1.) What do you mean by Asymptotic Notations. Define different asymptotic notation with example.

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

Asymptotic means tending to infinity.

Different types of asymptotic notation are :-

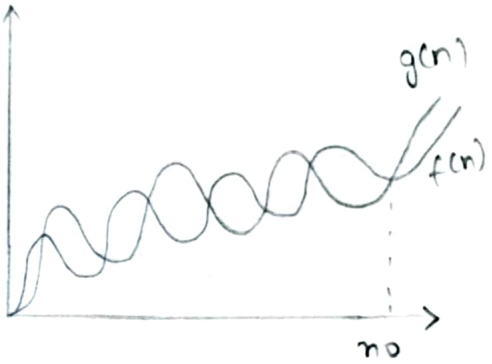
i) Big-O Notation :-

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

$\forall\ n > n_0$

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

$\because$ c is constant here.



ex: $f(n) = 2n+3$

$2n+3 \leq 2n^2+3n^2$

$2n+3 \leq 5n^2$

$f(n) \quad g(n)$

$f(n) = O(n)$

$g(n) = O(n^2)$

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

ii) Omega (Ω) Notation

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

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

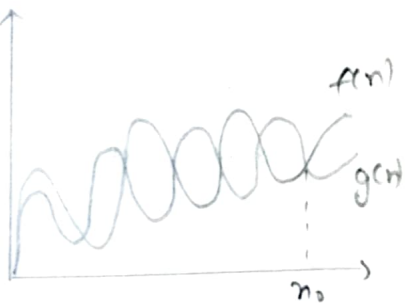
$g(n)$ is tight lower bound of $f(n)$

$f(n) = 2n+3$

$2n+3 \geq 1 \times \log n \quad \forall n \geq 1$

$f(n) \quad \uparrow c \quad g(n)$

$\boxed{f(n) = \Omega(n)}$



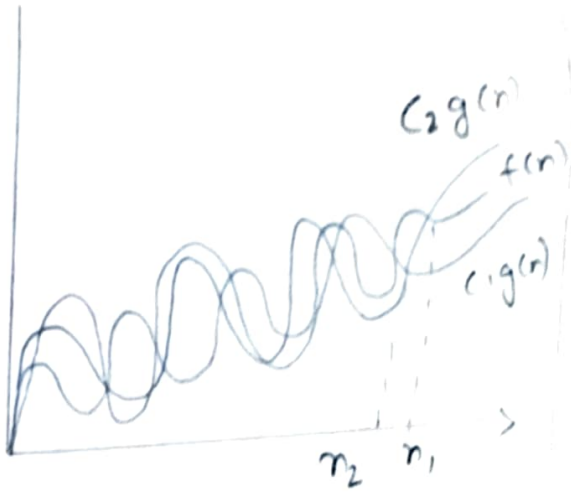
iii) Theta (Θ) Notation :-

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

iff $c_2 g(n) \geq f(n) \geq c_1 g(n)$

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

$c_1 > 0$ & $c_2 > 0$



ex: $f(n) = 2n+3$

$1 \times n \leq 2n+3 \leq 5 \times n$

c_1 $g(n)$ $f(n)$ c_2 $g(n)$

$\boxed{f(n) = \Theta(n)}$

2.) Time complexity of :

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

1, 2, 4, 8, 16, 32 --- k terms

k^{th} term $= a \cdot r^{k-1}$ $\{r = \frac{4}{2} = 2\}$ $a=1$

k^{th} term $\Rightarrow 1 \cdot 2^{k-1}$

$n = 2^{k-1}$

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

taking log both sides

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

$\log_2 2n = k$

$k = \log_2 2 + \log_2 n$

$k = 1 + \log_2 n$

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

$\approx O(\log_2 n)$

---

5) Time complexity of :

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

i = 1, 2, 3, 4, 5, 6 ---

s = 1, 3, 6, 10 --- k^{th} term

loop will run till $s <= n$

s is the sum of k numbers

k^{th} term $= \frac{k*(k+1)}{2}$

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

$2n = k^2 + k$

$k(k+1) = 2n$

$k^2 + k = 2n$

$k^2 = n$

$k = \sqrt{n}$

~~$T(n) = (\sqrt{2})$~~

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

3) $T(n) = 3T(n-1)$ — ①

$T(0) = 1$ — ②

Putting $n = n-1$ in ①

$T(n-1) = 3T(n-2)$ — ③

Putting $n = n-2$ in ①

$T(n-2) = 3T(n-3)$ — ④

Putting $n = n-3$ in ①

$T(n-3) = 3T(n-4)$ — ⑤

Putting ⑤ in ④

$T(n-2) = 9T(n-4)$ — ⑥

Putting ⑥ in ③

$T(n-1) = 3^3 T(n-4)$ — ⑦

Putting ⑦ in ①

$T(n) = 3^4 T(n-4)$ — ⑧

$T(n) = 3^k T(n-k)$ — ⑨ (for k terms)

