Fundaments of Algorithm

SYBSC (Computer Science)

Why Analysis of Algorithm

Goal of Analysis of Algorithm

- Compare algorithm in term of time , space and other factor

What is Running time analysis

issue: how processing time increases as the size of problems increases

Input may be different type

- Size of array
- Polynomial degree
- Number of Element in Matrix
- Number of bits in binary representation of input
- Vertices and edges in graph

How to Compare Algorithm

- **Execution Times**

  not good as it varies from pc to pc

- **Number of Statements**

  Not Good as it depends on programming language and style of programmer

- **Ideal Solutions**

  we expresse running time of given algo as a function of input size n i.e. f(n) and compare these different function corrosponding to running times

Rate of Growth

$$n^4+2n^2+100n+500 \;=n^4$$

Types of Analysis

To analyze algorithm we need to know on what input algorithm takes less time and long time.

Best Case

Best case performance used in computer science to describe an algorithm's behaviour under optimal conditions.

An example of best case performance would be trying to sort a list that is already sorted using some sorting algorithm.

Defines the input for which the algorithm takes lowest time
Input is the one for which the algorithm runs fastest.

Average Case

Average case performance measured using the average optimal conditions to solve the problem.

For example a list that is neither best case nor, worst case order that you want to be sorted in a certain order.

Provide a prediction about the running time of a algorithm
Assumes that the input is random

Worst Case
Worst case performance used to analyze the algorithm's behavior under worst case input and least possible to solve the problem.

It determines when the algorithm will perform worst for the given inputs.

An example of the worst case performance would be a a list of names already sorted in ascending order that you want to sort in descending order.

Defines the input for which the algorithm takes long time
Input is the one for which the algorithm runs slower

lower bound<=Average bound <= Upper bound

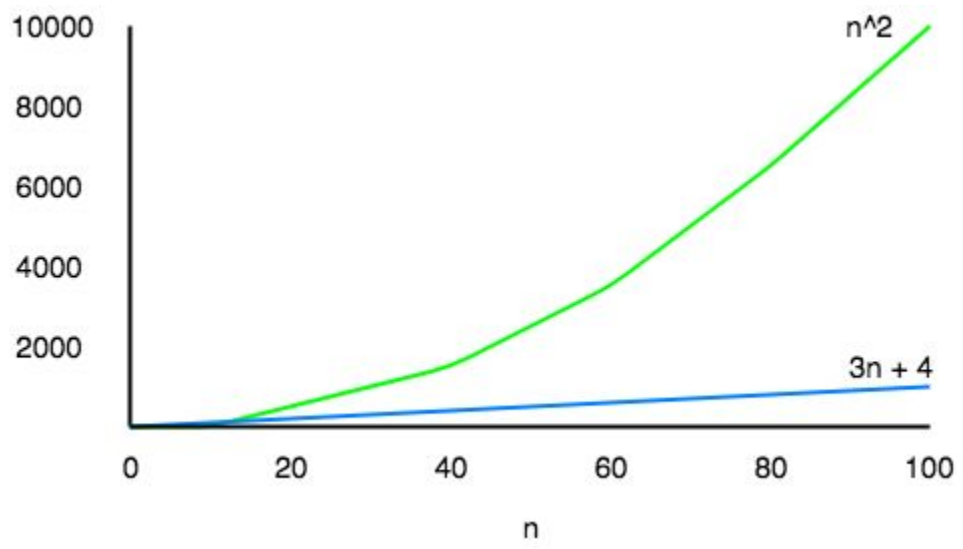
Asymptotic Notations.

When it comes to analysing the complexity of any algorithm in terms of time and space, we can never provide an exact number to define the time required and the space required by the algorithm, instead we express it using some standard notations, also known as **Asymptotic Notations**.

When we analyse any algorithm, we generally get a formula to represent the amount of time required for execution or the time required by the computer to run the lines of code of the algorithm, number of memory accesses, number of comparisons, temporary variables occupying memory space etc. but don't really tell us anything about the running time.

