constants and non-dominant growth terms.

If the number of steps taken by an algorithm A is $10n$ and another algorithm B is $1000n$, then both have $\Theta(n)$ running time, but algorithm A is superior in practice.

If two algorithms have the same asymptotic running time, implementation and experimentation is necessary to determine the best.

Nevertheless if algorithm A is asymptotically slower than B, then A is very likely to be (much) slower than B in practice.

Basic Facts on Exponents and Logarithms

$c^{-n} = \frac{1}{c^n}$ ----------> $c^{-1} = \frac{1}{c}$

$c^{1/n} = \sqrt[n]{c}$ ----------> $c^{1/2} = \sqrt{c}$

$(c^m)^n = (c^n)^m = c^{mn}$ -------> $2^{2n} = (2^n)^2 \neq \Theta(2^n)$; set $x = 2^n$, then $x^2 \neq \Theta(x)$

$c^m c^n = c^{m+n}$ ----------> $2^{n+2} = 2^2 2^n = \Theta(2^n)$; $\Theta(2^{n+c}) = \Theta(2^n)$

$\log_c(ab) = \log_c a + \log_c b$

$\log_c(b^a) = a \log_c b$ ----------> $\log_2(n^3) = 3 \log_2 n = \Theta(\log n)$

In general, $\log(n^c) = \Theta(\log n)$

$\log_b a = \log_c a / \log_c b$ -----> $\log_{10} n = \log_2 n / \log_2 10 = \Theta(\log n)$.

($\log_2 10$ is a constant)

In general, the base of the logarithm is unimportant and $\log_c n = \Theta(\log n)$, unless it appears in an exponent.

$\log_c(1/a) = \log_c(a^{-1}) = -\log_c a$

$\log_b a = \log_a a / \log_a b = 1/\log_a b$

$a^{\log_b n} = n^{\log_b a}$ --------> $4^{\log_2 n} = n^{\log_2 4} = n^2$

Important note on growth of functions

constant < logarithmic < polynomial < exponential

$9999^{9999^{9999}} < \log^{10} n < n^{0.1} < n \log n < n^2 < 2^n$

$9999^{9999^{9999}} = \Theta(1) = O(\log\log n)$

$\log(\log n) = O(\log n)$ ----------> set $m = \log n$, then $\log n = O(m)$

$\log^{10} n = O(n^{0.1})$

$n \log n = O\left(\frac{n^2}{\log n}\right)$

$n^{0.1} + \log^{10} n = \Theta(n^{0.1})$

$n^{100} = O(2^n)$

Exercise 1

Note: constant < logarithmic < polynomial < exponential

Only the largest term is important

For each of the following statements, answer whether the statement is true or false.

(a) $1000n + n\log n = O(n\log n)$

True. Also Ω and Θ

(b) $n^2 + n\log(n^3) = O(n\log(n^3))$

False. $n^2 + n\log(n^3) = n^2 + 3n\log n = \Theta(n^2)$

(c) $n^3 = \Omega(n)$

True.

(d) $n^2 + n = \Omega(n^3)$

False. $n^2 + n = \Omega(n^2)$. (also Ω of any function smaller than n^2)

(e) $n^3 = O(n^{10})$

True.

(f) $n^3 + 1000n^{2.9} = \Theta(n^3)$

True.

(g) $n^3 - n^2 = \Theta(n)$

False. $n^3 - n^2 = \Theta(n^3)$

Exercise 2

Let $f(n)$ and $g(n)$ be non-negative functions. Using the basic definition of Θ-notation, prove that $\max\big(f(n), g(n)\big) = \Theta(f(n) + g(n))$

- We will prove that $\max\big(f(n), g(n)\big) = O(f(n) + g(n))$ and $\max\big(f(n), g(n)\big) = \Omega(f(n) + g(n))$. Therefore, $\max\big(f(n), g(n)\big) = \Theta\big(f(n) + g(n)\big)$.

- For any value of n, $\max(f(n), g(n))$ is either equal to $f(n)$ or equal to $g(n)$. Therefore, for all n, $\max\big(f(n), g(n)\big) \leq f(n) + g(n)$ and $\max\big(f(n), g(n)\big) = O(f(n) + g(n))$.

