Advanced Algorithms & Data Structures

Asymptotic efficiency

- Asymptotic efficiency means study of algorithms efficiency for large inputs.

- To compare two algorithms with running times *f(n)* and *g(n),* we need a rough measure that characterizes how fast each function grows as n grows.

- *Hint:* use *rate of growth*
- Compare functions **asymptotically!**
  (i.e., for large values of *n*)

Functions ordered by growth rate

Function	Name
1	Growth is constant
logn	Growth is logarithmic
n	Growth is linear
nlogn	Growth is n-log-n
n^2	Growth is quadratic
n^3	Growth is cubic
2^n	Growth is exponential
n!	Growth is factorial

$1 < \log n < n < n\log n < n^2 < n^3 < 2^n < n!$

– To get a feel for how the various functions grow with n, you are advised to study the following figs:

Time	Name	Instance characteristic n					
		1	2	4	8	16	32
1	Constant	1	1	1	1	1	1
$\log n$	Logarithmic	0	1	2	3	4	5
n	Linear	1	2	4	8	16	32
$n \log n$	Log linear	0	2	8	24	64	160
n^2	Quadratic	1	4	16	64	256	1024
n^3	Cubic	1	8	64	512	4096	32768
2^n	Exponential	2	4	16	256	65536	4294967296
$n!$	Factorial	1	2	24	40326	20922789888000	26313×10^{33}

Figure 1.7 Function values

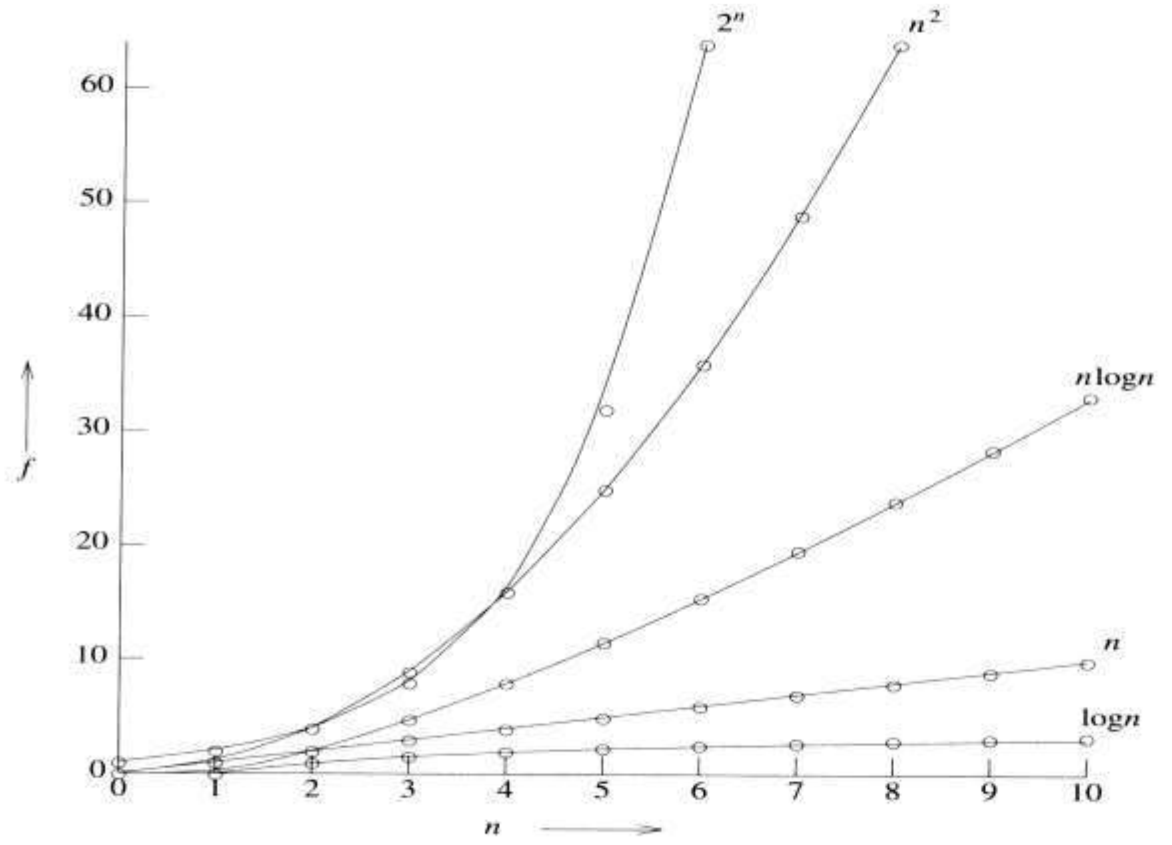


Figure 1.8 Plot of function values

- The low order terms and constants in a function are relatively insignificant for **large** *n*

