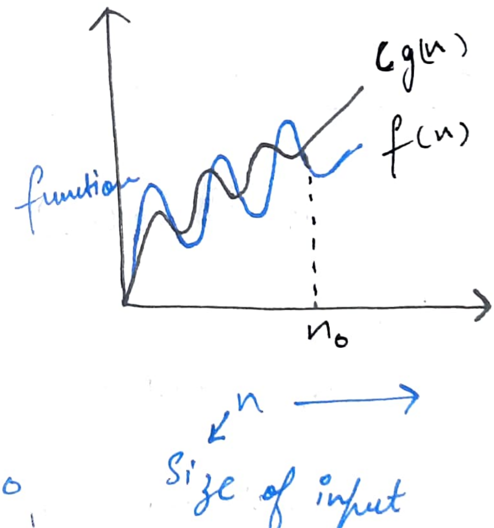
Q.1 What do you understand by Asymptotic notations. Define different Asymptotic notation with examples:

Sol They help us to find the complexity of an algorithm when input is very large.

① Big O (O) :-

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

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



iff

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

$\forall \; n \geq n_0$

for some constant, $c > 0$

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

② Big Omega (Ω) :- $f(n) = \Omega(g(n))$

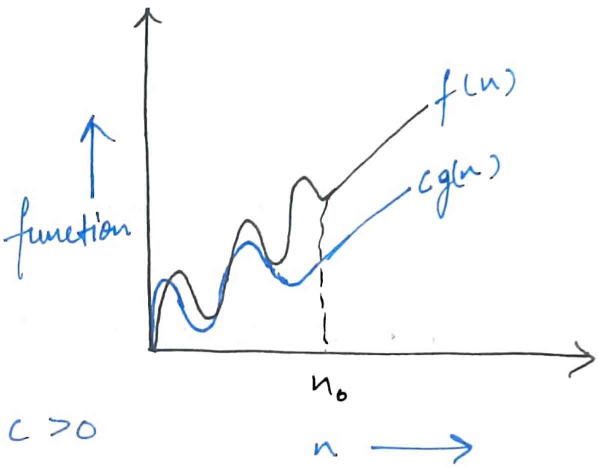
$f(n) = \Omega(g(n))$

iff

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

$\forall\ n \geq n_0$

for some constant, $c > 0$



g(n) is "tight" lower bond of function f(n)

③ Theta (Θ) :- $f(n) = \Theta(g(n))$

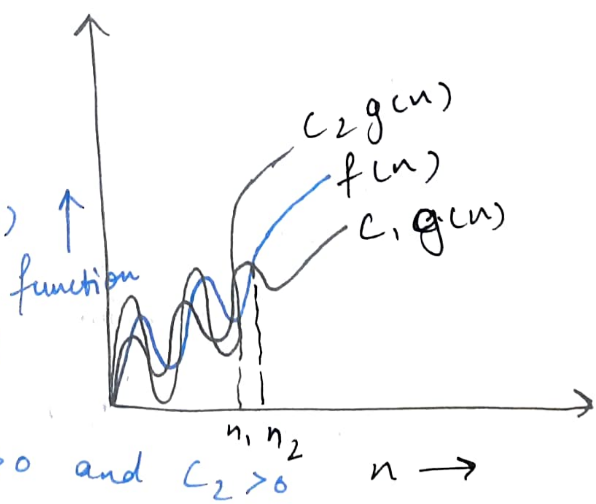
$f(n) = \Theta(g(n))$

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 $c_2 > 0$



g(n) is both "tight" upper and lower bond of function f(n).

(4) Small o (o) :- $f(n) = o(g(n))$

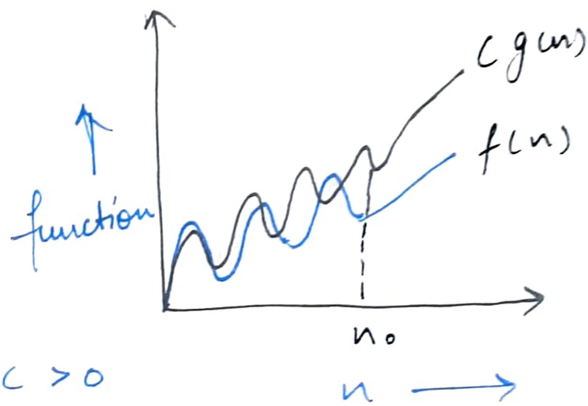
$f(n) = o(g(n))$

when

$f(n) < g(n)$

$\forall\ n > n_0$

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



g(n) is upper bond of function f(n).

(5) Small Omega (ω) :- $f(n) = \omega(g(n))$

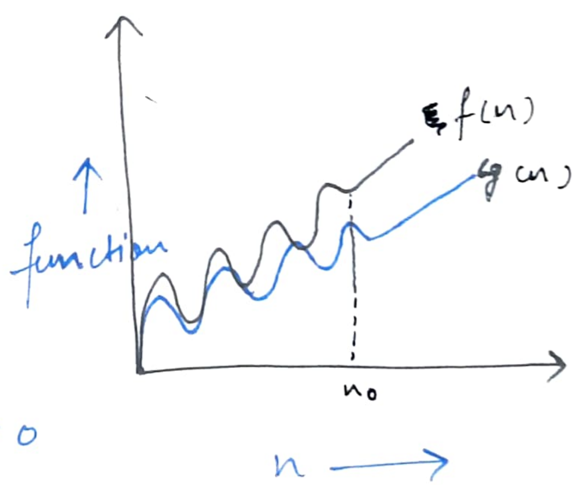
$f(n) = \omega(g(n))$

When

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

$\forall\ n > n_0$

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



g(n) is lower bond of function f(n).

