CE2001/ CZ2001: Algorithms

Quantifying Order of Growth of Time Complexity Functions Through Asymptotic Analysis

Dr. Loke Yuan Ren

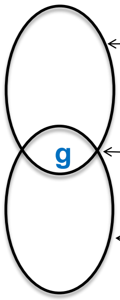
Learning Objectives

At the end of this lecture, students should be able to:

- Define Big-Oh (O)
- Define Big-Omega (Ω)
- Define Big-Theta (Θ)

Meanings of O, Ω and θ

Big-Oh (O) , Big-Omega (Ω) and Big-Theta (Θ) are asymptotic (set) notations used for describing the order of growth of a given function.



$\leftarrow f \in \Omega(g)$: Set of functions that grow at higher or same rate as **g**

$\leftarrow f \in \Theta(g)$: Set of functions that grow at same rate as **g**

$\leftarrow f \in O(g)$: Set of functions that grow at lower or same rate as **g**

Big-Oh Notation

Definition: Let f and g be 2 functions such that

f(n): N → R^+ and g(n): N → R^+,

f(n) is said to be in O(g(n)), denoted by $f(n) \in O(g(n))$

if f(n) is bounded above by some $f(n) = O(g(n))$

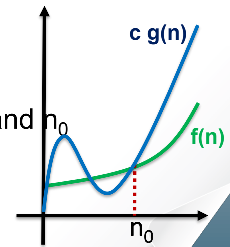
constant multiple of g(n) for all large n,

(Or more formally)

if there exist nonnegative constants c and n_0

such that

f(n) ≤ c*g(n) for all **n ≥ n_0**



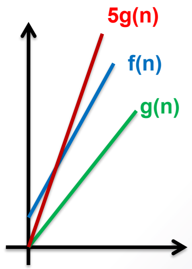
Big-Oh Notation

Consider: **f(n) = 4n+3, g(n) = n**

Let c = 5, n_0 = 3

Then f(n) ≤ 5g(n), i.e., 4n+3≤5n for all n≥3

➡ **f(n) = O(g(n)), i.e. 4n+3 ∈ O(n)**



Big-Oh Notation

Consider: **f(n) = 4n+3, g(n) = n**

Let c = 5, n_0 = 3

Then f(n) ≤ 5g(n), i.e., 4n+3 ≤ 5n for all n ≥ 3

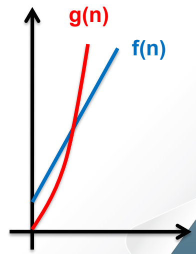
➡ **f(n) = O(g(n)), i.e. 4n+3 ∈ O(n)**

Consider: **f(n) = 4n+3, g(n) = n^3**

Let c = 1, n_0 = 3

Then f(n) ≤ g(n), i.e., 4n+3 ≤ n^3 for all n ≥ 3

➡ **f(n) = O(g(n)), i.e. 4n+3 ∈ O(n^3)**



Big-Oh Notation

Alternative Definition:

Let **f** and **g** be 2 functions such that

f(n) : N → R^+ and g(n) : N → R^+,

if $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

then $f(n) \in O(g(n))$ or *f(n) = O(g(n))*

Big-Oh Notation

Consider: **f(n) = 4n+3, g(n) = n** (again)

Another way:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{4n+3}{n} = 4 < \infty$$

➡ **f(n) = O(g(n)), i.e. 4n+3 ∈ O(n)**

Consider: **g(n) = n^3**

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{4n+3}{n^3} = 0 < \infty$$

➡ **f(n) = O(g(n)), i.e. 4n+3 ∈ O(n^3)**

Big-Oh Notation

Consider: f(n) = 4n, g(n) = e^n, $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{4n}{e^n} = ?$

L'Hôpital's Rule: Given functions f(x) and g(x),

if $\lim_{n\to\infty} f(x) = \lim_{n\to\infty} g(x) = \pm\infty$

then $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f'(n)}{g'(n)}$

where the prime (') denotes the derivative.