$$n^2 + 100n + \log_{10}n + 1000 \quad \sim \quad n^2$$

i.e., we say that $n^2 + 100n + \log_{10}n + 1000$ and n^2 have the same **rate of growth**

Some more examples

- $n^4 + 100n^2 + 10n + 50$ is $\sim$ n^4
- $10n^3 + 2n^2$ is $\sim$ n^3
- $n^3 - n^2$ is $\sim$ n^3
- constants
  - 10 is $\sim$ 1
  - 1273 is $\sim$ 1

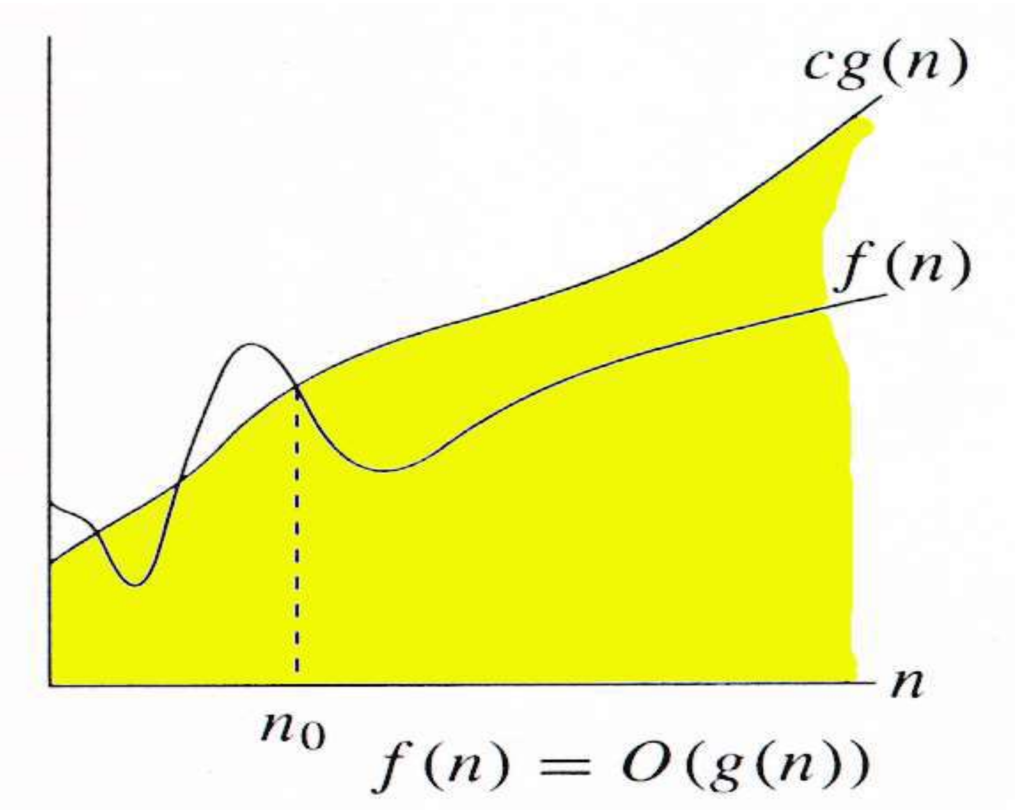
Asymptotic/order Notations

- Asymptotic/order notation describes the behavior of functions for the large inputs.
- **Big Oh(*O*)** notation:
  - The big oh notation describes an upper bound on the asymptotic growth rate of the function f.

Definition: [Big "oh"]

- f(*n*) = O(g(*n*)) (read as "*f* of *n* is big oh of *g* of *n*") iff there exist positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for all n, $n \geq n_0$.



