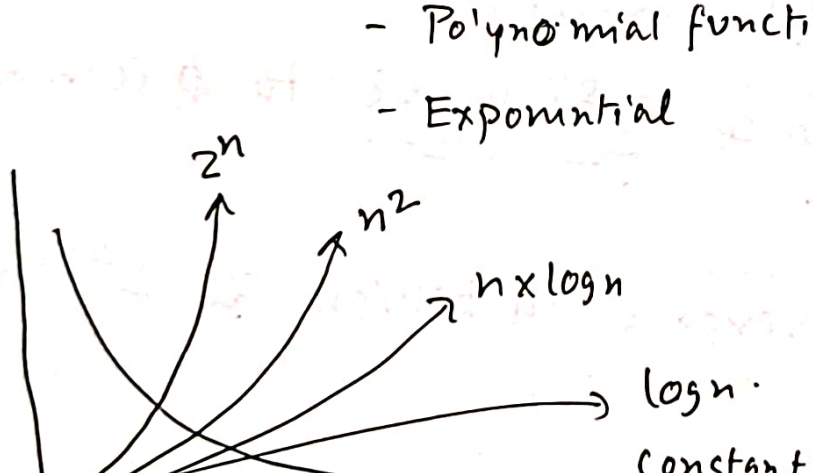
- COVERED IN GOODNOTES
  - Priori & Posteriori
  - Importance of Algorithmic Analysis
  - Importance of Growth rates
  - Big O Notation
  - Big O → Linear Search, Binary Search
  - Growth rates
  - Problems - (7-8)
  - Common Runtimes
  - Functions in Asymptotic notations
    - Decrement
    - constant
    - Logarithmic functions
    - Polynomial functions.
    - Exponential

* Take print from Good Notes [except: Big Ω]



→ • Big Ω (Omega) notation. (described here)

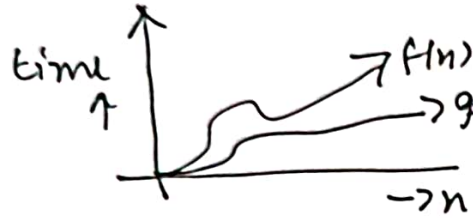
Big Ω Omega Notation

- Describes lower bound of an algorithm

$f(n) \geq c \cdot g(n)$.

↳ lower bound.

- Big Ω can be used for "Best case"
  - ~~Best case~~ run time
  - ~~"~~ memory space

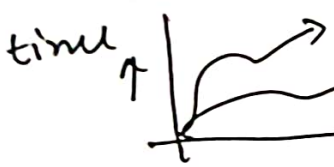


← Ω(n)

f(n) can't grow less than g(n)

- Big O → "worst case"

$f(n) \leq c \times g(n)$. ← upper bound.



* f(n) will not grow bigger than c×g(n).

← O(n)

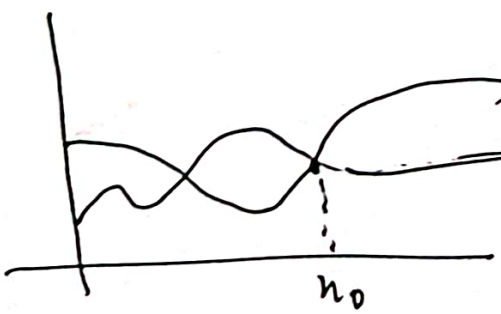
Definition

- Assume f(n) & g(n) are = non-negative functions
- $f(n) = \Omega(g(n))$ if $f(n) \geq c \cdot g(n)$ for some $c > 0$ and for all values of n where $n \geq n_0$
- c, n_0 = are constants.

- g(n) is asymptotic tight lower bound of f(n)

↓

Closest lower bound

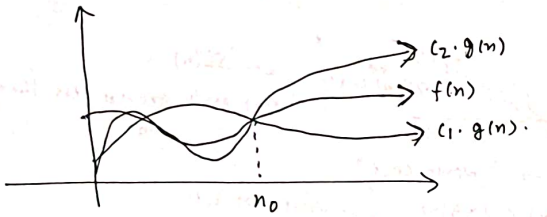


* After n_0, f(n) is always greater than g(n).

Theta
Big θ notation

Def

Assuming $f(n)$ & $g(n)$ are non-negative functions.
$f(n) = \theta(g(n))$ iff $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all values of n where $n \geq n_0$ and c_1, c_2 and n_0 are constants.



- $c_2 \cdot g(n)$ => upper bound of $f(n)$.
- $c_1 \cdot g(n)$ => lower bound of $f(n)$.

* we are interested in the growth rates after n_0.

[$f(n)$ can't grow more than $c_2 \cdot g(n)$
n n n less than $c_1 \cdot g(n)$]

Question :- Why do we need to know $\theta(n)$.?

- Observe :
  - $g(n)$ is both CLOSEST upper bound of $f(n)$. & lower

* Here $g(n)$ is the asymptotic tight bound of $f(n)$.

closest lower & upper bound

Problems

Topics

- ☐ Big Omega Notation - Solved Problem
- ☐ Big Theta Notation - Solved Problem.

Ex 1. Find the lower bound for $f(n) = 10n^2 + 5$.

↓
finding a function that grows asymptotically lesser than $f(n)$.

- How did we find the upper bound ?

Steps
(1) Find the dominant term
(2) Assume some $g(n)$ that is near to the dominant term.
(3) Try $f(n) \leq c \cdot g(n)$

find a relevant c and find n_0.

* Here, the first two steps are same.

(i) Find the dominant term.
$10n^2$

(ii) $g(n) = n^2$ ← I'll choose this, $n^2 \log n$, n^3, n^4.

According to Big Omega notation,
$f(n) = \Omega(g(n))$ iff $f(n) \geq c \cdot g(n)$ for some $c > 0$ & for all values of n where $n \geq n_0$ and c and n_0 are constants.

Let say $c=10$

$f(n) \geq c \cdot g(n)$ $\quad c=10$, $g(n)=n^2$

$10n^2+5 \geq 10n^2$ $\Rightarrow$ This is true for all values of $n \geq 1$

note:-

$n \neq 0$ (n = i/p size).

$\therefore$ $10n^2+5 = \Omega(n^2)$ for $c=10$ and $n_0=1$

Ex 2:- Show that $f(n) = n^3 + 3n^2 = \theta(n^3)$.

Steps

(1) Find the dominant term $= n^3$.

Defn.

~~We will have a~~

$f(n) = \theta(g(n))$ iff $c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$ for all values of n where $n \geq n_0$ and c_1, c_2 and n_0 are constants.

(a) Let's first prove $f(n) \leq c_2 g(n)$.

(b) Then we will show $c_1 g(n) \leq f(n)$.

For (a), let's assume $c_2 = 2$.

$n^3 + 3n^2 \leq 2n^3$.

$\Rightarrow n^3 - 3n^2 \geq 0$.

$\Rightarrow n^2(n-3) \geq 0$.

~~This is true for all values of $n \geq 1$~~

This is true for all values of $n \geq 3$.

Hence, we can say $n^3+3n^2 = O(n^3)$ for $f(n) \leq c_2 g(n)$ for $c_2 = 2$ and $n_0 = 3$

Now for (b), we have to prove

$f(n) \geq c_1 g(n)$. $\rightarrow \Omega(n)$.

$g(n) = n^3$, $f(n) = n^3 + 3n^2$

$n^3 + 3n^2 \geq n^3$. Let say $c_1 = 1$

$\Rightarrow 3n^2 \geq 0$. $\rightarrow$ for all values of $n \geq 1$ - this satisfies.

$\therefore f(n) = \Omega(n^3)$ for $c_1 = 1$ and $n_0 = 1$

$\therefore c_1 g(n) \leq f(n) \leq c_2 g(n)$.

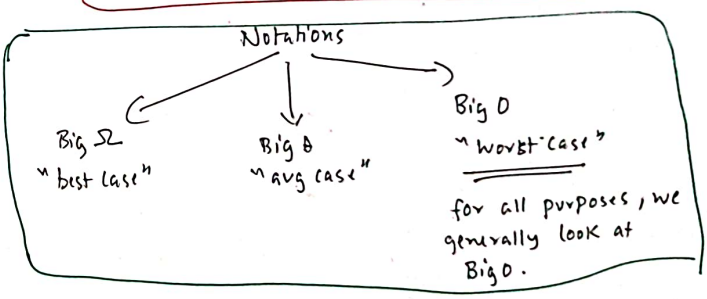
$\therefore$ $f(n) = \theta(n^3)$ for

$c_1 = 1$, $n_0 = 1$, where $f(n) = \Omega(n^3)$.

$c_2 = 2$, $n_0 = 3$ where $f(n) = O(n^3)$.

Here we only take $n_0 = 3$

$c_1 = 1$, $c_2 = 2$, $n_0 = 3$



Let say $c = 10$

$f(n) \geq c \cdot g(n)$ $\quad c = 10$, $g(n) = n^2$

$10n^2 + 5 \geq 10n^2$ $\Rightarrow$ This is true for all values of $n \geq 1$

note:- $n \neq 0$ (n = i/p size).

$\therefore$ $10n^2 + 5 = \Omega(n^2)$ for $c = 10$ and $n_0 = 1$

Ex 2:- Show that $f(n) = n^3 + 3n^2 = \theta(n^3)$.

Steps

(1) Find the dominant term $= n^3$.

Defn.

~~We will have a~~

$f(n) = \theta(g(n))$ iff $c_1 \times g(n) \leq f(n) \leq c_2 * g(n)$ for all values of n where $n \geq n_0$ and c_1, c_2 and n_0 are constants.

(a) Let's first prove $f(n) \leq c_2 g(n)$.

(b) Then we will show $c_1 g(n) \leq f(n)$.

For (a), let's assume $c_2 = 2$.

$n^3 + 3n^2 \leq 2n^3$.

$\Rightarrow n^3 - 3n^2 \geq 0$.

$\Rightarrow n^2(n-3) \geq 0$.

~~This is true for all values of $n \geq 1$~~

This is true for all values of $n \geq 3$.

Hence, we can say $n^3 + 3n^2 = O(n^3)$ for $f(n) \leq c_2 g(n)$ for $c_2 = 2$ and $n_0 = 3$

Now for (b), we have to prove

$f(n) \geq c_1 g(n)$. $\rightarrow \Omega(n)$.

$g(n) = n^3$, $f(n) = n^3 + 3n^2$

$n^3 + 3n^2 \geq n^3$. Let say $c_1 = 1$

$\Rightarrow 3n^2 \geq 0$. $\rightarrow$ for all values of $n \geq 1$ - this satisfies.

$\therefore f(n) = \Omega(n^3)$ for $c_1 = 1$ and $n_0 = 1$

$\therefore c_1 g(n) \leq f(n) \leq c_2 g(n)$.

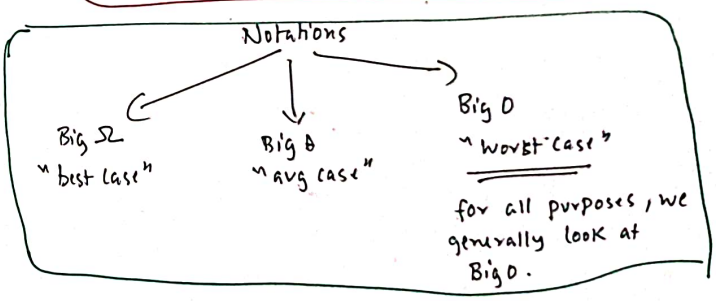
$\therefore$ $f(n) = \theta(n^3)$ for

$c_1 = 1$, $n_0 = 1$, where $f(n) = \Omega(n^3)$.

$c_2 = 2$, $n_0 = 3$ where $f(n) = O(n^3)$.

Here we only take $n_0 = 3$

$c_1 = 1, c_2 = 2$
$n_0 = 3$



Properties of Asymptotic Notations

General Property

If $f(n) = O(g(n))$ for all

which means $f(n) \leq c \cdot g(n)$ where $c > 0$, and $n \geq n_0$

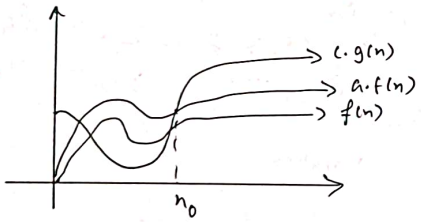
where c, n_0 = some constants → we know this's from Big O.

Now, the general property states that

$a \times f(n) = O(g(n))$

$a \times f(n) \leq c \cdot g(n)$ → By definition

or $c \cdot g(n)$ will be asymptotically bigger than $a \times f(n)$



Note: We come up with 1 n_0 like we did in the last problem.

Example :- $f(n) = 3n^2 + 9$ and $g(n) = n^2$. Is $f(n) = O(g(n))$

$3n^2 + 9 \leq c \cdot n^2$

If $c = 4$

$n^2 \geq 9$

$n \geq 3$

$a \times f(n) = a(3n^2 + 9)$

Let $a = 1000$

$\therefore a \times f(n) = 1000(3n^2 + 9)$

Now,

$1000(3n^2 + 9) = O(g(n))$

Yes → we can always take "c" ≥ 3000 to establish that.

$\therefore$ a = scaling factor : will not make any difference for the approximation

$a \times f(n) = O(g(n))$.

Extending that:

If $f(n) = O(g(n))$, then $a \times f(n) = O(g(n))$.

If $f(n) = \Omega(g(n))$, then $a \times f(n) = \Omega(g(n))$. → g can always choose a "c" accordingly

If $f(n) = \Theta(g(n))$, then $a * f(n) = \Theta(g(n))$.

↳ all are true

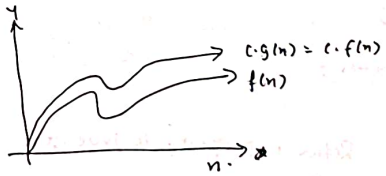
Reflexive property

If $f(n)$ is given, then $f(n) = O(f(n))$

. $f(n)$ can grow asymptotically bigger than $f(n)$.

multiply a constant → $f(n)$.

(c)



ex:- Let $f(n) = 3n$. Is $f(n) = O(f(n))$?

We know from the definition

$f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of $n \geq n_0$, $c > 0$ & c, n are some constants.

$\therefore f(n) \leq c \cdot f(n)$ $\quad c = 2$

$3n \leq 6n$! → This is true for all $n \geq 1$

∴ $\boxed{c=2,\ n_0=1}$.

∴ $f(n) = O(g(n))$

Surprisingly

$f(n) = \Omega(f(n))$.

$f(n) \geq c \cdot f(n)$

$3n \geq 1.5n$ · True for all $n \geq 1$

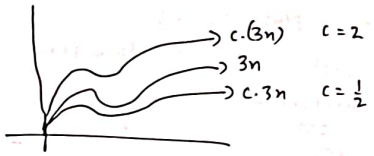
If $f(n) = 3n$ ~~or~~ (let $c=1$) or ($c = \frac{1}{2}$)

or we could choose $c = \frac{1}{2}$

∴ $3n \geq c \cdot 3n$

$3n \geq 3n$ → This is true for all values of n.

- How is $f(n)$ is both = (Lower + upper) bound for $f(n)$.



- If $f(n) = \Theta(f(n))$

Summary

$f(n) = O(f(n))$
$f(n) = \Omega(f(n))$
$f(n) = \Theta(f(n))$

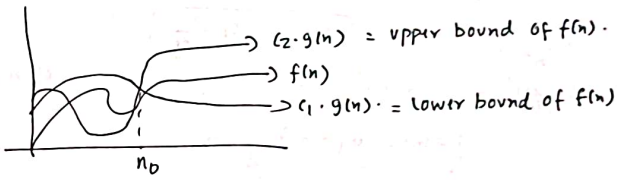
Reflexive property is true for all

Symmetric Property

If $f(n) = \Theta(g(n))$, then $g(n) = \Theta(f(n))$.

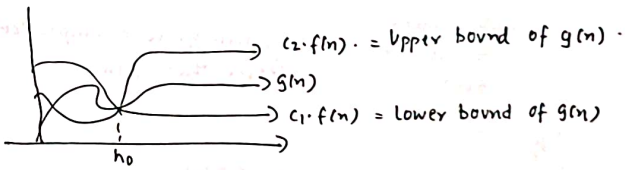
- Applicable to Θ.
- If $f(n)$ is asymptotically equal to $g(n)$ [This is what $\Theta(n)$ means] then $g(n)$ is also asymptotically equal to $f(n)$.

$f(n) = \Theta(g(n))$ - graph.



$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. — By definition.

$g(n) = \Theta(f(n))$



$c_2 f(n) \leq g(n) \leq c_1 \cdot f(n)$

Ex:- Let $f(n) = 2n^3$ and $g(n) = n^3$.

$f(n) = \Theta(g(n))$ iff. $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. for all values of $n \geq n_0$ and c_1, c_2 and n_0 are constants.

$c_1 \cdot n^3 \leq 2n^3 \leq c_2 \cdot n^3$.

$c_1 = 1$ and $c_2 = 3$

$n^3 \leq 2n^3 \leq 3n^3$ → True for all values of $n \geq 1$

Now, let see if we can prove otherwise

$c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$

$c_1 \times 2n^3 \leq n^3 \leq c_2 \times 2n^3$.

$c_2 = 1$ $\quad c_1 = \frac{1}{2}$

$\Rightarrow \frac{n^3}{2} \leq n^3 \leq 2n^3$. → True for all values of $n \geq 1$

$\therefore$ $g(n) = \theta(f(n))$ ⊙

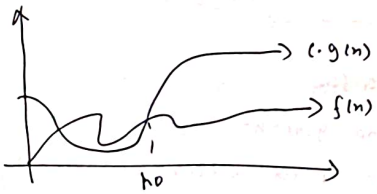
NOTE

* Symmetric property is not applicable to O and Ω

Transpose - Symmetric Property

If $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$.