Thus, $\lim_{n \to \infty} \frac{4n}{e^n} = \lim_{n \to \infty} \frac{4}{e^n} = 0$ ➡ f(n) $\in$ O(g(n))

Big-Omega Notation

Definition: Let f and g be 2 functions such that

f(n) : N -> R^+ and g(n) : N -> R^+,

f(n) is said to be in Ω(g(n)), denoted by $f(n) \in \Omega(g(n))$, $f(n) = \Omega(g(n))$

if f(n) is bounded below by

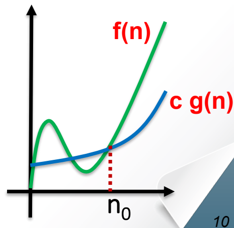
some constant multiple of g(n) for all large n,

Or more formally

if there exist positive constants c and n_0

such that

f(n) ≥ c*g(n) for all n ≥ n_0



Big-Omega Notation

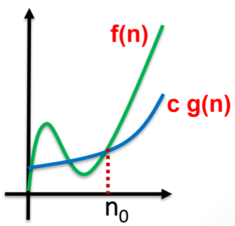
Alternative Definition:

Let f and g be 2 functions such that

f(n) : N -> R^+ and g(n) : N -> R^+,

if $\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$

then $f(n) \in \Omega(g(n))$ or f(n) = Ω(g(n)).



Big-Omega Notation

Consider: f(n)=4n+3, g(n)=5n,

Let c=1/5, n_0 =0

Then f(n) ≥ (1/5)g(n), i.e., 4n+3 ≥ (1/5)5n for all n≥0

Another way: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{4n+3}{5n} = \frac{4}{5} > 0$

➡ f(n) = Ω(g(n))

Consider: f(n)=n³+2n, g(n)=5n,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{n^3+2n}{5n} = \infty > 0$$

➡ f(n) = Ω(g(n))

The Big Theta Notation

Definition: Let f and g be 2 functions such that

f(n) : N -> R^+ and g(n) : N -> R^+,

if there exist positive constants c_1, c_2 and n_0 such that

c_1*g(n) ≤ f(n) ≤ c_2*g(n) for all $n \geq n_0$

then $f(n) \in \Theta(g(n))$ or $f(n) = \Theta(g(n))$

Alternative definition: if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$

then $f(n) \in \Theta(g(n))$ or $f(n) = \Theta(g(n))$

For example, $\lim_{n \to \infty} \frac{2n^2+7}{7n^2+n} = \lim_{n \to \infty} \frac{4n}{14n+1} = \frac{2}{7}$

➡ $2n^2+7 = \Theta(7n^2+n)$

Summary of Limit Definition

$\lim_{n \to \infty} \frac{f(n)}{g(n)}$	f(n) ∊ O(g(n))	f(n) ∊ Ω(g(n))	f(n) ∊ Θ(g(n))
0	✓		
$0 < C < \infty$			
∞			

Summary of Limit Definition

$\lim_{n \to \infty} \frac{f(n)}{g(n)}$	f(n) ∊ O(g(n))	f(n) ∊ Ω(g(n))	f(n) ∊ Θ(g(n))
0	✓		
$0 < C < \infty$			
∞		✓	

Summary of Limit Definition

$\lim_{n \to \infty} \frac{f(n)}{g(n)}$	f(n) ∊ O(g(n))	f(n) ∊ Ω(g(n))	f(n) ∊ Θ(g(n))
0	✓		
$0 < C < \infty$	✓	✓	✓
∞		✓	

Recap

- Formal definition of O, Ω and Θ
- Limit of ratio definition of O, Ω and Θ
- Comparison of functions using O, Ω and Θ

CE2001/ CZ2001: Algorithms

Use of O, Ω and Θ in Asymptotic Analysis

Dr. Loke Yuan Ren

Learning Objectives

At the end of this lecture, students should be able to:

- Use O, Ω and Θ to quantify order of growth
- Properties of O, Ω and Θ
- Explain the simplification rules that can be used in asymptotic analysis

Use of O, Ω and Θ

- The O, Ω and Θ notations are used in studying the asymptotic efficiency of an algorithm.
  - If f(n) = O(g(n)), we say
    g(n) is asymptotic upper bound of f(n)
  - If f(n) = Ω(g(n)), we say
    g(n) is asymptotic lower bound of f(n)
  - If f(n) = Θ(g(n)), we say
    g(n) is asymptotic tight bound of f(n)

How to derive Complexity Class of Algorithms?

