Tutorial - 1

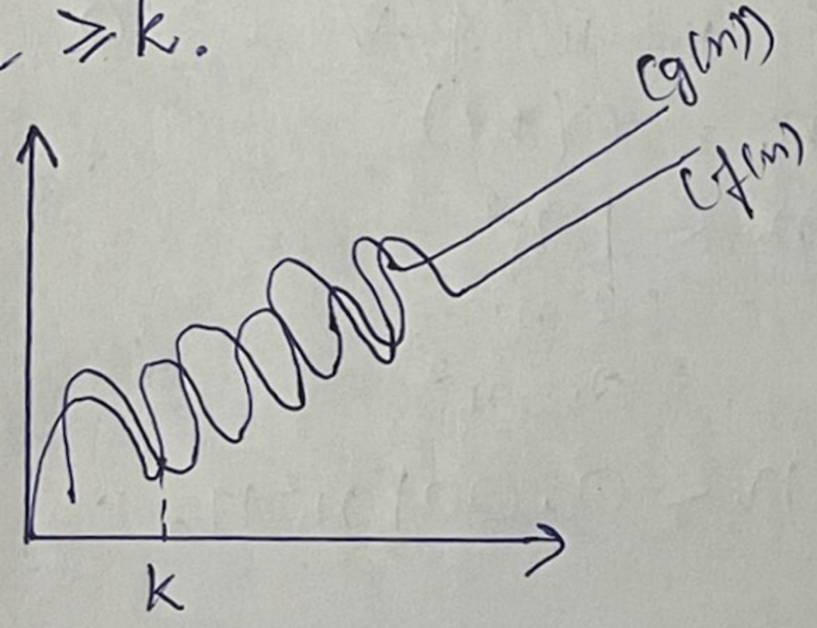
Sol-1) Asymptotic Notation →

(i) These notation are used to all the complexity of an algorithm when the input is very large.

(ii) It describes the algo. efficiency and performances in a meaningful way. It describes the behaviour of time or space complexity for large instance character is ties.

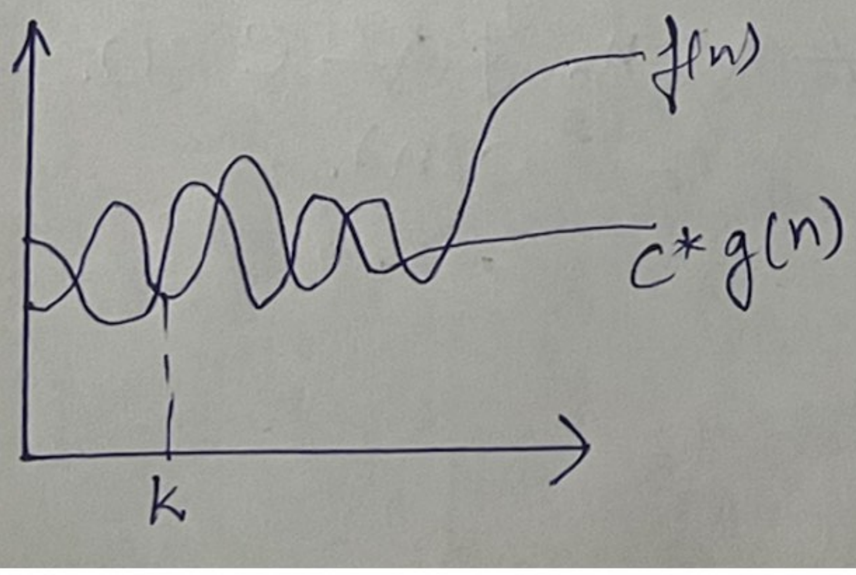
→ It is of 5 types →

(i) Big Oh notation (O) :→ (Asymptotic upper Bound) The function $f(n) = O(g(n))$, if and only if there exist a +ve constant c and k such that $f(n) \leq c * g(n)$ for all n, $n \geq k$.



$f(n) = O(g(n))$
iff
$f(n) \leq c \cdot g(n)$
$\forall\ n \geq n_0$,
Some constant, $c > 0$.