- The definition states that the function f(n) is at most c times the function g(n) except when n is smaller than n_0.
- *In other words, f(n)* grows slower than or same rate as" g(n).
- When providing an upper –bound function g for f, we normally use a single term in n.
- **Examples**
  - $f(n) = 3n+2$
    - $3n + 2 <= 4n$, for all $n >= 2$, $\therefore 3n + 2 = O(n)$
  - $f(n) = 10n^2+4n+2$
    - $10n^2+4n+2 <= 11n^2$, for all $n >= 5$, $\therefore 10n^2+4n+2 = O(n^2)$
  - $f(n)=6*2^n+n^2=O(2^n)$ /* $6*2^n+n^2 \leq 7*2^n$ for $n \geq 4$ */

- It also possible to write $10n^2+4n+2 = O(n^3)$ since $10n^2+4n+2 <= 7n^3$ for $n>=2$

- Although n^3 is an upper bound for $10n^2+4n+2$, it is not a tight upper bound; we can find a smaller function ($n^{2)}$ that satisfies big oh relation.

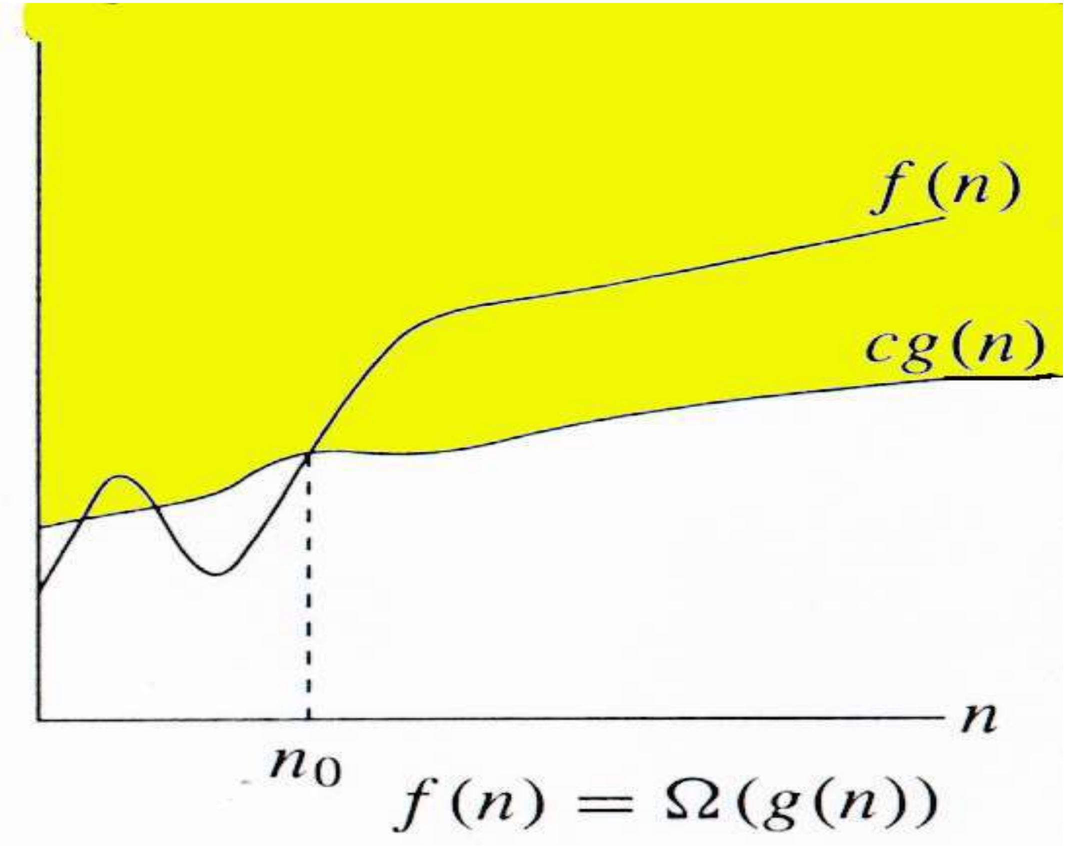
- But, we can not write $10n^2+4n+2 = O(n)$, since it does not satisfy the big oh relation for sufficiently large input.

- ***Omega*** **(Ω)** notation:
  - The omega notation describes a lower bound on the asymptotic growth rate of the function f.

