Asymptotic Notations

MD. MUKTAR HOSSAIN

LECTURER

DEPT. OF CSE

VARENDRA UNIVERSITY

Introduction

- Asymptotic Notations are mathematical tools used to analyze the performance of algorithms by understanding how their efficiency changes as the input size grows.
- There are mainly three asymptotic notations:
  - Big-O Notation (O-notation) ---- Worst Case
  - Omega Notation (Ω-notation) ---- Best Case
  - Theta Notation (Θ-notation) ---- Average Case

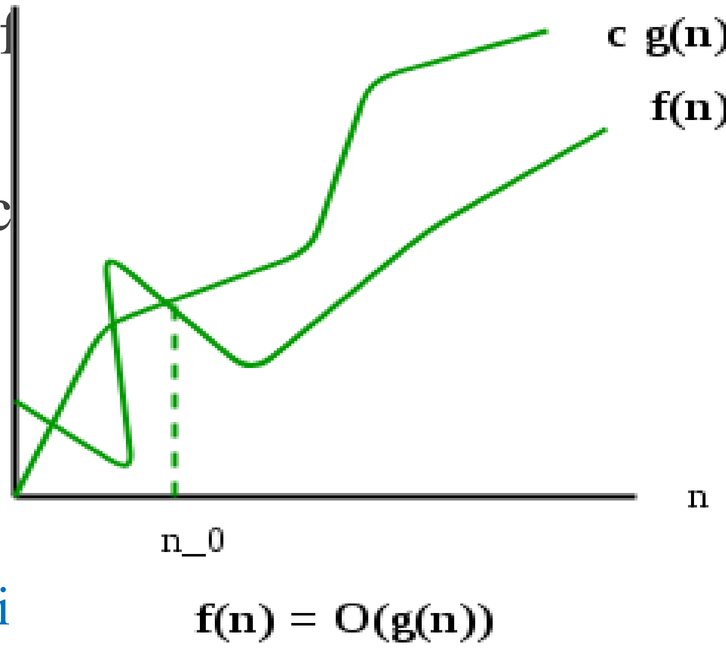
Big-O Notation (O-notation)

- The upper bound of the running time of an algorithm. Theref case complexity of an algorithm.
- $O(g(n))$ = { $f(n)$: there exist positive constants c and n0 suc for all $n \geq n0$ }
- Example: Let us consider a given function, $f(n)=4n^3+10n^2+5n+1$

  Considering $g(n)=n^3$,

  $f(n) \leqslant 20g(n)$, for all the values of $n>0$

  Hence, the complexity of $f(n)$ can be represented as $O(g(n))$, i



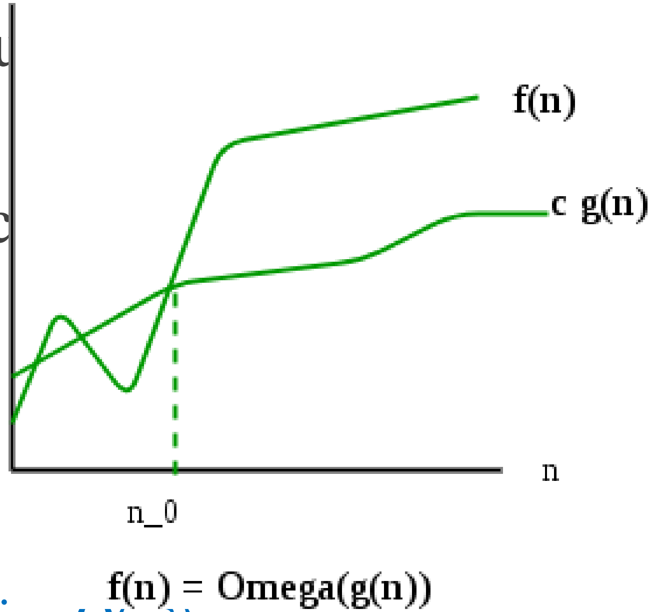
Omega Notation (Ω-Notation)