We need to identify lower and upper bounds.
To represent these lower and upper bounds we need some kind of syntax.

if some algorithm has a time complexity of $T(n) = (n^2 + 3n + 4)$, For large values of n, the 3n + 4 part will become insignificant compared to the n^2 part.

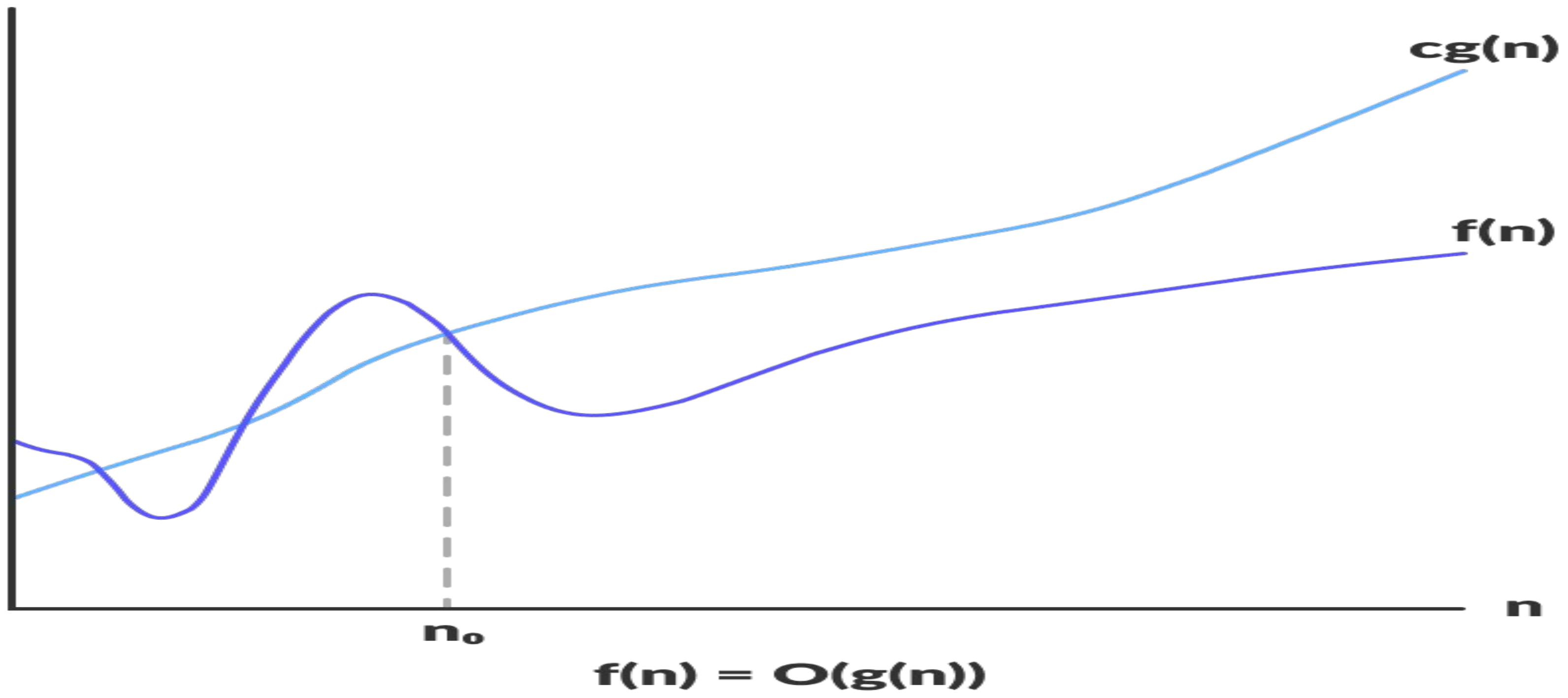


Upper Bounds: Big-O
This gives us tight upper bound of given function.
It is Represented as

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

This means at larger value of n the upper bound of $f(n)$ is $g(n)$.
if
$F(n)=n4+100n2+10n+3$

