Thursday

Tutorial - 1

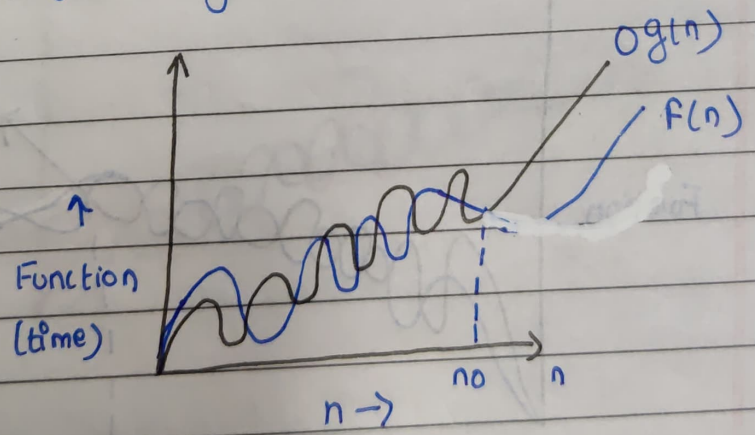
DAA

Q1) What do you understand by Asymptotic notations. Define different Asymptotic notations with examples.

Asymptotic notation is used to ~~desc~~ describe the running time of an algorithm. How much time an algorithm takes with a given input n. There are three different notation : big O, big theta (Θ), big omega (Ω).

→ Big O

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



Size of input / size of element.

iff

$f(n) \leq g(n)$

$\forall \; n \geq n_0$

for some constant, $c > 0$.

g(n) is "tight" upper boond of f(n)

2. Big Omega (Ω)

$F(n) = \Omega\, g(n)$

g(n) is 'tight" lower boond of fonction f(x)

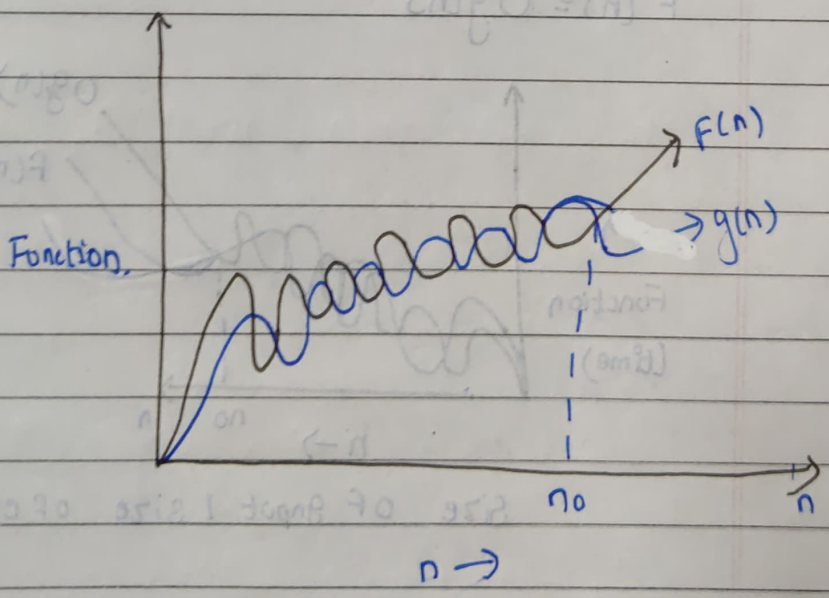
$f(n) = \Omega\, g(n)$

iff

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

$\forall \; n \geq n_0$

for some constant $c > 0$.



3⇒ Theta (Θ)

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

g(n) is both 'tight' upper and lower bound of function f(n)