When time complexity of algorithm A grows faster than algorithm B for the same problem, we say A is inferior to B.

- How to determine the big-Oh notation?
  1. Count primitive operations to derive complexity function f (in terms of problem size)
  2. Discard constant terms and multipliers in f
  3. Determine dominant term in f
  4. Dominant term = big-Oh notation for f (= big-Oh notation for algorithm)

Asymptotic Notation in Equations

When an asymptotic notation appears in an equation, we interpret it as standing for some anonymous function that we do not care to name.

Examples:

- $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$
- $T(n) = T(n/2) + \Theta(n)$
- $2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$

Simplification Rules for Asymptotic Analysis

1. If $f(n) = O(cg(n)+)$ for any constant $c>0$, then $f(n) = O(g(n))$

2. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.
   e.g. $f(n) = 2n$, $g(n) = n^2$, $h(n) = n^3$

3. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)+ f_2(n) = O(\max(g_1(n),g_2(n)))$.
   e.g. $5n + 3\lg n = O(n)$

4. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$.
   e.g. $f_1(n) = 3n^2$, $f_2(n) = \lg n$, $f_1(n) = O(n^2)$, $f_2(n) = O(\lg n)$, then $3n^2 \lg(n) = O(n^2 \lg(n))$

Recap

- Application of O, Ω and Θ
- How to derive O of algorithms?
- Simplification Rules

NANYANG TECHNOLOGICAL UNIVERSITY

CE2001/ CZ2001: Algorithms

Properties of O, Ω and θ

and

Time Complexity Classes

Dr. Loke Yuan Ren

Learning Objectives

At the end of this lecture, students should be able to:

- Review the properties of O, Ω and Θ
- Review Common Complexity Classes

Properties of O, Ω and Θ

- O, Ω and Θ are reflexive

  f(n)=O(f(n)), f(n)=Ω(f(n)), f(n)=Θ(f(n))

- O, Ω and Θ are transitive

  f(n)=O(g(n)) and g(n)=O(h(n)) => f(n)=O(h(n))

  f(n)=Ω(g(n)) and g(n)=Ω(h(n)) => f(n)=Ω(h(n))

  f(n)= Θ(g(n)) and g(n)= Θ(h(n)) => f(n)= Θ(h(n))

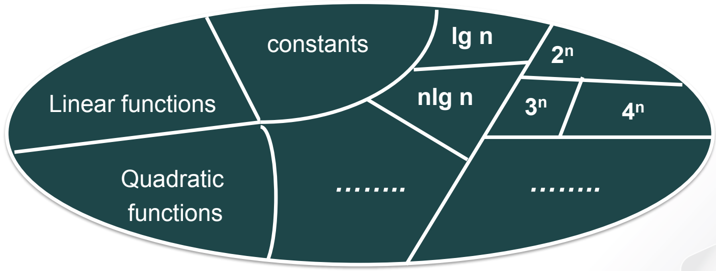
- Θ is symmetric

  f(n)= Θ(g(n)) => g(n)= Θ(f(n))

Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

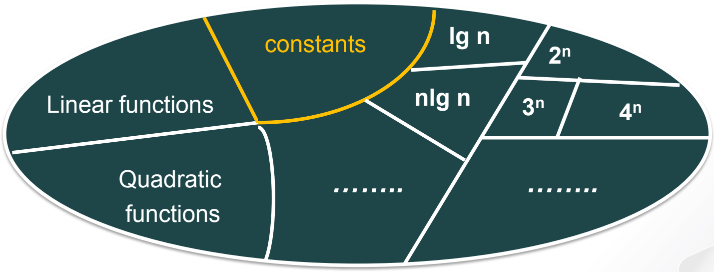
Set of all functions of n



Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

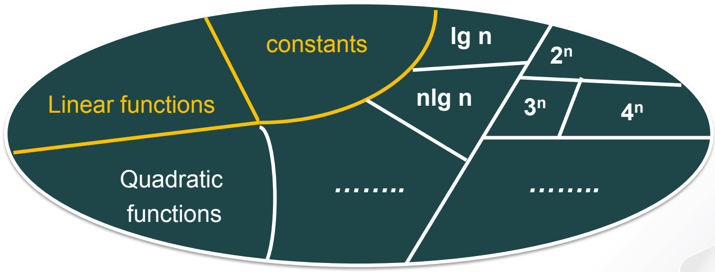
Set of all functions of n



Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

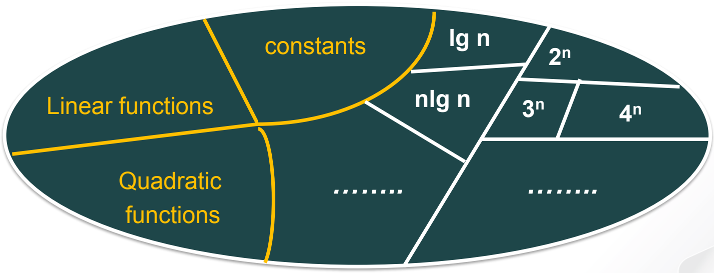
Set of all functions of n



Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

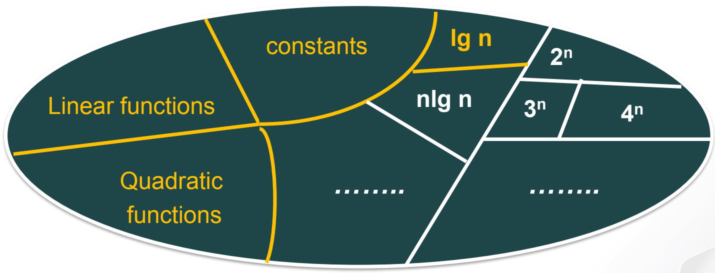
Set of all functions of n



Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

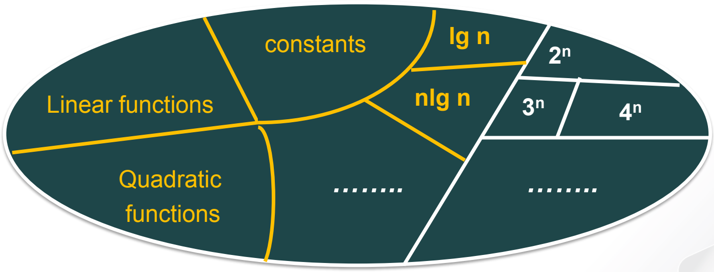
Set of all functions of n



Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

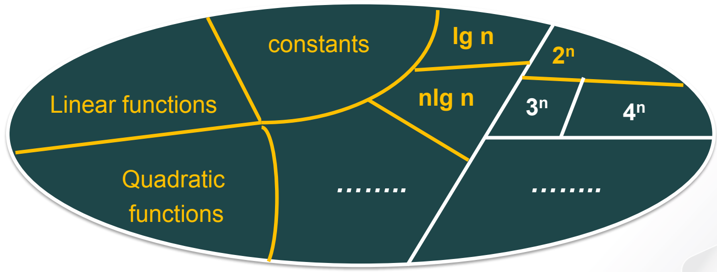
Set of all functions of n



Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

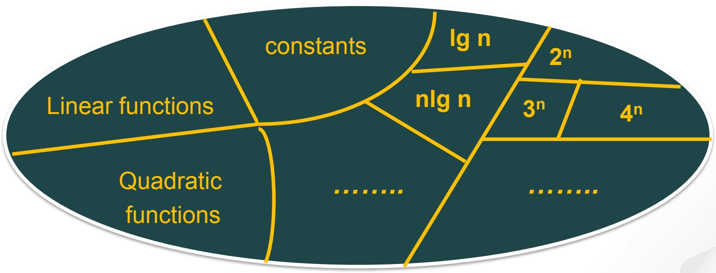
Set of all functions of n



Properties of O, Ω and θ

So θ defines an equivalence relation on functions. The equivalence classes in this relation are called the complexity classes.