$n-k = 0$

$n = k$

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

$T(n) = 3^n$

complexity ⇒ $O(3^n)$

6)

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

$1, 2^2, 3^2, 4^2, 5^2 \dots k^{th}$

k^{th} term $= k * k$

k^{th} term $<= n$

$k * k <= n$

$k^2 = n$

$k = \sqrt{n}$

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

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

Time complexity of inner most loop

$k = 1$ to n, $k = k * 2$

$1, 2, 4, 8, 16 \cdots k^{th}$ term

k^{th} term $= 2^{k-1}$

$$n = \frac{2^k}{2} \Rightarrow 2n = 2^{k-1}$$

taking $\log_2$ both sides

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

$$\log_2 2n = k$$

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

$$\boxed{k = 1 + \log_2 n}$$

Complexity of middle loop

$j = 1$ to n; $j = j * 2$

$1, 2, 4, 8, 16 \cdots k^{th}$ term

$\Rightarrow (1 + \log_2 n)$

It means for each value of i, this loop runs $(1 + \log_2 n)$ times

complexity of outer most loop

$i = n/2$ to n; $i++$

$n/2, \frac{n}{2}+1, \frac{n}{2}+2, \frac{n}{2}+3 \cdots k^{th}$

k^{th} term $= \frac{n}{2} + k$

$$n = \frac{n}{2} + k \Rightarrow k = n - \frac{n}{2} \Rightarrow \frac{n}{2}$$

$$T(n) = \frac{n}{2} * (1 + \log_2 n) * (1 + \log_2 n)$$

$$\Rightarrow \frac{n}{2} + \frac{n}{2}\log_2 n + \frac{n}{2}(\log_2 n)^2 + \frac{n}{2}\log_2 n$$

$$\Rightarrow O(n\{\log_2 n\}^2)$$

Q4.) $T(n) = 2T(n-1) - 1$

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

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

$\Rightarrow 2^2(2T(n-3) - 1) - 2 - 1$

$\Rightarrow 2^3 T(n-3) - 2^2 - 2 - 1$

Similarly, after k steps we have

$\Rightarrow 2^k T(n-k) - 2^{k-1} - 2^{k-2} - \dots - 2^2 - 2^1 - 2^0$

considering $T(1) = 1$, let's take $n-k=1 \Rightarrow k = n-1$

therefore, substituting $k = n-1$

$T(n) = 2^{n-1} T(1) - [2^0 + 2^1 + 2^2 + \dots + 2^{n-3} + 2^{n-2}]$

$\Rightarrow 2^{n-1} \times 1 - [2^{n-1} - 1]$

$\Rightarrow 2^{n-1} - 2^{n-1} + 1$

$\boxed{T(n) = O(1)}$

Q9.)

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

Outer loop will run n times (i)

for $i=1$; j will run n times

for $i=2$; j will run $n/2$ times

for $i=3$; j will run n/n times

Inner loop will run = $(n + \frac{n}{2} + \frac{n}{3} + \dots \frac{n}{n-1} + \frac{n}{n})$ terms

$\Rightarrow n(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n})$

$\Rightarrow n \cdot \log n \Rightarrow O(n \log n)$

Q8)
```
function (int n)
{
    if (n==1)
      return;
    for (i to n)
    {
      for (j to n)
      {
        printf("*");
      }
      function (n-3);
    }
```

i	j
1	1 → n
2	1 → n
3	1 → n
4	1 → n
.	
n	

⇒ n * n times

for func. (n-3)

n, n-3, n-6, n-9 ... k^{th} term

n, n-1*3, n-2*3, n-3*3 ... k^{th} term

k^{th} term = n-(k-1)*3 ⇒ n-3k+3

1 = n-3k+3

⇒ n-3k-k+1 = 0

⇒ n-3k-4 = 0

$k = \frac{n-4}{3}$

Inner most loop will execute ⇒ $n * n * \frac{n-4}{3}$

complexity ⇒ $\frac{n^3-4n^2}{3}$

$T(n) \Rightarrow O(n^3)$