- For all n, $\max(f(n), g(n)) \geq f(n)$ and $\max(f(n), g(n)) \geq g(n)$.
  Then $2 \cdot \max\big(f(n), g(n)\big) \geq f(n) + g(n)$
  $$\Longrightarrow \max\big(f(n), g(n)\big) \geq \frac{1}{2}\big(f(n) + g(n)\big)$$
  $$\Longrightarrow \max\big(f(n), g(n)\big) = \Omega\big(f(n) + g(n)\big).$$

Exercise 3

For each pair of expressions (A, B) below, indicate whether A is O, Ω, or Θ of B. List all correct relations that hold for each pair

(a) $A = n^3 + n\log n, B = n^3 + n^2 \log n$

Ω, Θ, O.

(b) $A = \log\sqrt{n}, B = \sqrt{\log n}$

Ω. Set $m = \sqrt{n}$. Then $A = \log\sqrt{n} = \log m = \Theta(\log m)$,

and $B = \sqrt{\log n} = \sqrt{\log(m^2)} = \sqrt{2\log m} = \Theta\left(\sqrt{\log m}\right)$.

(c) $A = n\log_3 n, B = n\log_4 n$

Ω, Θ, O.

(d) $A = 2^n, B = 2^{n/2}$

Ω. Set $m = 2^{n/2}, B = 2^{n/2} = m, A = 2^n = 2^{n/2}2^{n/2} = m^2$.

(e) $A = \log(2^n), B = \log(3^n)$.

Ω, Θ, O. $\log(2^n) = n\log 2 = \Theta(n)$. $\log(3^n) = n\log 3 = \Theta(n)$.

Exercise 4a Bounds of series, Arithmetic Series

Prove that $\sum_{i=1}^{n} i = 1 + 2 + .. + (n-1) + n = \Theta(n^2)$

First, I will prove that $\sum_{i=1}^{n} i = O(n^2)$

$\sum_{i=1}^{n} i \leq \sum_{i=1}^{n} n = n \cdot n = O(n^2)$ (replace each i with n)

Then, I will prove that $\sum_{i=1}^{n} i = \Omega(n^2)$

$\sum_{i=1}^{n} i \geq \frac{n}{2} + \left(\frac{n}{2} + 1\right) + \cdots + n \geq \frac{n}{2} . \frac{n}{2} = \frac{n^2}{4} = \Omega(n^2)$

(remove all terms < n/2, replace the remaining ones with n/2)

Thus,

$\sum_{i=1}^{n} i = \Theta(n^2)$

Closed formula for arithmetic series $\sum_{i=1}^{n} i = n(n+1)/2$

We will solve it in the next class.

Exercise 4b Polynomial Series

Prove that $\sum_{i=1}^{n} i^c = \Theta(n^{c+1})$, where c is a constant

First, I will prove that $\sum_{i=1}^{n} i^c = O(n^{c+1})$

$\sum_{i=1}^{n} i^c \leq \sum_{i=1}^{n} n^c = n \cdot n^c = O(n^{c+1})$

Then, I will prove that $\sum_{i=1}^{n} i^c = \Omega(n^{c+1})$

$\sum_{i=1}^{n} i^c \geq \left(\frac{n}{2}\right)^c + \left(\frac{n}{2}+1\right)^c + \cdots + nc \geq \frac{n}{2} . \left(\frac{n}{2}\right)^c = \frac{n^{c+1}}{2^{c+1}} = \Omega(n^{c+1})$

Thus,

$\sum_{i=1}^{n} i^c = \Theta(n^{c+1})$

No closed formula for polynomial series

Exercise 4c Harmonic Series H_n

$$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \cdots + \frac{1}{n-1} + \frac{1}{n}.$$

Prove that $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$. Let $k = \log_2 n$, i.e., $n = 2^k$.

lower bound	H_n	upper bound
1/2	1	1
2X1/4=1/2	(1/2)+(1/3)	2X1/2=1
4X1/8=1/2	(1/4)+(1/5)+(1/6)+(1/7)	4X1/4=1
2^{k-1}x$1/2^k$=1/2	$(1/2^{k-1})+(1/(2^{k-1}+1))+..(1/(2^k-1))$	2^{k-1}x$1/2^{k-1}$=1
0	$1/2^k$=1/n	1

Each of the right-hand sums (upper bounds) is 1. The number of sums is $\log n + 1$. Thus, $H_n \leq \log n + 1 = O(\log n)$.

Each of the left-hand sums is 1/2 except for the last one. Thus, $\frac{1}{2}(\log n) \leq H_n = \Omega(\log n)$.

Combining: $H_n = \Theta(\log n)$