Definition: [Omega]

- $f(n) = \Omega(g(n))$ (read as "*f* of *n* is omega of *g* of *n*") iff there exist positive constants *c* and n_0 such that $f(n) \geq cg(n)$ for all *n*,

$n \geq n_0$.

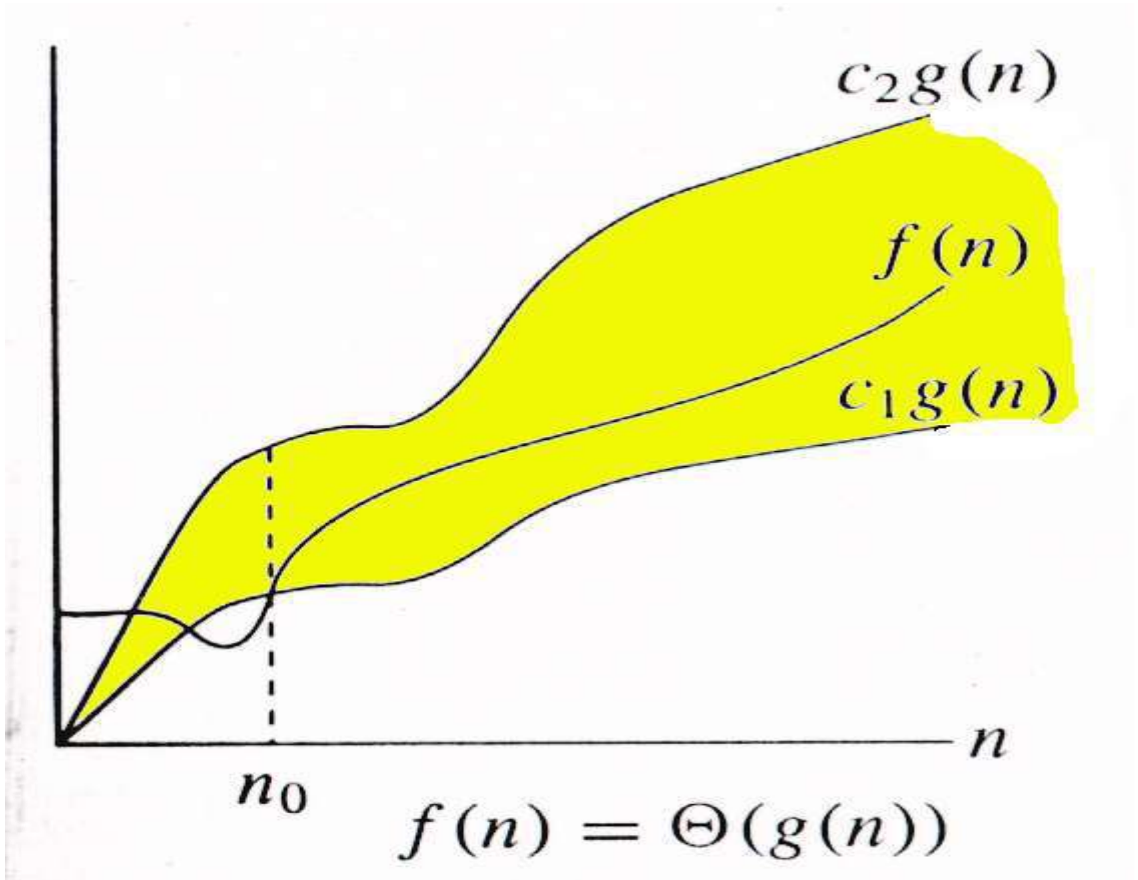


- The definition states that the function f(n) is at least c times the function g(n) except when n is smaller than n_0.
- *In other words, f(n)* grows faster than or same rate as" g(n).
- **Examples**
  - $f(n) = 3n+2$
    - $3n + 2 >= 3n$, for all $n >= 1$, $\therefore 3n + 2 = \Omega(n)$
  - $f(n) = 10n^2+4n+2$
    - $10n^2+4n+2 >= n^2$, for all $n >= 1$, $\therefore 10n^2+4n+2 = \Omega(n^2)$
- It also possible to write $10n^2+4n+2 = \Omega(n)$ since $10n^2+4n+2 >= n$ for $n>=0$
- Although n is a lower bound for $10n^2+4n+2$, it is not a tight lower bound; we can find a larger function (n^2) that satisfies omega relation.
- But, we can not write $10n^2+4n+2 = \Omega(n^3)$, since it does not satisfy the omega relation for sufficiently large input.