$g(n)$ is upper bound for $f(n)$ ⇒ $f(n)$ will grow asymptotically lesser than $g(n)$.



Ex:- Let $f(n) = n^2$ and $g(n) = n^3$.

Is $f(n) = O(g(n))$.?

$f(n) \leq c \cdot g(n)$ for all values of $n \geq n_0$ and c, n_0 : constants.

$n^2 \leq c \cdot n^3$.

If $c = 1$. ⇒ $n^3 \geq n^2$. for all values $n \geq 1$

$\therefore$ $n_0 = 1$

Is $g(n) = \Omega(f(n))$

$g(n) \geq c \cdot f(n)$ $\forall$ $n \geq n_0$, c, n_0 = constants.

$n^3 \geq c \cdot n^2$

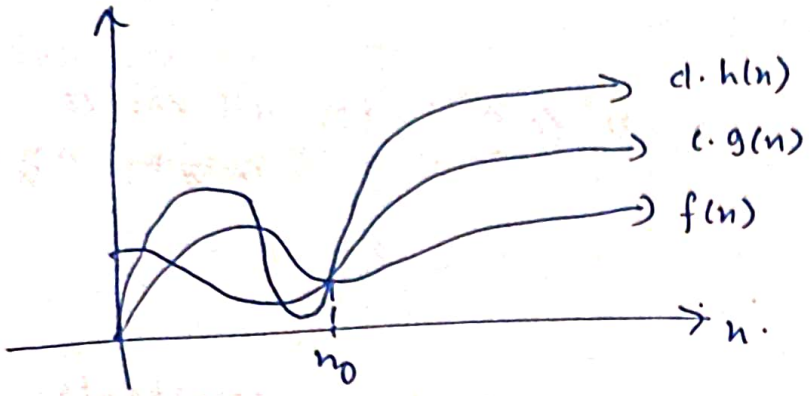
$c = 1$ $\quad n_0 = 0$

$n^3 \geq n^2$ → True for all values of $n \geq 1$.

Transitive property

If $f(n) = O(g(n))$, and $g(n) = O(h(n))$, then

$f(n) = O(h(n))$.



Ex:- $f(n) = n$, $g(n) = n^2$, $h(n) = n^3$.

Is $f(n) = O(h(n))$?

Steps

(1) We will show $f(n) = O(g(n))$

(2) Then we will show $g(n) = O(h(n))$.

(3) Then, we will show $f(n) = O(h(n))$.

(1) $f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$, c and n_0 are constants.

$$n \leq c \cdot n^2$$

Let $c = 1$

$n^2 \geq n$ → This is true for $n \geq 1$

$\therefore n_0 = 1$

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

(2) Now, let us show $g(n) = O(h(n))$

$$n^2 \leq d \cdot n^3$$

Let $d = 1$

$n^3 \geq n^2$ → True for all values of $n \geq 1$

∴ Here $n_0 = 1$.

(3) Let us prove $f(n) = O(h(n))$.

$n \leq c \cdot n^3$.

If $c = 1$

$n^3 \geq n$ → True for all $n \geq 1$.

∴ $(n_0)_{combined} = 1$

and $\boxed{f(n) = O(h(n))}$.

Addition & Multiplication Properties

Addition Property

$$\boxed{f(n) + g(n) = O(\max(f(n), g(n)))}$$

Example

$f(n) = n$ & $g(n) = n^2$

According to the above property

dominating term → $n^2 + n = O(\max(n, n^2)) = O(n^2)$.

* we can always multiply the right side with a constant.

$$\boxed{f(n) + g(n) = \Omega(\max(f(n), g(n)))}$$

$n + n^2 = \Omega(n^2)$.

$f(n) + g(n) \geq c \cdot n^2$

$n + n^2 \geq c \cdot n^2$ → for $c = 1$

↳ This property will hold true.

* We know if O, Ω satisfies, that will satisfy Θ also.

$\therefore$ $f(n) + g(n) = \Theta(\max(f(n), g(n)))$.

Multiplication property

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

Ex Let say $f(n) = n$ $g(n) = \log n$.

now, if $n \log n = O(n \log n)$. it means

$n \log n \leq c \cdot n \cdot \log n$.

For $c = 1$ or higher, the above inequality will hold true.

Note:-

$f(n) \times g(n) = \Omega(f(n) \times g(n))$

$\therefore$ $f(n) \times g(n) = \Theta(f(n) \times g(n))$

Little o Notation

- Assume $f(n)$ and $g(n)$ are non-negative functions.
- $f(n) = o(g(n))$ iff $f(n) < c \cdot g(n)$ ← "strictly less"
  - * → for all values of $c > 0$
    for all values of n where $n \geq n_0$

  where c, n_0 are constants.
- Here $g(n)$ = Strict upper bound of $f(n)$.

Example

Let $f(n) = n$ and $g(n) = n^2$

Can we say $f(n) = o(g(n))$.

For $c > 0$, $f(n) < c \cdot g(n)$.

Let say $c = \frac{1}{8}$.

$n < \frac{1}{8} n^2$

$\Rightarrow 8n < n^2$

$\Rightarrow n > 8 \quad (n > 1)$.

by definition.

→ * So, for all $n > 8$

$f(n) < c \cdot g(n)$.

where $c > 0$.

For

$\therefore n_0 = 9. \quad c = \frac{1}{8}$.

$f(n) = o(g(n))$.

. our intrusion tells us

- n^2 will asymptotically grow more than n. (for all values of $c > 0$).

- we just showed 1 configuration.

$g(n)$ will be strictly greater than $f(n)$ } for all values $n > 9$ when $c = \frac{1}{8}$. } $n > n_0$.

(in general).

Ex 2 $f(n) = n \quad g(n) = n$.. Can we say $f(n) = o(g(n))$?

For $c > 0 \quad f(n) < c \cdot g(n)$

$n < c \cdot n$.

$\Rightarrow$ This is true if $c < 1$ but that is not the only consideration.

$\Rightarrow$ The relation has to hold true for all values of $c > 0$.

$\therefore$ $\boxed{f(n) \neq o(g(n))}$

Relation with Big O Notation

- If $f(n) = o(g(n))$, then $f(n) = O(g(n))$ → True

  little → Big's.

This is understandable.

If $f(n) < c \cdot g(n)$ ⇒ then $f(n) \leq c \cdot g(n)$

By little O definition → Big O

But the opposite may not be true.

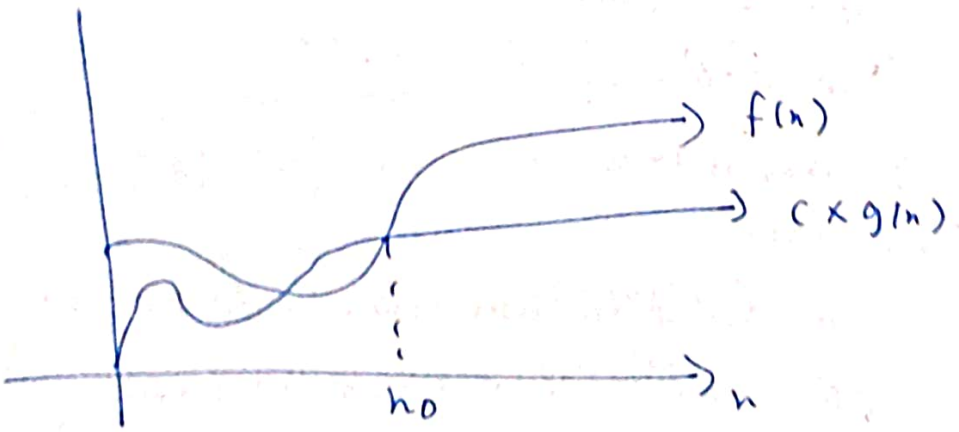
If $f(n) = O(g(n))$, then $f(n) \neq o(g(n))$.

Not True.

Little ω Notation (omega)

- Big Omega Notation → Recap.
- Little Omega (ω) of Little omega (ω) with Big Ω notation.
- Relationship ~~between~~ with Big Ω notation.

The Big Omega (Ω) Notation



$f(n) = \Omega(g(n))$ iff $f(n) \geq c \cdot g(n)$ for all values of $n \geq n_0$ and $c > 0$ where c, n_0 are both constants

Little (ω) notation

$f(n) = \omega(g(n))$ iff $f(n) > c \cdot g(n)$ $\forall$ $n \geq n_0$.

Strictly greater

little

and $c > 0$, c, n_0 are constants for all +ve values

Big O / Ω

- $c > 0$
- for all some restricted values of c

Little o / ω

$c > 0$

- for all +ve values of c

Note

(1) you will need to determine n_0 for which all $c > 0$ will satisfy the inequality

$f(n) > c \cdot g(n)$

Ex 1. Let $f(n) = n^2$, $g(n) = n$. Is $f(n) = \omega(g(n))$?

$f(n) > c \cdot g(n)$.

$n^2 > c \cdot n$.

~~If $n = 2$, $c = 3$~~

Let's take $c = 1$

$n = 2$

$4 > 2 \rightarrow$ True

$\therefore$ $n_0 = 2$ $c = 1$

$f(n) > c \cdot g(n)$ for all values of $n \geq n_0$ (2) and for all $c > 0$

$\therefore$ $f(n) = \omega(g(n))$.

Relation of ω with Ω

If $f(n) = \omega(g(n)) \Rightarrow$ then $f(n) = \Omega(g(n))$

But the reverse is not true.

strict lower bound

lower bound

Ex:- $f(n) = n$ and $g(n) = n$.

For $c = 1$, $f(n) = \Omega(g(n))$

$\Rightarrow f(n) \geq c \cdot g(n)$

$n \geq n$ -> True

But $f(n) \neq \omega(g(n))$ for $c = 1$ which is +ve.

Summary

Notation	Def	meaning	Example
$O(n)$ Big O	$f(n) \leq c \cdot g(n)$	$g(n)$ = tight upper bound of $f(n)$ ~~tight~~	$n^2 = O(n^2)$.
$\Omega(n)$	$f(n) \geq c \cdot g(n)$	$g(n)$ = tight ~~upper~~ lower bound of $f(n)$	$n^2 = \Omega(n^2)$.
$\Theta(n)$	$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$	$g(n)$ = Tight Bound of $f(n)$ Acts on both - lower - upper	$n^2 = \Theta(n^2)$

Notation	Def	Meaning	Example
$o(n)$ Small o	$f(n) < c \cdot g(n)$	$g(n)$ is Strict upper bound	$n^2 = o(n^3)$.
$\omega(n)$ Small omega	$f(n) > c \cdot g(n)$	$g(n)$ is strict lower bound	$n^2 = \omega(n)$.

Solved Problem - 1

(1) Let $f(n) = n^2 \log n$ and $g(n) = n \cdot (\log n)^{10}$ be two positive functions of n. Which of the following statements are correct? <GATE 2001>

(A) $f(n) = O(g(n))$ and $g(n) \neq O(f(n))$.

(B) $g(n) = O(f(n))$ and $f(n) \neq O(g(n))$

(C) $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$.

(D) $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

Let's consider $f(n) = O(g(n))$., and then let see if any of the property is true.

$f(n) \leq c \cdot g(n)$ for some $c > 0$ and for all $n \geq n_0$ where c, n_0 are constants.

$n^2 \log n \leq c \cdot n \cdot (\log n)^{10}$.

Since $n \geq 1$

$\Rightarrow n \log n \leq c \cdot (\log n)^{10}$.

$c \geq \frac{n}{(\log n)^9}$

$n > 1$

$c \geq \frac{n}{(\log n)^{10}}$ will hold true for $n \geq 2$

$\therefore$ Yes $f(n) = O(g(n))$.

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

$n \times (\log n)^{10} \leq c \cdot n^2 \log n$

$\Rightarrow \quad c \geq (\log n)^9$.

yes we can choose such a value for c and $n > 1$

$\therefore \quad f(n) = O(g(n))$ and $g(n) = O(f(n))$.

Option (D)

$f(n)$		$g(n)$
$n^2 \log n$		$n \cdot (\log n)^{10}$
$n \log n$		$(\log n)^{10}$
n	$>$	$(\log n)^9$
(Polynomial)		(Logarithmic)

Decr. functions $<$ Const. functions $<$ Log functions $<$ Poly fns $<$ Exp.

$\therefore$ $g(n) = O(f(n))$.

$\Rightarrow$ $f(n)$ is asymptotically bigger than $g(n)$.

[Option B.]

(2) Consider the following 3 claims.

(i) $(n+K)^m = \Theta(n^m)$. where K and m are constants.

(ii) $2^{n+1} = O(2^n)$

(iii) $2^{2n+1} = O(2^n)$.

Which ~~are~~ of the following statements are correct?

$$(n+K)^m = {}^mC_0 n^m + {}^mC_1 n^{m-1} K + {}^mC_2 n^{m-2} K^2 + \cdots + {}^mC_m K^m.$$

$$\hookrightarrow \underbrace{n^m}_{\text{Polynomial}} + \underbrace{K^m}_{\text{constant}}$$

$${}^mC_0 = \frac{m!}{0!\,(m!)} = 1$$

$${}^mC_m = 1$$

* $f(n) = \Theta(g(n))$.

$c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$.

<GATE 2003>.

(A) (i) and (ii) (B) (i) and (iii) (C) (ii) and (iii) (D) (i), (ii) & (iii).

Options.

(i) $(n+K)^m$

$\Theta(n^m)$.

$n^m + K^m$

Poly > const

dominating term.

Proof.

we know

$n^m = \Theta(n^m)$.

[Refer problem in Pg: 19, 20]

$n^3 + 3n^2 = O(n^3)$.

(ii) $\underbrace{2^{n+1}}_{f(n)} = O(\underbrace{2^n}_{g(n)})$.

$2^{n+1} \leq c \cdot 2^n$.

$\Rightarrow 2 \cdot 2^n \leq c \cdot 2^n$

$\Rightarrow \boxed{c \geq 2}$

$\therefore 2^{n+1} \leq c \cdot 2^n$ for $c \geq 2$

$\therefore$ This ~~claim~~ claim is also correct.

(iii) $2^{2n+1} = O(2^n)$.

Using definition,

$2 \cdot 2^{2n} \leq c \cdot 2^n$

$c \geq 2 \cdot 2^n \Rightarrow c \geq 2^{n+1}$

~~Let n → ∞~~

~~⇒ c ≥ 2~~

~~⇒ c ≥ 4~~

Can we say,

$\underbrace{2^{n+1}}_{\text{Exp. const.}} \leq \underbrace{c}$

~~Poly~~ Exp. const.

~~Yes this claim is also correct~~

No matter what c we choose, 2^{n+1} will surpass c at some point of time.

moreover, 2^{n+1} => Exponential function

c => Constant

$\therefore \boxed{\text{Exp} > \text{const.}}$

$\therefore$ This claim is False.

Option A

(3) Let $f(n)$, $g(n)$, $h(n)$ be functions defined for +ve integers. such that

$f(n) = O(g(n))$
$g(n) \neq O(f(n))$
→ from these two we can say $g(n)$ is asymptotically bigger than $f(n)$

Let $f(n) = n$ & $g(n) = n^2$

$g(n) = O(h(n))$
$h(n) = O(g(n))$
→ from these two we can say $g(n) = h(n)$ asymptotically.

Let $h(n) = n^2$

Which one of the following statements is false ?

(A) $f(n) + g(n) = O(h(n) + h(n))$
(B) $f(n) = O(h(n))$
(C) $h(n) \neq O(f(n))$
(D) $f(n) \cdot h(n) \neq O(g(n) \cdot h(n))$.

Let's take the option (A)

$n + n^2 = O(n^2 + n^2)$ => TRUE

Option (B)

$g(n) \neq O(f(n))$
$n^2 \neq O(n)$ => TRUE

Option (C)

$h(n) \neq O(f(n))$
$n^2 \neq O(n)$ => TRUE

Option (D)

$f(n) \cdot h(n) \neq O(g(n) \cdot h(n))$
$n \cdot n^2 \neq O(n^2 n^2)$ => FALSE.

(4.) Consider the following functions.

$f(n) = 2^n$ $g(n) = n!$ & $h(n) = n^{\log n}$.

Which one is true ?

(A) $f(n) = O(g(n))$; $g(n) = O(h(n))$.

(B) $f(n) = \Omega(g(n))$; $g(n) = O(h(n))$.

(C) $g(n) = O(f(n))$; $h(n) = O(f(n))$.

(D) $h(n) = O(f(n))$; $g(n) = \Omega(f(n))$.

Let's compare the functions $f(n)$, $h(n)$

2^n $n^{\log n}$.

, We can easily compare 2^n, 3^n.

n^2, n^3.

But how do we compare

2^n, $n^{\log n}$.

→ CONFLICTING FUNCTIONS

$\frac{f(n)}{2^n}$ $\frac{h(n)}{n^{\log n}}$.

Let's take log on both

$n \log_2 2$

$= n$

Poly

$\log_2 n \log_2 n = (\log_2 n)^2$

Log.

log < Poly

∴ ~~f(n)~~ $h(n) = O(f(n))$.

Option (C), (D) are closest to investigate.

Remember :-

Def:- A function whose growth rate increases rapidly ex:- 2^n, 3^n, $n!$, n^n, a^n etc => Exponential function.

2^n ↑ fixed base

$n!$ ↓ Super exp

$n = 5 \quad 2^5 = 32$

$n = 5 \quad n! = 5! = 120$.

∴ This mean.

Option (C) is false ∵ $g(n) \neq O(f(n))$.

$f(n) = 2^n$.

Option (D) => TRUE ∵ $g(n) = \Omega(f(n))$.

~~f(n)~~ $f(n) \leq c \cdot g(n)$.