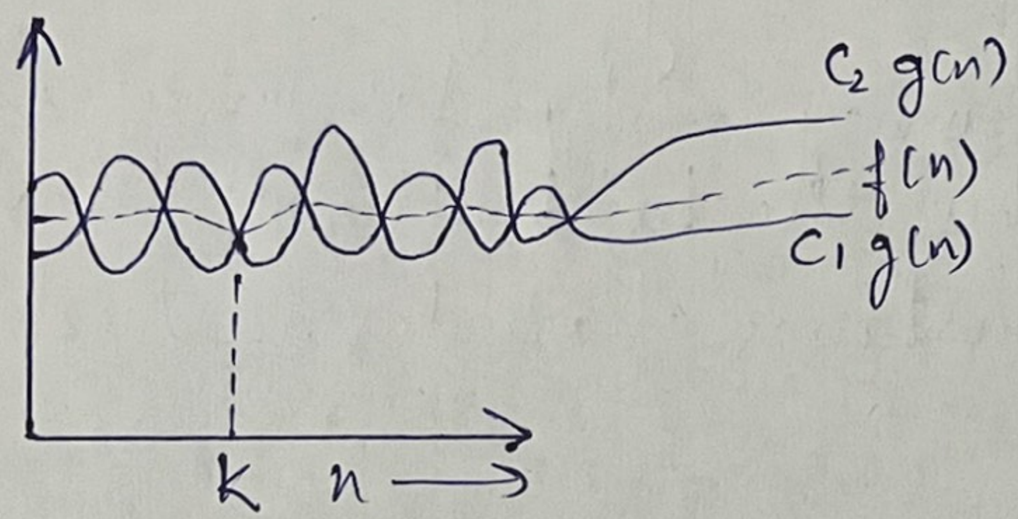
(ii) Big Omega notation (Ω) :→ (Asymptotic lower bound) The function $f(n) = \Omega(g(n))$, iff there exists a +ve constant c and k such that $f(n) \geq c * g(n)$ for all n, $n \geq k$.



$f(n) = \Omega\ g(n)$ iff
$f(n) \geq c \cdot g(n)$
$\forall\ n \geq n_0$ & Some const $c > 0$.

(iii) Big theta notation (Θ) :→ (Asymptotic tight bound) ②

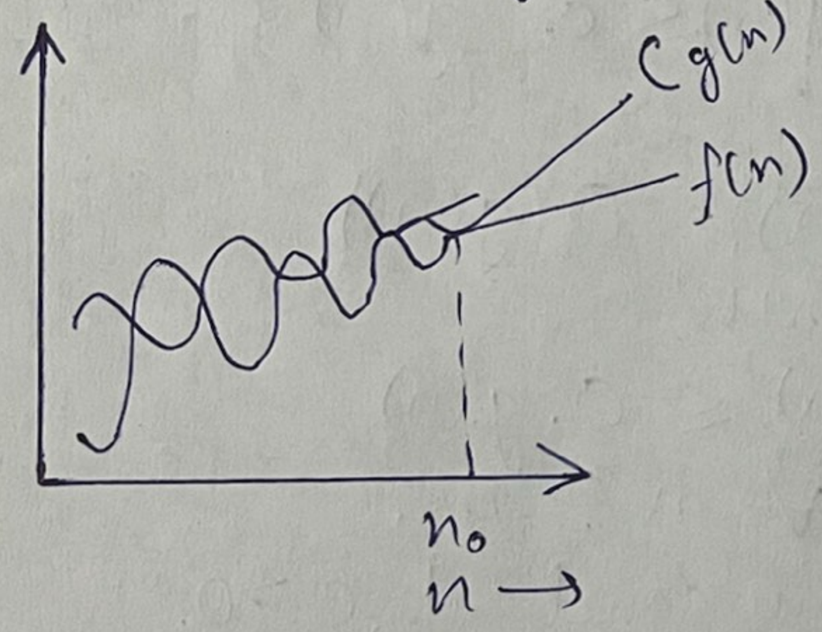
The fxn $f(n) = \Theta(g(n))$, iff there exists a +ve constant $c1, c2 \in k$ such that $c1 * g(n) < f(n) < c2 * g(n)$ for all n, $n \geq k$.



