جامعة طرابلس ـ كلية تقنية المعلومات

Design and Analysis Algorithms

تصميم وتحليل خوارزميات

ITGS301

المحاضرة الثانية: Lecture 2

Tripoli University - Faculty of Information Technology
Marwa Solla
m.solla@uot.edu.ly

خريف 2024

Contents

Common order-of-growth classifications

Asymptotic Notation

- Big Oh , Omega , Theta

Asymptotic Analysis

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

Asymptotic analysis is an analysis of algorithms that focuses on

- Analyzing problems of large input size
- Consider only the leading term of the formula
- Ignore the coefficient of the leading term

There are mainly three asymptotic notations:

- Big-O notation
- Omega notation
- Theta notation

Why Choose Leading Term?

Lower order terms contribute lesser to the overall cost as the input grows larger

Example

- $f(n) = 2n^2 + 100n$
- $f(1000) \quad = 2(1000)^2 + 100(1000)$
  $= 2{,}000{,}000 + 100{,}000$
- $f(100000) \quad = 2(100000)^2 + 100(100000)$
  $= 20{,}000{,}000{,}000 + 10{,}000{,}000$

Hence, lower order terms can be ignored

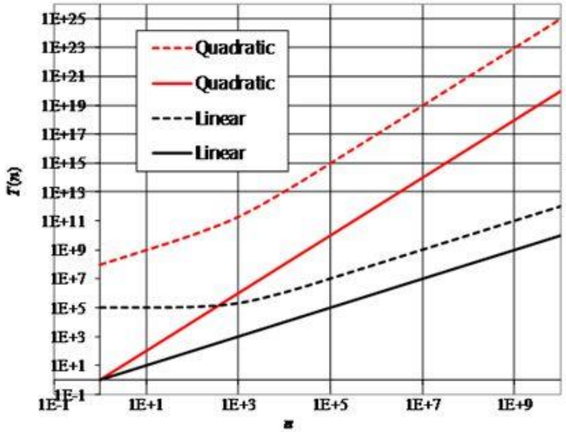
Lower-Order Terms and Constant Factors

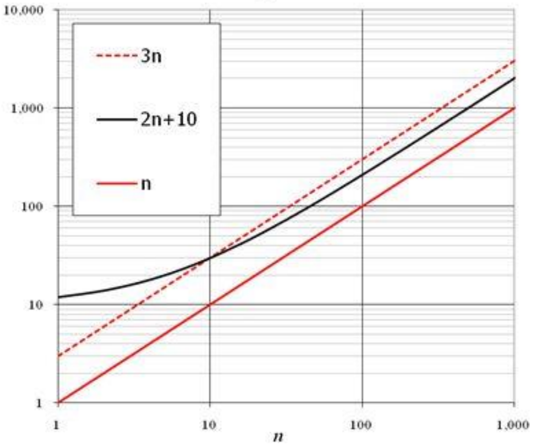
- The growth rate is not affected by
  - lower-order terms or
  - constant factors
- Examples

Quadratic function:
- $10^5 n^2 + 10^8 n$
- $10^5 n^2$

Linear function:
- $10^2 n + 10^5$
- $10^2 n$

Different linear function:
- $3n$
- $2n+10$
- n





Examples: Leading Terms

- $a(n) = \frac{1}{2} n + 4$
  - Leading term: $\frac{1}{2} n$
- $b(n) = 240n + 0.001n^2$
  - Leading term: $0.001n^2$
- $c(n) = n \lg(n) + \lg(n) + n \lg(\lg(n))$
  - Leading term: $n \lg(n)$
  - Note that $\lg(n) = \log_2(n)$

These terms can be obtained through **asymptotic analysis**

Order of growth

The fundamental reason is that for large values of n, any function that contains an n^2 term will grow faster than a function whose leading term is n. The **leading term** is the term with the highest exponent.

we expect an algorithm with a smaller leading term to be a better algorithm for large problems, but for smaller problems, there may be a **crossover point** where another algorithm is better.

Order of growth

Suppose you have analyzed two algorithms and expressed their run times in terms of the size of the input: Algorithm A takes $100\,n + 1$ steps to solve a problem with size n; Algorithm B takes $n^2 + n + 1$ steps.

The following table shows the run time of these algorithms for different problem sizes:

Input size	Run time of Algorithm A	Run time of Algorithm B
10	1 001	111
100	10 001	10 101
1 000	100 001	1 001 001
10 000	1 000 001	$> 10^{10}$

What is Order of Growth?