- The lower bound of the running time of an algorithm. Thu case complexity of an algorithm.
- $\Omega(g(n))$ = { f(n): there exist positive constants c and n0 suc for all $n \geq n0$ }
- Example: Let us consider a given function, $f(n)=4n^3+10n^2+5n+1$

  Considering $g(n)=n^3$

  $f(n) \geqslant 4g(n)$, for all the values of $n>0$

  Hence, the complexity of f(n) can be represented as $\Omega(g(n))$, i.e. $\Omega(n^3)$



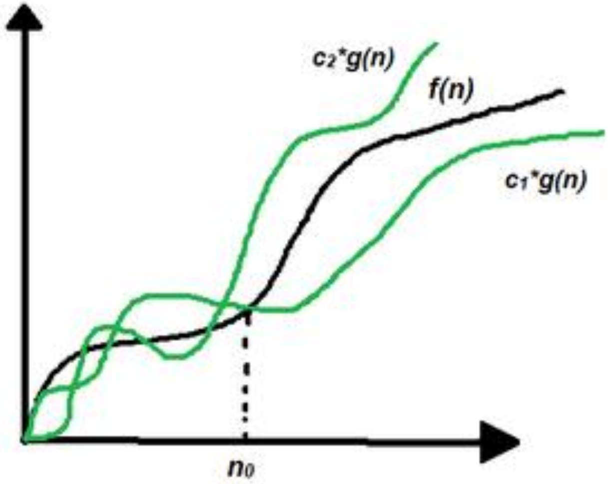
Theta Notation (Θ-Notation)

- The upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.
- $\Theta(g(n)) = \{f(n)$: there exist positive constants c1, c2 and n0 $\leq f(n) \leq c2 * g(n)$ for all $n \geq n0\}$
- Example: Let us consider a given function, $f(n)=4n^3+10n^2+5n+1$

  Considering $g(n)=n^3$

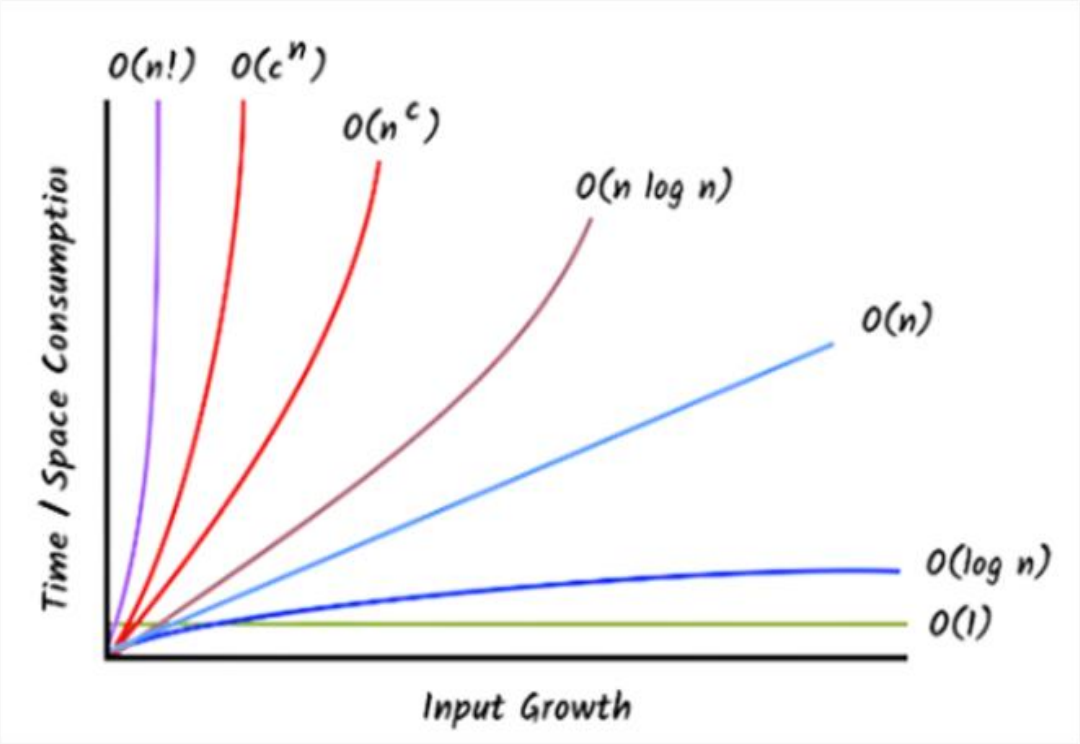
  $4.g(n)\leqslant f(n)\leqslant 20.g(n)$, for all the large values of n.

