TUTORIAL-01

Answer-1:- Asymptotic Notation:-

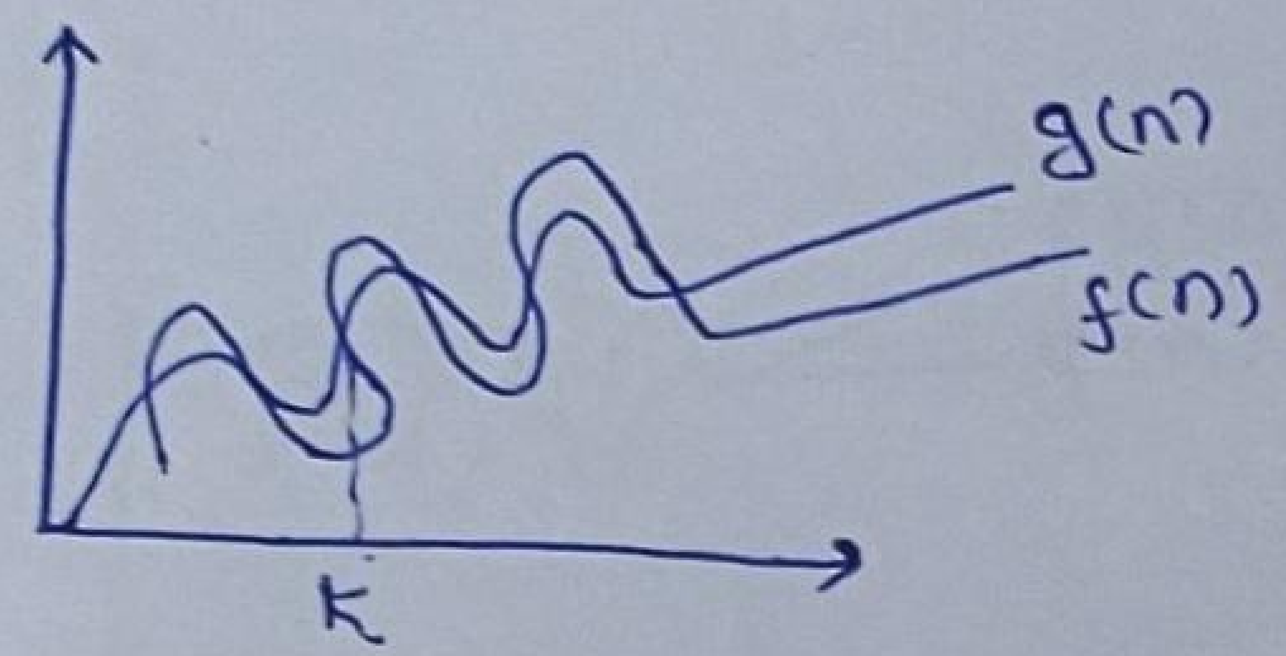
→ These notations are used to tell the complexity of an algorithm when the input is very large.

→ It describes the algorithm efficiency and performance in a meaningful way. It describes the behaviour of time or space complexity for large instance characteristics.