Q.2 What should be time complexity of -

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

Sol for $n = 1$, i will be from 1 to 1, $\Rightarrow$ 1 time

for $n = 2$, i = 2 times

upto n time $\Rightarrow (\log_2 n + 1)$ times

$\Rightarrow O(\log_2 n + 1)$

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

Sol $T(n) = 3T(n-1)$ —— (1)

Put $n = n-1$

$T(n-1) = 3T(n-2)$ —— (2)

Put in eq.(1)

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

$T(n) = 9T(n-2)$ —— (3)

Put $n = n-2$ in eq.(1)

$T(n-2) = 3T(n-3)$ —— (4)

Put in eq. 3

$$T(n) = 27T(n-3) \quad — 5$$

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

$T(0) = 1$

Put $n-K=0$

$n = K$

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

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

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

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

<u>Q-4</u> $T(n) = \{2T(n-1)-1$ if $n>0$, otherwise $1\}$

<u>Sol</u>

$$T(n) = 2T(n-1)-1 \quad — 1$$

Put $n = n-1$

$$T(n-1) = 2T(n-2)-1 \quad — 2$$

Put in eq. 1

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

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

$$T(n) = 2^2 T(n-2)-3 \quad — 3$$

Put $n = n-2$

$$T(n-2) = 2T(n-3) - 1 \quad \text{--- } (4)$$

Put ~~in~~ eq. (3)

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

$$T(n) = 8T(n-3) - 4 - 2 - 2^0 \quad \text{--- } (5)$$

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

Put $n - K = 0$
$n = K$

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

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

$$= 2^n - [1 + 2^1 + 2^2 + \cdots + 2^{n-1}]$$

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

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

$$T(n) = O(1)$$

Q-5 what should be time complexity of -

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

Sol

for $i=2, \quad s=3 \rightarrow$ After 1'st iteration

$i=3, \quad s=6 \rightarrow$ After 2nd iteration

$i=4, \quad s=10 \rightarrow$ After 3rd iteration

|

$K \qquad \frac{K(K+1)}{2} \rightarrow$ when $i=K$,

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

$$\frac{K^2+K}{2} <= n$$

$$O(K^2) <= n$$

$$K = O(\sqrt{n})$$

$$\Rightarrow \quad T(n) = O(\sqrt{n})$$

Q.6 Time Complexity of -

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

Sol

for $n = 1$, $i = 1$ time

for $n = 2$, $i = 1$ time

$i = 2$ times

for $n = 4$,

$i = 3$ times

for $n = 9$,

$= \sqrt{n}$ times

for $n = n$

$$\sum_{i=1}^{n} 1 + 1 + 1 + 2 + \cdots \sqrt{n} \text{ times}$$

$\therefore O(\sqrt{n})$

$T(n) = O(\sqrt{n})$ or $O(n^{1/2})$

Q.7 Time Complexity of -

```
Void function (int n)
{
  int i, j, K, count = 0;
  for (i = n/2 ; i <= n; i++)
   for (j = 1 ; j <= n ; j = j * 2)
    for (K = 1 ; K <= n ; K = K * 2)
       count ++;
 }
```

Sol for $n = 2$, i = 2 times

for $n = 16$, i = 9 times

for outer loop = $\left(\frac{n}{2} + 1\right)$ times.

for both inner loops = $(\log_2 n)$ times

$T(n) = O(i * j * K)$

$= O\left(\left(\frac{n}{2} + 1\right) * (\log_2 n) * (\log_2 n)\right)$

$T(n) = O\left[n(\log_2 n)^2\right]$

Q-8 Time complexity of.

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

Sol

$T(n) = T(n-3) + n^2$ — (1)

$T(1) = 1$

Put $n = n-3$ in (1)

$T(n-3) = T(n-6) + (n-3)^2$ — (2)

Put $n = n-6$ in (1)

$T(n-6) = T(n-9) + (n-6)^2$ — (3)

Putting $T(n-9) + (n-6)^2$

$\Rightarrow T(n) = T(n-6) + (n-3)^2 + n^2$ — (4)

Putting $T(n-6)$ in (4)

$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n^2$

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

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

$- - - - - - n^2$

putting $n - 3K = 1$

$$n = 1 + 3K$$

$$K = \frac{n-1}{3}$$

$$T(n) = T(1) + (1+3)^2 + (1+6)^2 - - - - n^2$$

$$= 1 + 4^2 + 7^2 + - - - - n^2$$

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

Q.9 Time Complexity of -

```
void function (int n)
{
  for (i = 1 to n) {
    for (j = 1; j <= n; j = j+i)
        printf (" * ");
  }
}
```

Sol

for $i = 1$, $j = 1, 2, 3, \ldots\ldots n$

for $i = 2$, $j = 1, 3, 5, \ldots\ldots n/2$

for $i = 3$, $j = 1, 4, 7, \ldots\ldots n/3$

⋮

$i = n$, $j = 1 + 1 + \ldots\ldots 1$

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \ldots + \frac{n}{n-1} - \log(n-1)$$

$$n\left\{1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-1}\right\} - \log(n-1)$$

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

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

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