  Hence, the complexity of f(n) can be represented as $\theta(g(n))$, i.e. $\theta(n^3)$



Common Asymptotic Notations

Name	Notation
Constant	$O(1)$
Logarithmic	$O(\log n)$
Linear	$O(n)$
Linearithmic, Log-linear	$O(n \log n)$
Quadratic	$O(n^2)$
Polynomial	$O(n^c)$
Exponential	$O(c^n)$; where $c>1$
Factorial	$O(n!)$



Problems for Practice

1. Consider the following functions representing the runtime of different algorithms:

$f(n)=8n^2+3n+10$; $g(n)=5n+100$; $h(n)= 2n^3+7$

i. Determine the asymptotic complexity (Big-O) of each function.
ii. Rank them in terms of efficiency for large n, explaining your reasoning.

2. Consider the following functions representing the runtime of different algorithms:

$f(n)=6n^3+2n^2+15$; $g(n)=4n+\log n$; $h(n)= 3n^2+100$

i. Determine the asymptotic lower bound (Big-Ω) of each function.
ii. Rank the functions based on their minimum growth rate and explain.

3. Consider the following functions representing the runtime of different algorithms:

$f(n)=2n^2+50n+20$; $g(n)=7n+5$; $h(n)= 10n^3+n^2+2$

i. Prove that each function belongs to a specific asymptotic class using Big-Θ notation.
ii. Rank them by asymptotic growth rate with explanation.

Complexity Analysis of Recurrence Relation

Definition

A recurrence relation is a mathematical expression that defines a sequence in terms of its previous terms.

*General form of a **Recurrence Relation:***

$$a_n = f(a_{n-1}, a_{n-2}, \ldots a_{n-k})$$

Example:

Fibonacci Sequence	*F(n) = F(n-1) + F(n-2)*
Factorial of a number n	*F(n) = n * F(n-1)*
Merge Sort	*T(n) = 2*T(n/2) + O(n)*
Binary Search	*T(n) = T(n/2) + 1*

Solving Technique

Solving recurrences plays a crucial role in the analysis, design, and optimization of algorithms.

There are mainly three ways of solving recurrences:

- Substitution Method
- Recurrence Tree Method (**Study yourself**)
- Master Method

Substitution Method

It uses following steps to find Time Complexity using recurrences:

- Take the main recurrence and try to write recurrences of previous terms
- Take just previous recurrence and substitute into main recurrence
- Again take one more previous recurrence and substitute into main recurrence
- Do this process until you reach to the initial condition
- After this substitute the the value from initial condition and get the solution

Substitution Method Cont'd

Recurrence Relation: $T(n) = \begin{cases} 1 & , \; if \; n = 1 \\ T\left(\frac{n}{2}\right) + C, & if \; n > 1 \end{cases}$

Solution:

$$T(n) = T\left(\frac{n}{2}\right) + C \dots\dots\dots\dots\dots\dots (1)$$

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + C \dots\dots\dots\dots\dots\dots (2)$$

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + C \dots\dots\dots\dots\dots\dots (3)$$