$g(n)$ will grow faster than $f(n)$

(5) Consider the following functions

$f_1 = 10^n$ $f_2 = n^{\log n}$ $f_3 = n^{\sqrt{n}}$.

Which of the following options arranges the functions in the increasing order of asymptotic growth rate?

(A) f_3, f_2, f_1

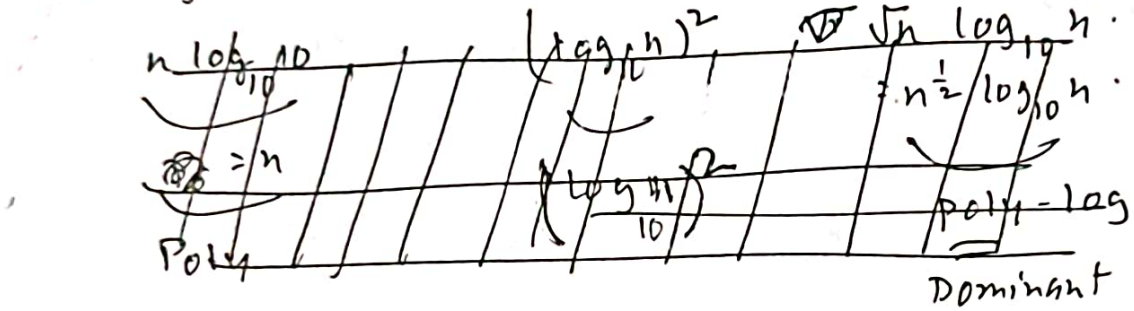
(B) f_3, f_1, f_2

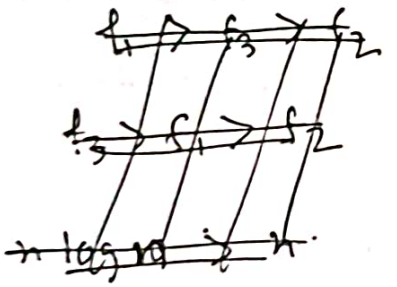
(C) f_1, f_2, f_3

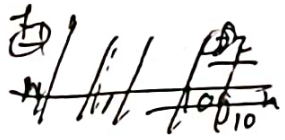
(D) f_2, f_3, f_1 ←

$\frac{f_1}{10^n}$ $\frac{f_2}{n^{\log_{10} n}}$ $\frac{f_3}{n^{\sqrt{n}}}$.

~~Take log for all functions~~







Let's first compare f_2 and f_3.

$\frac{n^{\log_{10} n}}{f_2}$ $\frac{n^{\sqrt{n}}}{f_3}$

Base = n = same for both. f_2 and f_3.

log. Poly

$\therefore$ Poly > log

$\boxed{f_3 > f_2}$ — (1)

Now let us compare f_1 and f_3.

f_1 : 10^n — f_3 : $n^{\sqrt{n}}$ — base = different = conflicting.

Take logarithm

$n \log_{10} 10$ — $\sqrt{n} \log_{10} n$.

$= n$ — $\sqrt{n} \log_{10} n$ → still this is not easy to compare.

$= \underline{\sqrt{n}} \times \underbrace{\sqrt{n}}$ — $\underline{\sqrt{n}}\ \underbrace{\log_{10} n}$.

Poly — log.

Poly > log.

$\therefore \boxed{f_1 > f_3}$. — (2)

Decreasing order

$\therefore$ We can say $\Rightarrow \boxed{f_1 > f_3 > f_2}$. — From (1) & (2)

& Increasing order $\Rightarrow f_2 < f_3 < f_1 \Rightarrow$ Option (D)

Time Complexity of Loops

(LB)

Time complexity

Topics

- Priori vs Posteriori Analysis - Recap.
- CPU computations and Main memory space.
- Understand the Time complexity
- Example

Priori vs Posteriori

Priori	Posteriori
• Estimation of time & memory space required by an algo before executing on the system.	• Calculation of time & memory space required by an algo after executing it on the system.
• Not actual time or space • It is estimation	• It is Actual time & space
• No H/W dependency	• There is H/W dependency.
• Practical	

→ (Estimation of Time) = Total no. of CPU computations that an algorithm needs

→ (Estimation of Space) = Estimation of main memory space.

- What is CPU computations?
  - task performed / instructions executed by the CPU.

- Main memory space
  - Temporarily store data & instructions that CPU needs for quick access during program execution.

Time complexity

Estimations of Total CPU computations required to execute an algorithm.

How? We will measure

Time complexity = Total No. of CPU computations.

* Method = Frequency count method

Time complexity = Sum of Frequency count of each instruction of an algorithm.

What is Frequency count?

Number of times an instruction is executed.

Example

```
Algo sum(a, n)        <- n: no. of elements; a: list of n elements.
{
    sum = 0  ------ (1).
    for i in rag                    (n+1)
    for (i=1; i<=n; i++).
        1 unit sum = sum + a[i]

    return sum
}
```

Let say,
1 instruction = 1 unit of time

Sum of FC of Instructions.
(freq count)

$= \underset{(sum=0)}{1} + \underset{(i=1)}{1} + \underset{(i<=n).}{(n+1)} + \underset{(i++)}{n} + \underset{[sum = sum + a[i]]}{2n}$

2n: 1 instr.
+ 1 (return sum). — 1 instruction.

$= 4n+4$

$i <= n \longrightarrow (n+1)$

Addition - 1 instr.
Assignment - 1 instr.

1 <= n
2 <= n
3 <= n.
⋮
n <= n.
— execute sum (True) → will also execute (i++)

n+1 <= n → will not execute (False)

Execution flow.

(1) i <= n ? condition check

↓

if true

(2) sum = sum + a[i]

↓

(3) (i++).

Note

∴ sum = sum + a[i]

2 instructions

(1) Addition => sum + a[i]

(2) Assignment => sum = x

∴ Time required to solve the algorithm = O(n).

linear time.

~~Abstract~~

Space complexity

Topics

- Priori vs Posteriori - Recap.
- Space complexity
- Example.

Recap

Priori	Posteriori
- Estimation of space done before executing it on the system	- Calculation of time + memory space required by an algorithm after executing it on the system.
- Practical	

Q. How do we estimate the main memory space?

= [① Space required - to store the source code
+
② Space required - for simple variables
+
③ Space required - for data structures used
+
④ Space required - for stack for recursive algorithms.]

Note

① , ② => Constants => Hence can be dropped.

③ -> data structure = f(n).

④ -> stack size = f(n).

How many times the function is called

Example

```
Algo sum(a, n)
{
    sum = 0
    for(i=1; i<=n; i++).
        sum = sum + a[i]
    return sum
}.
```

Simple Variables = n, sum, i = constant

data structure = a

Let us assume, each unit of "a" consumes 1 unit of memory space.

$\sum_{i=1}^{n} a[i] = n.$

a[1] + a[2] + - - n.

1 1

= n x 1 = n units.

Space complexity = O(n).

[P-3]

Analysis of Algorithms

Logarithms & Summations

Topics

- Understanding logarithms
- commonly used logarithms
- commonly used Summation

$\log_b a$ ← argument

base

How many times 'b' will be multiplied with itself to get a

ex:- $\log_2 128$

How many times 2 nuds to be multiplied with itself to get = 128?

Am :- 7 times

Commonly used log

(1) $\log_a a = 1$

ex:- $\log_7 49 = ?$

$\log_7 49 = x$

$7^x = 49 = 7^2$

$\boxed{x = 2}$

ex:- $\log_{\frac{1}{4}} 256 = ?$

let $\log_{\frac{1}{4}} 256 = x$

$\left(\frac{1}{4}\right)^x = 256$

$\Rightarrow 4^x = \frac{1}{256} = \left(\frac{1}{2}\right)^8$

$\Rightarrow 2^{2x} = \left(\frac{1}{2}\right)^8 = 2^{-8}$

$\Rightarrow 2x = -8$ $\quad \boxed{x = -4}$

ex:- $5^{\log_5 125} = ?$ let say it is $= x$

~~Take $\log_5$ on both sides~~

Let $\log_5 125 = x$

$5^x = 125 = 5^3$

$x = 3$

$\therefore \boxed{5^3 = 125}$

Observe

$5^{\log_5 (125)} = 125$

$a^{\log_a (b)} = b$

This is a property.

ex:- $\underbrace{\log_{0.01} 1000}_{e} + \underbrace{\log_{0.1} 0.0001}_{e_2} = ?$

$e_1 = \log_{\frac{1}{100}} 1000$

$\Rightarrow \left(\frac{1}{100}\right)^{e_1} = 10^3$

$\Rightarrow 100^{e_1} = \left(\frac{1}{10}\right)^3$

$\Rightarrow (10)^{2e_1} = 10^{-3}$

$\Rightarrow 2e_1 = -3$

$\Rightarrow e_1 = -\frac{3}{2}$

$e_2 = \log_{\frac{1}{10}} \frac{1}{10000}$

$\left(\frac{1}{10}\right)^{e_2} = \left(\frac{1}{10}\right)^4$

$10^{e_2} = 10^4$

$e_2 = 4$

$\therefore e_1 + e_2$

$= -\frac{3}{2} + 4 = \frac{8-3}{2} = \frac{5}{2}$

ex:- $\log \sqrt{5\sqrt{5\sqrt{5}}}$

Remember surds

$\sqrt[5]{5\sqrt{5\sqrt{5}}}$

$y^{\frac{2^n - 1}{2^n}}$

$y = 5 \quad n = \text{no. of roots} = 3$

$5^{\frac{2^3 - 1}{2^3}} = 5^{\frac{7}{8}}$

$\log_5 5^{\frac{7}{8}} = \frac{7}{8}$ Ans.

ex:- If $\log_{\sqrt{8}} b = \frac{10}{3}$ what is $b = ?$

$$b = (\sqrt{8})^{\frac{10}{3}} = (8^{\frac{1}{2}})^{\frac{10}{3}} = 8^{\frac{5}{3}} = 2^5 = 32$$

ex:- If x, y, z be positive real numbers such that

$$\log_{2x}(z) = 3 \quad -(1)$$

$$\log_{5y}(z) = 6 \quad -(2)$$

$$\log_{xy}(z) = \frac{2}{3} \quad -(3)$$

What is $z = ?$

$(2x)^3 = z$ From (1)

$(5y)^6 = z$ From (2)

$\therefore$ ~~$(2x)^3 = (5y)^6$~~

~~$8x^3 =$~~

$$x = \frac{z^{\frac{1}{3}}}{2} \qquad y = \frac{z^{\frac{1}{6}}}{5}$$

$$(xy)^{2/3} = z$$

$$z = (xy)^{\frac{2}{3}} = \left[\frac{z^{\frac{1}{3}}}{2} \times \frac{z^{\frac{1}{6}}}{5}\right]^{2/3}$$

$$= \frac{(z^{\frac{1}{3}+\frac{1}{6}})^{\frac{2}{3}}}{(10)^{2/3}}$$

$z \neq 0$ ~~$\frac{1}{3}$~~ 1

$$\frac{1}{3} + \frac{1}{6} = \frac{2+1}{6} = \frac{3}{6} = \frac{1}{2}.$$

$$= \frac{z^{\frac{1}{2}\times\frac{2}{3}}}{(10)^{\frac{2}{3}}} = \frac{z^{\frac{1}{3}}}{10^{\frac{2}{3}}}$$

$$\Rightarrow z^{1-\frac{1}{3}} = \frac{1}{10^{2/3}}$$

$$\Rightarrow z^{\frac{2}{3}} = \text{~~$10^{\frac{2}{3}}$~~} = \left(\frac{1}{10}\right)^{\frac{2}{3}}$$

$\therefore \boxed{z = \frac{1}{10}}$

Note $\log_b a$

$b > 0 \quad b \neq 1, \quad a > 0$

$\rightarrow \log_b a < 0$ if $0 < a < 1$

Properties of log – (1)

$\rightarrow$ (1) $\log_a m + \log_a n = \log_a mn$ $\quad a>0 \quad m>0 \quad n>0.$ $\quad a \neq 1$

- Why log (-ve) is not possible.

$\log_b a = x$

$b^x = a$

if $b > 0$ and $b \neq 1$

- There is no way, I can get a -ve value for (b^x).
- $\therefore$ a cannot be -ve.

$\rightarrow$ (2) $\log_a m - \log_a n = \log_a \frac{m}{n}$

$\rightarrow$ (3) $\log_a m^n = n \log_a m.$

---

eg:- Find the value of $(\log_{10}2)^3 + \log_{10}8 \cdot \log_{10}5 + (\log_{10}5)^3$

Let $a = \log_{10}2$, $b = \log_{10}5$

$a^3+b^3 = (a+b)^3 - 3ab(a+b)$.

$(a+b)^3 = a^3 + b^3 + 3a^2b + 3ab^2$.

$\therefore a^3+b^3 = \Rightarrow (\log_{10}2)^3 + (\log_{10}5)^3$

$= (\log_{10}2 + \log_{10}5)^3 - 3\log_{10}2 \cdot \log_{10}5 (\log_{10}2 + \log_{10}5)$

$= 1 - 3\log_{10}2 \cdot \log_{10}5$

$= 1 - \log_{10}2^3 \cdot \log_{10}5$

$\therefore$ The original term is $(\log_{10}2)^3 + \log_{10}8 \cdot \log_{10}5 + (\log_{10}5)^3$

$\therefore \Rightarrow 1 - \log_{10}8 \cdot \log_{10}5 + \log_{10}8 \cdot \log_{10}5$

$= 1$

---

ex-> If $3^x = 4^{x-1}$ what is $x = ?$

$\Rightarrow$ We are dealing with different bases here.

$\Rightarrow \log_{10}3^x = \log_{10}4^{x-1}$

$\Rightarrow x \log_{10}3 = (x-1)\log_{10}4$

$\Rightarrow x \log_2 3 = x \log_{10}4 - \log_{10}4$

$\Rightarrow x(\log_{10}4 - \log_{10}3) = \log_{10}4$

$\log_2 4 = 2$

$\therefore \boxed{x = \frac{2}{2 - \log_2 3}}$

other properties of log -(2)

→ (4) $\log_a b = \frac{\log_c b}{\log_c a}$ $\quad c>0,\ c\neq 1$ $\quad a>0$ $\quad b>0$ $\quad a\neq 1$

$\Rightarrow \boxed{\log_a b \times \log_c a = \log_c b}$

Now, observe

$\boxed{\log_a b = \frac{1}{\log_b a}}$ Why?

$\log_a b = \frac{\log_c b}{\log_c a}$ ⋘ ← Here Let's take $b = c$

$\Rightarrow \log_a b = \frac{1}{\log_b a}.$

→ (5) (a) $^{\log_a N} = N$

→ (6) (a) $^{\log_c b} = (b)^{\log_c a}.$ ← Interchangeable

(6a.)

$\boxed{\log_{a^n} b = \frac{1}{n} \log_a b}$

Q. Simplify $(81)^{\frac{1}{\log_5 3}}$.

$\Rightarrow (3^4)^{\log_3 5}$

$\Rightarrow 3^{4\log_3 5} \Rightarrow 3^{\log_3 5^4} = 5^4 = 625.$

Q. P.T $x^{\log(\frac{y}{z})} \times y^{\log(\frac{z}{x})} \times z^{\log(\frac{x}{y})} = 1$

L.H.S

$x^{(\log y - \log z)} \times y^{(\log z - \log x)} \times z^{(\log x - \log y)}$

$\Rightarrow \frac{x^{\log y}}{x^{\log z}} \times \frac{y^{\log z}}{y^{\log x}} \times \frac{z^{\log x}}{z^{\log y}} = 1$

Note:-

$x^{\log y} = y^{\log x}$

$x^{\log z} = z^{\log x}$

$y^{\log z} = z^{\log y}$

} interchangeable property.

Graph of log

$y = \log_2 x$

$x = \frac{1}{2}$

$\log_2 \frac{1}{2} = \underbrace{\log_2 1}_{0} - \underbrace{\log_2 2}_{1} = -1$

$x = \frac{1}{4}$

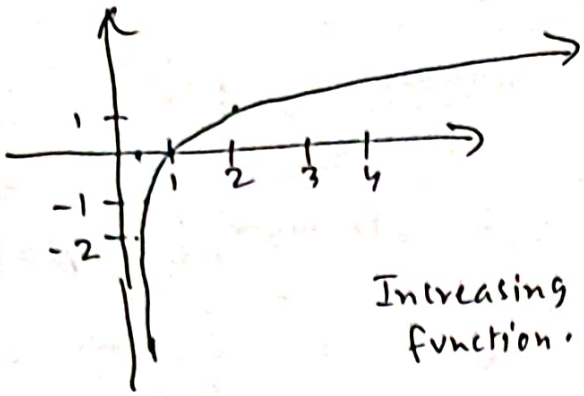
$\log_2 \frac{1}{4} = \log_2 1 - \log_2 4 = -2$

$x = 2$

$\log_2 x = 1$

$x = 4$

$\log_2 4 = 2$



$\log_2 x$

Increasing function.