- The assymptotic notation of an algorithm is classified in 5 types-

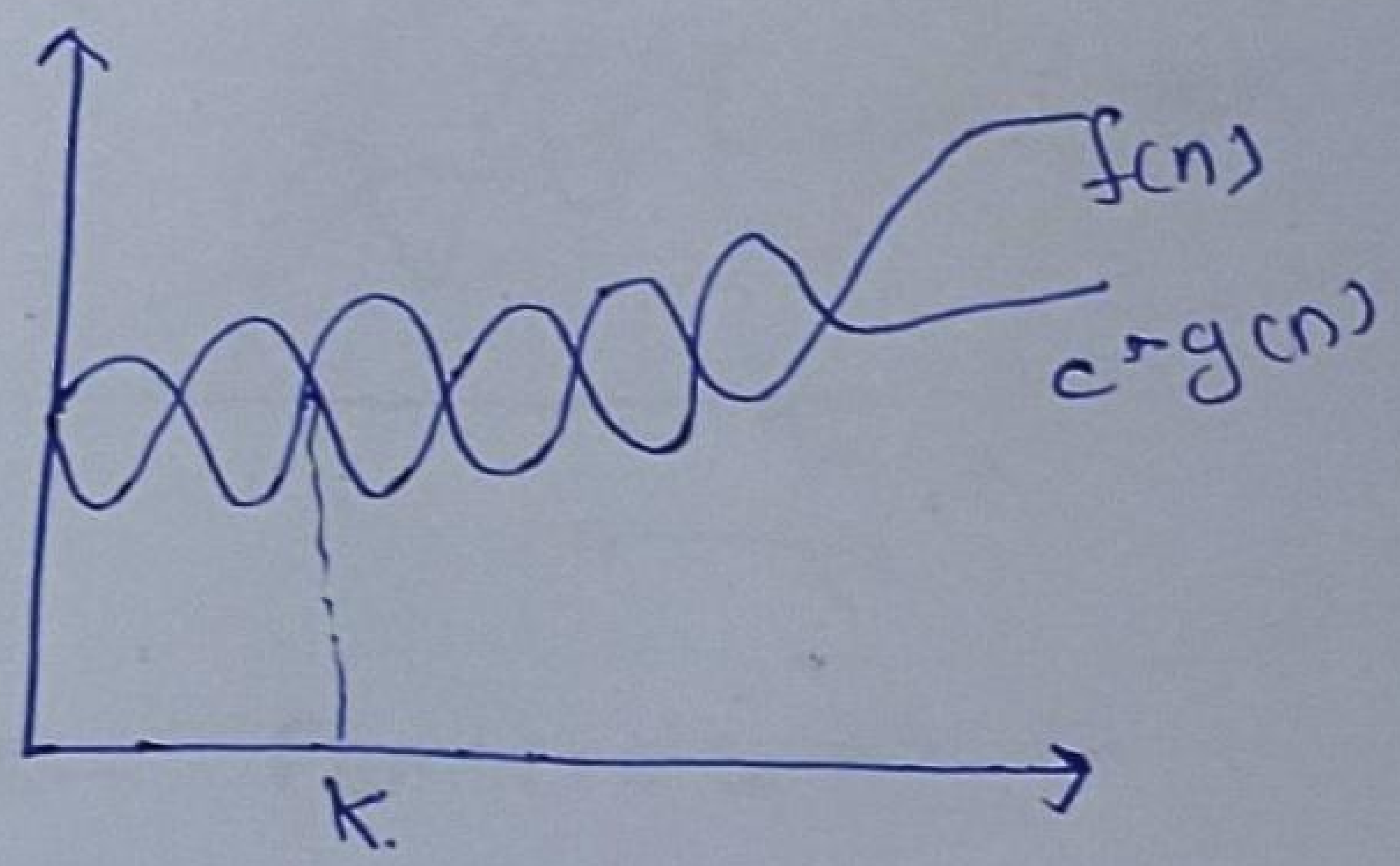
(i) Big oh notation (O):- (Asymptotic upper Bound) The function $f(n) = O(g(n))$, if and only if there exist a +ve constant c and K such that $f(n) \le c * g(n)$ for all n.

$n \ge K$



$f(n) = O(g(n))$
iff
$f(n) \le c \cdot g(n)$
$\forall\ n \ge n_0$,
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 $f(n) \ge c * g(n)$ for all n, $n \ge k$.



