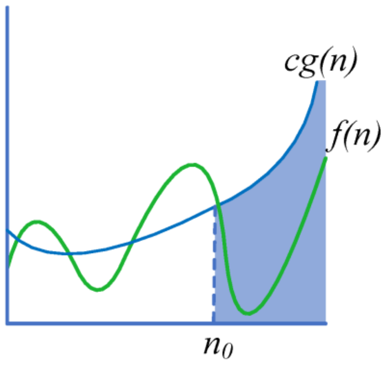
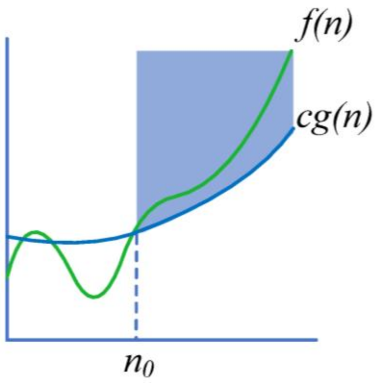
DSE 2256 DESIGN & ANALYSIS OF ALGORITHMS

Lecture 4 & 5 :

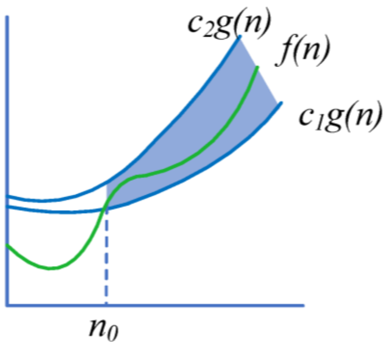
Worst, Best & Average Case Efficiencies & Asymptotic Notations



(a) Big-O Notation



(b) Big-Omega Notation



(c) Big-Theta Notation

Recap of L2 & L3

- Fundamental Data Structures

- Algorithm Analysis Framework
  - Measuring an Input's Size
  - Units for measuring Running Time
  - Orders of growth

$$T(n) \approx c_{op}\, C(n)$$

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Best-case, average-case, worst-case efficiencies I

For some algorithms, efficiency depends on form of input.

Example:

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

//Searches for a given value in a given array by sequential search
//Input: An array $A[0..n-1]$ and a search key K
//Output: The index of the first element in A that matches K
// or -1 if there are no matching elements
$i \leftarrow 0$
while $i < n$ **and** $A[i] \neq K$ **do**
    $i \leftarrow i + 1$
if $i < n$ **return** i
else return -1

<u>For this example:</u>

- How would the **worst** possible form of input look like ?
  - What about C(n) in this case?
- How would the **best** possible form of input look like ?
  - What about C(n) in this case?

Best-case, average-case, worst-case efficiencies II

- Worst case: $C_{worst}(n)$ – maximum over inputs of size n
- Best case: $C_{best}(n)$ – minimum over inputs of size n
- Average case: $C_{avg}(n)$ – "average" over inputs of size n

How to analyze an algorithm's average case efficiency?

Best-case, average-case, worst-case efficiencies III

For the sequential search example, assume the following:

- The probability of a successful search is equal to *p* (where, 0<= p <= 1)
- The probability of the first match occurring in the i^{th} position of the list is same for every *i*.

$$C_{avg}(n) = [\, 1 \,.\, {}^{p}/_{n} + 2.{}^{p}/_{n} + 3.{}^{p}/_{n} + \ldots. + i\,.\,{}^{p}/_{n} + \ldots + n \,.\, {}^{p}/_{n} \,] + n\,.\,(1 - p)$$

$$= {}^{p}/_{n}\,[\, 1 + 2 + 3 + \ldots.. + n] + n\,(1 - p)$$

$$= {}^{p}/_{n}\,. + n\,(1 - p)$$

$$C_{avg}(n) = p\,\frac{(n-1)}{2} + n\,(1 - p)$$

Best-case, average-case, worst-case efficiencies IV

Remember that average case efficiency is:

- Not the average of worst and best case.

- Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs.

  So, average = expectation under uniform distribution.

Asymptotic Notations I

The order of growth of $C(n)$ is the principal indicator of an algorithm's efficiency.

Three notations to compare and rank the order of growths:

1. **$O(g(n))$** is the set of all functions with a smaller or same order of growth as $g(n)$.