Note: $\log_a x$

$a > 1$

increasing function

$y = \log_{\frac{1}{2}} x$

(A) $x = \frac{1}{2}$ $\quad y = 1$

(B) $x = \frac{1}{4}$ $\quad y = \log_{\frac{1}{2}} \frac{1}{4} = 2.$

$\Rightarrow \log_{\frac{1}{2}} 1 - \log_{\frac{1}{2}} 4$

$= 0 - \log_{\frac{1}{2}} 4$

$= 0 - (-2) = +2$

Let $\log_{\frac{1}{2}} 4 = x$

$\left(\frac{1}{2}\right)^x = 2^2$

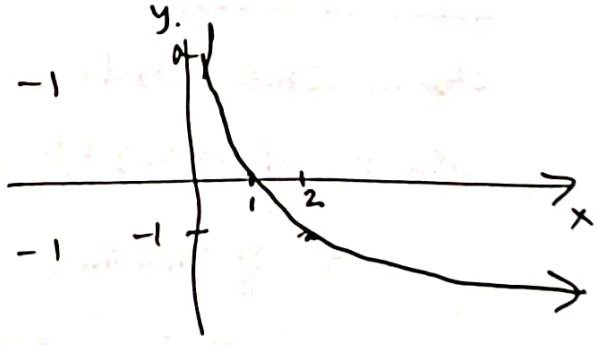
$2^x = \left(\frac{1}{2}\right)^2 = 2^{-2}$

$x = -2$

(C) $x = 2 \quad y = -1$

$y = \log_{\frac{1}{2}} 2 = \log_{\frac{1}{2}} \left(\frac{1}{2}\right)^{-1} = -1$

or $\frac{\log_2 2}{\log_2 \frac{1}{2}} = \frac{1}{0-1} = -1$



Decreasing

$\log_a x$

if $0 < a < 1 \Rightarrow$ Decreasing.

Commonly used Summations

1. $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \Theta(n^2)$

$\hookrightarrow \Theta\left(\frac{n^2}{2} + \frac{n}{2}\right)$

$\hookrightarrow \frac{1}{2}\Theta(n^2+1)$

$= \Theta(n^2)$

AP : $T_n = a + (n-1)d \Rightarrow n^{th}$ term.

$S_n = 1 + 2 + 3 + \cdots + n.$

$S_n = n + (n-1) + (n-2) + \cdots + 1$

Writing in rev. order

$+$

$2S_n = (n+1) + (n+1) + (n+1) + \cdots (n+1)$

$2S_n = n(n+1)$

$$\boxed{S_n = \frac{n(n+1)}{2}}$$

2. $1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$

General Sum formula for AP:

$S_n = a \quad a+d \quad a+2d \quad \cdots \quad a+(n-1)d$ (last term a_n)

$S_n = a_n \quad a_n - d \quad a_n - 2d \quad \cdots \quad a$

$2S_n = (a + a_n) \times n$

$S_n = \frac{n}{2}(a + a_n)$

Now, what is a_n

$a_n = a + (n-1)d$

$= \frac{n}{2}[a + a + (n-1)d] = \frac{n}{2}[2a + (n-1)d]$

For GP

$T_n = ar^{n-1}$

$S_n = a + ar + ar^2 + \cdots + ar^{n-1}$

$= a\left[1 + r + r^2 + \cdots + r^{n-1}\right]$

$S_n = a\dfrac{(r^n - 1)}{(r-1)}$ if $r > 1$

$= a\dfrac{(1-r^n)}{(1-r)}$ if $r < 1$

Proof.

$S_n = a + ar + ar^2 + \cdots + ar^{n-1}$

$rS_n = ar + ar^2 + ar^3 + \cdots + ar^n$

$S_n(1-r) = a - ar^n = a(1-r^n)$

$S_n = \dfrac{a(1-r^n)}{1-r}$ or $\dfrac{a(r^n-1)}{r-1}$

*

Proof of $1^2 + 2^2 + \cdots + n^2 = \dfrac{n(n+1)(2n+1)}{6}$.

$a^3 - b^3 = (a-b)(a^2 + ab + b^2)$.

(1) $n^3 - (n-1)^3 = [n - n + 1][n^2 + n(n-1) + (n-1)^2]$

$= [n^2 + n^2 - n + n^2 - 2n + 1]$

$= [3n^2 - 3n + 1]$

(2) $(n-1)^3 - (n-2)^3 = 3(n-1)^2 - 3(n-1) + 1$

(3) $(n-2)^3 - (n-3)^3 = 3(n-2)^2 - 3(n-2) + 1$

$2^3 - 1^3 = 3(2)^2 - 3(2) + 1$

(n) $1^3 - 0^3 = 3(1)^2 - 3(1) + 1$

Adding (1) + (2) + - - - + (n).

~~$\Rightarrow n^3 = 3[1^2 + 2^2 + \cdots (n-1)^2] - 3[1+2+\cdots(n-1)] + n$.~~

~~$\Rightarrow n^3 + n^2 = 3[\quad]^2$~~

~~$\Rightarrow n^3 + 3n^2 = 3[1^2 + 2^2 + \cdots + n^2] - 3[1+2+\cdots(n-1)] + n$~~

~~$\Rightarrow n^3 + 3n^2 - 3n = 3X - 3\frac{n(n+1)}{2} + n$.~~

~~$\Rightarrow n^3 + 3n^2 - 4n + 3\frac{n(n+1)}{2} = 3X$~~

~~$\Rightarrow 2n^3 + 6n^2 - 8n + 3n^2 + 3n = 6X$~~

~~$\Rightarrow 8n^3 + 9n^2 - 5n = 6X$~~

$$n^3 = 3\sum_{n=1}^{n} n^2 - 3\sum_{n=1}^{n} n + n.$$

$$\Rightarrow n^3 = 3\sum_{n=1}^{n} n^2 - 3\frac{n(n+1)}{2} + n.$$

$$\Rightarrow \sum_{n=1}^{n} n^2 = \frac{1}{3}\left[n^3 + \frac{3n(n+1)}{2} - n\right]$$

$$= \frac{1}{3} n\left[n^2 + \frac{3(n+1)}{2} - 1\right]$$

$$= \frac{n}{3}\left[\frac{2n^2 + 3n + 3 - 2}{2}\right]$$

$$= \frac{n}{6}\left[2n^2 + 3n + 1\right]$$

$2n^2 + 2n + n + 1$

$\Rightarrow 2n(n+1) + 1(n+1)$

$\Rightarrow (n+1)(2n+1)$

$$\boxed{\sum_{n=1}^{n} n^2 = \frac{n}{6}(n+1)(2n+1)}$$

Proof.

Proof of

$1^3+2^3+\cdots+n^3=\left(\frac{n(n+1)}{2}\right)^2$ for all $n\in N$. *

~~By induction~~

Steps

(1) This statement is true for $n=1$

(2) Assume given statement is true for $n=r$.

(3) Prove the given statement is true for $n=r+1$

Soln

$$P(n)=1^3+2^3+\cdots+n^3=\left[\frac{n(n+1)}{2}\right]^2 \quad -(1).$$

$P(1)=1^3=1=LHS$

$RHS=\left[\frac{1(1+1)}{2}\right]^2=1^2=1$

$\therefore$ LHS = RHS

$\therefore$ P(1) = ~~works~~ true.

$\therefore$ Assume given statement is true for $n=r$.

$$P(r) \Rightarrow 1^3+2^3+\cdots+r^3=\left(\frac{r(r+1)}{2}\right)^2.$$

Now,

$$P(r+1) \Rightarrow \underline{1^3+2^3+\cdots+r^3}+(r+1)^3.$$

$$\Rightarrow \left(\frac{r(r+1)}{2}\right)^2+(r+1)^3.$$

$$\Rightarrow (r+1)^2\left[\frac{r^2}{4}+r+1\right]$$

$$\Rightarrow (r+1)^2\left[\frac{r^2+4r+4}{4}\right] \Rightarrow (r+1)^2\ \frac{(r+2)^2}{4}.$$

$$P(r+1) = \left(\frac{(r+1)(r+2)}{2} \right)^2 .$$

∴ The above relation is true.

$$\therefore \quad 1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2} \right)^2$$

Note:-

(a) $1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$

As n^3 is the dominating term.

(b) $1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4} = \Theta(n^4)$.

---

4. ~~Proof~~ -

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \log_e n \cdot = \Theta(\log n)$$

Harmonic series $= \sum_{k=1}^{n} \frac{1}{k}$

(Hn)

* The proof can be done after learning integral calculus.

Question :- Why are we writing Θ here instead of Big O?

5. $${}^nC_0 + {}^nC_1 + {}^nC_2 + \cdots + {}^nC_n = 2^n .$$

~~RHS~~ RHS $= 2^n = (1+1)^n$

$$(q+p)^n = {}^nC_0 q^n + {}^nC_1 q^{n-1} p + {}^nC_2 q^{n-2} p^2 + \cdots + {}^nC_n p^n .$$

Here $q = 1 \quad p = 1$

$$2^n = {}^nC_0 + {}^nC_1 + {}^nC_2 + \cdots + {}^nC_n$$

$$= \Theta(2^n)$$

Summary

1. $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \Theta(n^2)$.
2. $1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$.
3. $1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2 = \Theta(n^4)$.
4. $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \log_e n = \Theta(\log n)$.

$$H_k = \sum_{k=1}^{n} 1 \Theta \frac{1}{k}$$

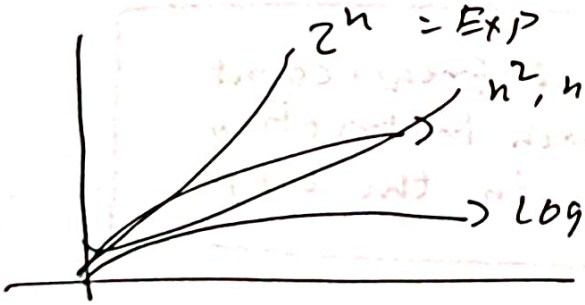
5. ${}^nC_0 + {}^nC_1 + {}^nC_2 + \cdots + {}^nC_n = 2^n = \Theta(2^n)$.

---

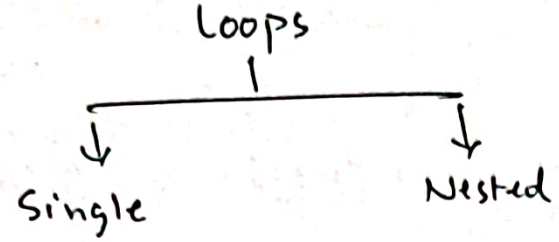
n^2, n^3, n^4 => Polynomial functions.

2^n => Exponential function

$\log n$ => Logarithmic.



Time Complexity - Single loops



Time complexity (TC) = Estimation of total CPU computation required to execute an algorithm.

$$TC = \sum (\text{freq count of each instruction})$$

Number of times an instruction is executed.

$$TC = (f_1)_{I_1} + (f_2)_{I_2} + \cdots + (f_n)_{I_n} = \sum_{i=1}^{n} (f_i)_{I_i}$$

f_i = freq count of instruction I_i

$$\boxed{TC = (f_1)_{I_1} + (f_2)_{I_2} + \cdots + (f_n)_{I_n} = \sum_{i=1}^{n} (f_i)_{I_i}}$$

f_i = freq count of instruction I_i

Time complexity of loop = Sum of freq. count of each instruction in the loop

- But what if I tell you this is not required.
- We just calculate the freq. count of the inner most instruction of the loop.

Example:-

```
for(i=1 ; i<=n ; i++)        (n+1) times, n times
    print("Welcome");        n times
```

1 → i=1

print("Welcome"); ← ["This is the inner most instruction of the loop."]

"we just need to find out the freq. count".

∴ TC = O(n).

↳ This can be derived directly from the innermost loop statement / instruction

- we just need 1 aspect

```
for(      )   → Aspect 1
print( )      → Aspect 2
```

> Any one is fine to proceed with.

* By convention, we take the print( ) statement.

Time complexity of single loops = Increment by constant.

1.
```
for(i=1; i<=n; i++)
    print("welcome")
```
← n times = O(n).

we saw that

2.
```
for(i=1; i<=n; i=i+2)
    print("welcome")
```
← ?

Let's consider each iteration

		iter
$i = 1$	$= (1 + 0 \times 2)$	1
$i = 3$	$= (1 + 1 \times 2)$ ← iter seq.	2
$i = 5$	$= (1 + 2 \times 2)$	3
$i = 7$	$\Rightarrow (1 + 2 \times 3)$	4
⋮	⋮	⋮
$i = n$	$\Rightarrow 1 + K \times 2$	$(K+1)$ iteration.

$1 + K \times 2 = n$

$2K = n - 1$

$$K = \frac{n-1}{2}$$

Use this format

$$K + 1 = \left(\frac{n-1}{2} + 1\right) \Rightarrow \frac{n-1+2}{2} = \frac{n+1}{2}$$

But this is still $O(n)$.

(3) for ($i = 1$; $i <= n$; $i = i + 10$)

print ("welcome").

$i = 1$	$\Rightarrow 1 + 0 \times 10$	
$i = 11$	$\Rightarrow 1 + 1 \times 10$	
$i = 21$	$\Rightarrow 1 + 2 \times 10$ ← iteration seq.	
⋮	⋮	
$i = n$	$\Rightarrow 1 + K \times 10$.	→ iteration $= (K+1)$.

$1 + K \times 10 = n$

$10K = n - 1$

$$K = \frac{n-1}{10}$$

iteration $= K+1 = \left(\frac{n-1}{10} + 1\right) = O(n)$.

The time complexity remains same. $= O(n)$.

Time complexity - Decrement variable by a constant

1. for (i = n; i >= 1; i--)
   print ("welcome").

How many times this will run?

n times = $O(n)$.

2. for (i = n; i >= 1; i = i - 2)
   print ("welcome").

$i = n$	$= n - 0 \times 2$	
$i = n-2$	$= n - 1 \times 2$	
$i = n-4$	$= n - 2 \times 2$	
⋮	⋮	
$i = 1$	$= n - K \times 2$	-- (K+1) iterations.

$n - K \times 2 = 1$

$2K = n - 1$

$K = \frac{n-1}{2}$

$K+1 = \frac{n-1}{2} + 1$. = Freq. count of the instruction.

T.C = $O(n)$.

Note

- The $i = i - 2$ is working in the same way like $i = i + 2$

3. for (i=n; ~~i>1~~ i>=1; i=i-10)
print ("welcome").

F.C = $\frac{n-1}{10} + 1$

T.C = O(n).

---

* Freq. count of loop = ~~n-1/k+1~~

Pg-71 for general formula

~~K = increment / decrement counter.~~

~~for (i=1; i<=n; i++).~~
~~print ("welcome").~~

General formula (This is when $i = i \pm k$).

$$= \left[\frac{(i_{last} - i_{first} + 1) - 1}{k} \pm 1 \right]$$

k → increment or decrement counter

GIF

General formula

$$\left[\frac{(i_{last} - i_{first} + 1) - 1}{k} \right]$$

Time complexity - Single loop

- multiply the variable by a constant

```
for (i=1; i<=n; i=i*5)
    print("Welcome");
```

	cond	itr	Result	Representation
i=1	1<=n	1	True	5^0
i=5	5<=n	2	n	5^1
i=25	25<=n	3	n	5^2
⋮				
i=n	n<=n	(K+1)	n	5^K

$\therefore 5^K = n$

$K = \log_5 n$

Total iteration $= (K+1) = \log_5 n + 1$

$\therefore$ **$FC = \log_5 n + 1$** $\Rightarrow \Theta(\log n)$

In general way,

$\log_\lambda n + 1$ (X)

λ = multiplying factor

NO

→ No this is not the general formula (P-70)

* The best, worst case performance = $\log n$
  - no early exits
  - no late exits

$\therefore \Theta(\log n)$ is used instead of ~~$\Theta(n)$~~ $O(\log n)$ or $\Omega(\log n)$

• You will often see people using $O(n)$ notation as they are concerned with worst case performance most of the time.

H.W

Determine the time complexity of the following loops.

1. for (i=1; i<=n; i = i*10).
   print("--")

~~$\Theta(\log 10)$~~. $\Theta(\log_{10} n)$. = TC.

FC = $\log_{10} n + 1$.

2. for (i=1; i<=n; i = i*3)
   print("--").

TC = $\Theta(\log_3 n)$

FC = $\log_3 n + 1$.

GENERAL FORMULA

for (i=A; i<=B; i = i*C)
print ("--").

		itr	o/p
i = A	A <= B	1	True.
i = A*C	AC <= B.	2	"
i = A*C^2	AC^2 <= B	3	"
i = A*C^3	AC^3 <= B	4	"
⋮			
i = A*C^K	AC^K <= B	(K+1)	

$AC^K = B$

$K = \log_C\left(\frac{B}{A}\right)$

$\therefore$ **FL = $\log_C\left(\frac{B}{A}\right) + 1$**

general formula.

General Formula - for Incr / Decr

```
for (i = A; i <= B; i = i+C)
    print("...");
```

		Itr	Result	Repr.
i = A	A <= B	1	True	A + 0·C
i = A+C	A+C <= B	2	True	A + 1·C
i = A+2C	A+2C <= B	3	True	A + 2C
⋮			⋮	
i = A+KC	A+KC <= B	(K+1)	True	

$A + KC = B$

$K = \frac{B-A}{C}$

$= \left(\frac{\text{Final value} - \text{Initial value}}{\text{Step size}} + 1 \right)$

$FC = \text{Freq. count} = K+1 = \left[\frac{B-A}{C} + 1 \right]$

⊗ greatest int. function.

If $A > B$ → happens for Decr.

```
for (i = A; i >= B; i = i-C)
    print
```

~~FC = Freq. count~~

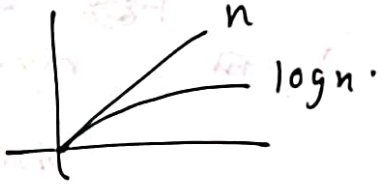
$FC = \left[\frac{A-B}{C} + 1 \right]$

Time complexity - Single loop
(Divide the update Expression)

Let us do the general calculation.

```
for (i = A; i >= B; i = i/c).
    print ("--").
```

		Iteration	Result
$i = A$	$A >= B$	1	True
$i = \frac{A}{c}$	$\frac{A}{c} >= B$	2	"
$i = \frac{A}{c^2}$	$\frac{A}{c^2} >= B$	3	"
$i = \frac{A}{c^3}$	$\frac{A}{c^3} >= B$	4	"
⋮			
$i = \frac{A}{c^k}$	$\frac{A}{c^k} = B$	(k+1)	"



$$\frac{A}{c^k} = B$$

$$\Rightarrow c^k = \frac{A}{B}$$

$$\Rightarrow k = \log_c \left(\frac{A}{B}\right)$$

$$FC = 1 + k = 1 + k = 1 + \log_c \left(\frac{A}{B}\right)$$

$$= \log_{\text{Step size}} \left(\frac{\text{initial val}}{\text{Final val}}\right) + 1$$

$$T.C = \Theta(\log n)$$

```
for (i=n; i>=1; i= i/5).
    print("--")
```

How many times the loop will run?

$$FC = \log_c\left(\frac{\text{initial}}{\text{final}}\right) + 1$$

$$= \log_5\left(\frac{n}{1}\right) + 1$$

$$\boxed{FC = \log_5 n + 1}$$

* Don't write log(base) in TC.

$$TC = \Theta(\log_5 n)$$

Best = worst case = Avg case

H.W

Determine the FC, TC for the following loops.