An **order of growth** is a set of functions whose asymptotic growth behavior is considered equivalent. For example, $2n$, $100n$ and $n + 1$ belong to the same order of growth, which is written $O(n)$ in **Big-Oh notation** and often called **linear** because every function in the set grows linearly with n.

How the **time/space complexity** of an algorithm **grows/changes** with the **input size**

		حجم المدخلات					
نوع معدل النمو	**معدل النمو**	**n**	**...**	**3**	**2**	**1**	**وقت الخوارزمية**
(Constant) ثابت	1	2	...	2	2	2	وقت الخوارزمية (A)
(Linear) خطي	n	2n	...	6	4	2	وقت الخوارزمية (B)
أسي (Exponential)	c^n	2^n	...	8	4	2	وقت الخوارزمية (K)

معدل تغير وقت أو مساحة الخوارزمية مع تغير حجم المدخلات

What is Order of Growth?

Algorithm 30 Minimum and Maximum Elements

Input: An array $A[1..n]$ of n elements.
Output: The minimum and maximum elements in A

$min \leftarrow A[1]$
$max \leftarrow A[1]$
for $i \leftarrow 2$ **to** n **do**
    if $(A[i] < min)$ **then**
        $min \leftarrow A[i]$
    end if
    if $(A[i] > max)$ **then**
        $max \leftarrow A[i]$
    end if
end for
return (min, max)

Algorithm 29 Minimum and Maximum Elements

Input: An array $A[1..n]$ of n elements sorted in ascending order.
Output: The minimum and maximum elements in A

$min \leftarrow A[1]$
$max \leftarrow A[n]$
return (min, max)

Orders of Common Functions

A list of classes of functions that are commonly encountered when analyzing algorithms.

constant	$O(1)$
logarithmic	$O(\log_2 N)$
Linear	$O(N)$
N log n	$O(n \log_2 N)$
Quadratic	$O(N^2)$
Cubic	$O(N^3)$
Exponential	$O(2^n)$
Factorial	$O(n!)$

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

Order of growth

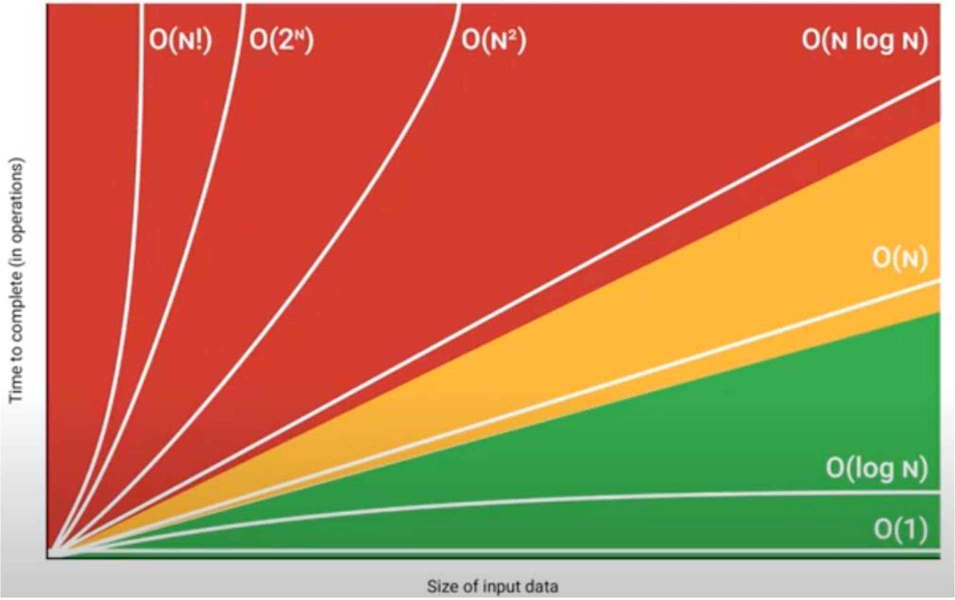
The following table shows some of the orders of growth that appear most commonly in algorithmic analysis.
For the logarithmic terms, the base of the logarithm doesn’t matter; changing bases is the equivalent of multiplying by a constant, which doesn’t change the order of growth.
Similarly, all exponential functions belong to the same order of growth regardless of the base of the exponent.
Exponential functions grow very quickly, so exponential algorithms are only useful for small problems.