Ex: $n \in O(n^2)$

2. **$\Omega(g(n))$** stands for the set of all functions with larger or same order of growth as $g(n)$.

Ex: $n^3 \in \Omega(n^2)$

3. **$\Theta(g(n))$** is the set of all functions that have same order of growth as $g(n)$.

Ex: $7n^2 \in \Theta(n^2)$

Asymptotic Notations II

- **O (g(n)) :** class of functions $t(n)$ that grow **no faster** than $g(n)$

- **Θ (g(n)) :** class of functions $t(n)$ that grow **at same rate** as $g(n)$

- **Ω (g(n)) :** class of functions $t(n)$ that grow **at least as fast** as $g(n)$

Big-oh (O) notation

Definition:

A function t(n) is said to be in O(g(n)), denoted as t(n) ∈ O(g(n)):

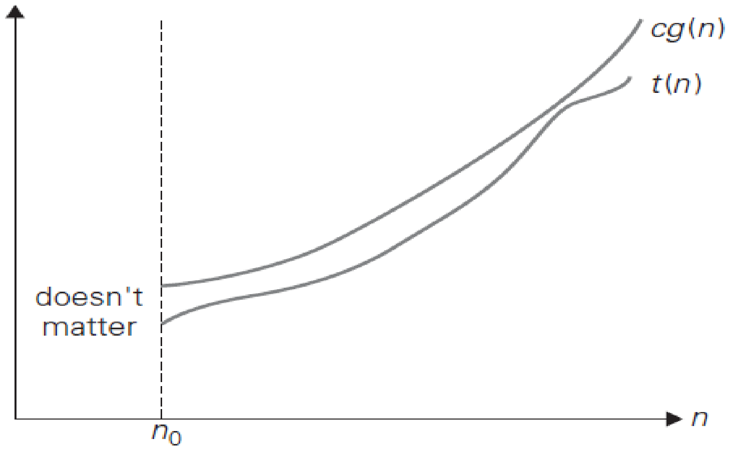
if t(n) is bounded above by some constant multiple of g(n) for all large n,

i.e., there exist positive constant *c* and non-negative integer n_0 such that:

$t(n) <= c\, g(n)$ for every $n \geq n0$

Examples:

- $10n \in O(n2)$
- $5n+20 \in O(n)$



Big-omega (Ω) notation

Definition:

A function t(n) is said to be in Ω(g(n)), denoted as t(n) ∈ Ω(g(n)):

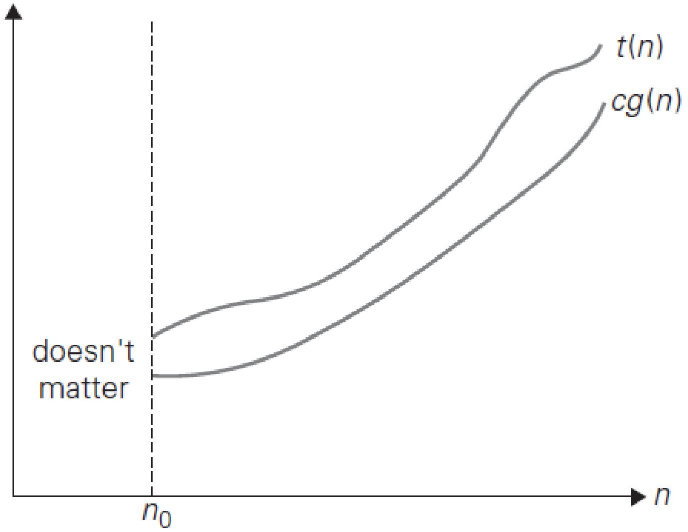
if t(n) is bounded below by some positive constant multiple of g(n) for all large n,

i.e., there exist positive constant *c* and non-negative integer n_0 such that:

***t*(*n*) >= *c g*(*n*) for every *n* ≥ *n*0**

<u>Examples:</u>

- $10n^2 \in \Omega(n^2)$



Big-theta(Θ) notation

Definition:

A function t(n) is said to be in Θ (g(n)), denoted as $t(n) \in \Theta(g(n))$:

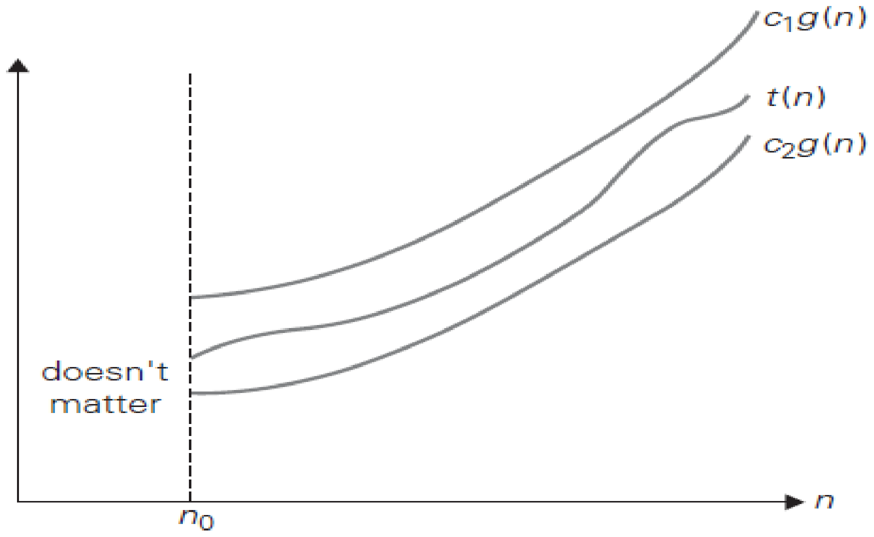
if t(n) is bounded both above and below by some positive constant multiple of g(n) for all large n,

i.e., there exist positive constant *c* and non-negative integer n_0 such that:

c2 g(n)= *t*(*n*) <= *c g*(*n*) for every *n* ≥ *n*0

<u>Examples:</u>

- $10n^2 \in \Theta(n^2)$
- $0.3n^2 - 2n \in \Theta(n^2)$
- $\frac{1}{2} n(n+1) \in \Theta(n^2)$



Big-theta notation: $t(n) \in \Theta(g(n))$.

Exercises

1. Use the definition of O, Ω and Θ to determine whether the following assertions are true or false.

a) $n(n + 1)/2 \in O(n^3)$

b) $n(n + 1)/2 \in O(n^2)$

c) $n(n + 1)/2 \in \Theta(n^3)$

a) $n(n + 1)/2 \in \Omega(n)$

Asymptotic Notations: Property

Theorem:

If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then

$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$.

Proof:

- For real numbers, If $a_1 <= b_1$ and $a_2 <= b_2$ then,

  $a_1 + a_2 <= 2 \max\{b_1, b_2\}$

- Since $t_1(n) \in O(g_1(n))$, there exist constants c_1, c_2, n_1, n_2 such that :

  $t_1(n) \leq c_1 * g_1(n)$, for all $n \geq n_1$

- Similarly, since $t_2(n) \in O(g_2(n))$: $t_2(n) \leq c_2 * g_2(n)$, for all $n \geq n_2$

Contd ..

Asymptotic Notations: Property

- Adding the two inequalities above yields the following:

$$t_1(n) + t_2(n) <= c_1g_1(n) + c_2g_2(n)$$

$$<= c_3g_1(n) + c_3\, g_2(n) = c_3\, [g_1(n) + g_2(n)]$$

$$<= c_3\, 2 \max\{g_1(n), g_2(n)\}$$

- Hence, $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$, with the constants c_2, c_3 and n_0 required by the O definition being $c_3=\max\{c_1,c_2\}$ and $n >= \max\{n_1,n_2\}$, respectively.

Using limits for comparing orders of growth

$$\lim_{n\to\infty} t(n) / g(n) = \begin{cases} \mathbf{0} & \text{order of growth of } T(n) < \text{ order of growth of } g(n) \\ \mathbf{c > 0} & \text{order of growth of } T(n) = \text{ order of growth of } g(n) \\ \mathbf{\infty} & \text{order of growth of } T(n) > \text{ order of growth of } g(n) \end{cases}$$

Basic asymptotic efficiency classes

Class	Name
1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n-log-n
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

Thank you!

Any queries?