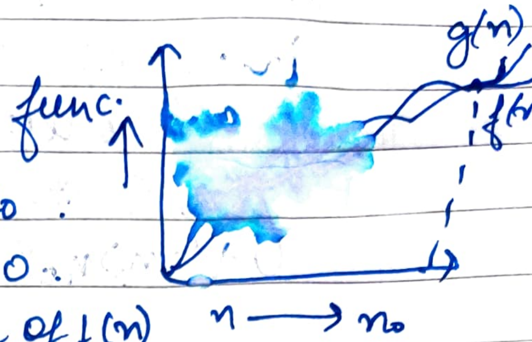
Tutorial - 01.

Q-1) (i) Big O(n)



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

if $f(n) \leq g(n) \times c \;\forall\; n \geq n_0$.

for same constant, $c > 0$.

g(n) is 'tight' upper bound of f(n)

eg:

(ii) Big omega (Ω)

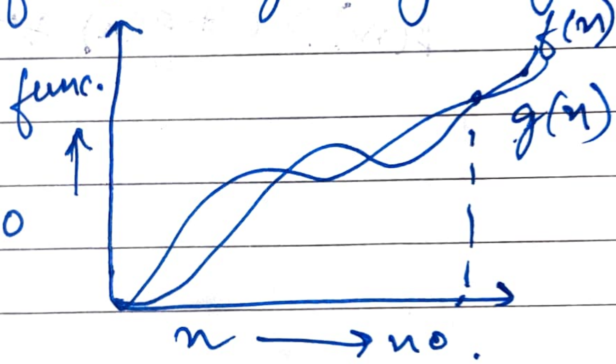
when $f(n) = \Omega(g(n))$ means g(n) is 'tight' lowerbound of f(n) i.e. f(n) can go beyond g(n)

i.e $f(n) = \Omega g(n)$

if and only if

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

$\forall\; n \geq n_0$ & c = constant > 0



(iii) Big Theta (Θ)

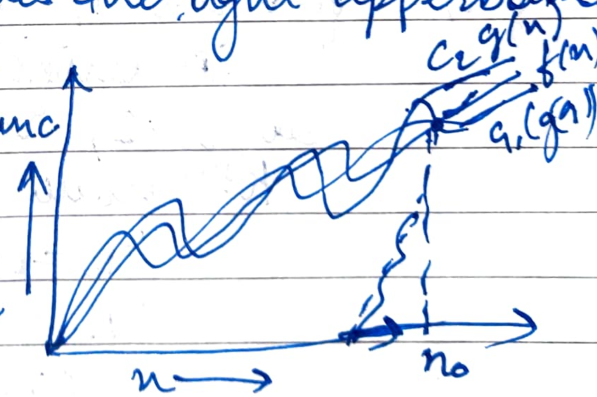
when $f(n) = \Theta(g(n))$ gives the tight upperbound and lowerbound both.

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

if and only if

$c_1 * g(n_1) \leq f(n) \leq c_2 * g(n_2)$

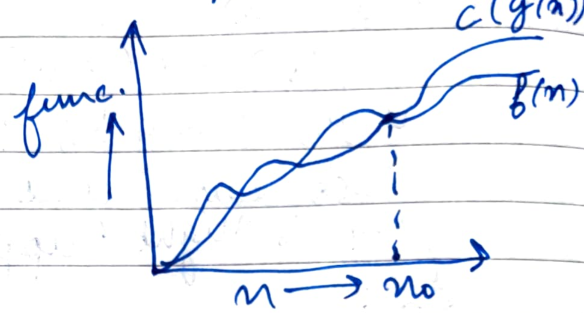
for all $n \geq \max(n_1, n_2)$, some constant $c_1 > 0$ & $c_2 > 0$



