1. What do you understand by Asymptotic Notations. Define different Asymtotic Notations with example.

Sol:

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

For example: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear i.e. the best case.
But, when the input array is in reverse condition, the algorithm take the maximum time (quadratic) to sort the elements i.e. the worst case.
When the input array is neither sorted nor in reverse order, then it takes average time. These durations are denoted using asymptotic notations.

There are mainly three asymptotic notations

- Big-O notation
- Omega notation
- Theta notation

- **Big-O Notation** (O-notation)

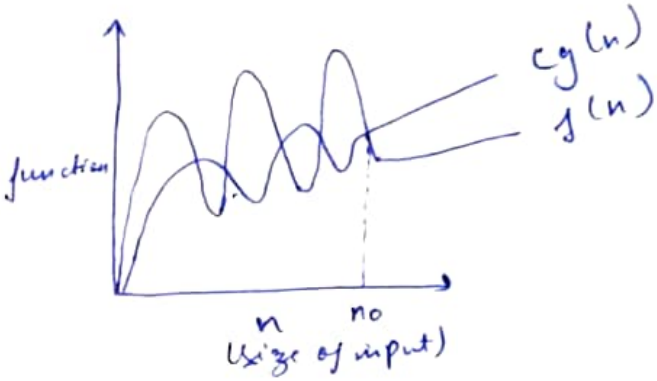
Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.

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

iff. $f(n) \leq c\,g(n) \quad \forall\ n \geq n_0$

for some constant $c > 0$

$g(n)$ is "tight" upper bound of $f(n)$



- **Omega Notation** (Ω-notation)

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithms.

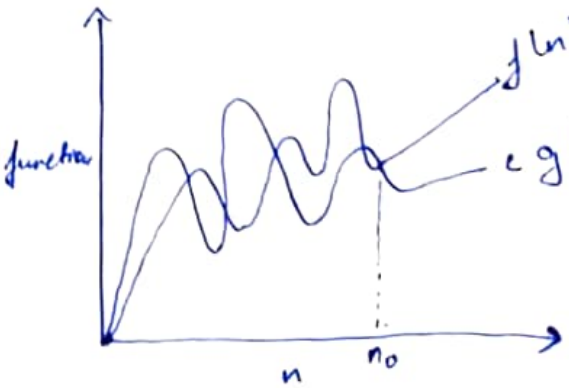
$$f(n) = \Omega(g(n))$$

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

$\forall\ n \geq n_0$

for some constant $c > 0$

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



• Theta Notation (Θ - notation)

Theta notation encloses the function from above and below. Since it represents the upper and lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

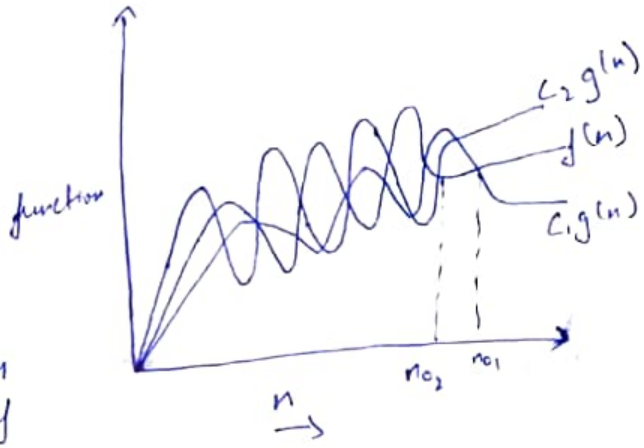
$f(n) = \Theta(g(n))$

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

$\forall\, n \geq n_0$

for some constant $c_1 > 0$

and $c_2 > 0$



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

2. What should be time complexity of -

```
for (i=1 to n)        // i = 1, 2, 4, 8, ... n
  { i = i * 2; }      // O(1)
```

$$\sum_{i=1} 1 + 2 + 4 + 8 + \ldots n$$

G.P. kth value $\Rightarrow T_k = ar^{k-1}$

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

Put $n = 2^{k-1}$

$\Rightarrow 2n = 2^k$

$\Rightarrow \log_2 2n = k \log_2 2$

$\Rightarrow \log_2^2 + \log_2^n = k$

$\Rightarrow \log n + 1 = k$

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

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

$T(0) = 1$

$T(n) = 3T(n-1)$ — (1)

Put $n = n-1$

$T(n-1) = 3T(n-2)$ — (2)

Put value of $T(n-1)$ in (1)

$Tn = 3 \times 3T(n-2)$ — (3)

$= 9T(n-2)$ — (3)

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

$T(n-2) = 3T(n-3)$ — (4)