- ***Theta*** **(Θ)** notation:
  - The Theta notation describes a tight bound on the asymptotic growth rate of the function f.

Definition: [Theta]

- $f(n) = \Theta(g(n))$ (read as "*f* of *n* is theta of *g* of *n*") iff there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all n, $n \ge n_0$.



- The definition states that the function f(n) lies between c1 times the function g(n) and c2 times the function g(n) except when n is smaller than n_0.
- *In other words, f(n)* grows same rate as" g(n).

- ***Examples:-***
  - $f(n) = 3n+2$
    - $3n <= 3n + 2 <= 4n$, for all $n >= 2$, $\therefore$ $3n + 2 = \Theta(n)$
  - $f(n) = 10n^2+4n+2$
    - $n^2 <= 10n^2+4n+2 <= 11n^2$, for all $n >= 5$, $\therefore$ $10n^2+4n+2 = \Theta(n^2)$
- But, we can not write either $10n^2+4n+2 = \Theta(n)$ or $10n^2+4n+2 = \Theta(n^3)$, since neither of these will satisfy the theta relation.

Big-Oh, Theta, Omega

Tips :

- Think of O(g(n)) as "less than or equal to" g(n)
  - Upper bound: "grows slower than or same rate as" g(n)

- Think of Ω(g(n)) as "greater than or equal to" g(n)
  - Lower bound: "grows faster than or same rate as" g(n)

- Think of Θ(g(n)) as "equal to" g(n)
  - "Tight" bound: same growth rate

- (*True for large N*)

Example 1 - Iterative sum of n numbers

Statement	s/e	frequency	Total steps
Algorithm sum(a, n)	0	--	0
{	0	--	0
s:=0 ;	1	1	O(1)
for i:=1 to n do	1	n+1	O(n+1)
s:=s+a[i];	1	n	O(n)
return s;	1	1	O(1)
}	0	--	0
Total			O(n)

Example 2 - Addition of two m×n matrices

Statement	s/e	frequency	Total steps
Algorithm Add(a,b,c,m, n)	0	--	0
{	0	--	0
for i:=1 to m do	1	m+1	O(m)
for j:=1 to n do	1	m(n+1)	O(mn)
c[i,j]:=a[i,j]+b[i,j] ;	1	mn	O(mn)
}	0	--	0
Total			O(mn)

Time complexity of Towers of Hanoi

– $T(n) = T(n-1) + 1 + T(n-1) = 2T(n-1) + 1$

$= 2 * (2 * T(n-2) + 1) + 1$

$= (2 ^ 2) * T(n-2) + 2^1 + 2^0$

:

$= (2^k) * T(n-k) + 2^(k-1) + 2^(k-2) + ... + 2^0$

Base condition T(0) == 1
n – k = 0 => n = k;
put, k = n

T(n)=2^n T(0)+2^(n-1)+....+2^1+2^0

It is GP series, and sum is 2^(n+1)-1
T(n) = O(2^n) which is exponential.

SAMPLE QUESTIONS

- What are asymptotic notations in algorithm analysis
- Why are they important in evaluating the efficiency of algorithms
- What is the difference between the best-case, worst-case, and average-case time complexities represented by Big O notation
- What is rate of growth
- Provide examples of algorithms and their corresponding time complexity expressed in Big O notation.
- Discuss the properties of Big O notation
- What are the rules for comparing and combining functions represented by Big O notation?
- Explain the concept of Omega notation.
- How does it differ from Big O notation, and what does it represent in terms of lower bounds of time complexity
- What is function value

SAMPLE QUESTIONS CONT..

- What is the purpose of Theta notation
- How does it provide a tighter bound on the time complexity of an algorithm compared to Big O notation?
- Explain the concept of space complexity in asymptotic notations. How can it be represented using Big O notation?
- Discuss the concept of logarithmic time complexity and how it is represented using asymptotic notations.
- Provide examples of exponential time complexity and how it is expressed in asymptotic notations.