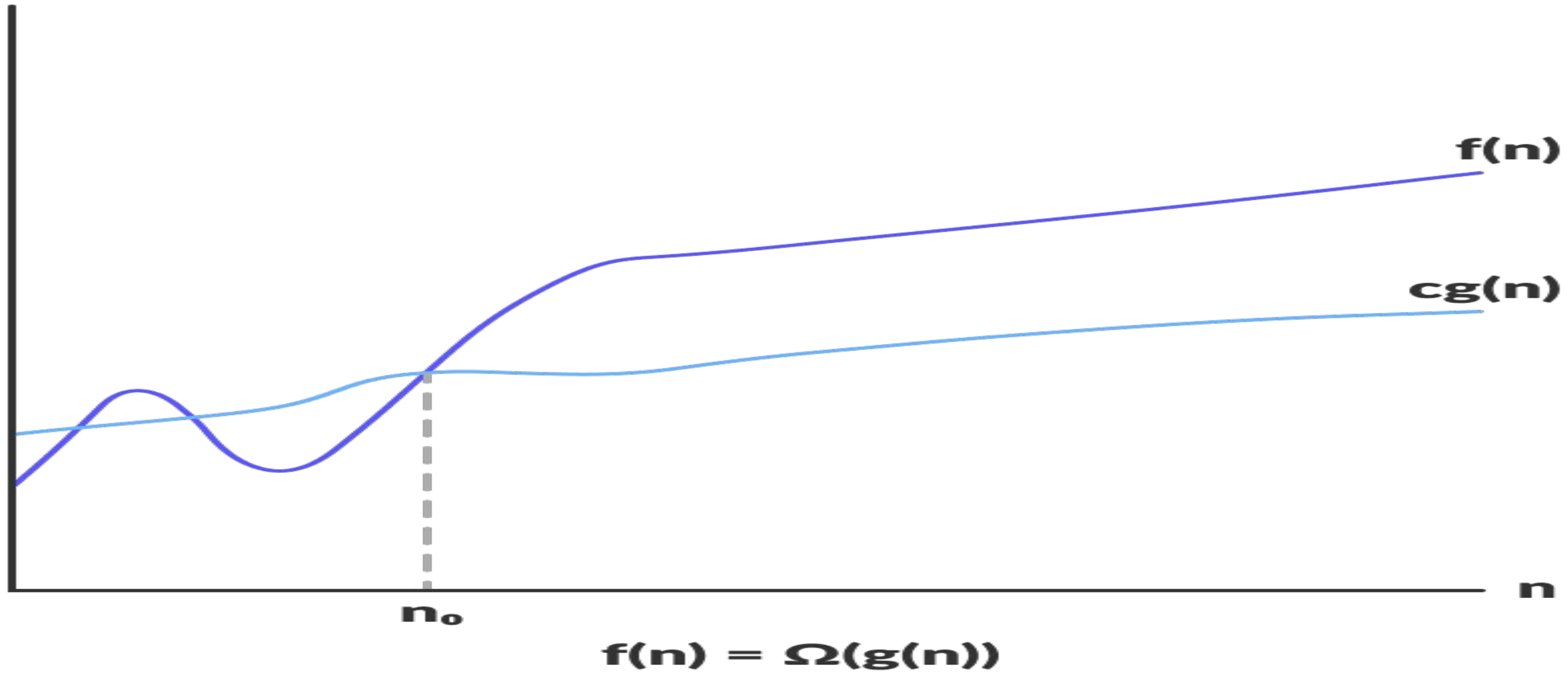
Then $g(n)= n4$ means $g(n)$ gives maximum rate of growth for $f(n)$ at larger value of n



$O(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ $\}$

Omega Notation (Ω-notation)