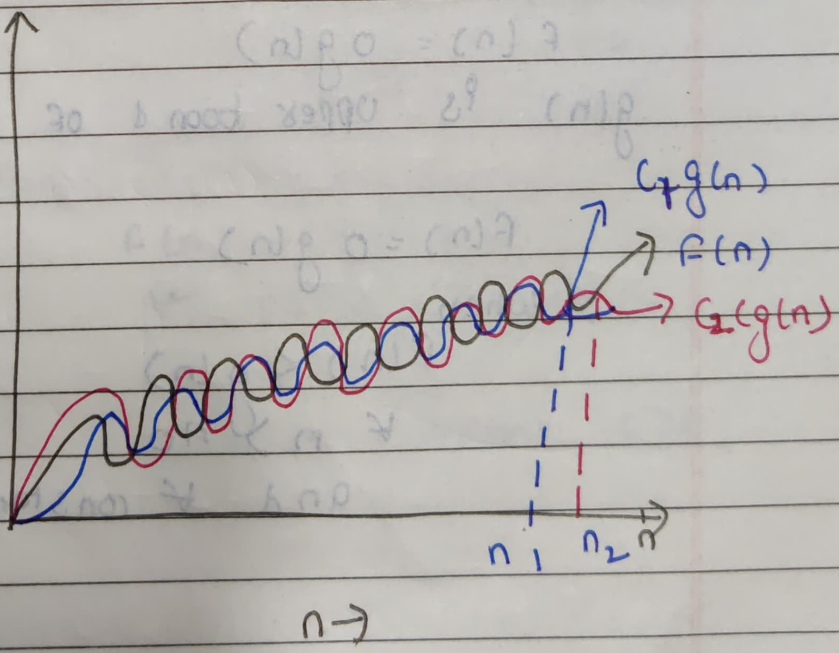
$f(n) = \Theta(g(n))$

iff

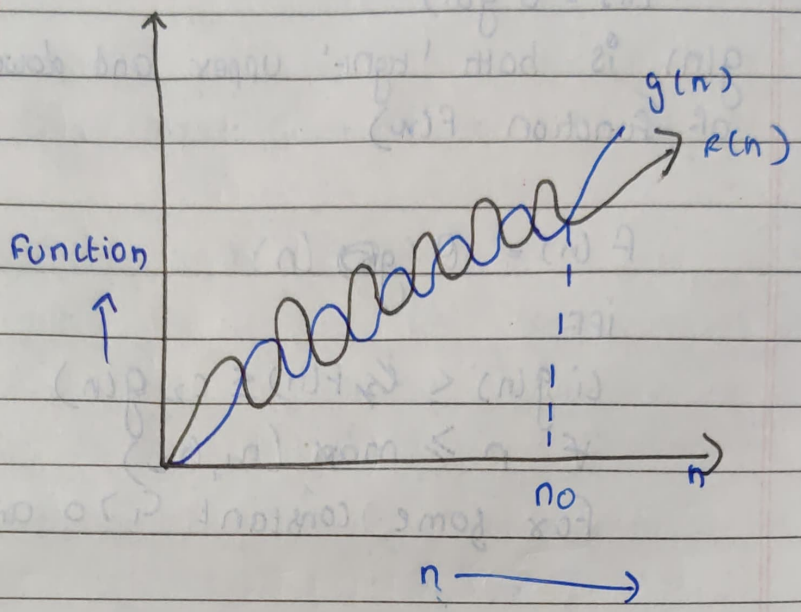
$c_1 g(n) \leq f(n) \leq c_2 g(n)$

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

for some constant $c_1 > 0$ and $c_2 > 0$



4. Small o (o)



$f(n) = o\,g(n)$

$g(n)$ is upper bound of function.

$f(n) = o\,g(n)$

when

$f(n) < g(n)$

$\forall\ n > n_0$

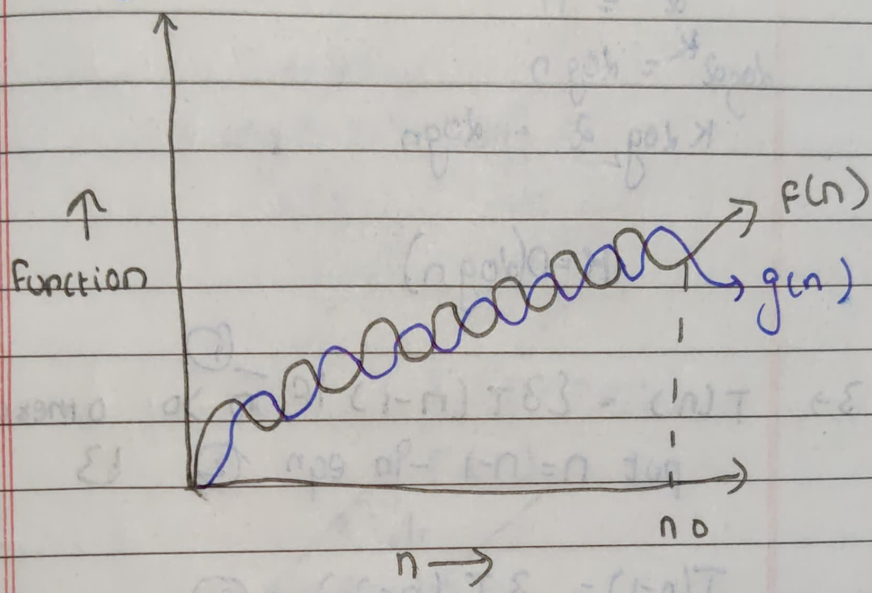
and $\forall$ constant, $c > 0$.