(1)
```
for (i=n; i>=1; i= i/10).
    print("--").
```

FC

multiplication $= \log_c\left(\frac{\text{Final}}{\text{Initial}}\right) + 1$

Division $= \log_c\left(\frac{\text{Initial}}{\text{Final}}\right) + 1$

Initial = n Final = 1 c = 10.

$$FC = \log_c\left(\frac{\text{Initial}}{\text{Final}}\right) + 1$$

$$\boxed{FC = \log_{10} n + 1} \qquad \boxed{TC = \Theta(\log_{10} n).}$$

(2)
```
for (i=n; i>=1; i= i/3)
    print("--")
```

$$\boxed{FC = \log_3 n + 1} \qquad \boxed{TC = \Theta(\log_3 n)}$$

Time complexity for single loop - powers

. for (i = A; i <= B; i = i^c)
print("--").

Here c > 1.

i =	A <= B	iter
i = A	A <= B	1
i = A^c	A^c <= B	2
i = (A^c)^c / A^c^2	A^c^2 <= B	3
i = ((A^c)^c)^c / A^c^3	A^c^3 <= B	4
⋮	⋮	⋮
i = A^c^K	A^c^K = B	(K+1)

$A^{c^K} = B$

$\Rightarrow c^K = \log_A B$.

$\Rightarrow K = \log_c \log_A B$.

$\therefore$ FC = K + 1 =

$= \log_c \log_A B + 1$

$\Rightarrow \log_{power} \log_{initial}$ Final + 1

TC = $\Theta(\log \log n)$.

ex:- for (i = 3; i <= n; i = i^2)
print

Using the above formula.

FC = $\log_2 \log_3 n + 1$

$\log_c \left(\frac{\log B}{\log A} \right) + 1$

another form-

Use this.

ex:- for (i = n; i >= 5; i = $\sqrt{i}$).
print("--").

Let us apply the general formula

for (i = A ; i >= B ; i = i / c)
    print (4 .. n)

Here A > B
c > 1.

		iter
$i = A$	$A >= B$	1
$i = A^{\frac{1}{c}}$	$A^{\frac{1}{c}} >= B$	2
$i = (A^{\frac{1}{c}})^{\frac{1}{c}}$	$A^{\frac{1}{c^2}} >= B$	3
$i = A^{\frac{1}{c^3}}$	$A^{\frac{1}{c^3}} >= B$	4
!	!	!
	$A^{\frac{1}{c^k}} = B$	(k+1)

$\log_c \left(\frac{x}{y}\right) = \log_{\frac{1}{c}} \left(\frac{y}{x}\right)$ This is true.

$A^{\frac{1}{c^k}} = B$

use this → $1 + \log_{\frac{1}{c}} \left(\frac{\log B}{\log A}\right)$. Another form.

$\frac{1}{c^k} = \log_A B$. => $c^k = \log_B A$

A = Initial
B = Final.

$k = \log_c \log_B A$.

$1 + k = 1 + \log_c \log_B A = 1 + \log_c \left(\frac{\log A}{\log B}\right)$

Initial. Final.

general way => $1 + \log_{power}$

just the "c^n" not $(\frac{1}{c})$ as used

$TC = \Theta(\log \log n)$.

ex:- for (i = n ; i >= 5 ; i = $\sqrt{i}$)
for (i = n ; i >= 5 ; i = $i^{\frac{1}{2}}$)

$FC = 1 + \log_2 \log_5 n$ $\quad n > 5$

$TC = \Theta(\log \log n)$

HW

a Determine the FC and TC of the following loops

1. for (i = 1; i <= n; i = i^2.5)

2. for (i = n; i >= 1; i = ∛i).

b ~~1. FC = 1 + log_power log n~~

(1) What is the general formula for the following

for (i = A; i <= B; i = i^c). c > 1

$$K+1 = \log_c\left(\frac{\log B}{\log A}\right) + 1$$

Here B > A.

Now if A = 1 => no. of iteration = ∞

for (i = 1; i <= 10; i = i^c).

$i_0 = 1$

$i_1 = 1$

$i_2 = 1$

All iterations will have i = 1.

(2) for (i = n; i >= 1; i = i^{1/3}).

~~K+1 = 1 + log_c (log A / log B)~~

Here c = 3.

$= 1 + \log_3 \frac{\log n}{\log 1}$ → Again this will enter into an infinite loop.

The Frequency count

$$= 1 + \log_{\frac{1}{2}} \left(\frac{\log B}{\log A} \right).$$

$$= 1 + \log_{\frac{1}{3}} \left(\frac{\log 1}{\log n} \right)$$

This term is undefined.

If $n = 10$, And we are doing square root each time.

$10, 10^{\frac{1}{2}}, 10^{\frac{1}{4}}$ - - - - But this will never reach 1.

∴ And that's why this is an infinite loop.

Time Complexity → ~~with a function~~
- Conditional statement with a function

function = Exp / Poly / log - - anything.

1. for (i = 1; $i <= 2^n$; i++) ← conditional statement
   print ("welcome").

Step (1) Determine the FC of the innermost statement of the loop.

(P.T.O)

		iter	Result
$i = 1$	$1 <= 2^n$	1	True
$i = 2$	$2 <= 2^n$	2	"
$i = 3$	$3 <= 2^n$	3.	"
⋮			
$i = k$	$k = 2^n$	(k) --->	which is nothing but 2^n

~~$n = \log_2 k$~~ $\quad FC = k$.

$TC = \theta(2^n)$.

2. for (i = 2; i <= 2^n; i = i*2).

		iter	Result	Rtyp.
$i = 2$	(2) $<= 2^n$	1		
$i = 2^2$	$2 \cdot 2$ (2) $<= 2^n$	2		
$i = 2^4$	$2 \cdot 2^3$ $<= 2^n$	3		
$i = 2^8$	$2 \cdot 2^4$ $<= 2^n$	4		

$GP = ar^{n-1}$ → nth term.

$a = 2 \quad r = 2$

$a = 2$

o iter / 2 → 2nd term of GP $= 2 \times 2$

3 → 3rd term $= 2 \times 2^{3-1} = 2 \times 2^2$

Let's observe the pattern here

$i = 2$	$2 <= 2^n$	itr Result	Representation:
		1	True
$i = 2^2$	$2^2 <= 2^n$	2	n
$i = (2^2)^2$	$2^2 <= 2^n$	3	
~~$i = (2^2)^3$~~			

$(2^4)^2 = 2^8 = 2^{2^3}$ (can be written)

$i = 2^{2^3}$	$2^{2^3} <= 2^n$	4	
$(2^8)^2 = 2^{16} = 2^{2^4}$			
$i = 2^{16}$	$2^{2^4} <= 2^n$	5	
⋮			
	$2^{2^K} <= 2^n$	(K+1)	

The pattern here is $2^{2^{(itr-1)}}$

$\therefore 2^{2^K} = 2^n$

$2^K = \log_2 2^n = n$

$K = \log_2 n$

$\therefore FC = 1 + K = 1 + \log_2 n$

$TC = O(\log_2 n)$

3. for(i=1; i^2 <= n; i++)
printf("welcome").

		iter
i=1	1^2 <= n	1
i=2	2^2 <= n	2
i=3	3^2 <= n	3
⋮		
i=K	K^2 <= n	K.

∴ $K^2 = n$.

$K = \sqrt{n}$.

∴ FC = $K = \sqrt{n}$.

TC = $\Theta(\sqrt{n})$.

HW

Determine the time complexity of the followins loops

(1) for(i=1; i<=$\sqrt[3]{n}$; i++)
print("welcome").

		iter
i=1	i <= $\sqrt[3]{n}$	1
i=2	2 <= $\sqrt[3]{n}$	2
i=3	3 <= $\sqrt[3]{n}$	3
⋮		
i=K	K <= $\sqrt[3]{n}$	K

$K = (n)^{\frac{1}{3}}$

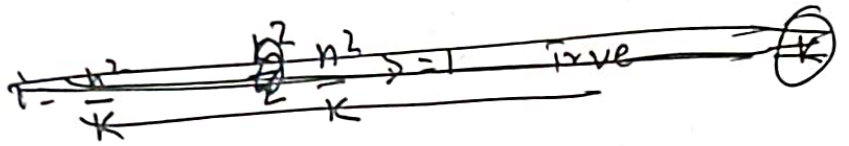
∴ FC = $\sqrt[3]{n}$ TC = $\Theta(\sqrt[3]{n})$.

Time complexity of single loop - Initialization statement is a function.

1. for (i = n^2 ; i >= 1 ; i = i/2).
   print ("n -- n") --> FC of this instruction

		Result	itr
$i = n^2$	$\frac{n^2}{1} >= 1$	True	1
$i = \frac{n^2}{2}$	$\frac{n^2}{2} >= 1$	True	2

$1 = 2^0$.



$\therefore$ $\frac{n^2}{K} >= 1$

$i = \frac{n^2}{4}$ $\quad \frac{n^2}{2^2} >= 1$ $\quad$ True $\quad$ 3.

$\frac{n^2}{2^K} >= 1$ $\quad (K+1)$

$\therefore$ $n^2 = 2^K$

$K = \log_2 n^2 = 2 \log_2 n$.

TC = $\theta(\log n)$

FC = $(2 \log_2 n + 1)$ times.

Ex.2 `for(i=5^n; i>=5; i=sqrt(5)i)`.

`print("...")`

		Res	iter
$i = 5^n$	$5^n >= 5$	True	1
$i = 5^{\frac{n}{5}}$	$5^{\frac{n}{5}} >= 5$	"	2
$i = 5^{\frac{n}{5^2}}$	$5^{\frac{n}{5^2}} >= 5$	"	3
			⋮
	$5^{\frac{n}{5^k}} >= 5$	True	(k+1)

$$5^{\frac{n}{5^k}} = 5 = 5^1$$

$$\Rightarrow \frac{n}{5^k} = 1$$

$$\Rightarrow 5^k = n \quad \Rightarrow k = \log_5 n.$$

FC = $\log_5 n + 1$

TC = $\Theta(\log n)$.

Time complexity - Nested loops.

Independent nested loops.

- At least 1 loop is enclosed within another.

Types

- Independent [in which inner loop variable is independent of outer loop variable]
- Dependent.

Time complexity of independent Nested loop =

(FC)$_{\text{outer loop}}$ × (FC)$_{\text{inner loop}}$

This can be extended as required.

Ex:-

```
for ( i=1;  i<=n;  i++)
    for ( j=1;  j<=n;  j++)
        x = x+1
```

Conditions for Nested loop

① At least one loop is within another loop

② Is this independent? ✓ (yes).

inner loop variable = j
outer " " = i
] are they related? (No).

'j' is independent of i.

(FC)$_{\text{loop j}}$ = n. (FC)$_{\text{loop i}}$ = n.

(n × n) times. = n^2

∴ ~~(FC) = n × n =~~ n^2

i=1 → j = 1 to n (n)

i=2 → j = 1 to n (n)

... i = n → j = 1 to n (n.)

FL = (FC)$_{\text{outer loop}}$ × (FC)$_{\text{inner loop}}$ = n × n = n^2 = $\underline{\underline{\Theta(n^2)}}$.

TC.

more examples - Independent Nested loop

Ex 1.
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++).
        for (k=1; k<=n; k++).
            x = x+1
```

FC = $n \times n \times n = n^3$ ⇒

TC = $\theta(n^3)$.

Ex 2.
```
for (i=3; i<=n; i=i^3)
    for (j=n; j>=1; j=j/2)
        for (k=1; k<=n; k=k*4).
            x = x+1.
```

Loop 1 (i)

		itr
$i = 3$	$3 <= n$	1
$i = 3^3$	$3^3 <= n$	2
$i = 3^9$	$(3^3)^3 <= n$	3.
	$(3^3)^{3^2} <= n$	4.
	⋮	
	$(3^3)^K = n$	(K+1)

$\therefore \quad 3^{3^K} = n$

$\Rightarrow \quad 3^K = \log_3 n$. $\quad (FC)_{loop1} = K+1 = \log_3 \log_3 n + 1$

$\Rightarrow \quad K = \log_3 \log_3 n$

(85) loop 2 - j

		itr
$j = n$	$n >= 1$	1
$j = \frac{n}{2}$	$\frac{n}{2} >= 1$	2
$j = \frac{n}{4}$	$\frac{n}{2^2} >= 1$	3.
$j = \frac{n}{8}$	$\frac{n}{2^3} >= 1$	4.
	:	
	$\frac{n}{2^k} >= 1$	(k+1)

$\frac{n}{2^k} = 1$

$\Rightarrow k = \log_2 n$.

$\therefore (FC)_{loop2} = \log_2 n + 1$.

loop 3 - k

		itr
$k = 1$	$1 <= n$	1
$k = 4$	$4 <= n$	2
$k = 4^2$	$4^2 <= n$	3
$k = 4^3$	$4^3 <= n$	4
	:	
	$4^k <= n$	(k+1)

$4^k = n$.

$k = \log_4 n$.

$(FC)_{loop3} = 1 + \log_4 n$.

$(FL) = (FC)_{loop1} \times (FC)_{loop2} \times (FC)_{loop3}$.

$= [1 + \log_3(\log_3 n)] \times [1 + \log_2 n] \times [1 + \log_4 n]$

$TC = \log_3(\log_3 n) \times \log_2 n \times \log_4 n$.

Observation

```
for(i=3; i<=n; i=i^3)  --> log3 log3 n . (powers -> 2 times log)
    for(j=n; j>=1; j=j/2)  --> log2 n .
        for(k=1; k<=n; k=k*4)  --> log4 n
            x = x+1
```

Whether it is multiplication or division, the TC = log n

$$TC = \theta(\log^2 n \cdot \log \log n)$$

$\log^2 n$: $\log_2 n \times \log_4 n$ ⊗ Forget the bases

$\log \log n$: Omit the bases

Time complexity - Dependent loops

Dependent nested loop

inner loop variable => depends on outer loop variable.

- How to find the TC?

Ex 1)
```
for(i=1; i<=n; i++)
    for(j=1; j<=n; j=j+i).
```
≈ dependency.

```
x = x+1
```

Approach:- Analyze the behaviour of the inner loop for each iteration of the outer loop.

(87)

i = 1

↳ inner for loop.

will run = n times.

i = 2

↳ inner for loop. ——→ ~~$\log_2(n-1)$~~.

for (j = 1; j <= n; j = j + 2). ——→ $\frac{n}{2}$ times.

j	j <= n	it+1
1	1 <= n	1
3	1+2 <= n	2
5	1+2+2 <= n	3.
	1+2x2	
7	1+3x2 <= n	4
	⋮	⋮
	1+Kx2 <= n	(K+1)

$2^K + 1 = n$

$2^K = n - 1$

$\Rightarrow K = \log_2(n-1)$.

(K+1) iteration = $1 + \log_2(n-1)$.

$2K + 1 = n$

$K = \frac{n-1}{2}$

$\simeq \frac{n}{2}$ times.

i = 3.

for (j = 1; j <= n; j = j + 3)

will run for $1 + \log_3(n-1)$.

Er

(P.T.O)

i = n.

$1 + \log_n(n-1)$.

(88)

$i = 1$ —> inner loop will run for n times.

$i = 2$ —> inner loop will run for $\frac{n}{2}$ times

$i = 3$ —> n n n n n $\frac{n}{3}$ times

⋮

$i = n$ —> n , , , , $\frac{n}{n}$ times
$= 1$ time.

Frequency count = Times the outer loop runs ↓ + How many times the inner loop runs: ↓

n + ~~1+2+ ---~~

$n + \frac{n}{2} + \frac{n}{3} + \cdots 1.$

$\Rightarrow n\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right]$

$\Rightarrow n \times \log_e n.$

$\therefore$ Frequency count $= n + n \times \log_e n.$

$= n\left[1 + \log_e n\right]$

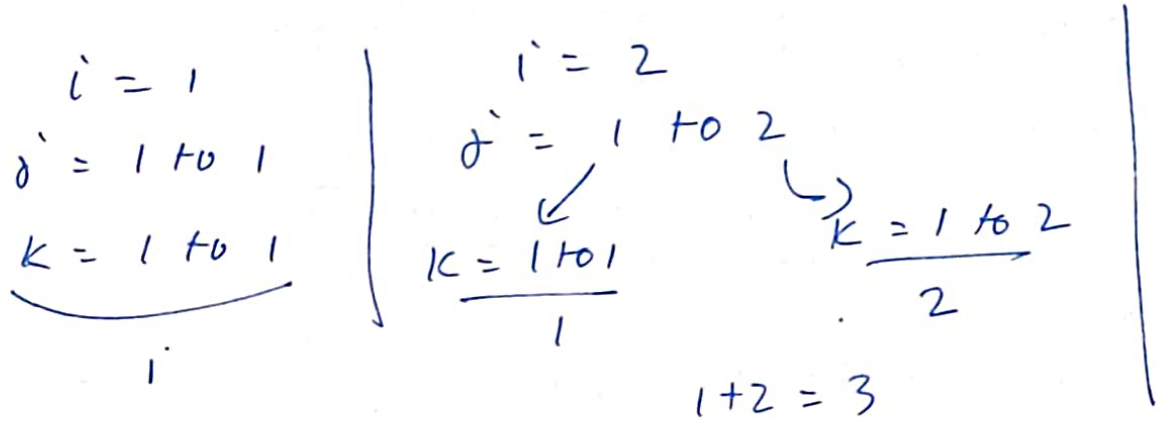
$\therefore$ TC $= \Theta(n \log n)$. [by dropping the constants]

Dependent loop - Examples

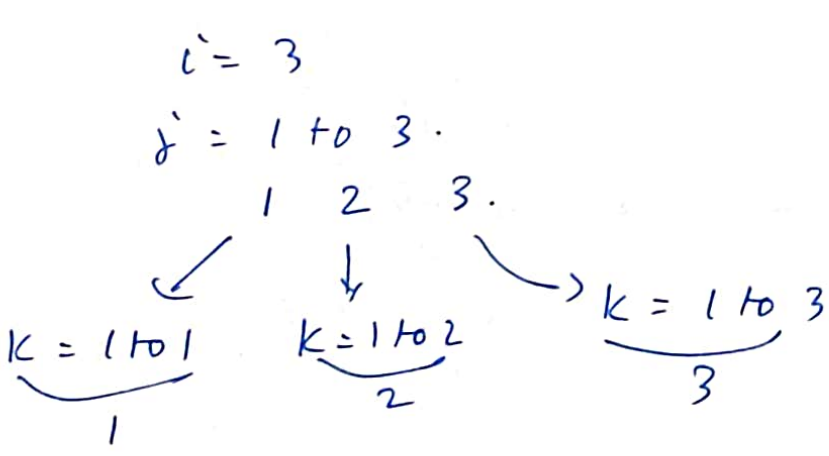
Ex 2)