Order of growth

Name	Function
Constant	c
Double Logarithmic	$\log \log n$
Logarithmic	$\log n$
Fractional Power	$n^c, 0 < c < 1$
Linear	$O(n)$
Loglinear	$n \log n$ and $\log n!$
Quadratic	n^2
Polynomial	$n^c, c > 1$
Exponential	$c^n, c > 1$
Factorial	$n!$
Super Exponential	n^n

Common order-of-growth classifications Running time complexity

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	O(1)	O(log n)	O(n)	O(n log n)	O(n^2)	O(n^3)	O(2^n)
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}



Exercise 1

Arrange the functions in increasing asymptotic order

(a) $n^{1/3}$
(b) e^{n}
(c) $n^{7/4}$
(d) $n\ \log n$
(e) 1.0000001^{n}

n	n log2(n)	n^(7/4)
2	2	3
4	8	11
8	24	38
16	64	128
32	160	431
64	384	1448

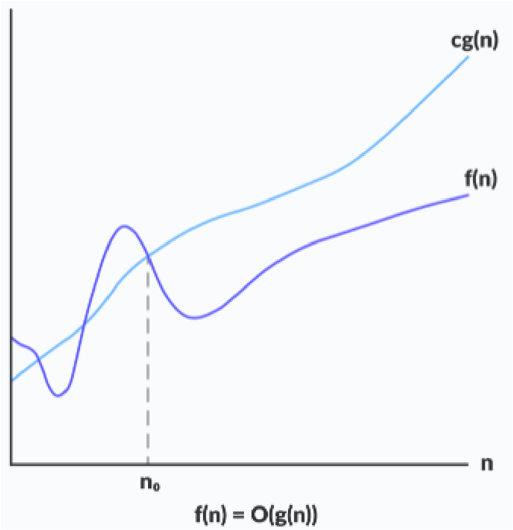
O-notation (Big-Oh)

- Big O Notation (Big-Oh)

Definition: Let f(n) , g(n) be functions, we say f(n) is of order g(n) if there is a constant c>0 such that $n \geq n_0$

$$f(n) = O(g(n))$$

$$\text{if } f(n) \leq C.g(n) \text{ for all } c, n_0 > 0 , n > n_0.$$



g(n) is asymptotic upper bound for f(n)

Note That:

- we use O-notation to provide an upper bound on the time for any input.
- the worst case running time of an algorithm is upper bound on the time for any input.
- the worst case running time gives us guarantee that the algorithm will never take any longer.

Example #1:

let $f(n) = n+ 5$ and $g(n) = n$ show that $f(n) = O(g(n))$ choose $c=6$.

answer:

$f(n) = O(g(n))$ if $f(n) <= c.g(n)$ for $c,n_0 > 0$

$n+5 <= c.n$

$n+5 <= 6n$

The condition has been proofed for any $n_0 > 0$

$$f(n) = O(n)$$

Example #2

Prove that the running time of $f(n) = 3n^2 + 10n$ is $O(n^2)$.

Proof:

by big oh definition

$$f(n) = O(n^2) \text{ if } f(n) <= C.g(n) \text{ for } c,n_0 > 0$$

$$3n^2 + 10n <= c.n^2$$

$$3 + 10/n <= c$$

when n_0 => 1 then

$$3+ 10 <= c$$

$$13 <= c$$

The condition has been proofed when c = 13 when n=1

Theory

if $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ *then* $f(n) = O(n^m)$

when a function is sum of several terms , its order of growth is determined by the fastest growth term.

Proof

$f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$

$f(n) = O(n^m)$ if $f(n) <= c.g(n)$ for $c, n_0 > 0$

$$| a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0 | <= c.n^m$$

$$(| a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0 |) / n^m <= c$$

when $n_0 = 1$

$$| a_m + a_{m-1} + \dots + a_1 + a_0 | <= c$$

.: $f(n) = O(n^m)$ when $c >= | a_m + a_{m-1} + \dots + a_1 + a_0 |$

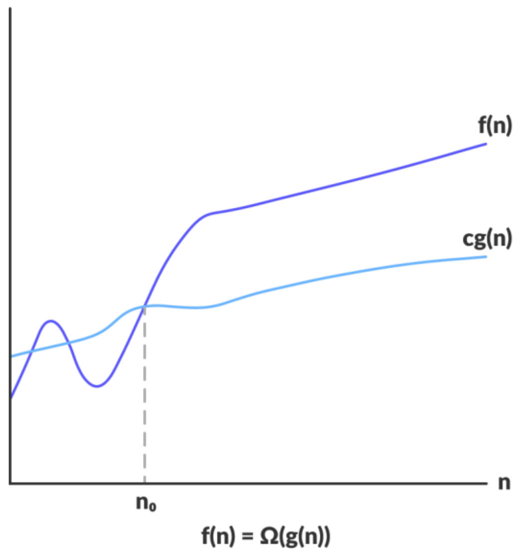
The condition has been proofed.