Date
Page

=> Small omega (ω)

$f(n) = \omega\, g(n)$

$g(n)$ is lowest bound of function $f(n)$



$f(n) = \omega\, g(n)$

$\forall\ f(n) > c\, g(n)$

$\forall\ n > n_0$

$\forall$ constant, $c > 0$

2. for (i = 1 to n)
{ i = i * 2 }

1, 2, 4, 8, ... n

$2^k = n$

$\log 2^k = \log n$

$k \log_2 2 = \log n$

$k = O(\log n)$

3. $T(n) = \{3T(n-1)$ if $n > 0$ otherwise 1 — (1)

put $n = n-1$ in eqn (2)

$T(n-1) = 3T(n-2)$ — (3)

Put (3) in eqn (1)

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

$= 9[T(n-2)]$ — (2)

put $n = n-2$ — (1)

$T(n-2) = \{3T(n-3)\}$ — (4)

Put eqn (4) in (2)

$T(n) = 9\{3T(n-3)\}$

$= 27[T(n-3)]$ — (5)

$T(n) = 2T[T(n-3)]$ - ⑤

$K = 3$

$3 T[T(n-K)]$

$n - 1K = 1$

$n - 1 = K$

$\log(n-1) = \log K$

$\frac{\log n}{\log 1} \cdot \log K$

$3^{n-1} [T(n-n+1)$

$T(1)$

$3^{n-1} \cdot 1$

$(n-1) \log 3$

$\Rightarrow O(n-1)$

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

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

$= O(3^n)$

4-) $T(n) = \{2T(n-1) - 1$ if $n > 0$, otherwise 1. — ①

Put $n = n-1$ in eqn ①

$T(n-1) = \{2[T(n-2) - 1]$ — ③

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

$= 4T(n-2) - 2$ — ④

Put $n = n-2$

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

$T(n) = 4[2T(n-3) - 1] - 2$

$= 8T(n-3) - 4 - 2 - 1$

$= 8T(n-3) - 7$

$= 2^k T(n-k) - 2^k + 1$

$= 2^k T(n-k) - 2^2 - 2^1 - 2^0$

$n - k = 1$

$n - 1 = k$

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

$2^{n-1} - (2^{n-1} - 1)$

$= 2^{n-1}[(1 - (1$

$= 2^k T(n-k) - 2^{k-1} - 2^{k-2} - 2^{k-3}$

$n - k = 1$

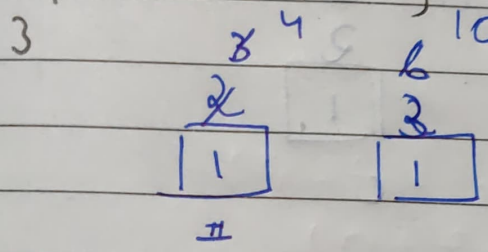
$n - 1 = k$

$= 2^{n-1} - 2^{n-2} - 2^{n-3} - 2^{n-4}$

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

5) What should be time complexity of

```
int i=1, s=1;
while (s<=n)
{
    i++;
    s=s+i;
    printf("#");
}
```



$1 + 3 + 6 + 10 \cdots K = n$

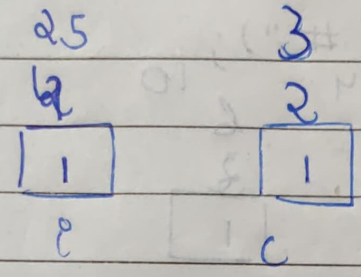
$\frac{K(K+1)(K+2)}{6} = n$

$O(K^3) = n$

$K = \sqrt[3]{n}$.

6 → Time complexity of →

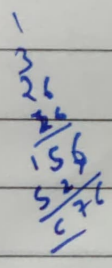
```
void function (int n)
{ int i, count=0;
  for (i=1; i*i<=n; i++)
    count++;
}
```



$1 + 4 + 25 + 676 \cdots n.$

$a = 1$

$S_n = a\left(\frac{r^n - 1}{r - 1}\right)$



3, 2, 1, 5, 1

i	j	k
$n/2$	1	1
↓	↓	2
		4
$n/2$	$\log n$	$n \to \log n$

$O(n/2 \times \log n \times \log n)$

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

Ans 8)

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

$1 + 4 + 7 \cdots n$

$n = 1 + 3(k-1)$

$= \frac{n+2}{3}$

no of terms.

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

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

$O\left[\frac{(n^2+3n+2)}{6} \times n^2\right]$

$O(n^4)$ Ans

Ans9 ->

```
void function (int n)
{ for (i=0 ton)
{
    for (j=1 ; j<=n ; j=j+1)
    {
        Printf ("* "); n^2
    }
}
```

O (n + n² + n² + n²)

O (3n² + n)

O (n²) Ans.

Q10> For the functions n^k and c^n what is the asymptotic relationship these function ? Assume $k >= 1$ and $c > 1$ are constant

Ans10 -> n^k is $O(c^n)$