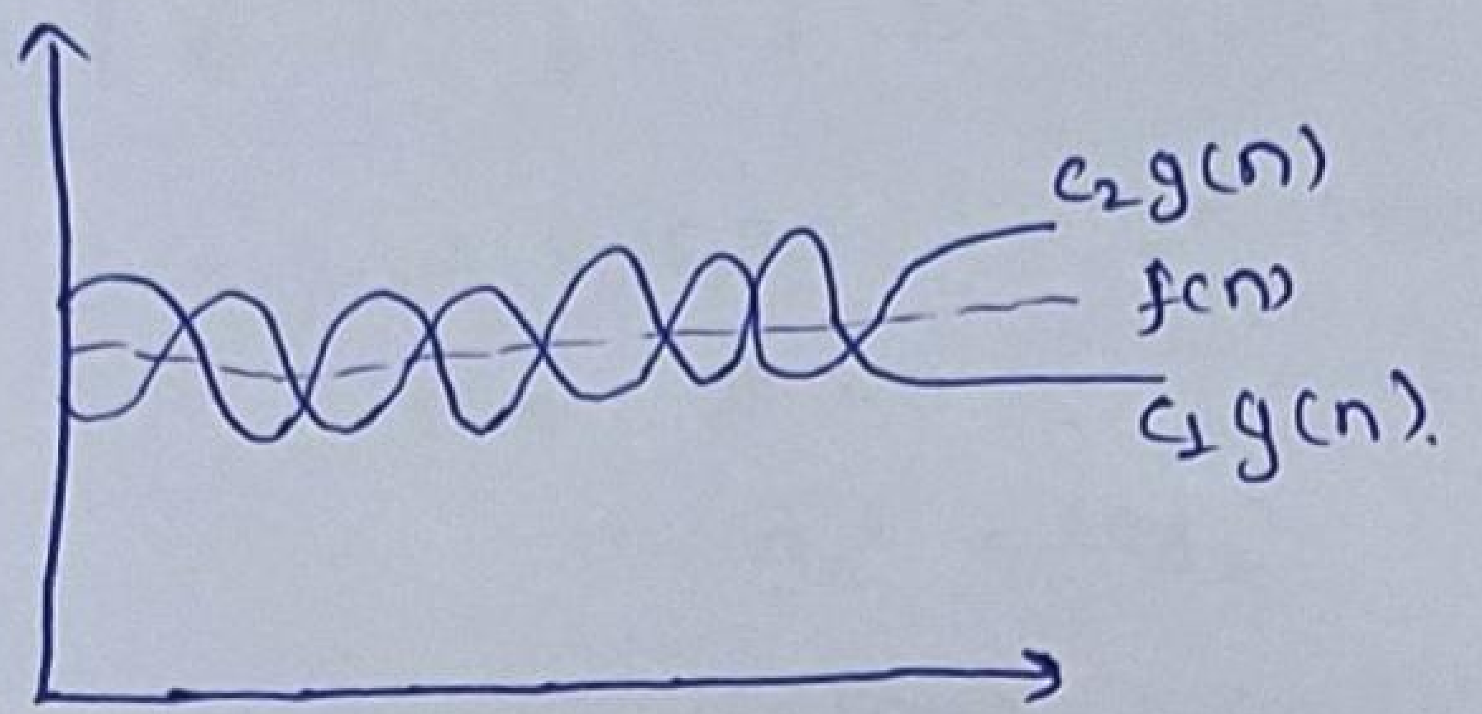
$f(n) = \Omega g(n)$
iff
$f(n) = c \cdot g(n)$

$\forall\ n \ge n_0$,

Some constant
$c > 0$.

(iii) Big theta notation (θ):- (Asymptotic tight bound)- The function $f(n) = \theta(g(n))$, iff there exists a +ve constant c_1, c_2 & k such that $c_1 * g(n) < f(n) < c_2 * g(n)$ for all n,

$n \geq k$.