```
for( i=1; i<=n; i++).
    for (j= 1; j<=i; j++).
        for ( k=1; k<=j; k++)
            x = x+1
```

Let us analyze how many times



The innermost loop k is what we are interested in.



pattern $\underbrace{i=1}_{1}$ $\underbrace{i=2}_{1+2}$ $\underbrace{i=3}_{1+2+3}$ ... $\underbrace{i=4}_{1+2+3+\cdots 4}$

(96)

Total = $1 + (1+2) + (1+2+3) + \cdots + (1+2+\cdots+n)$

$= x_1 + x_2 + x_3 + \cdots + x_n$.

$\sum_{i=1}^{n} x_i$ where x_i = sum of 1st i natural nos.

$= \sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{1}{2} \sum_{i=1}^{n} (i^2 + i)$.

$= \frac{1}{2} \left[\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i \right]$

$= \frac{1}{2} \left[\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right]$

$= \frac{1}{2} \left[\frac{n(n+1)}{2} \left(\frac{2n+1}{3} + 1 \right) \right]$

$= \frac{1}{2} \frac{n(n+1)}{2} \times \frac{2n+4}{3}$

$\frac{2n+4}{3} \rightarrow \frac{2(n+2)}{3}$

$= \frac{n(n+1)(n+2)}{6} \simeq \Theta(n^3)$

dominant term.

(91)

Ex 3

```
for (i=1; i<=n; i++)
    for (j=i; j<=n; j++)
        for (k=j; k<=n; k++)
            x = x+1
```

FC = How many times x = x+1 will run.

Another way of solving Ex 2

$$\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} 1$$

$$\sum_{k=1}^{j} 1 = 1 + 1 + \cdots \text{ j times} = j$$

$$\Rightarrow \sum_{i=1}^{n} \sum_{j=1}^{i} j$$

Let's evaluate $\sum_{j=1}^{i} j = \frac{i(i+1)}{2}$

$$\Rightarrow \sum_{i=1}^{n} \frac{i(i+1)}{2}$$

$$\Rightarrow \frac{1}{2} \sum_{i=1}^{n} [i^2 + i] = \frac{1}{2} \left[\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i \right]$$

$$= \frac{1}{2} \left[\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right]$$

$$\simeq \frac{n(n+1)(n+2)}{6} \simeq \Theta(n^3).$$

Ex 3

```
for( i=1; i<=n; i++)
    for (j=i; j<=n; j++)
        for (k=j; k<=n; k++)
            x = x+1
```

$$\sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=j}^{n} 1$$

Step

(1) $\sum_{k=j}^{n} 1 = ?$

Let's take an example here

$$\sum_{k=2}^{5} 1 = 1 + 1 + 1 + 1 = 4$$

iter = 2 3 4 5

$(5 - 2 + 1)$.

Let's take another example

$$\sum_{k=3}^{7} 1 = 1 + 1 + 1 + 1 + 1 = 5$$

$k = 3, 4, 5, 6, 7$

$(5) = (7 - 3 + 1)$.

$$\therefore \sum_{k=j}^{n} 1 = (n - j + 1).$$

(2) $\sum_{i=1}^{n} \sum_{j=i}^{n} (n - j + 1)$.

(93)

$\sum_{j=i}^{n} (n-j+1)$ = Sum of first $(n-i+1)$ natural nos.

$= \sum_{j=i}^{n} n - \sum_{j=i}^{n} j + \sum_{j=i}^{n} 1$

$\sum_{j=1}^{n} j$ $\sum_{j=1}^{m} j = \frac{m(m+1)}{2}$.

$m = (n-i+1) = \frac{(n-i+1)(n-i+2)}{2}$.

Let's take an example

$\sum_{j=i}^{n} (n-j+1)$.

$n = 10$ $i = 5$

$\sum_{j=5}^{10} (10+1-j) = \sum_{j=5}^{10} (11-j)$.

$= (11-5) + (11-6) + (11-7)$
$+ (11-8) + (11-9) + (11-10)$.

$= 6 + 5 + 4 + 3 + 2 + 1$

$= \frac{6 \times 7}{2} = 3 \times 7 = 21$

$= \sum_{j=5}^{10} (10-j+1)$ = Sum of first $(10-5+1)$ natural nos
$= 6$.

=

(94)

Can 9 write

$\sum_{j=i}^{n} (n-j+1)$ = Sum of first $(n-i+1)$ natural nos.

$\therefore \quad \frac{(n-i+1)(n-i+2)}{2}$.

Now.

(3) $\sum_{i=1}^{n} \sum_{j=i}^{n} (n-j+1)$

~~1+1+... 5 times~~
~~= 5×1=5~~

$= \sum_{i=1}^{n} \frac{(n-i+1)(n-i+2)}{2}$

$= \frac{1}{2} \sum_{i=1}^{n} (n-i+1)(n-i+2)$.

$= \frac{1}{2} \sum_{i=1}^{n} [n^2 - in + 2n - in + i^2 - 2i + n - i + 2]$

$= n^2 - 2in + 3n - 3i + i^2 + 2$

$\sum_{i=1}^{n} n^2 = n^3$. $(n^2+n^2+\cdots n \text{ times}) = n \times n^2$

$\sum_{i=1}^{n} in = 1\times n + 2\times n + 3\times n + \cdots + n\times n$

$= n(1+2+3+\cdots+n) = n \frac{n(n+1)}{2}$

$\sum_{i=1}^{n} n = n^2$

$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \underline{\underline{\theta(n^3)}}$

T.C.

Ex 4)

```
for( i=n, j=0; i>0; i/=2 | j+=i ).
```

← init → condition ↓ 'i' update ; ↳ j update

Let val(j) denotes the value stored in the variable j after termination of the for loop. Which one of the following is true?

(A) $val(j) = \theta(\log n)$ (B) $val(j) = \theta(\sqrt{n})$

(C) $val(j) = \theta(n)$ (D) $val(j) = \theta(n \log n)$.

[GATE 2006]

$i = n$ ---> Decreases by $\frac{1}{2}$ in each iteration.

$i > 0$ ---> condition

$j = 0$, $j = j + i$

~~Let's look at how~~

~~Let us look at how the~~

first let us look at what will happen with just i.

i	i>0	i+1	Result	val(j)
				0
$i = n$	$n > 0$	1	true	$0 + \frac{n}{2} = \frac{n}{2}$
$i = \frac{n}{2}$	$\frac{n}{2} > 0$	2	true	$\frac{n}{2} + \frac{n}{4} = \frac{2n+n}{4} = \frac{3n}{4}$
$i = \frac{n}{4}$	$\frac{n}{2^2} > 0$	3	true	$\frac{3n}{4} + \frac{n}{8} = \frac{6n+n}{8} = \frac{7n}{8}$
$i = \frac{n}{8}$	$\frac{n}{2^3} > 0$	4		$0, \frac{n}{2}, \frac{3n}{4}, \frac{7n}{8}, \ldots$
	!			
	$\frac{n}{2^K} > 0$	(K+1)		$\frac{(2K+1)n}{2^K}$ $j = \frac{(2K-1)n}{2^K}$

$\frac{n}{2^K} = 0$ $\frac{2^K}{n} = \frac{1}{0} = \infty$

$K = \infty$ → The loop will never end.

* We did correctly but slight issue in representation of val(j).

		itr	Result	val(j)
$i = n$	$n > 0$	1	True	0
$i = \frac{n}{2}$	$\frac{n}{2} > 0$	2	True	$0 + \frac{n}{2} = \frac{n}{2}$
$i = \frac{n}{4}$	$\frac{n}{4} > 0$ $\downarrow$ $\frac{n}{2^2}$	3	True	$\frac{n}{2} + \frac{n}{2^2}$
⋮				
$\frac{n}{2^k} \gtrdot 0 = 1$		(k+1)	True	$\frac{n}{2} + \frac{n}{2^2} + \cdots \left(\frac{n}{2^k}\right)$

$\therefore$ val(j) = $n \left[\underbrace{\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^k}}_{GP} \right]$

S_n for GP = $\frac{a(1-r^n)}{1-r}$ $(r < 1)$.

$a = \frac{1}{2}$ $\quad r = \frac{1}{2}$ $\quad n = K$.

$= \frac{\frac{1}{2}\left(1 - \frac{1}{2^k}\right)}{1 - \frac{1}{2}} = \frac{2^k - 1}{2^k}$

$\therefore$ val(j) = $n \cdot \left(\frac{2^k - 1}{2^k}\right)$.

Now, let's say in (k+1)th iteration

$$val(j) = \frac{n}{2^k}$$

we assume $\frac{n}{2^k} = 1$ (Why?)

$i = n, \frac{n}{2}, \frac{n}{2^2} \cdots \left(\frac{n}{2^k}\right)$ — last term for which the loop executes

$2^k = n$

$k = \log_2 n$.

$$val(j) = n \cdot \frac{(2^K - 1)}{2^K} = n \times \frac{n-1}{n} = (n-1) = \theta(n).$$

Space Complexity

We will study

- Space complexity - Recursive algorithms.
- Time complexity - Recursive algorithms.

Space Complexity

most of the algorithms that we studied are (iterative algorithms.) so far.

Recollect → What is Space complexity?

1. Space required - to store the source code
2. Space required - for simple variables
3. Space required - for data structures used
4. Space required - for stack of recursive algorithms.

(1) & (2) => irrelevant => Constants.

(3) & (4) => dependent on the i/p data size.

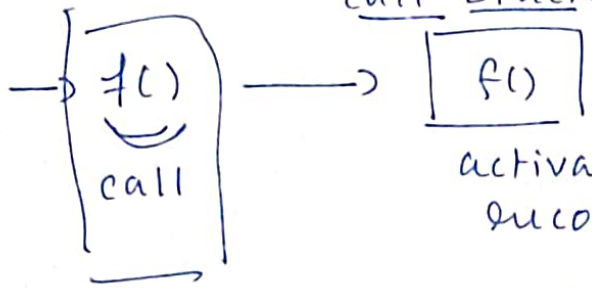
* For iterative algorithms → we generally do not use data structures generally. That is why we have not discussed the space complexity there.

But for Recursive algorithms → we need to disuss the space complexity.

Call Stack

- A specific memory space used to manage function calls.

call stack → follows LIFO.



- return address
- variables of that function → local

where the function should return after its completion

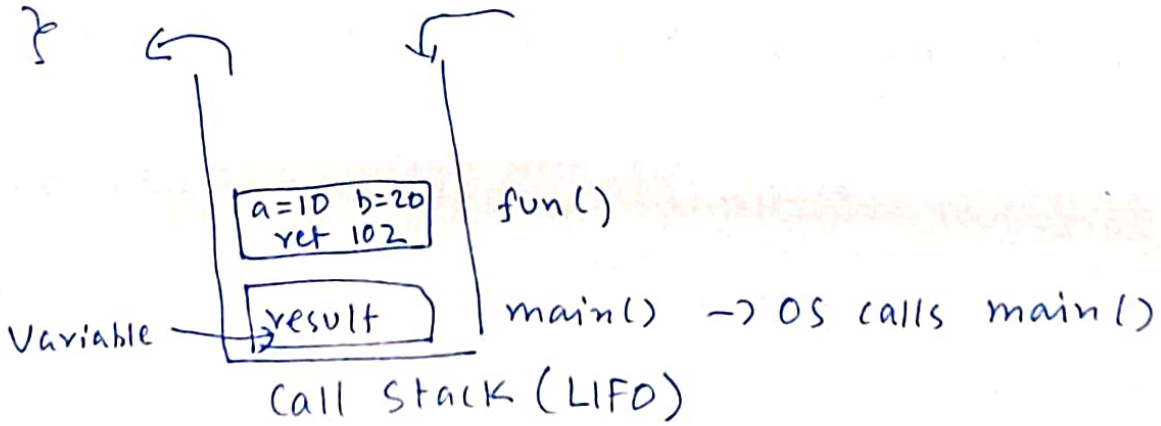
ex:-

```
int fun (){
}
main (){
    int result;
    result = fun(10,20);
    return 0;
}
```

addresses of the instr. 101, 102, 103

* Write the int fun() here describing below because space is not there

```
int fun (int a, int b) {
    return a+b
}
```



Note

* The top ~~of the~~ record of call stack shows the current function.

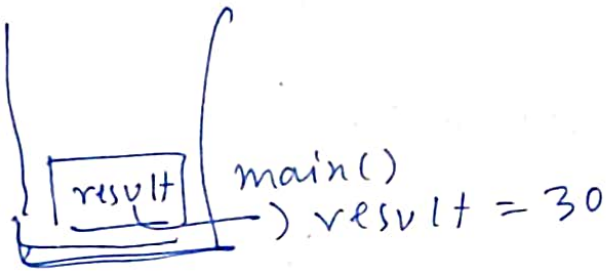
- After the function executes, it should return to the specific spot from where it was called.

  fun()

  That's why [ret 102] is important.

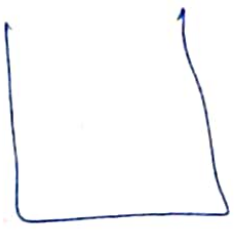
- return a + b

  The values will be accessed from the activation record only.



call stack
(after return from fun()).

[103] → return 0 → call goes back to OS



=> done with the program.

call stack
(empty)

Why did we study the call stack?

In recursive algorithms → call stack plays an important role.

Recursive Function

- A Recursive function
- Behavior of the call stack.

Def:- A function that calls itself directly or indirectly.

Types → Direct Recursion.
Types → Indirect Recursion

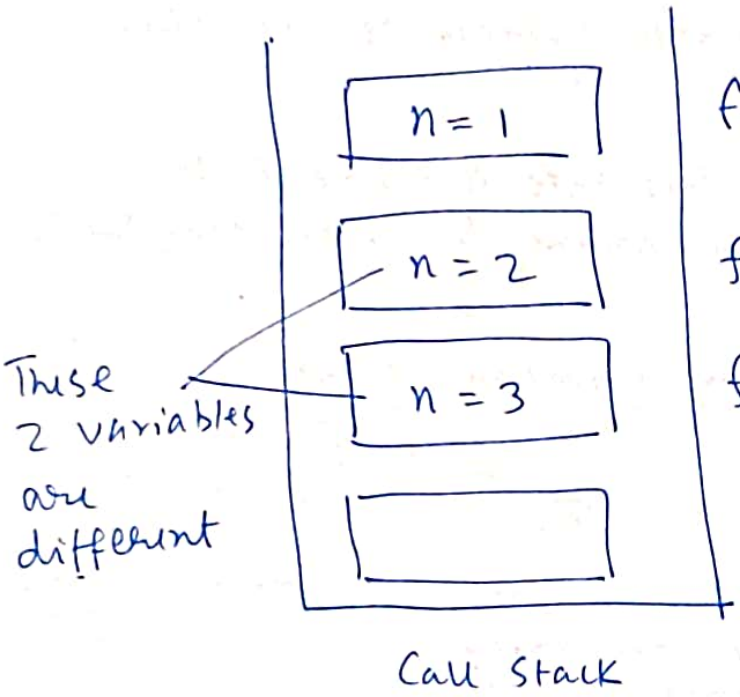
Direct Recursion

```
fun() {
    ...
    ..
    fun()  // calling itself directly.
    ..
}
```

Indirect Recursion

