Date
Page

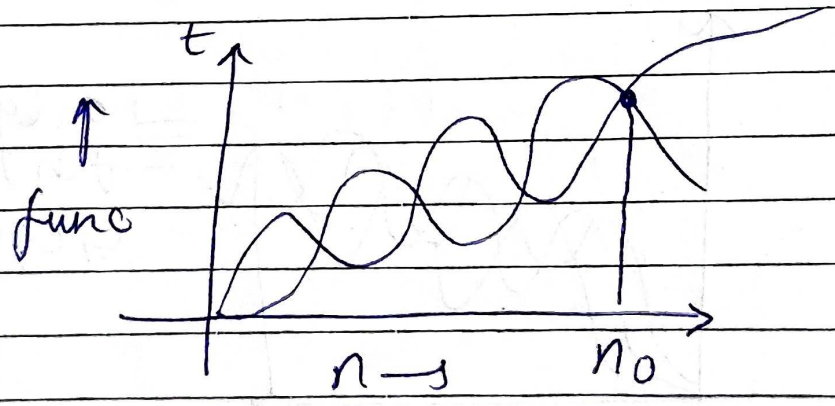
Q1) **Asymptotic notations** :- Asymptotic means tending to infinity or is used for large numbers of size. These are used to tell the complexities of an algorithm when input is very large.

Types < Time complexity.
Space complexity.

Space complexity – any extra space that we take apart from the input. We do not consider the variable as extra space as it is constant if it comes in relevance to 'n'. Then only it is extra space.

(1) **Big Oh (O)** :-

let $f(n) = O(g(n))$
iff $f(n) \leq c\,(g(n))$
$n \geq n_0$, $c > 0$
also $g(n)$ is tight upper bound of $f(n)$.



Date
Page

(2) Big omega (Ω)

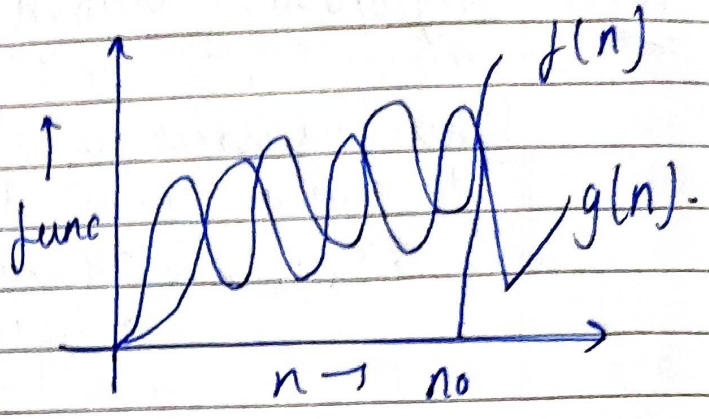
$f(n) = \Omega(g(n))$

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

if and only if

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

for all $n \geq n_0$ and some $c > 0$.



(3) Theta (Θ) :-

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

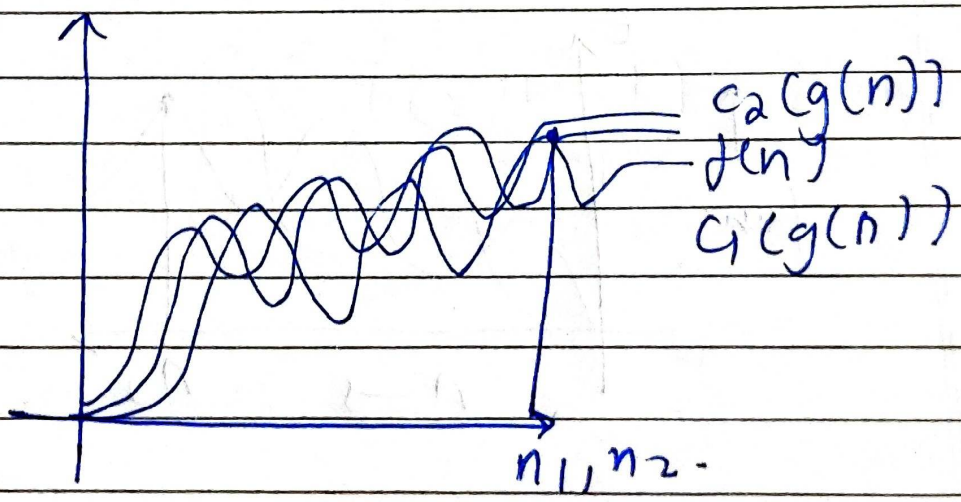
{exact bound}

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

$\forall n > \max(n_1, n_2)$

and constants $c_1 > 0$

$c_2 > 0$.

