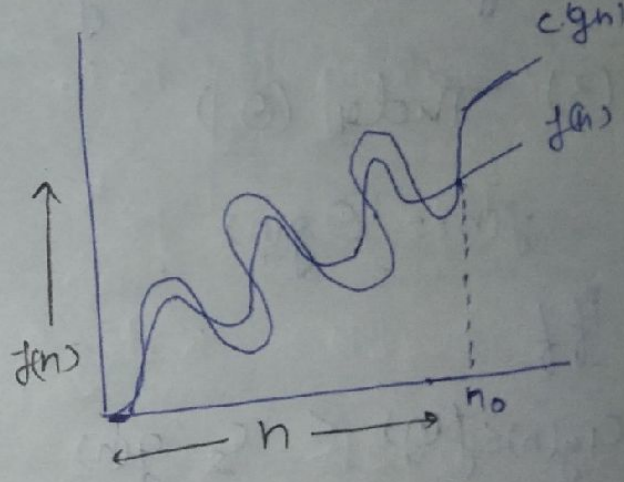
Design and Analysis of Algorithms

Tutorial- 1

(1) **Asympotic notations** : These are the mathematic representation, which helps to find the complexity of an algorithm when input is very large.

There are 5 types of different Asympotic notation

(1) **Big O (O)**:



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

iff

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

$\forall n \geq n_0$

for some constant $c, > 0$;

* $g(n)$ is "tight" upper bound of $f(n)$
* Used for worst time complexity.

(2) Big (Ω)

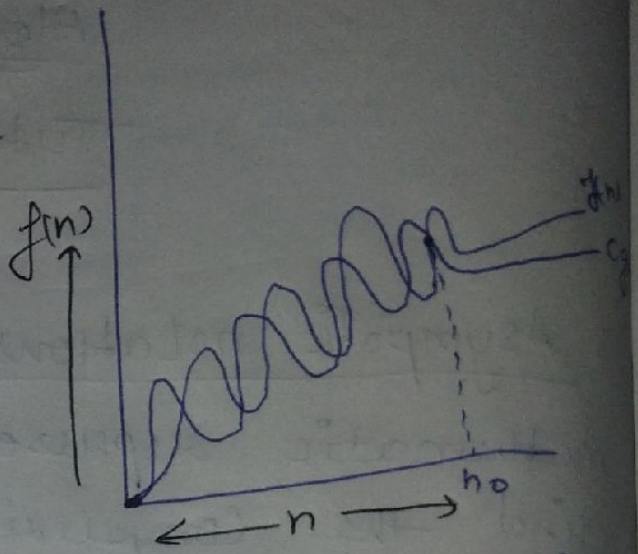
$f(n) = \Omega(g(n))$

iff

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

$\forall \; n \geq n_0$

for some constant, $c > 0$.



* used for Best time Complexity

* $g(n)$ is "tight" lower bound of function $f(n)$

(3) Theta (Θ)

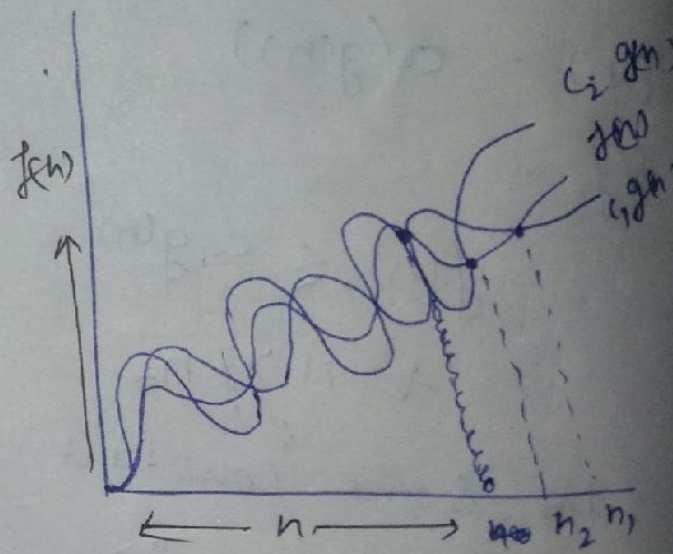
$f(n) = \Theta g(n)$

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 $c_2 > 0$.