```
fun1() {
    ..
    ..
    fun2()  ---> fun2() {
    ..                ..
}                     fun1()  // Indirect call.
                      ..
                  }
```

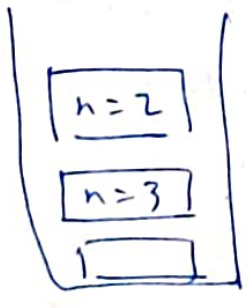
```
int fun (int n) {
        if (n == 1)
            return 1
        else
            return fun (n-1);
}
main () {
    fun(3).
    return 0
}
```



↳ Note
- no local variables
- ~~no return addresses~~

note
- we are not mentioning the return addresses but the exist ~~implicitly~~.

After n=1 goes out



fun()
↓
fun()
↓
main()
↓

n=1 fun() goes out.

=> * What n=2 fun() got in return?
1

one by one the ~~code~~ activation record gets out & call stack becomes empty.

Observe

(1) Here in this case, the maximum number of activation records in the call stack = 3 (excluding the main())

but that is for this specific case.

but the space complexity here can be determined by the size of the i/p = 3.

For $n = 3$, the maximum number of activation records in call stack = 3.

↳ This info is essential to determine the space complexity.

Q:- Is it not dependent on the design of the function?

What if fun(n-2) had happened?

n	no. of activation records
3	3 .. for this program.
n	n

Finding Space complexity

fun(3) → fun(2) → fun(1)

1 2 3

```
int fun(n){
    if (n==1)
        return 1
    else
        return fun(n-1)
}

main(){
    fun(3)
    return 0
}
```

This idea

- If n = 3, no. of function calls = 3. needs to be tested.
- What if

```
int fun(n){
    if (n<=100)
        return fun(n+1)
    else
        return
```

This is still a recursive function but will be called 100 times. —> Talking about ($n <= 100$) case.

Number of function calls = Highest number of activation records

~~This idea is not clear yet to me~~. Not necessarily.

Another

Function calls & activation records

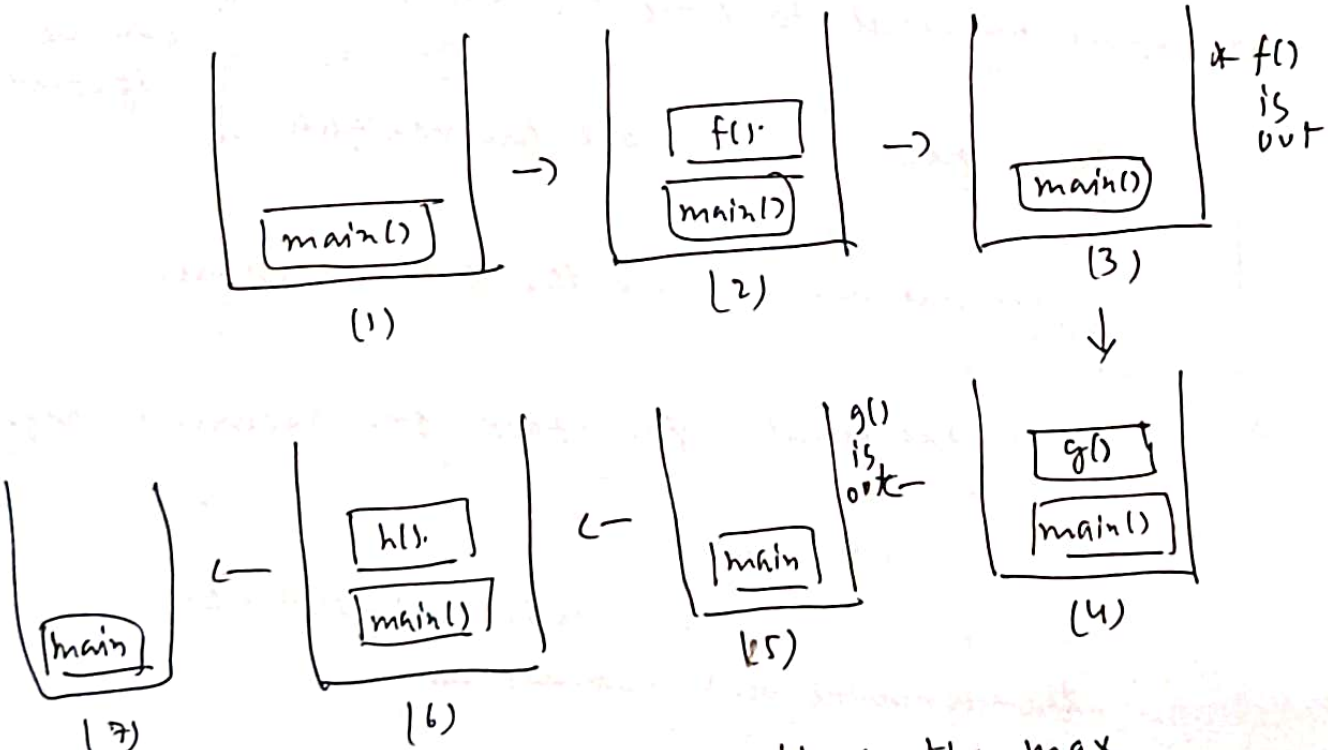
```
main()
{
   f()
   g()
   h();
}

f()
{
   int x = 10
   x = x+1
}

g()
{
   int y = 20
   y = y x 2
}

h()
{
   int z = 30
   z = z/3
}
```

Let's look at ~~that~~ this example.

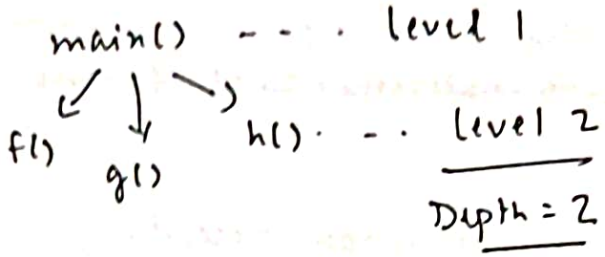


At any given time, the max no. of activation records = 2

* What is the idea?

* We can't depend on the function calls to determine the number of activation records.

, What is depth of the function calls?



We also know the highest number of activation records = 2

∴ We can say, if we know the depth of function calls, we can find the max. number of activation records.

Space complexity - algorithm

= Space required to store the source code (constants) can be ignored
+
Space required - store the variables
+
(3) Space required - store the data structures
+
(4) Space required - for stack for recursive algo. call stack.

~~(3) → can be ignored as we are not using the complex data structures.~~

(4) -> Depth of recursion.

Space complexity of recursive algorithm = space for the data structures used + Depth of Recursion.

Ex :- 1) Space complexity — n!

$n! = n \times (n-1) \times (n-2) \times \cdots 1$.

$n \in N$.

$n! = n \times (n-1)!$

Immidiate next step

```
fact(n):
    if n == 1:
        return 1
    else
        return n * fact(n-1)
```

Terminate condition

base case (exit)

recursive case

fact(4)

4 x fact(3) = 24

3 x fact(2) = 3x2 = 6.

2 x fact(1). = 2x1 = 2

for 4 -> 4 depth

for n -> n depth.

Space complexity = $\Theta(n)$.

Space complexity = space of DS used (ignored) + Depth of Recursion.

for n! => (n).

$\theta(n)$ = Better choice => [best + worst case] shows.

Ex. 2 Space complexity - n^{th} fibonacci

What is fibonacci sequence?

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, - - - -

~~fib(n) if~~
~~if n<=0~~
~~return 0~~
~~elif n==~~
~~else:~~
~~return~~
~~fib(n-1) + fib(n-2)~~

$fib(0) = 0$
$fib(1) = 1$

Known ↓

$fib(n) = fib(n-1) + fib(n-2)$

if n=2 → fib(1), fib(0)] We have to deal with

Algo fib(n):

```
if n == 0
    return 0
elif n == 1
    return 1
else
    return fib(n-2) + fib(n-1).
```

base case.

(— recursive case.

Find Space complexity

Space complexity = Space of DS used + Depth of Recursion.

Space for the call stack

no complex DS used => (Space of DS) can be eliminated.

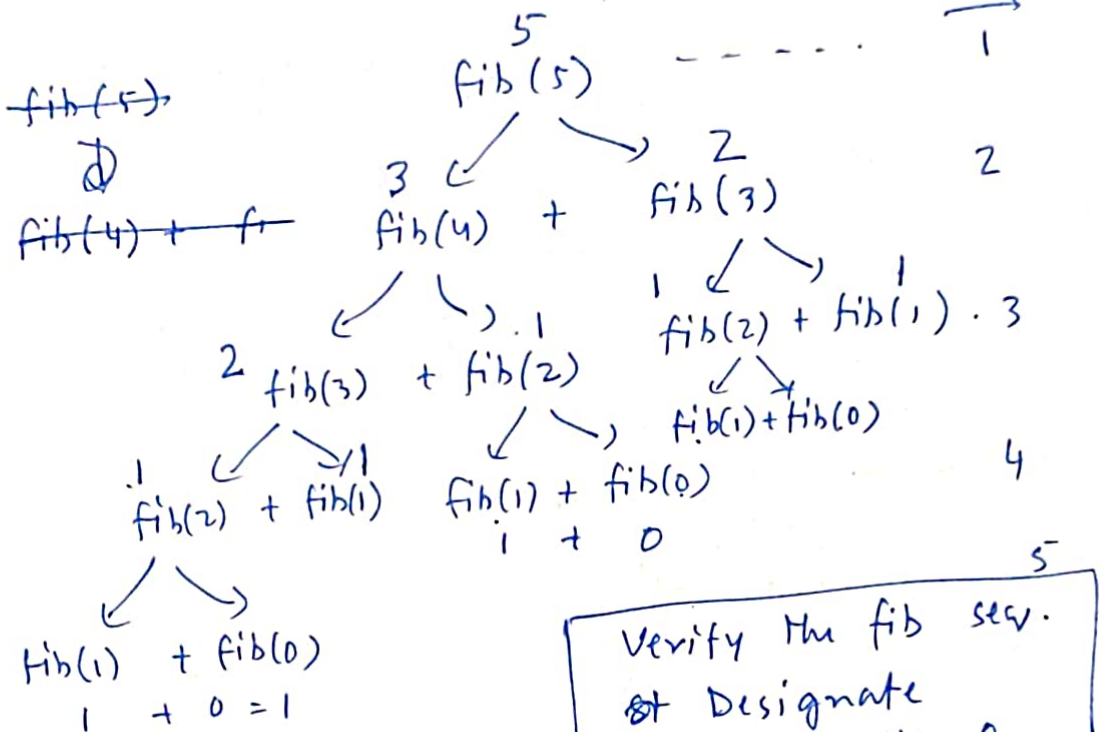
Depth of Recursion

→ We represent space complexity in asymptotic notations = θ(depth of recursion)

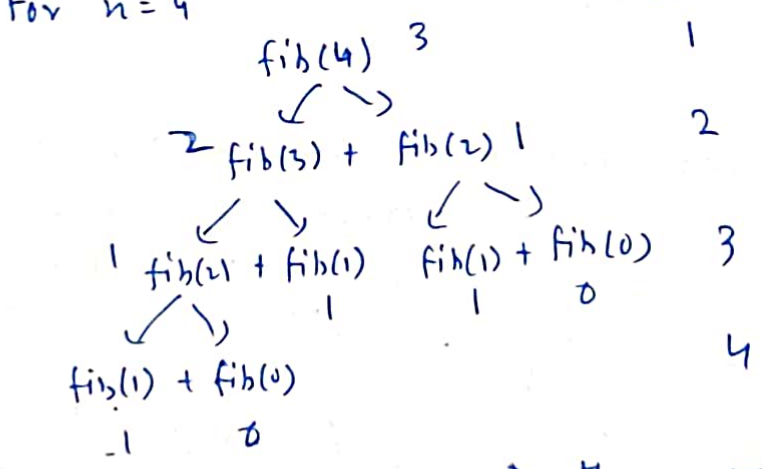
How to find the depth of recursion?

- We need to build the tree and then generalize it.
- But first we need to build the tree with a sample number.

$n = 5$



Verify the fib seq.
Designate "0" with 0



∴ For

n	Depth
5	5
4	4
n	(n)

Space complexity = O(n).

Solved Problems

(1) What is the space complexity of the following algorithm?

```
Algo f(n)
{
    if n == 1
        return 1
    else
        return (f(n/2) + 1)
}
```

Space complexity = O(Depth of Recursion)

Let us see a normal case.

f(4) 3.

Level

1

1 + f(2) 2

1 + 1 + f(1) 3

f(5) => ?

But this approach will not be helpful in solving the problem we have here.

Note:-

(1) We are not interested in knowing the value at any level. We are just interested in knowing the Depth of recursion

Level

f(n) 1

f(n/2) 2

f(n/2²) 3

f(n/2³) 4

⋮

no. of function calls → Not true for all

f(n/2^k) (k+1)

↳ Let say this returns 1

$$\frac{n}{2^k} = 1$$

$$\Rightarrow 2^k = n.$$

$$k = \log_2 n.$$

iteration $\Rightarrow$ $k+1 = 1 + \log_2 n.$

- ~~for every recursion~~

Space complexity = $\Theta(\log_2 n)$.

---

(2) What is the space complexity of the following algorithm.

```
Algo f(n)
{
    if n <= 10
        return 1
    else
        return (f(√n) + 1).
}.
```

	Level
f(n)	1
↓	
$f(n^{\frac{1}{2}})$	2
↓	
$f(n^{\frac{1}{2^2}})$	3
⋮	⋮
$f(n^{\frac{1}{2^k}})$.	(K+1)

$$n^{\frac{1}{2^k}} = 10. \quad \text{--(K+1)}$$

$$\Rightarrow \frac{1}{2^k} = \log_n 10$$

$$\Rightarrow 2^k = \log_{10} n.$$

$$\Rightarrow k = \log_2 \log_{10} n.$$

$$\Rightarrow 1+k = 1 + \log_2 \log_{10} n.$$

Space complexity = $O(\log \log n)$.

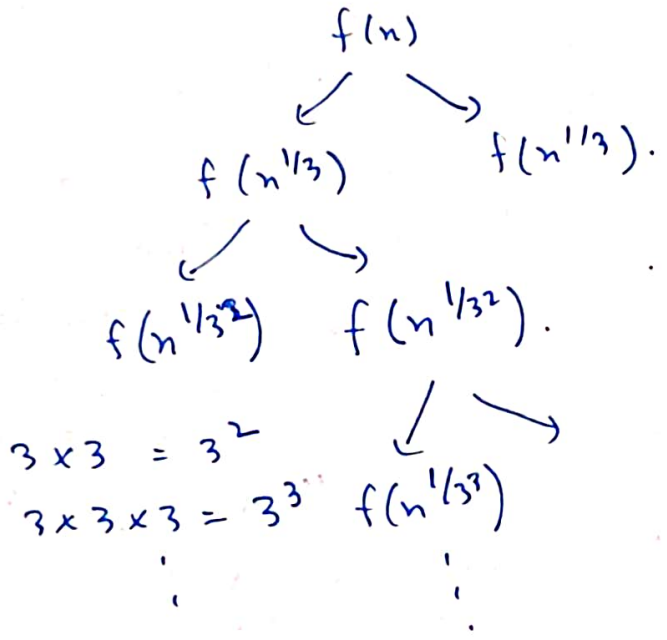
Why $O(\log \log n)$ Why not $\Theta(\log \log n)$?

- The Best case space complexity will be achieved if $n <= 10$.
- or else it is the worst case and that's why Big O

(3) Space complexity ?

```
Algo f(n)
{
    if n<=7
        return 1
    else
        return (f(∛n) + f(∛n) + 1).
}
```

We need to know the flow of function calls.



Let $f(n^{1/3^k})$ is when the exit condition comes.

$$n^{\frac{1}{3^k}} = 7$$

$$\frac{1}{3^k} = \log_n 7$$

$$3^k = \log_7 n.$$

$$\Rightarrow k = \log_3 \log_7 n.$$

$$\Rightarrow (k+1) = 1 + \log_3 \log_7 n.$$

Space complexity = $\Theta(\log \log_3 n)$

(11)

Note:- Why it is Big O?

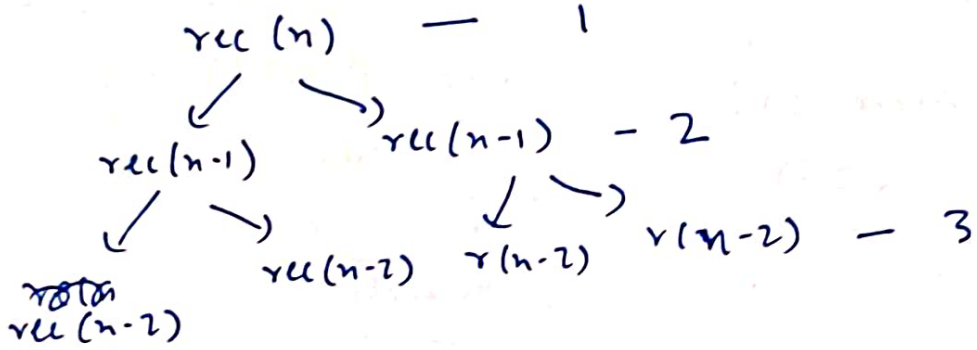
The best case space complexity will happen if.

$n <= 7$ or else it is Big O;

---

(4)

```
int rec (int n) {
    if (n == 1)
        return 1;
    else
        return (rec(n-1) + rec(n-1))
}.
```



$\therefore$ dot $(n-K) = 1$ $\quad \therefore -(K+1)$

$K = n-1$ $\quad \therefore \boxed{K+1 = n}$ → which is also the depth.

$\therefore$ Space complexity = $O(n)$.

Best case = $\Omega(1)$.

Rapid Five Quiz. (14 Qs).

1. What info is typically stored in the activation record of a stack.

[(b)] local variables
function arguments
ret. address

2. Which of the following is an example of indirect recursion?

(a)
```
void func1() {
    func1()
}
```
--> Direct

[(b)]
```
void func1() {
    func2();
}

void func2() {
    func1()
}
```
↳ Indirect

(c)
```
void func1() {
}
```
-> This not recursion at all

(d)
```
void func1() {
    if (cond)
        func1()
}
```
--> Direct.

3. What happens to the activation record of a function when the function returns?

[(b)] The activation record is popped out from the call stack

(4) Which one is true?

(a) max no. of activation record = the no. of function calls
×
max no. of parameters of the function x

(b) Depth of Recursion ✓

(c) None of the above

(d) Same.

---

(5). What is the space complexity of a recursive function that performs a single recursive call in each function execution? (Assume the problem size = n).