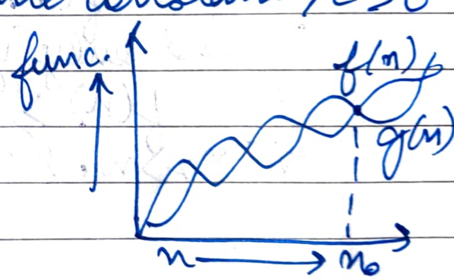
i.e. f(n) can never go beyond $c_2 g(n)$ & will never come down of $c_1 g(n)$

PAGE
DATE

(iv) Small o(θ)
when $f(n) = Og(n)$ gives the upper bound.
i.e. $f(n) = Og(n)$
and only if
$) < c * g(n)$
$0 \forall n > n_0$ & $n > 0$



(v) Small omega (ω)
It gives the 'lower bound' ie. $f(n) = \omega(g(n))$.
where $g(n)$ is lower bound of $f(n)$ if and only if
$f(n) > c * g(n) \forall n > n_0$ & some constant, $c > 0$.



Q-2) for $i \Rightarrow 1, 2, 4, 6, 8 ---$ n times.
ie series. is a G.P.
So $a = 1$, $r = 2/1$.
Kth value of G.P:
$t_K = ar^{k-1}$
$t_K = 1(2)^{k-1}$
$2n = 2^K$.
$\log_n (2n) = K\log 2$
$\log_2 2 + \log_2 n = k \Rightarrow \log_2 n + 1 = k$ (neglecting 1)
So, time complexity $T(n) \Rightarrow O(\log_2 n)$. - Ans.

DATE

Q4) $T(n) = \{ 2T(n-1)-1$ if $n>0$,
otherwise 1
}

$T(n) = 2T(n-1)-1 \longrightarrow (1)$
put $n = n-1$.
$T(n-1) = 2T(n-2)-1 \longrightarrow (2)$
put in (1)
$T(n) = 2\times(2T(n-2)-1)-1 \Rightarrow 4T(n-2)-2-1 — (3)$
put $n = n-2$ in (1)

$T(n-2) = 2T(n-3)-1$.
put in (1)
$T(n) = 8T(n-3)-4-2-1 \longrightarrow (4)$

<u>kth term</u>

let $n-k=1 \Rightarrow k=n-1$.

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

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

in series in GP
$a = 1/2$, $r = 1/2$

so

$T(n) = 2^{n-1}\left(1-\left(\frac{1/2(1-(1/2)^{n-1})}{1-1/2}\right)\right)$

$= 2^{n-1}\left(1-1+(1/2)^{n-1}\right)$

$= \frac{2^{n-1}}{2^{n-1}} \Rightarrow T(n) = O(1)$ Ans.

Q-6) Time complexity of :

↳ As $i^2 = n$

$i = \sqrt{n}$

$i = 1, 2, 3, 4, \ldots \sqrt{n}$.

$\sum_{i=1} 1+2+3+4+ \ldots + \sqrt{n}$.

$$T(n) = \frac{\sqrt{n} * (\sqrt{n}+1)}{2}$$

$$T(n) = \frac{n * \sqrt{n}}{2} \Rightarrow T(n) = O(n) \text{ Ans.}$$

Q-7) Since for $k = k^2$

$K = 1, 2, 4, 8 \ldots k$.

∴ series is in GP.

So $a = 1, r = 2$

$$\frac{a(r^n - 1)}{r-1} \Rightarrow \frac{1(2K-1)}{1}$$

$$n = 2^K - 1 \Rightarrow n+1 = 2^K \Rightarrow \log_2(n) = K.$$

i	j	K
1	log(n)	log(n) * log(n)
⋮	log(n)	log(n) * log(n)
2	log(n)	⋮
⋮	⋮	⋮
n	log(n)	log(n) * log(n)

$T.C \Rightarrow O(n * \log n * \log n)$

$\Rightarrow O(n \log^2(n)) \rightarrow$ Ans.

Q8 for (i = 1 to n)
we get j = n times every turn.
$\therefore \quad i*j = n^2$
k th,

Now,
$T(n) = n^2 + T(n-3);$
$T(n-3) = (n-3)^2 + T(n-6);$
$T(n-6) = (n-6)^2 + T(n-9);$
and $T(1) = 1$.
$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \text{-----} + 1.$
Let.