Ω Notation (Big Omega)

Ω Notation

Given two functions $f(n)$ and $g(n)$, we say that $f(n)$ is $\Omega(g(n))$ if there exists positive constants $n0$ and and c such that:

$$f(n) \geq c\, g(n) \quad \forall\, n \geq n_0$$



Example #1

show that $f(n) = 5n^2$ is $\Omega(n^2)$ when c=5 and $n_0=1$.

answer:

$$f(n) = \Omega(g(n)) \quad \text{if} \quad f(n) \Rightarrow c.g(n) \text{ for } c, n_0 > 0$$

$$5n^2 \Rightarrow c.n^2$$

$$5n^2 \Rightarrow 5n^2$$

when $n_0=1$

$$5 \Rightarrow 5$$

The condition is true.

Example #2

show that $f(n) = n^2$ is $\Omega(n)$ when c= 3

answer:

$f(n) = \Omega(g(n))$ if $f(n) => c.g(n)$ for $c, n_0 > 0$

$n^2 => c.n$

$n^2 => 3n$

when c= 3

$3^2 => 3*3$

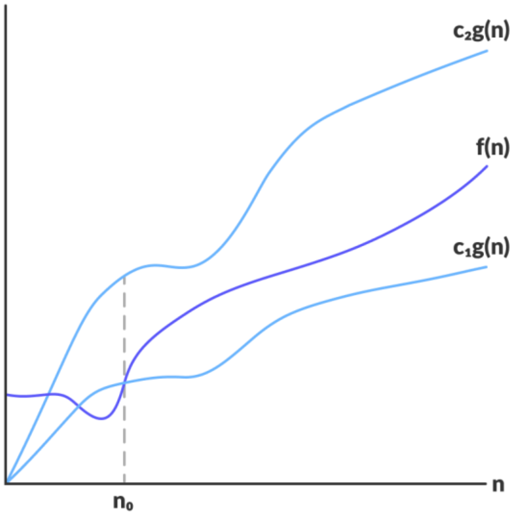
Then $f(n) = \Omega(n)$ when $n_0 = 3$

Θ Notation (Big Theta)

Θ Notation

Given two functions $f(n)$ and $g(n)$, we say that $f(n)$ is $\Theta(g(n))$ if there exists positive constants $n0$, $c1$ and $c2$ such that:

$$\forall\, n \geq n_0, c_1\, g(n) \leq f(n) \leq c_2\, g(n)$$



f(n) = Θ(g(n))

Example #1

let $f(n) = 3n+2$, $g(n) = n$ show that $f(n) = \Theta(g(n))$ when $c_1 = 3$, $c_2 = 4$.

answer:

$f(n) = \Theta(g(n))$ if $C_1.g(n) <= f(n) <= C_2 .g(n)$

$$3n <= 3n+2 <= 4n$$

when n=2

$$6 <= 8 <= 8$$

the condition has been proofed when c=3,c=4 for all n>1

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

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

Tripoli University - Faculty of Information Technology
Marwa Solla
m.solla@uot.edu.ly

Note That:

- $f(n) = \Theta(g(n))$ is both upper and lower bound on $f(n)$, this means that the worst and the best case require the same amount of time with in constant factor.

- the Θ-notation called a tight bound.

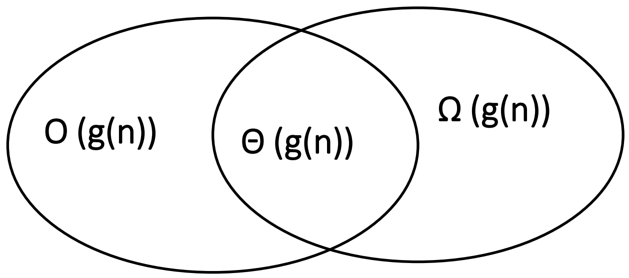
Theory:

For any 2 functions $f(n)$ and $g(n)$ we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O\ (g(n))$ and $f(n) = \Omega\ (g(n))$.

Big O (O()) describes the **upper bound** of the complexity.

Omega (Ω()) describes the **lower bound** of the complexity.

Theta (Θ()) describes the **exact bound** of the complexity.



Exercise 2

Write True or False :

$$T(n) = 5n^3 + 2n^2 + 4 \log n$$

1. $T(n) \in O(n^4)$
2. $T(n) \in O(n^2)$
3. $T(n) \in \Theta(n^3)$
4. $T(n) \in O(\log n)$
5. $T(n) \in \Theta(n^4)$
6. $T(n) \in \Omega(n^2)$

The End .