Set of all functions of n



Complexity Classes and Examples

Order of Growth	Class	Example
1	constant	Finding midpoint of an array
$\log_2 n$	logarithmic	Binary search
n	linear	Linear Search
$n \log_2 n$	linearithmic	Merge Sort
n^2	quadratic	Insertion Sort
n^3	cubic	Matrix Inversion (Gauss-Jordan elimination)
2^n	exponential	The Tower of Hanoi Problem
$n!$	factorial	Travelling Salesman Problem

Common Complexity Classes

Common Complexity Classes

1. **f(n) = O(1)** means *f*(*n*) is of constant order,
   i.e. the running time is independent of problem size n.

   Example: sum = (1+n)*n/2;
   $f(n) = 4 = O(4) = O(1)$

   Formal verification: $\forall n \geq 0,\ n \in N,\ |f(n)| \leq 4 \cdot 1$
   $\therefore\ f(n) = O(1)$

Common Complexity Classes

2. **f(n) = O(log(n))** means f(n) is of logarithmic order;
   the running time increases slower than n.

   Example: for (i=n; i>=1; i/=2) sum++;
   $f(n) = \lfloor \log_2(n) \rfloor + 1 = O(\log(n))$

   Note 1: log(n) increases slower than n^ε for all $\varepsilon > 0$.

   Proof: $\lim_{n\to\infty} \frac{\lg n}{n^\varepsilon} = \lim_{n\to\infty} \frac{c(1/n)}{\varepsilon n^{\varepsilon-1}} = \lim_{n\to\infty} \frac{c}{\varepsilon n^\varepsilon} = 0$

   Note 2: Base of log is not important
   $\forall\ b,c \in R,\ b>1 \wedge c>1 \rightarrow \log_b n = (\log_c n) / (\log_c b)$
   E.g. $\log_{10} n = \log_2 n / \log_2 10$

Common Complexity Classes

3. **f(n) = O(n)** means f(n) is of linear order
   [optimal for an algorithm that must process n inputs].

   E.g. for (j=1; j<=n; j++) sum++;
   f(n) = n = O(n)

4. **f(n) = O(nlogn)** is seen in algorithms that break a problem into sub-problems, solve them independently and combine the solutions.

   E.g. W(2) = 1
   W(n) = 2W(n/2) + n-1

Common Complexity Classes

5. $\exists p \in N$, **$f(n) = O(n^p)$** means f(n) is of polynomial order.

E.g.
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        for (k=1; k<=n; k++)
            M[i][j] = A[i][k]*B[k][j];
```

$f(n) = O(n^3)$

6. $\exists a \in N$, $a>1$, **$f(n) = O(a^n)$** means f(n) is of exponential order; not practical for normal use.

E.g. Print all subsets of a set of n elements

$f(n) = c*2^n$
$\therefore f(n) = O(2^n)$

Recap

- Properties of O, Ω and θ
- Common Complexity Classes

CE2001/ CZ2001: Algorithms

Space Complexity

Dr. Loke Yuan Ren

Space Complexity

- Determine number of entities in problem (also called problem size)
- Count number of basic units in algorithm

Basic units

- Things that can be represented in a constant amount of storage space

  E.g. integer, float and character.

How about Structure?

- Space requirements for an array of n integers - $\theta(n)$
- If a matrix is used to store edge information of a graph,

  i.e. G[x][y] = 1 if there exists an edge from x to y,

  space requirement for a graph with n vertices is $\theta(n^2)$

Space/time tradeoff principle

Reduction in time can be achieved by sacrificing space and vice-versa.

Recap

- Concepts on Space Complexity