3. Asymptotic Notations

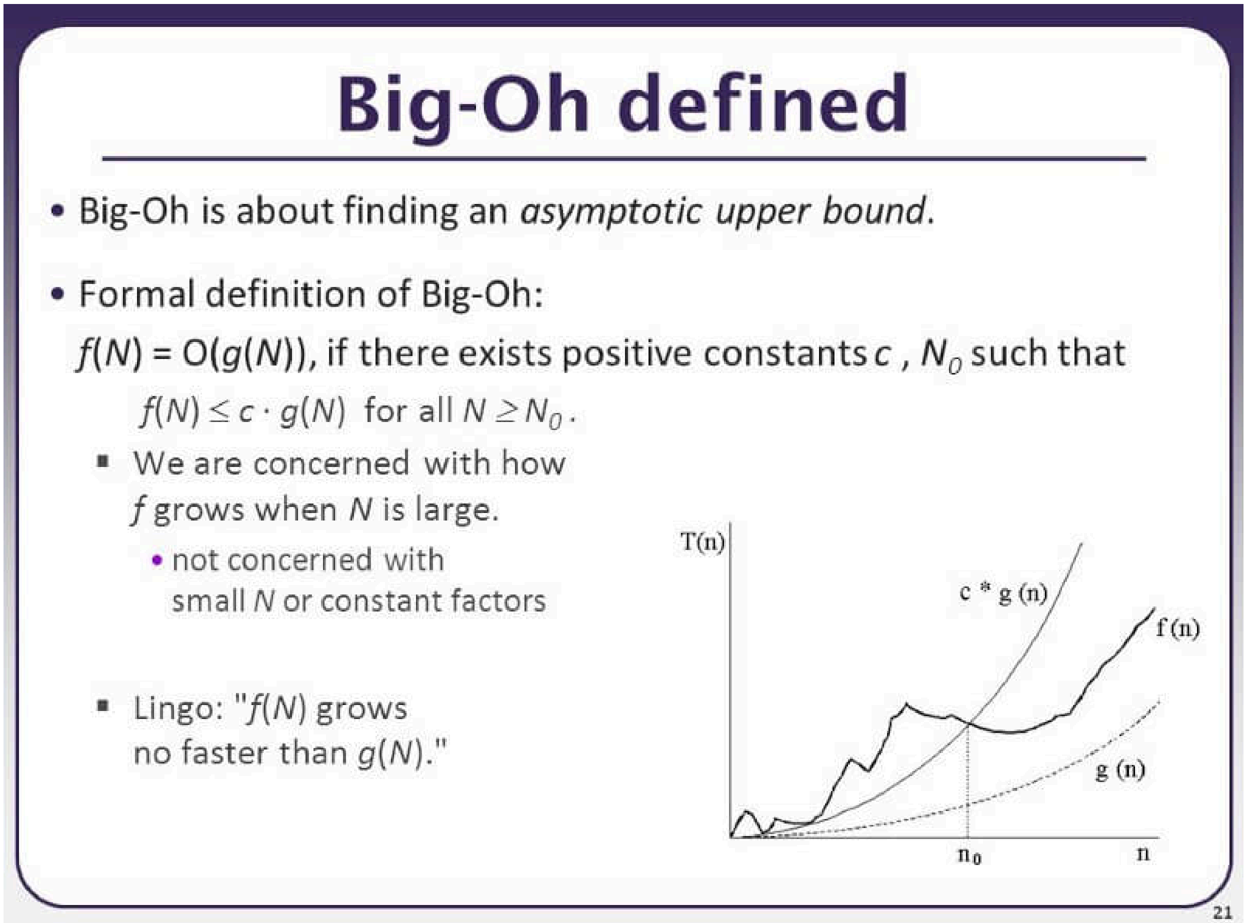
To measure 2. Complexities and Efficiencies of algorithm without running the code we use some mathematical tools known as Asymptotic Notations.

- We use asymptotic notations to compare the efficiencies of algorithm.
- It's a mathematical tool that estimates time based on input size without running the code.
- It focuses on how many basic operations the program perform, giving us an idea of how the algorithm behaves as input size increases.

Types of Asymptotic Notations

- Big O -> Describes that worst-case scenario or the upper bound of how the algorithm performs as the input size increases.
- Omega -> Describes the best case scenario or the lower bound.
- Theta -> Describes the average case or how the algorithm performs generally as input grows.

Big O Notation



n axis -> number of input

T axis -> time of processing

f(n) -> function which is working

C and n_0 -> these are the constants.

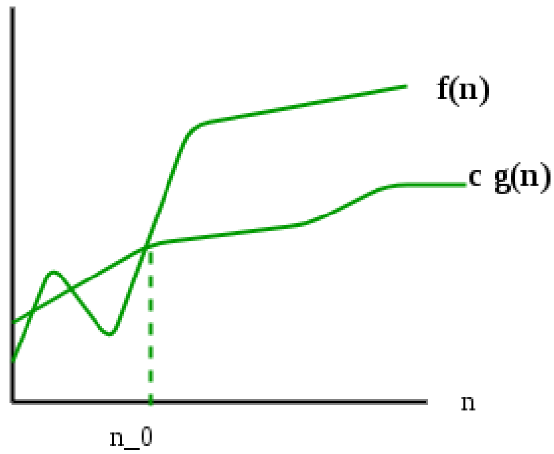
C g(n) -> another function g(n) which shows upper limit and C is the constant (this is actually which shows upper limit i.e. showing Big O notation in presence of C)

Explain : for f(n) whatever value you put for value of n it will always be smaller than C g(n) in terms of time.