Put value of $T(n-2)$ in (3)

$\Rightarrow 27T(n-3)$

$\Rightarrow T(n) = 3^k T(n-k)$

Put $n = k$

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

$T(n) = 3^n$

$\Rightarrow O(3^n)$

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

$T(n) = 2T(n-1) - 1$ — (1)

$T(0) = 1$

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

$T(n-1) = 2T(n-2) - 1$ — (2)

Put value of $T(n-1)$ in (1)

$T(n) = 4T(n-2) - 2 - 1$ — (3)

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

$T(n-2) = 2T(n-3) - 1$ — (4)

Put value of $T(n-2)$ in (3)

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

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

Consider the G.P. $2^{k-1} + 2^{k-2} + \ldots 1$

$a = 2^{k-1}$

$r = \frac{1}{2}$

$\Rightarrow \frac{a(1-r^n)}{1-r}$

$= \frac{2^{k-1}\left(1 - \left(\frac{1}{2}\right)^k\right)}{1/2}$

$= 2^k \left(1 - \frac{1}{2}^k\right)$

$= 2^k - 1$

Let $n - k = 0$

$n = k$

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

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

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

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

5. What should be time complexity of -

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

We can define the term 's' according to the relation $s_i = s_{i-1} + i$

$$s = 1 + 3 + 6 + 10 + 15 + \ldots \quad n$$

The value contained in s at the ~~4~~ i^{th} iteration is the sum of first 'i' positive integers. If K is total number of iterations taken by the program, then while loop terminates if :

$$1 + 2 + 3 + \ldots + K = [K(K+1)/2] > n$$

$$\Rightarrow K(K+1)/2 = n$$

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

$$\Rightarrow \quad k^2 + k = 2n$$

$$\Rightarrow \quad k^2 = 2n - k$$

$$k = \sqrt{2n - k}$$

$$= O\sqrt{n}$$

&

$\therefore$ Time complexity of the above function $O\sqrt{n}$.

Q. 6 Time complexity of -

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

as $i^2 <= n$

$\Rightarrow i <= \sqrt{n}$

$i = 1, 2, 3, 4 \ldots, \sqrt{n}$

$\sum_{i=1}^{n} 1 + 2 + 3 + 4 + \ldots + \sqrt{n}$

$\Rightarrow T(n) = \frac{\sqrt{n} \times (\sqrt{n} + 1)}{2}$

$\Rightarrow T(n) = \frac{n \times \sqrt{n}}{2}$

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

Q.7. Time complexity of :

```
void fn(int n)
{ int i, j, k, count = 0;
  for(i = n/2 ; i <= n ; ++i)
    for(i = 1; j <= n ; j = j * 2)
      for(k = 1; k <= n ; k = k * 2)
        count++;
}
```

for $k = k * 2$

$k = 1, 2, 4, 8, \ldots \quad n$

G.P. :- $a = 1, r = 2$

$$\rightarrow \frac{1(2^k - 1)}{1}$$

$n \rightarrow 2^k$

$\log n = k$

i	j	k
1	$\log n$	$\log n * \log n$
2	$\log n$	$\log n * \log n$
⋮	⋮	⋮
n	$\log n$	$\log n * \log n$

$O(n * \log n * \log n)$

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

8) Time complexity of

```
function (int n)
{
    if (n == 1)
      return;
    for (i = 1 to n)                    — O(n)
    {  for (j = 1 to n)
       {  printf("*");                  — O(n²)
       }
    }
    function(n-3);                      — T(n/3)
}
```

$\Rightarrow T(n) = T(n/3) + n^2$

$a = 1, \quad b = 3 \quad f(n) = n^2$

$c = \log_3 1 = 0$

$\Rightarrow n^0 = 1 \quad (f(n) = n^2)$

$\Rightarrow T(n) = \theta(n^2)$

Q. 9.

Time complexity of -

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

for $i=1 \Rightarrow j = 1, 2, 3, 4, \ldots n = n$

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

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

for $i=n \Rightarrow j = 1 \ldots = 1$

$$\Rightarrow \sum_{j=n}^{1} n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \ldots + 1$$

$$\sum_{j=n}^{1} n\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots \frac{1}{n}\right)$$

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

10. For the functions, n^k and c^n. What is asymptotic relationship between these functions?

Assume that $k >= 1$ and $c > 1$ are constants. Find out the value of c and n_0 for which relation holds.

The relation between n^k and c^n

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

as $n^k < a c^n$

$\forall n \geq n_0$ and some constant $a > 0$

for $n_0 = 1$

$c = 2$

$\Rightarrow 1^k \leq a 2^1$

$\Rightarrow n_0 = 1$ and $a = 2$