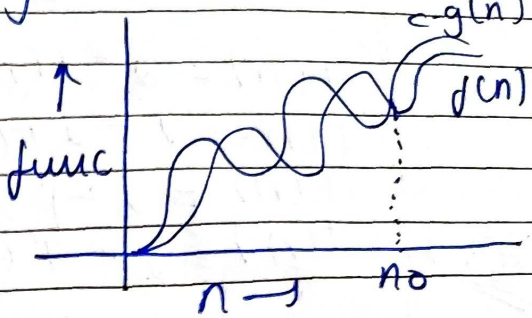


Teacher's Signature.....

Date
Page

(4) small oh (o) :-

$f(n) = g(n)$
if and only if $f(n) < c \cdot g(n)$
$\forall (n > n_0)$
$g(n)$ is upper bound
of $f(n)$
for all $c > 0$.



(5) small omega (Ω)

$f(n) = \omega(g(n))$
$f(n) = \omega(g(n))$
if $f(n) > c \cdot g \cdot g(n)$
$\forall$ $n > n_0$ for all $c > 0$

Q2 →

Q3-1 $T(n) = 3T(n-1)$ if $n > 0$, otherwise 1.

$T(1) = 0$

$T(2) = 3T(2-1)$

$= 3T(1)$

$= 3 * 0 = 0$

$T(n-1) = 3T(n-1-1)$

$= 3T(n-2) \rightarrow (2)$.

put the value of $T(n-1)$ from (2) to (1).

$T(n) = 3 \times [3T(n-2)]$

$= 9T(n-2)$

$T(n-2) = 3T(n-3)$

$= 9[3T(n-3)]$

$= 3^3[T(n-3)]$

:

:

k times.

$T(n) = 3^k T(n-k)$

put $n-k = 0$

$\boxed{n = k}$

$T(n) = 3^n T(0)$

$= 3^n (1)$

$= 3^n$

$\boxed{O(3^n)}$

Date
Page

Q4) $T(n) = [T(n-1) - 1]$ if $n > 0$ otherwise 1. → (1)

Sol:

$T(n-1) = 2T(n-2) - 1$

$T(n) = 2(2T(n-2) - 1) - 1$

$T(n) = 2^2 T(n-2) - 2 - 1$ → (2)

$T(n-2) = 2T(n-3) - 1$

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

$T(n) = 2^3 T(n-3) - 2^2 - 2 - 1$ → (3)

⋮ k times

⋮

$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - \cdots - 2^2 - 2^1 - 2^0$

$= 2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} - \cdots - 2^2 - 2^1 - 2^0$

$= 2^n - (1 + 2 + 2^2 - \cdots - 2^{n-2} + 2^{n-1})$

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

$= 2^n - 2n + 1$

$= 1$

$O(1)$

$$S = \frac{a(r^n - 1)}{r - 1} \quad r > 1$$

Page No :
Date :

Q5 →

i	
1	1
2	1+2 = 3
3	1+2+3 = 6
4	1+2+3+4 = 10
⋮	
n times	1+2+3 ..... n

Assume $j > m$

$\therefore s = (1+2+3 \cdots n)$

$= \frac{m(m+1)}{2}$

$\frac{m(m+1)}{2} > n$

$m^2 > n$

$m = \sqrt{n}$

$O(\sqrt{n})$

Q6 →

i	i * i > n
1^2	i * i = k^2
2^2	$k^2 > n$
3^2	$k > \sqrt{n}$
4^2	$O(\sqrt{n})$
⋮	
k^2	

Q7→

```
void function (int n)
{
 int i, j, k, count=0;
 for (i=n/2 ; i<=n ; i++)  // log n
  {
    for (j=1 ; j<=n ; j=j*2) // log n x log n
     {
       for(k=1 ; k<=n ; k=k+2) // log n
          {
            count ++ ;
          }
     }
  }
```

$O(\log^3 n)$

Q8→

```
function ( int n)
 if (n==1) → 1
   return ;
 for (i=1 to n) → n+1
    { for (j=1 to n)  n(n+1)
        { print("*");
        }
    }
 func(n-3)
 }
```

$T(n) = T(n-3) + n^2 + n-1 \; n+1 + 1$

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

$$\left[T(n) = aT(n-b) + f(n) \quad a \geq 0 \quad f(n) = O(n^k) \right]$$

where $k \geq 0$

$a = 1$

$b = 3$

$f(n) = n^2 = O(n^2)$

$k = 2$

case 2: if $a = 1$

$O(n^{k+1})$

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

$\boxed{O(n^3)}$

---

(9)

i	J
1	1 $= 1+1$
2	3 $= 1+2+1$
3	6 $= 1+2+3+10$.
4	10
:	
:	
n times	

assume $J > n$

$J = 1+2+3 \; ---- + n \qquad J = \frac{n(n+1)}{2}$

$n^2 > n$

$n > \sqrt{n}$

$\boxed{O(n^{3/2})}$

(10) If k is a constant & $c > 1$
Then,
c^n grows faster than n^k as $n \to \infty$
because;
rate of growth of c^n is exponential
while rate of growth of n^k is polynomial

Or we can say $O = \boxed{O(n^k)}$.

(9)