20-01-22 | See Abdul Bari videos on youtube for computing complexity.

TIME COMPLEXITY: fn that tells us relationship of how the time is going to grows as the input grows. size grows

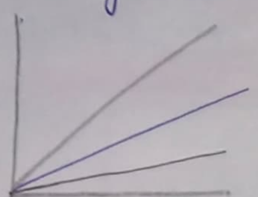
mathematical fn

→ Time complexity != Time taken

→ Time taken varies from machine to machine, so we see relationship b/w time grows when input grows

→ Things consider when thinking about complexity:

* Always look for worst case complexity.
* Always look at complexity for large / ∞ data.
*



} O(N)

$y = x$

$y = 2x$

$y = 4x$

Even though the value ($n, 2n, 4n$) is differ, they are all growing linearly.

we don't care about actual time.

Do we need constants?

* we only care about relationship of how time grows, when input grows.
  So, No we don't need constants. Above are some pts. why we ignore all constants.

* Always ignore less dominating terms.

eg. lets say, you've complexity

$$O(N^3 + \log(N))$$

pt 2). we'll look complexity for large / ∞ data

∴ we data amnt. → 1 million/sec

$N = 1$ million

$\Rightarrow O((1M)^3 + \log(1M))$

$\Rightarrow (1M)^3$ sec + $\boxed{6 \text{ sec}}$ very small compared to $(1M)^3$. Hence, ignore it example.

$O(3N^3 + 4N^2 + 5N + 6)$

① ignore constants

$N^3 + N^2 + N$

② ignore less dominating terms

$N^3 \rightarrow O(N^3)$

$O(3N^3 + 4N^2 + 5N + 6) \Rightarrow O(N^3)$

→ we write const. as $O(1)$ because constants doesn't count and necessary so for all we write it as $O(1)$

Big-oh Notation : Denoted by O

in simple language →

let say, we have $O(N^3)$ → upper Bound

Here, N^3 refers to size and Big-oh saying, that the complexity will not exceed N^3. (N^3 is the upper bound)

meaning ↗

mathematically →

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

$$\lim_{n \to \infty} = \frac{f(n)}{g(n)} < \infty$$ It is actually some finite value.

we have to look for worst case & complexity for large/∞ data

* Our algo will not exceed the complexity. It can be solved in less complexity but in any case it will never exceed.

It can be better but never exceed.

~~not imp~~

Big-omega notation : opposite of Big-oh-

in words → : ~~denoted by Ω~~

eg — $\Omega(N^3)$ → lower bound

it means that, it'll take atleast N^3 time complexity.
it can take above N^3 but not below N^3.

lower bound → minimum N^3 time complexity required.

mathematically →

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$$

⇒ we always look at the worst case, so we do actually care about Big-oh-notation.

not imp

Big-theta Notation : combining big-oh-notation and big-omega notation.

→ θ

$$0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

not imp

little-oh-Notation : This is also giving upper bound but, this is not strict upper bound.

→ denoted by o.

in words →

→ loose upper bound

Big-oh	little-oh
$f = O(g)$	$f = o(g)$
growth of f is no faster than g	f is smaller than g
$\therefore f \leq g$	$\therefore f < g$
	→ strictly lower than g.

→ mathematically →

If f is strictly slower than g, then you can say numerator is slower than denominator, it should give us 0/zero.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

$f = n^2$, $g = n^3$

$$\lim_{n \to \infty} \frac{n^2}{n^3} \Rightarrow \lim_{n \to \infty} \frac{1}{n} = 0$$

not imp

little omega : loosely lower bound.

: denoted by ω

Big Ω	little ω
$f = \Omega(g)$ means $f \geq g$ lower bound	$f = \omega(g)$ means $f > g$ strictly greater difference.

mathematically

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} = \infty$$

SPACE COMPLEXITY

: input space + auxiliary space

: Total space taken by algo w.r.t. input size.

→ **Auxiliary Space** – extra space or temporary space used by algorithm.