```
f(n) {
  if n == 1
    return 1
  f(n-1)
}
```

n → n-1 → n-2 → ... → (n-K) -- (K+1)

n - K = 1

K = n - 1

∴ K + 1 = n → Depth of recursion.

∴ O(n) → Space complexity

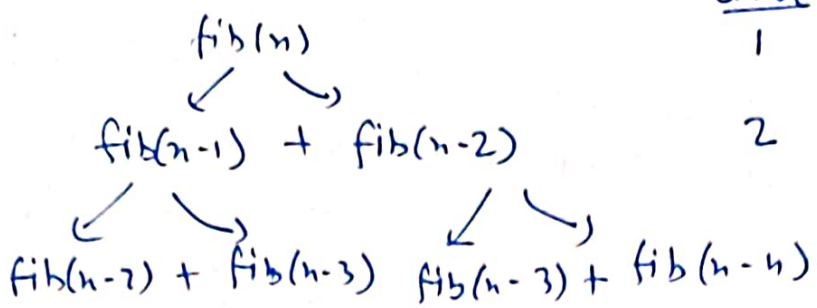
---

7. For a recursive function that has a branching factor of 2 → it calls itself twice in each call. What is the space complexity?

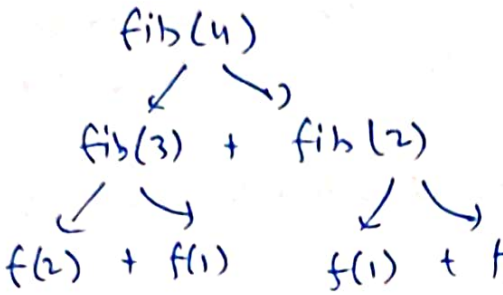
Similar to fib(n).

```
fib(n):
    if n == 0
        return 0
    if n == 1
        return 1
    else
        fib(n-2) + fib(n-1).
```

(114)



Level
1
2
3.



Level
1
2
3.

∴ Depth of recursion = ~~(n-1)~~ . n.

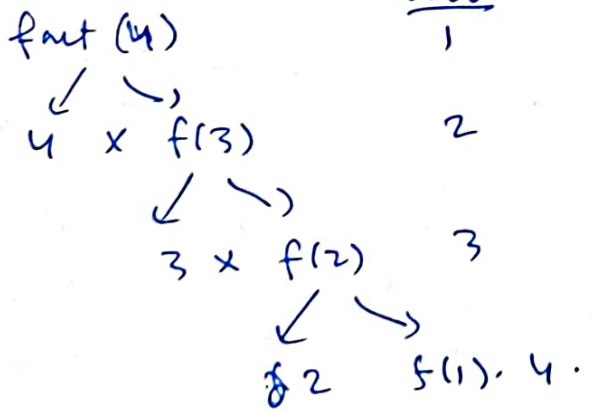
∴ Space complexity = O(n).

---

(8) Simple

(9)

```
fact(n):
    if n == 1
        return 1.
    else
        return n x fact(n-1).
```



Level
1
2
3

For n	Depth
4	4
n	n.

Space complexity = ~~O(n)~~.
= O(n).

* Each recursive calls uses O(1) space. Therefore there are n levels of recursion.

(11 5)

(10) For a recursive algo that uses an auxiliary array size n and has a recursion depth of n. What is the overall space complexity.?

size of DS = n. → O(n).

Depth of Recursion = n. → O(n).

~~The overall space complexity =~~

For each n. → Depth of rec = O(n).

∴ Space Complexity = $O(n^2)$.

---

(12) Given the following algorithm that uses 2 nested loops, what is the space complexity?

```
void printPairs(int n){
    for (int i=0; i<n; i++).{
        for (int j=0; j<n; j++).{
            printf("%d %d \n", i,j).
        }
    }
```

Space complexity = O(1).

Time Complexity - Recursive Algorithms

⑥ Writing Recurrence Relations

Steps

① Write the recurrence relation of the algorithm
② Solve the recurrence relation.
③ Represent recurrence relation using asymptotic notation

~~What is a recurrence relation?~~

* What is a recurrence relation?

math expression. = Describes overall cost of the problem

↓

in terms of solving the smaller sub problems.

cost can be
- Time
- Ret value
- no. of OPS
- space.

ex:-
```
Algo fact(n) {
    if n == 1
        return 1
    else
        return n * fact(n-1).
}
```

Let T(n) represents the time to solve fact(n).

COST

. What is the overall problem? → Time required to complete fact(n)

• now we need to represent $T(n)$ in terms of solving smaller sub problems

What are the smaller sub problems?

Algo fact(n) ⟵ $T(n)$.

Sub prob 1 — if $n == 1$ return 1

Sub prob 2 — else return $n \times$ fact$(n-1)$

What is the base case?

$T(1)$ —> if $n == 1$ return 1

$T(1) = c$ (some constant)
↓
we don't know what c is.

~~fact(n)~~

$T(n)$ —> takes to solve fact(n)

$T(n-1)$ —> " " " fact$(n-1)$.

But then what about the multiplication?

$n \times$ fact$(n-1)$
?

This will be constant time.

∴ we can say $n \times$ fact$(n-1)$ => will take $T(n-1) + c$

The constant C
=> perform multiplication
+
=> peform return operation.

Recurrence relation —> This is $(n-1)$

$$T(n) = \begin{cases} T(n-1) + c & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

How to solve the recurrence relation? - (Using Substitution method)

$$T(n) = \begin{cases} T(n-1) + c & \text{if } n > 1 \\ c & \text{if } n = 1. \end{cases}$$

This is for fact(n).

Substitution method

① Start with the recursive case.

$T(n) = T(n-1) + c$

② $T(n-1) = T(n-2) + c.$

$\therefore T(n) = [T(n-2) + c] + c$

$= T(n-2) + 2c$

But why it is

↑ But why?

The constant should be different? No they should not be.

$T(n) = T(n-3) + 3c$ --> Why?

$= T(n-4) + 4c$

:

$T(n) = T(n-k) + kc$

What the constants are made up of? [multiplication + return] operation.

Let us assume fact(n-k) is the last recursive call.

∴ $T(n-K)$ must satisfy the base case

$T(n-K) = T(1) = C$.

∴ $n-K=1 \Rightarrow \boxed{K = n-1}$

∴ $T(n) = T(n-(n-1)) + (n-1)\cdot C$.

$T(n) = T(1) + (n-1)C$ ... (1)

∴ $T(n) = C + (n-1)C = nC$.

$\boxed{T(n) = nC}$

∴ In terms of asymptotic notation,

$\boxed{T(n) = O(n)}$

- How to solve Recurrence relation of Return value using substitution method.

```
Algo fact(n) {
    if n==1
        return 1
    else
        return n x fact(n-1)
}
```

- We learnt how to write recurrence relation of time.
- Here we will see how to write recurrence relation for ret value.

Recurrence relation → Describes cost of the overall problem in terms of cost of smaller sub problem.

What are the smaller subproblems

Base [if n == 1 return 1] subproblem 1 — R(1) = 1

Recursive [else return n x fact(n-1).] Subproblem 2.

Let R(n) = Ret value for fact(n).

① R(1) = Ret value for fact(1). [for n = 1] Subproblem 1

R(1) = 1

② Subproblem 2

return n x fact(n-1).

↓ ↓

n x R(n-1).

$$R(n) = \begin{cases} n \times R(n-1) & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Now how to solve the recurrence relation by substitution method.

$$R(n) = n \times R(n-1) \quad \text{---} \ 1$$

$n = n-1$

$\Rightarrow R(n-1) = (n-1) \times R(n-2).$

$\therefore R(n) = n \times (n-1) \times R(n-2) \quad \text{---} \ 2$

~~Continue this up to k~~

$R(n) = R(n-3) \times n \times (n-1) \times (n-2) \cdots 3.$

Rewriting this

$R(n) = R(n-3) \times (n-2) \times (n-1) \times n \cdots 3.$

Continue this up to K.

$$R(n) = R(n-K) \times (n-(K-1)) \times (n-(K-2)) \times \cdots \times n.$$

$R(n-K)$ = Represents the ret value of fact $(n-K)$.

Let's assume

fact $(n-K)$ = last recursive call. = base case must be satisfied.

$n - K = 1$

$\Rightarrow K = n-1$

now replace k in that expression

$R(n) = R(n-n+1) \times [n+1-n+1] \times [n+2-n+1] \cdots \times n.$

$= R(1) \times (2 \times 3 \times \cdots n)$

$= 1 \times 2 \times 3 \times \cdots n \quad [\because R(1) = 1]$

$= n!$

$\therefore \boxed{R(n) = n!}$ → And this is what the Algorithm is about.

How to solve recurrence relations of multiplication using substitution method

- Write recurrence relation of multiplication.
- Solve n ! using substitution method.

(b) Algo fact(n) {
    if n == 1
        return 1
    else
        return n x fact(n-1).
}

- Here we want to know how many multiplication are needed to compute n!

Recurrence relation = cost of problem in terms of cost of smaller subproblems

Here the cost is in terms of multiplication.

Let M(n) = cost of fact(n).

Base case -> 0 multiplication
(if n == 1
  return 1.) -> no multiplication op.

∴ M(1) = 0.

M(n) = M(n-1) + 1
↓ for fact(n-1)
This multiplication is for n x fact(n-1)
This 1 multiplication operation.

Recurrence relation

$$M(n) = \begin{cases} M(n-1) + 1 & \text{if } n > 1 \\ 0 & \text{if } n = 1 \end{cases}$$ ← this is for M(1)

Now how to solve the recurrence relation via substitution.

(i) first write the recursive part first

$M(n) = M(n-1) + 1$

(a) $M(n) = [M(n-2)+1] + 1 = M(n-2) + 2$

(b) $M(n) = M(n-3) + 3.$

⋮

for K iterations

Let $M(n-K) = M(1)$.

$\Rightarrow n - K = 1$

$\boxed{K = n-1}$.

(c) $M(n) = M(n-K) + K.$

↓

no. of multiplication required to compute fact $(n-K)$. $\Rightarrow$ Let us consider this is the last recursive call.

(d) $M(n) = M(n-(n-1)) + K = M(1) + K = 0 + K = K.$

$\therefore \boxed{M(n) = (n-1)}$

$\therefore$ In order to compute fact(n), a total of $(n-1)$ multiplications are needed.

$\therefore \boxed{M(n) = O(n)}$ → linear.

Verify

~~fact~~ fact(5) = 5! = $5 \times 4 \times 3 \times 2 \times 1$

1 2 3 4.

Total = (5-1) multiplications.

Solved Problem-1

(1) Determine the TC of the following algorithm.

```
Algo rec(n)
{
    if n==1                    ]  Sub Prob 1
        return 1               ]
    else return 2 x rec(n-1) + n.  ]  Sub Prob 2
}.
```

$T(n)$ ⟶ time that it takes to solve ⟶ rec(n).

$T(n-1)$ ⟶ n ⟶ rec(n-1).

$$T(n) = \begin{cases} T(n-1) + C & \text{if } n > 1 \\ T(1) & \text{if } n = 1. \end{cases}$$

C = Constant time = (Ret) + (multiply) + (Add)

$T(n) = T(n-1) + C$

$= (T(n-2) + C) + C = T(n-2) + 2C$

⋮

$T(n) = T(n-K) + KC$

$n - K = 1$

$K = n - 1.$

$T(n-K)$ — it reaches the base case

$\therefore T(n) = T(1) + (n-1)C = C + (n-1)C = nC$

$\therefore T(n) = nC = O(n)$.

~~Rember~~ Remember, we have not computed the return value recurrence relation:

Solved Problem - 2

Determine the recurrence relation of the same algorithm for the O/P.

$$rec(n) = \begin{cases} 2 \times rec(n-1) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

$\therefore rec(n-1) = 2 \times rec(n-2) + (n-1)$.

$rec(n) = [2 \times rec(n-1) + n]$

$= 2 \times [2 \times rec(n-2) + (n-1)] + n$.

~~$= 4 \times rec(n-2) + 2n - 2 + n$.~~

~~$rec(n) = 4 \times rec(n-2) + 3n - 2$~~

~~What is it~~ ~~What is~~

What is required here?

-> come up with a sustainable pattern. for all iteration.

of me

$rec(n) = 4 \times rec(n-2) + 3n - 2$ —> I am unable to form a pattern

(126)

$R(n) = 2 R(n-1) + n \quad -(1)$.

$R(n) = 2\,[2 \times R(n-2) + (n-1)] + n$.

$= 2^2 \times R(n-2) + 2(n-1) + n$.

$R(n-2) = 2 \times R(n-3) + (n-2)$.

$= 2^2\,[2 \times R(n-3) + (n-2)] + 2(n-1) + n$.

$R(n) = 2^3 \times R(n-3) + 2^2(n-2) + 2(n-1) + 2^0 n$.

⋮

Continues for K iterations.

$= 2^K R(n-K) + 2^{K-1}(n-(K-1)) + \cdots + 2^2(n-2) + 2^1(n-1) + n$.

$R(1) = R(n-K)$

$n - K = 1$

$K = n-1$

$K-1 = n-2$

$R(n) = 2^{n-1} + 2^{n-2} \times 2 + \cdots + 2^2(n-2) + 2(n-1) + n$.

This is AP + GP combination.

$R(n) = 2^0 n + 2^1 \times (n-1) + 2^2 \times (n-2) + \cdots + 2^{n-1}$. $-(I)$

$2 \times R(n) = 2^1 n + 2^2 \times (n-1) + 2^3 \times (n-2) + \cdots + 2^n$. $-(II)$

$(II) - (I) = -n + 2 + 2^2 + \cdots + 2^3 + \cdots + 2^n$

$n - (n-1) = 1$

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

(127).

$$R(n) = \underbrace{2 + 2^2 + \cdots + 2^n} - n.$$

$$= \frac{2(2^n - 1)}{2 - 1} - n = 2^{n+1} - 2 - n.$$

$$= 2^{n+1} - (n+2)$$

$$\therefore \boxed{R(n) = 2^{n+1} - (n+2)}.$$

output.

~~Solved~~ Solved Problem - 3

Find the time complexity of the following algorithm.

```
Algo rec(n).
{
    if n==1
        return 1
    else
        rec(n-1)
        for(i=1; i<=n; i++)
            print("welcome")
}.
```

my attempt.

~~rec(n) =~~ What is the T.C of the following

```
for(i=1; i<=n; i++)
    print("...n").
```

$$TC = 1 + \frac{B - A}{C} = 1 + \frac{n-1}{1} = n.$$

$$\therefore rec(n) = rec(n-1) + n.$$

Recurrence relation

$$rec(n) = \begin{cases} rec(n-1) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1. \end{cases}$$

→ recursive case

$$rec(n) = rec(n-1) + n.$$

$$= [rec(n-2) + (n-1)] + n.$$

$$= rec(n-3) + (n-2) + (n-1) + n.$$

$$\vdots$$

$$= \underline{rec(n-K)} + (n-(K-1)) + (n-(K-2)) + \cdots + n.$$

let $n - K = 1$

$K = n - 1.$

$K - 1 = n - 2$

$n - (K-1)$

$= n - (n-2) = 2.$

$$= rec(1) + 2 + 3 + \cdots + n.$$

$$= 1 + 2 + 3 + \cdots + n$$

$$= \frac{n(n+1)}{2}$$

$\therefore TC = O(n^2).$

Instead of $rec(n)$ we use $T(n)$.

$$T(n) = \begin{cases} T(n-1) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

$T(n) = O(n^2).$ ⟶ worst case time complexity.

Problem-4

```
Algo rec(n)
{
    if n==1
        return 1
    else
        rec(n-1)
        rec(n-1).
        for(i=1; i<=n; i++)
            print("n..n").
}
```

rec(n) —> T(n) time.

$$T(n) = \begin{cases} 2T(n-1) + n. & \text{if } n > 1 \\ 1 & \text{if } n = 1. \end{cases}$$

→ Recurrence relation.

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

$= 2[2T(n-2) + (n-1)] + n.$

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

$= 2^2 [2T(n-3) + (n-2)] + 2(n-1) + n$

$T(n). = 2^3 T(n-3) + 2^2(n-2) + 2(n-1) + n.$

$\vdots$

$= 2^K T(n-K) + 2^{K-1}(n-(K-1)) + \cdots + 2^2(n-2) + 2^1(n-1) + 2^0 . n.$

Let $n-K = 1$

$\therefore K = n-1$

$T(n) = 2^{n-1} . T(1) + 2^{n-2} . 2 + 2^{n-3} . 3 + \cdots + 2^2(n-2) + 2(n-1) + n$

(P.T.O)

Writing this in reverse order.

$T(n) = 2^0 \cdot n + 2^1 \cdot (n-1) + 2^2 \cdot (n-2) + \cdots + 2^{n-1}$

$2T(n) = 2^1 \cdot n + 2^2 \cdot (n-1) + 2^3 \cdot (n-2) + \cdots + 2^n$

$T(n) = -n + 2 + 2^2 + \cdots + 2^n$

$= 2 + 2^2 + \cdots + 2^n - n$

$= \frac{2(2^n - 1)}{2-1} - n \quad = \quad 2^{n+1} - 2 - n$

$= 2^{n+1} - (n+2)$.

$\therefore \quad T(n) = \underbrace{2^{n+1}}_{\text{Exp}} - \underbrace{(n+2)}_{\text{Poly}}$

$T(n) = O(2^n)$. $\longrightarrow$ Exponential time.

Book 1 - over

Book 2 - will start from Pg - 131.