Mathematically :

$$0 <= f(n) <= C\ g(n)$$

Omega Notation



f(n) = Omega(g(n))

n axis -> number of input

T axis -> time of processing

C and n_0 -> these are the constants.

f(n) -> function which is working

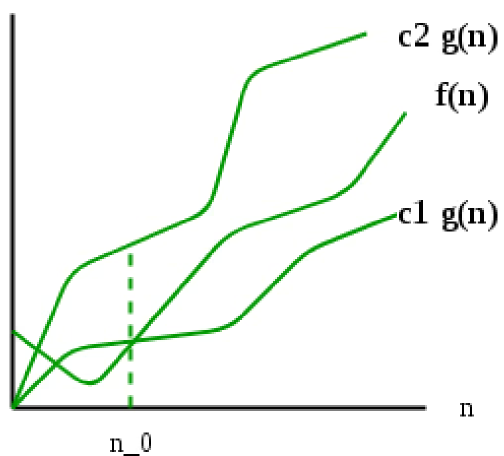
C g(n) -> another function g(n) which shows lower limit and C is the constant (this shows that c g(n) is at the lower limit and is the best case i.e. it will not come more low than that point i.e. the best-case)

Explain : for f(n) whatever value you put for value of n it will always be greater than c g(n) because that is the maximum lower limit algorithm can achieve for processing.

Mathematically :

f(n) = Omega(g(n))

Theta Notation



f(n) = theta(g(n))

n axis -> number of input
T axis -> time of processing
C1, C2 and n_0 -> these are the constants.
f(n) -> function which is working
C1 g(n) -> another function g(n) which shows lower limit and C is the constant (this shows that c g(n) is at the lower limit and is the best case i.e. it will not come more low than that point i.e. the best-case)
C2 g(n) -> another function g(n) which shows upper limit and C is the constant (this is actually which shows upper limit i.e. showing Big O notation in presence of C, i.e. no other function can perform beyond this point as because this is the worst case)

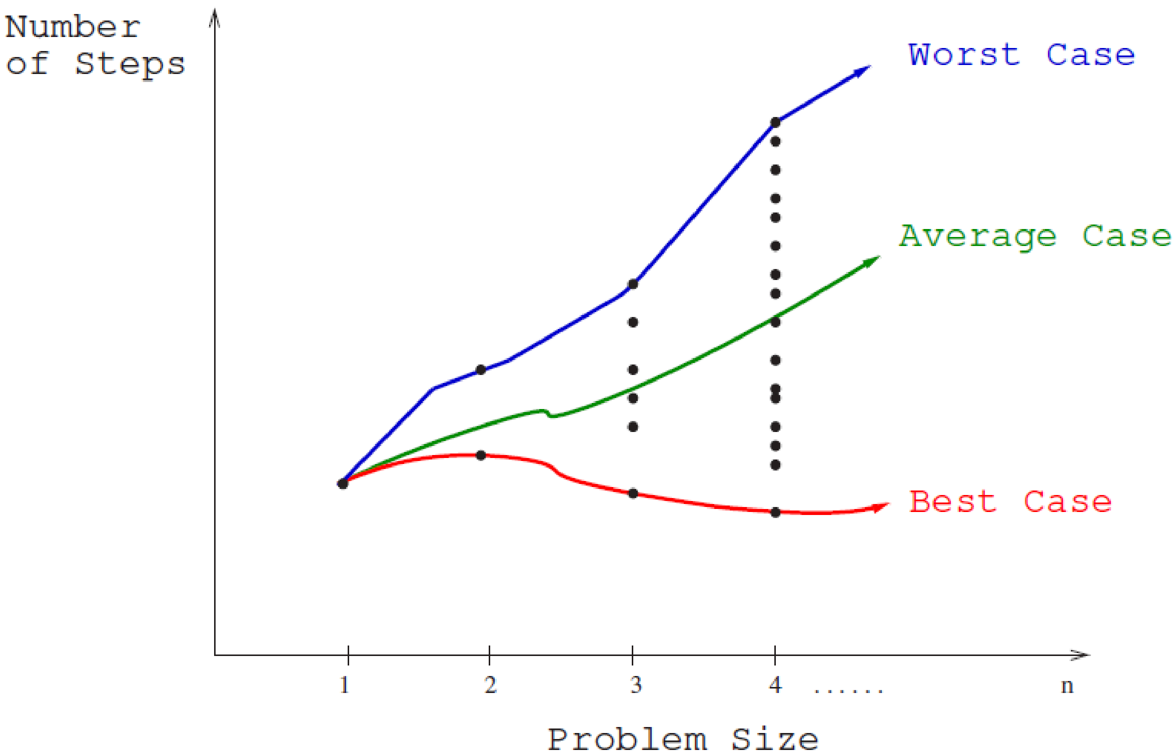
Explain : for f(n) whatever value you put for value of n it will always in between C1g(n) and C2g(n) which shows average case algorithm will work.
Mathematically :

$0<=c1g(n)<=f(n)<=C2g(n)$

Example :

Searching an element using Linear Search
3,4,5,6,7,8

Best case : when we have to search 3 in 1st attempt we can find it.
Worst case : when we have to search 8, algorithm has to traverse n times for time i.e. size of array to search the last element
Average case : when we have to search 6, algorithm will not find the element in 1st attempt and algorithm has not to traverse entire array to find the required element.


Number of Steps
Worst Case
Average Case
Best Case
1
2
3
4 ......
n
Problem Size