Asymptotic Notations

Asymptotic notations give us an idea about how good a given algorithm is compared to some other algorithm
Let us see the mathematical definition of "order of" now.

Primarily there are three types of widely used asymptotic notations.

1. Big oh notation (O)
2. Big omega notation (Ω)
3. Big theta notation (Θ) → Widely used one!

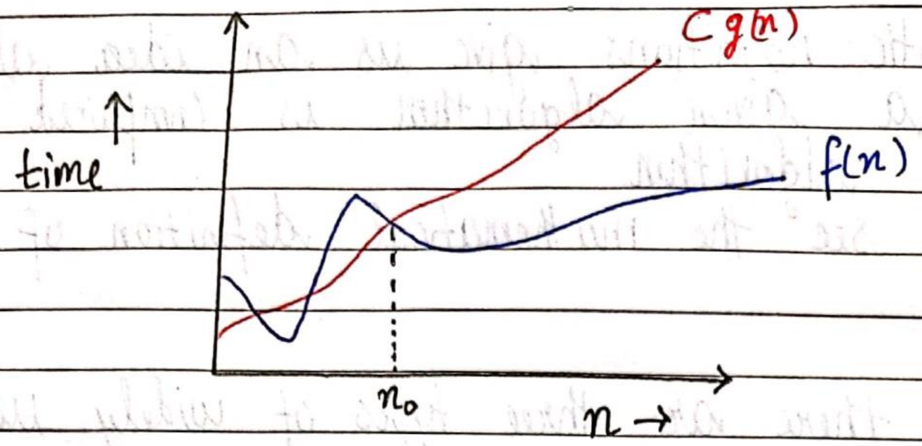
Big oh notation
Big oh notation is used to describe asymptotic upper bound
Mathematically, if $f(n)$ describes running time of an algorithm; $f(n)$ is $O(g(n))$ iff there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq c\,g(n) \quad \text{for all } n \geq n_0$$

⇓ used to give upper bound on a function.

if a function is $O(n)$, it is automatically $O(n^2)$ as well!

Graphic example for Big oh (O)



Big Omega notation

Just like O notation provides an asymptotic upper bound, Ω notation provides asymptotic lower bound. Let f(n) define running time of an algorithm;

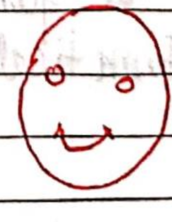
f(n) is said to be $\Omega(g\,n)$ if there exists positive constants c and n_0 such that

$$0 \leq c\,g(n) \leq f(n) \quad \text{for all } n \geq n_0$$

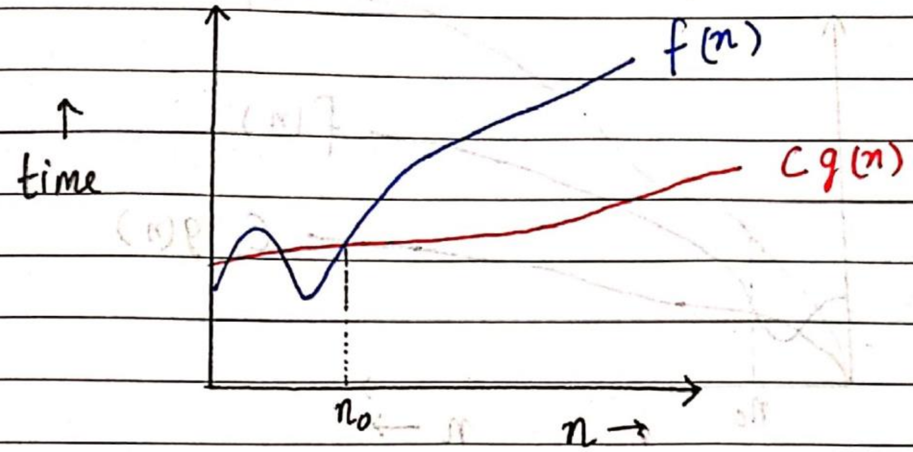
$\Downarrow$

used to give lower bound on a function

if a function is $O(n^2)$ it is automatically $O(n)$ as well



Graphic example for Big omega (Ω)



Big theta notation

Let $f(n)$ define running time of an algorithm

$f(n)$ is said to be $\Theta(g n)$ iff $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$