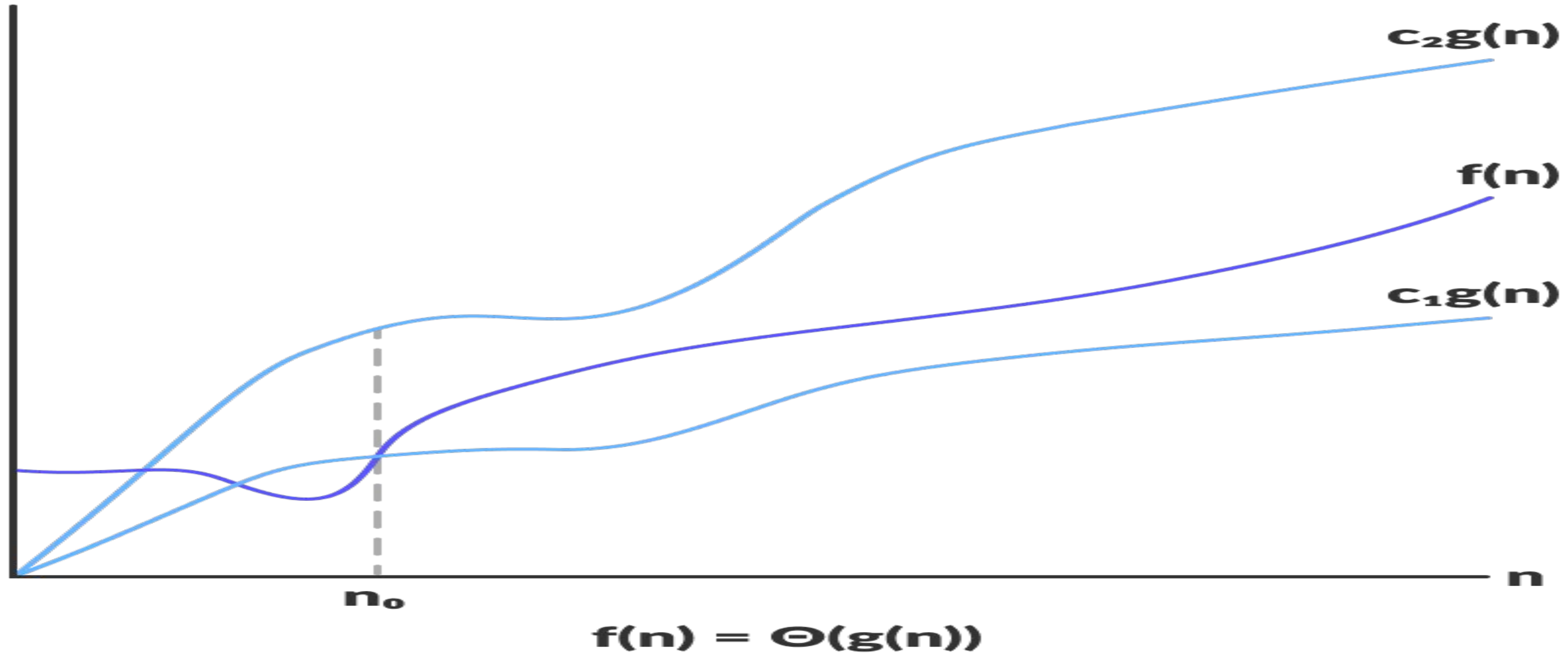
Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.



Ω(g(n)) = { f(n): there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$ }

Theta Notation (Θ-notation)

Theta notation encloses the function from above and below. Since it represents 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 c_1, c_2 and n_0 such that $0 \leq c_1g(n) \leq f(n) \leq c_2g(n)$ for all $n \geq n_0$ $\}$

Master theorem for Divide and Conquer

- Divide the problem into sub program then perform additional work to get final answer

- The above theorem is used to determine running time of divide and conquer algo.

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

where n = size of the problem
a = number of subproblems in the recursion and a >= 1
n/b = size of each subproblem
b > 1, k >= 0 and p is a real number.
Then,

if $a > b^k$, then $T(n) = \theta(n^{\log_b a})$
if $a = b^k$, then
(a) if $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
(b) if $p = -1$, then $T(n) = \theta(n^{\log_b a} \log\log n)$
(c) if $p < -1$, then $T(n) = \theta(n^{\log_b a})$

if $a < b^k$, then
(a) if $p >= 0$, then $T(n) = \theta(n^k \log^p n)$
(b) if $p < 0$, then $T(n) = \theta(n^k)$