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

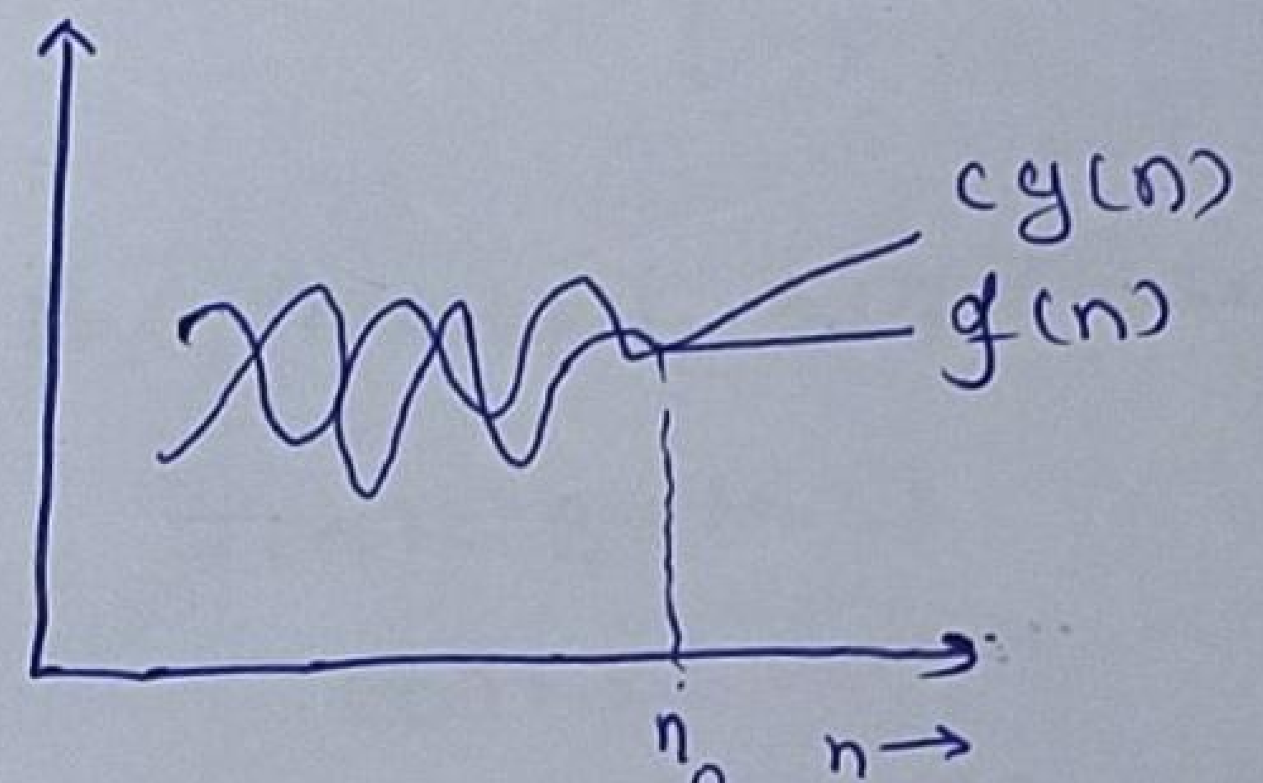
iff

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

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

(iv) Small-oh (o) :- o gives us upper bound.