After substituting (2) into (1), we get,

$$T(n) = T\left(\frac{n}{4}\right) + 2C$$

$$= T\left(\frac{n}{4}\right) + 2C = T\left(\frac{n}{2^2}\right) + 2C$$

$$= T\left(\frac{n}{8}\right) + 3C$$

$$= T\left(\frac{n}{2^3}\right) + 3C$$

$$\dots\dots\dots\dots\dots\dots$$

$$= T\left(\frac{n}{2^k}\right) + kC$$

$$= T(1) + kC \quad , [\text{Assume, } \frac{n}{2^k} = 1 \; or \; n = \; 2^k]$$

$$= 1 + kC$$

Here, $n = \; 2^k$.

So, $k = \log n$

Hence, $\boldsymbol{T(n) = O(\log n)}$

Substitution Method Cont'd

Calculate the Big-O notation of the following recurrences using Substitution method:

1. $$T(n) = \begin{Bmatrix} 1 & , \ if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & if\ n > 1 \end{Bmatrix}$$

2. $$T(n) = \begin{Bmatrix} 1 & , \ if\ n = 1 \\ n * T(n-1), & if\ n > 1 \end{Bmatrix}$$

Master Method (Theorem)

Usually used for divide and conquer algorithm.

Required Form:

$$\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)} \text{ or } \boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + \theta\left(n^k log^p n\right)} \quad , \ where\ a \geq 1, b > 1, k \geq 0$$

Solution:

Here,

n = size of the problem

a = number of sub-problems and *a >= 1*

n/b = size of each sub-problem

b > 1, k >= 0 and *p* is a real number.

Condition		T(n)
$\boldsymbol{a > b^k}$		$\boldsymbol{\theta(n^{log_b a})}$
$\boldsymbol{a = b^k}$	$p > -1$	$\boldsymbol{\theta(n^{log_b a} log^{p+1} n)}$
	$p = -1$	$\boldsymbol{\theta(n^{log_b a} log\ log\ n}$
	$p < -1$	$\boldsymbol{\theta(n^{log_b a})}$
$\boldsymbol{a < b^k}$	$p \geq 0$	$\boldsymbol{\theta\left(n^k log^p n\right)}$
	$p < 0$	$\boldsymbol{\theta\left(n^k\right)}$

Master Method Cont'd

Recurrence Relation: $T(n) = \begin{cases} 1 & , \ if\ n = 1 \\ T\left(\frac{n}{2}\right) + C, & if\ n > 1 \end{cases}$

Solution:

Here, a = 1, b = 2, k = 0, p = 0.

$a = 1 = 2^0 = b^k$ and p > -1.

So,

$$T(n) = \theta(n^{log_2 1} log^{0+1} n)$$
$$= \theta(n^0 \log n)$$
$$= \theta\ (\log n)$$

Recurrence Relation: $T(n) = \begin{cases} 1 & , \ if\ n = 1 \\ 3T\left(\frac{n}{2}\right) + log^2 n, & if\ n > 1 \end{cases}$

Solution:

Here, a = 3, b = 2, k = 0, p = 2.

$a > 1 = 2^0 = b^k$

So,

$$T(n) = \theta(n^{log_2 3})$$

Master Method Cont'd

Calculate the Big-O notation of the following recurrences using Master method:

1. $$T(n) = \begin{cases} 1 & , \; if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & if\ n > 1 \end{cases}$$

2. $$T(n) = \begin{cases} 1 & , \; if\ n = 1 \\ 8T\left(\frac{n}{2}\right) + n^2, & if\ n > 1 \end{cases}$$

3. $$T(n) = \begin{cases} 1 & , \; if\ n = 1 \\ 3T\left(\frac{n}{2}\right) + n^2, & if\ n > 1 \end{cases}$$

Homework

1. Determine when it is suitable to apply Substitution Method and when it is not.

2. Determine when it is suitable to apply Master Method and when it is not.