Mathematically,

$$0 \leq f(n) \leq C_1 g(n) \quad \forall n \geq n_0$$

$$0 \leq C_2 g(n) \leq f(n) \quad \forall n \geq n_0$$

Sufficiently large value of n

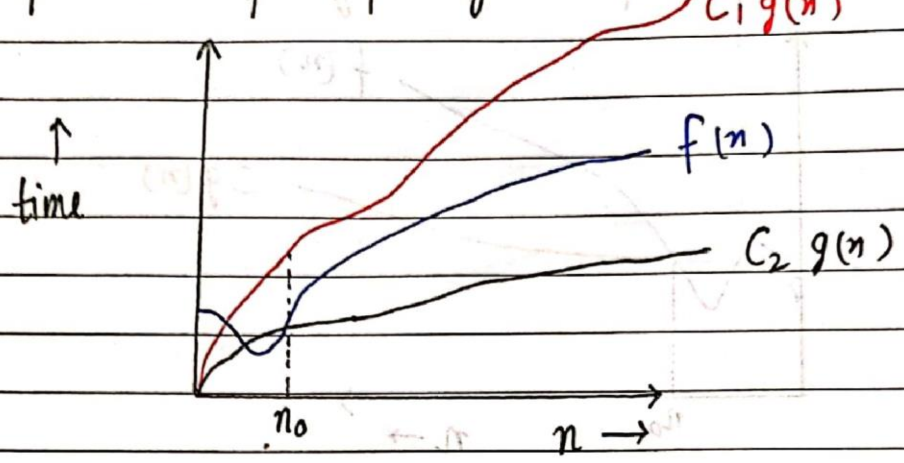
Merging both the equations, we get:

$$0 \leq C_2 g(n) \leq f(n) \leq C_1 g(n) \quad \forall n \geq n_0$$

The equation simply means there exist positive constants C_1 and C_2 such that $f(n)$ is sandwiched between $C_2 g(n)$ and $C_1 g(n)$

URBAN EDGE

Graphic example of Big theta



Which one of these to use ?

Since Big theta gives a better picture of runtime for a given Algorithm, most of the interviewers expect you to provide an answer in terms of Big theta when they say "Order of".

Quick Quiz :- Prove that n^2+n+1 is $O(n^3)$, $\Omega(n^2)$ and $\Theta(n^2)$ using respective definitions.

Increasing order of Common runtimes

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

Better

Worse

Common runtimes from better to worse

Best, worst and Expected Case

Sometimes we get lucky in life. Exams cancelled when you were not prepared, Surprise test when you were prepared etc. ⟹ Best case

Some times we get unlucky. Questions you never prepared asked in exams, rain during sports period etc. ⟹ Worst case

But overall the life remains balance with the mixture of lucky and unlucky times. ⟹ Expected case.

Analysis of a search algorithm

Consider an array which is sorted in increasing order

1	7	18	28	50	180

We have to search a given number in this array and report whether its present in the array or not.

Algo 1 → Start from first element until an element greater than or equal to the number to be searched is found.

Algo 2 → Check whether the first or the last element is equal to the number. If not find the number between these two elements (center of the array). If the center element is greater than the number to be searched, repeat the process for first half else repeat for second half until the number is found.

Analyzing Algo 1
If we really get lucky, the first element of the array might turn out to be the element we are searching for. Hence we made just one comparison.

Best case Complexity = $O(1)$

If we are really unlucky, the element we are searching for might be the last one.

Worst case complexity = $O(n)$

For calculating Average case time, we sum the list of all the possible case's runtime and divide it with the total number of cases.

⇓
Sometimes calculation of average case time gets very complicated

Analyzing Algo 2
If we get really lucky, the first element will be the only one which gets compared

Best case Complexity = $O(1)$

If we get unlucky, we will have to keep dividing the array into halves until we get a single element (the array gets finished)

URBAN EDGE

Worst case Complexity = $O(\log n)$

What $\log(n)$? What is that

$\log(n)$ → Number of times you need to half the array of size n before it gets exhausted

$$\log 8 = 3 \Rightarrow \frac{8}{2} \rightarrow \frac{4}{2} \rightarrow \frac{2}{2} \rightarrow \text{Cant break anymore.}$$

$$1 + 1 + 1$$

$$\log 4 = 2 \Rightarrow \frac{4}{2} \rightarrow \frac{2}{2} \rightarrow \text{Cant break anymore.}$$

$$1 + 1$$

Log n simply means how many time I need to divide n units such that we cannot divide them (into halves) anymore.