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



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

$\forall\ n > n_0$ & $\forall\ c > 0$

$n = o(n^2)$

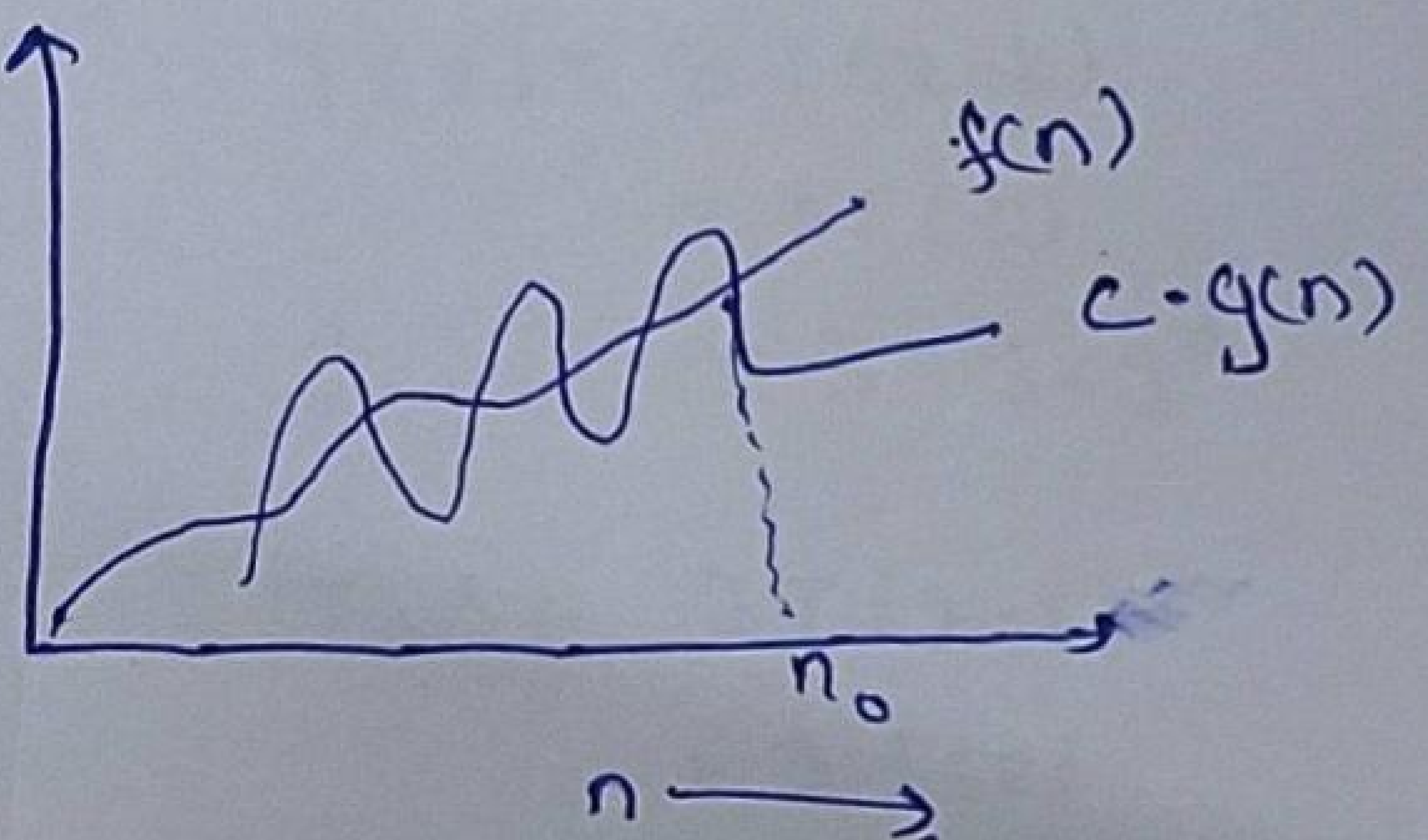
$n < 1 \cdot n^2$

$2n^2$

$0.5 n^2$

$n < 0.001 n^2\ n_0$

(v) Small-omega (ω) :-



lower bound

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

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

$\forall\ n > n_0$ & $\forall\ c > 0$

$n^2 = \omega(n)$

Answer:-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

$i = 1, 2, 4, 8, 16, 32, \ldots, 2^k$ this means k times

i.e. $2^k = n$

$k \log_2 2 = \log_2 n$

$k = \log n \quad \{\log_2 2 = 1\}$

$\therefore$ T.C = $O(\log n)$.

Answer-3:-

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

By forward substitution,

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

$T(0) = 1$

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

$= 3T(0)$

$= 3 \times 1 = 3$

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

$= 3T(1)$

$= 3 * 3 = 3^2$

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

$= 3T(2)$

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

$T(n) = 3^n$

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

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

By forward substitution,

$T(0) = 1$

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

$= (2-1)$

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

$= 2T(1) - 1$

$= 2(2-1) - 1$

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

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

$= 2T(2) - 1$

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

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

$\vdots$

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

$\therefore \boxed{T.C = 1}$

Answer-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 ith iteration is the sum of the first 'i' +ve integers. If k is the total number of iteration taken by any program then while loop terminates if: $1+2+3+\dots+k$.

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

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

$\therefore \boxed{T \cdot C = O(\sqrt{n})}$

Answer-6 :- void function (int n)

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

Time complexity :- $O(n)$

Answer-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 ++;
}
```

$T \cdot C = \log n * \log n$

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

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

Answer-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);
}
```

Time complexity :- $O(n^2)$

Answer-9:-

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

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

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

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

$n^k = O(c^n)$ ✗