$f(n) = \Theta(g(n))$
iff
$c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\forall\ n \geq \max(n_1, n_2)$

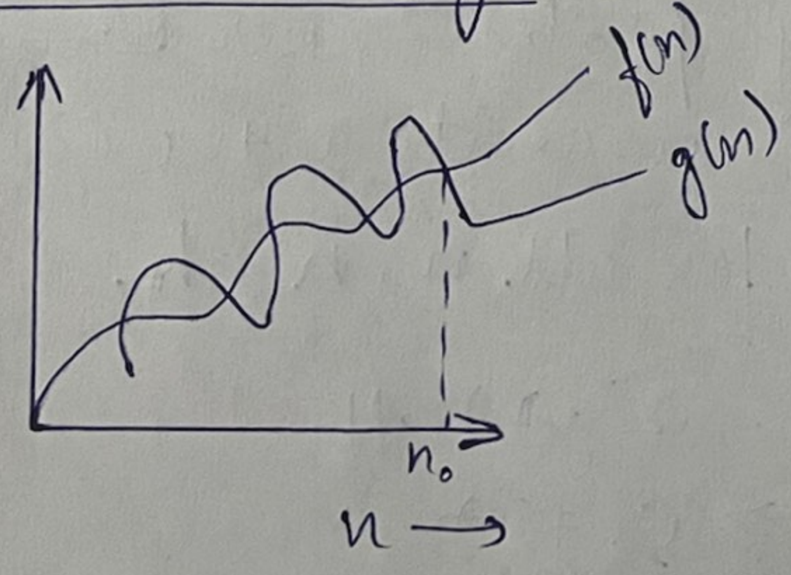
(iv) Small-Oh (O) :→ O gives upper bound.

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



$f(n) < c\ g(n)$
$\forall\ n > n_0$ & $\forall\ c > 0$
$n = O(n^2)$
$n < 1\ n^2$
$2\ n^2$
$0.5 n^2$
$n < 0.001\ n^2 n_0$

(v) Small-omega (ω) :→



lower bound
$f(n) = \omega\ g(n)$
$f(n) > c \cdot g(n)$
$\forall\ n > n_0$ & $\forall\ c > 0$.
$n^2 = \omega(n)$.

Sol-2:)

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

Time complexity for a loop means no. of times loop has run.

→ For the above loop, the loop will run for the following values of i :→

i	1	2	4	8	16	32	----	2^K
value	2^0	2^1	2^2	2^3	2^4	2^5	----	n

$i = 1, 2, 4, 8, 16, 32 ----, 2^K$ this means k times.

i.e. $2^K = n$

$k \log_2 2 = \log_2 n$

$k = \log n \quad [\log_2 2 = 1]$

$\therefore \boxed{TC = O(\log n)}$

Sol-3:) $T(n) = \begin{cases} 3T(n-1) & , n > 0 \\ 1 & \end{cases}$

By forward Substitution,

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

$T(0) = 1$

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

$= 3T(0)$

$= 3$

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

$= 3T(1)$

$= 3T = 3 * 3 = 3^2$

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

$= 3T(2)$

$= 3 * 3^2 = 3^3$

⋮

$T(n) = 3^n$

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

Sol-4:) $T(n) = \begin{cases} 2T(n-1) - 1, & n > 0 \\ 1 \end{cases}$

By forward Substitution,

$T(0) = 1$

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

$= (2 - 1)$

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

$= 2T(1) - 1$

$= 2(2-1) - 1$

$= 2^2 - 2^1 - 1$

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

$= 2T(2) - 1$

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

$= 2^3 - 2^2 - 2^1 - 1$

$\vdots$

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

$\Rightarrow 2^n - (2^n - 1)$

$\Rightarrow 2^n - 2^n - 1$

$\therefore TC = 1$

Sol-5:)

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

$S_i = S_{i-1} + i$

The value of 'i' increases by one for each value contained in 'S' at the i^{th} iteration is the sum of the first 'i' tve integers. If k is the total no. of iterations taken by any program then while loop terminates if:

$1 + 2 + 3 + \cdots + k.$

$= [k(k+1)/2] > n$

So, $k = O(\sqrt{n})$

$\therefore$ **TC = $O(\sqrt{n})$**

Sol-6:)

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

T.C. = O(n)

Sol-7:)

```
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)        O(log n)
            for(k = 1; k<=n; k = k*2)    O(log n)
                count++;
}
```

T.C = $\log n * \log n = O(n \log^2 n)$

TC = $O(n \log^2 n)$

Sol-8:)

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

$$\boxed{TC = O(n)}$$

Sol-9:)

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

$$TC = O(n) * O(n) = O(n^2)$$

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

Sol-10:) n^k is $O(c^n)$

$$\boxed{n^k = O(c^n)}$$