Space Complexity

Time is not the only thing we worry about while analyzing algorithms. Space is equally important.

Creating an array of size n → $O(n)$ space

↳ Size of input

If a function calls itself recursively n times its space complexity is $O(n)$

Quick Quiz → Calculate Space Complexity of a function which calculates factorial of a given number n.

Why cant we calculate Complexity in seconds?

→ Not everyone's Computer is equally powerful

→ Asymptotic analysis is the measure of how time (runtime) grows with input

Time Complexity & Big O notation

This morning I wanted to eat some pizzas; So I asked my brother to get me some from Dominos (3 km far)
He got me the pizza and I was happy only to realize it was too less for 29 friends who came to my house for a surprize visit!

My brother can get 2 pizzas for me on his bike but pizza for 29 friends is too huge of an input for him which he cannot handle.

2 pizzas →  okay! not a big deal!

68 pizzas →  Not possible! in short time

What is Time Complexity?
Time Complexity is the study of efficiency of algorithms.

Time Complexity = How time taken to execute an algorithm grows with the size of the input!

Consider two developers who created an algorithm to sort n numbers. Shubham and Rohan did this independently.

When ran for input size n, following results were recorded.

no of elements (n)	Shubham's Algo	Rohan's Algo
10 elements	90 ms	122 ms
70 elements	110 ms	124 ms
110 elements	180 ms	131 ms
1000 elements	2 s	800 ms

We can see that initially Shubham's algorithm was shining for smaller input but as the number of elements increases rohan's algorithm looks good!

Quick Quiz: Who's Algorithm is better?

Time Complexity: Sending GTA V to a friend

Let us say you have a friend living 5 kms away from your place. You want to send him a game.

Final exams are over and you want him to get this 60 GB file from you. How will you send it to him?

Note that both of you are using JIO 4G with 1 Gb/day data limit.

The best way to send him the game is by delevering it to his house.
Copy the game to a Hard disk and send it!

Will yo do the same thing for sending a game like minesweeper which is in kbs of size?
No because you cand send it via internet.

As the file size grows, time taken by online sending increases linearly $\rightarrow O(n^1)$
As the file size grows, time taken by physical sending remains constant. $O(n^0)$ or $O(1)$

Calculating Order in terms of Input size
In order to calculate the order, most impactful term containing n is taken into account.
$\hookrightarrow$ Size of input

Let us assume that formula of an algorithm in terms of input size n looks like this:

$$\text{Algo 1} \rightarrow \underbrace{k_1 n^2}_{\text{Highest order term}} + \underbrace{k_2 n + 36}_{\text{Can ignore lower order terms}} \Rightarrow O(n^2)$$