→ Average time Complexity

→ Exact time

4. Small o(o)

$f_n = o(g(n))$

$g(n)$ is upper bound of fxn $f(n)$

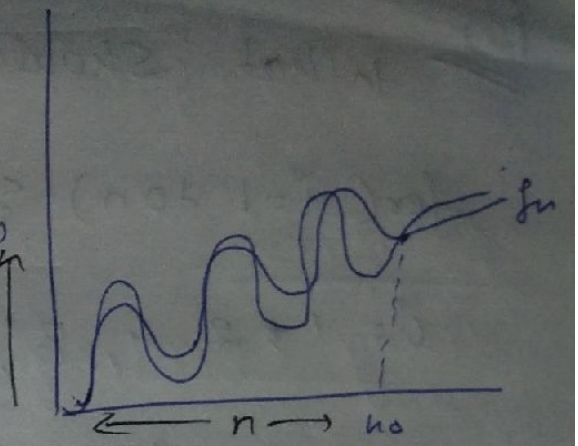
$f(n) = o(g(n))$

when

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

$\forall \; n > n_0$

$c \cdot g(n)$ and $\forall$ Constant, $c > 0$



5. Small omega (ω):

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

$g(n)$ is lower bound of function

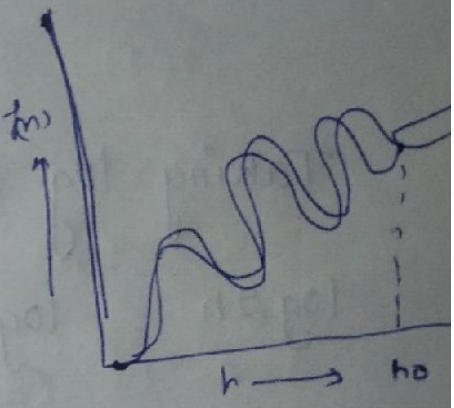
$f(n) = \omega(g(n))$

when

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

$\forall \; n > n_0$

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



Q2 What should be time complexity of -

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

i = 1 * 2, 4, 8, 16 ... n

$a = 1, r = \frac{t_1}{t_2} = 2$

GP k^{th} value = $t_k = ar^{k-1}$

$n = 1 \times 2^{k-1}$

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

$2n = 2^k$

Taking log both side

$\log 2n = \log 2^k$ [$\because \log a^b = b \log a$]

$\log 2k = k \log 2$

$\log 2k = k$ [$\log ab = \log a + \log b$]

$\log 2 + \log n = k$

$1 + \log n = k$ [$\because k = 1$]

since

$= O(1 + \log_2 n)$

$= O(\log_2 n)$ is the time Complexity.

(3) $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

Solution :-

Using here Substitution metho (Back Substitution)

$T(n) = 3T(n-1)$ —(1)

Put $n = n-1$

$T(n-1) = 3T(n-2)$ —(2)

Putting the value of $(n-1)$ in eqn (1)

$T(n) = 3^2 T(n-2)$ —(3)

Now put $n = (n-2)$

$T(n-2) = 3T(n-3)$ —(4)

Putting $(n-2)$'s value in eqn (3)

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

⋮

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

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

$= 3^n$

Therefore, the complexity of this fun is $O(3^n)$.

(4) $T_n = \begin{cases} 2T(n-1)-1 & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

Solution:

Using here forward Substitution meth

$T(0) = 1$

$T(1) = 2T(0) - 1 = 1$

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

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

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

...

So $T(n) = 1$ for any value of n

i.e it remains constant.

time Complexity = $O(1)$

(05) what should be time complexity of

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

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

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

$= O(k^3) = n$

$O(\sqrt{n})$

Q6 Time Complexity of

```
Void function (int n) {
    int i, j, k ; count = 0;
    for (i = n/2 ; i <= n ; i++) ---> O(n)
        for (j = 1 ; j <= n ; j = j * 2) ---> O(log n)
            for (k = 1 ; k <= n ; k = k * 2 ---> O(log n)
                count ++
}:
```

Time complexity = $O(n) * O(\log n \times \log n)$

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

Q7 Time complexity of

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

the time complexity of above function is $O(n)$.

Q8 Time complexity of

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

Time complexity = $O(n \times n)$

$= O(n^2)$ Ans

Q9 Time complexity of

```
void function (int n)
    for (i=1 to n) {          ← O(n)
        for (j=1 ; j<=n ; j=j+1)   O(n)
            Print ("*");
```

$$\sum_{j=n} \; n + n/2 + n/3 + n/4 + \cdots + 1$$

$$\sum_{j=n} \; n\,[\,1 + 1/2 + 1/3 + 1/4 + \cdots 1/n\,]$$

$$\sum_{j=n} \; n\,[\log n]$$

$$T(n) = [n \log n]$$

$$T(n) = O(n \log n)$$

Q10 for function, n^k and c^n, what is the asymptotic relation...

as given n^k and c^n

relation b/w n^n and c^n is

$$n^k = O(c^n)$$

as $n^n \leq O(c^n)$,

$\forall n \geq n_0$ and some const $a > 0$.

for $n_0 = 1$

$c = 2$.

$\Rightarrow 1^k \leq a_2 1$

$n_0 = 1$ and $c = 2$.