$k^n - 3k = 1.$
$k = (n-1)/3$ total terms $= k+1$.
$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \text{-- --} + 1.$
$T(n) \simeq kn^2$
$T(n) \simeq (k-1)/3 - n^2.$
So

$T(n) = O(n^3)$ Ans.

Q10 As given n^k and c^n
Relationship b/w n^k & c^n is
$n^k = O(c^n)$
$n^k \leq a(c^n)$
$\forall \; n \geq n_0$ & constant, $a > 0$
for $n_0 = 1, \; c = 2.$
$\Rightarrow \; 1^k < a2$

$\Rightarrow \; n_0 = 1$ & $c = 2$ Ans.

$$\text{max level} = \frac{n}{2^k} = 1$$

$$= k = \log_2 n$$

$$T(n) = c\left(n^2 + (5/16)n^2 + (5/16)^2 n^2 + \dots + (5/16)^{\log_2 n}\right)$$

$$T(n) = Cn^2 \times 1 \times \left(\frac{1-(5/16)^{\log n}}{1-(5/16)}\right)$$

$$T(n) = Cn^2 \times \frac{4}{5} \times \left(1-(5/16)^{\log n}\right)$$

$$T(n) = O(n^2 c)$$

$O(cn^2)$ Ans.

Q-5) → for

→ for

i	j
1	1
2	1+3+5
3	1+4+7
⋮	1+5+9
n	

$j = (n-1)/i$ times.

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

$$\because T(n) = \frac{(n-1)}{1} + \frac{(n-1)}{2} + \frac{(n-1)}{3} + \dots + \frac{(n-1)}{n}.$$

$$T(n) = n[1 + 1/2 + 1/3 + \dots + 1/n] - 1 \times [1 + 1/2 + 1/3 + \dots + 1/n]$$

$$\Rightarrow n\log n - \log n$$

$T(n) = O(n \log n) \longrightarrow$ Ans.

Q-6)

→ for

i

2^1

2^k

2^{k^2}

2^{k^3}

⋮

2^{k^m}

where
$2^{k^m} <= n$
$k^m = \log_2 n$
$m = \log_k \log_2 n$

$\therefore \sum_{i=1}^{m} 1$.

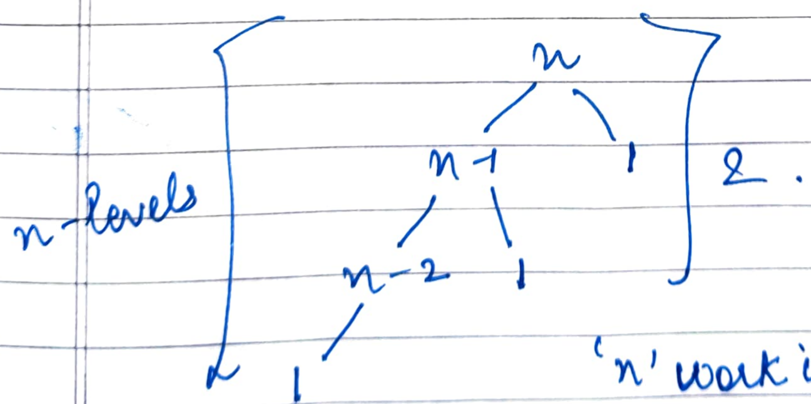
$1 + 1 + 1 - - - m$ times.

$T(n) = O(\log_k \log n) \longrightarrow$ Ans.

Q-7)

→ Given algorithm divides array in 99% & 1% part -

$\therefore T(n) = T(n-1) + O(1)$



'n' work is done at each level..

$T(n) = (T(n-1) + T(n-2) + - - - + T(1) + O(1)) * n$
$= n \times n$

$\therefore \boxed{T(n) = O(n^2)}$

DATE

lowest height $= 2$.

highest height $= n$.

$\boxed{\therefore \text{difference} = n-2}$ as $n>1$

The given algo. produces linear result.