$$\text{Algo 2} \rightarrow k_1 k_2^2 + k_3 k_2 + 8$$

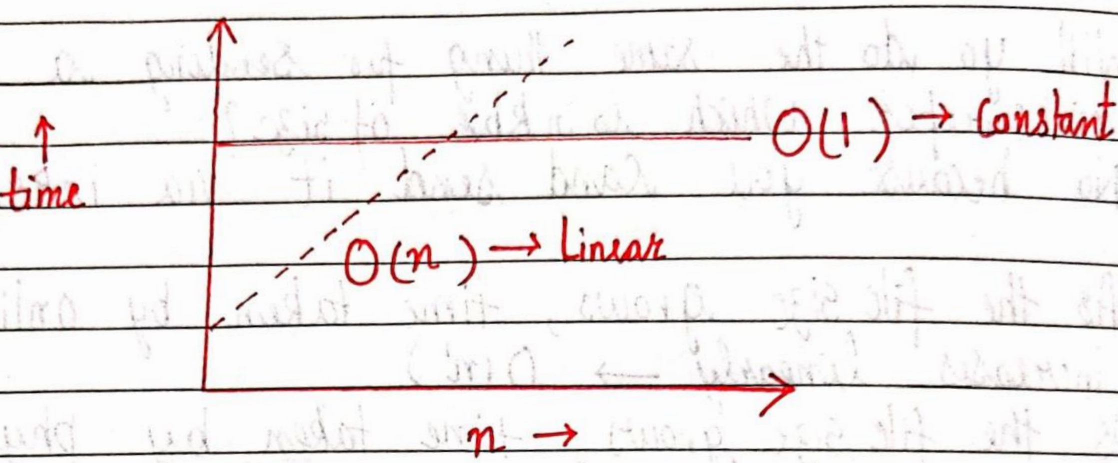
$$\Downarrow$$

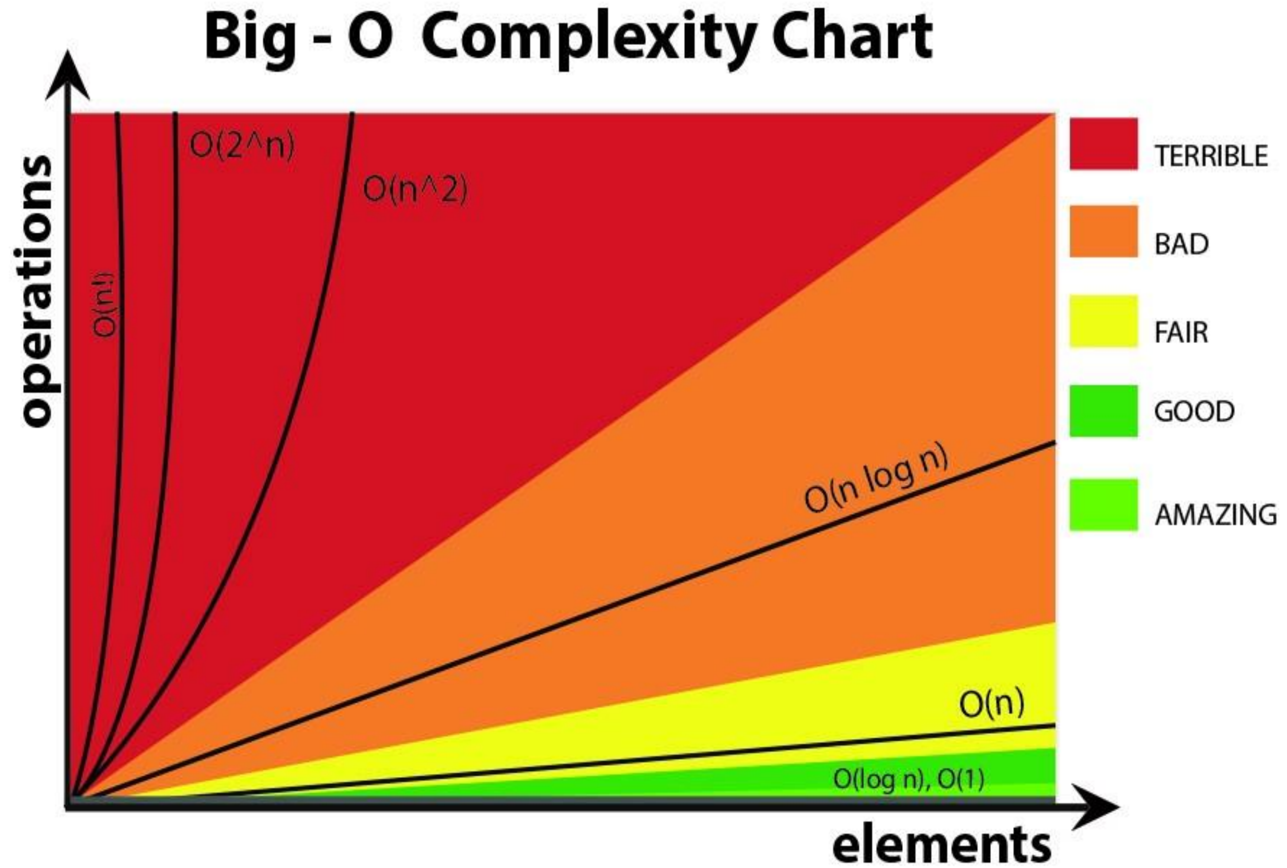
$$k_1 k_2^2 n^0 + k_3 k_2 + 8 \Rightarrow O(n^0) \text{ or } O(1)$$

Note that these are the formulas for time taken by them.

Visualising Big O

If we were to plot O(1) and O(n) on a graph, they will look something like this :





Source: https://stackoverflow.com/questions/3255/big-o-how-do-you-calculate-approximate-it