Name :- Shikha Singh

Section :- CST

Semester :- IV

class Roll no :- 18

Univ. Roll no :- 2017554

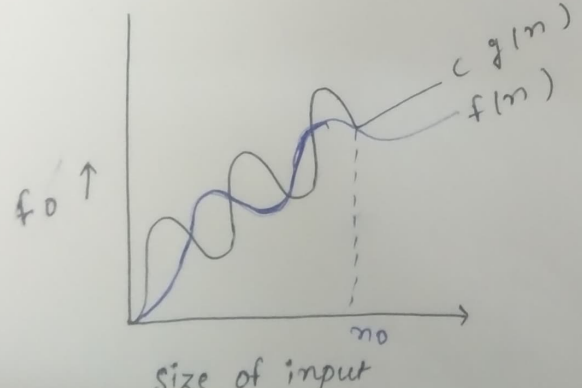
Date :- 10-March-2022

Tutorial-1

Que (1) What do you understand by Asymptotic notations. Define different Asymptotic notation with examples.

→ Asymptotic notations :- It is used to describe the running time of an algorithm - how much time an algorithm takes with a given input n.

(i) Big O (O) :-



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

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

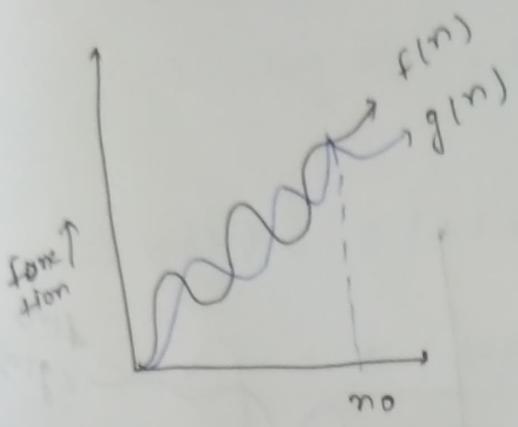
$\forall\ n \geq n_0$

for some constant $c > 0$

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

Shikha Singh

(ii) Big Omega (Ω):-



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

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

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

iff $f(n) \geq cg(n)$

$\forall \quad n \geq n_0$

for some constant, $c > 0$

(iii) Theta (θ)

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

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

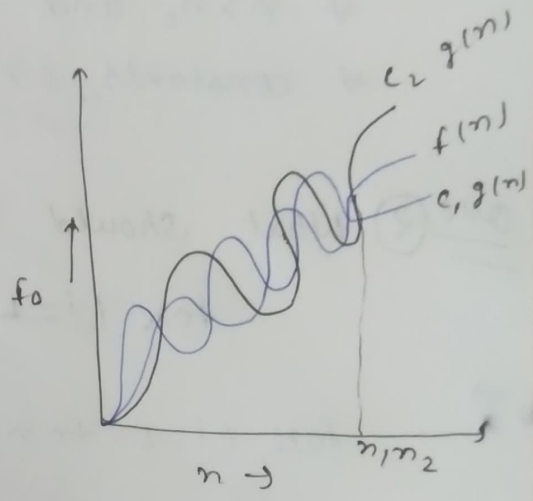
$f(n) = \theta(g(n))$

Iff

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

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

for some constant $c_1 > 0$ and $c_2 > 0$



(iv) Small o (o):-

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

$g(n)$ is upper bound of function $f(n)$

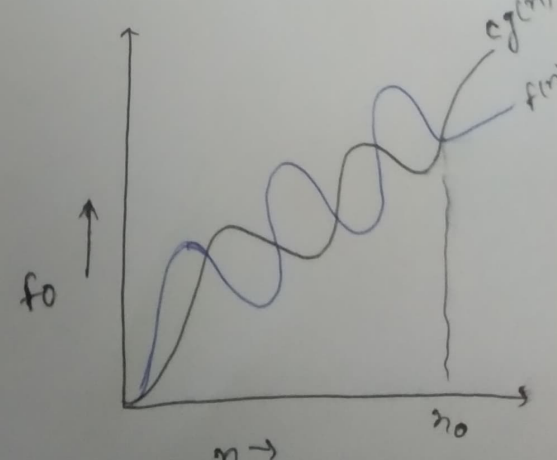
$f(n) = o(g(n))$

when

$f(n) < cg(n)$

$\forall \quad n > n_0$

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



Shivani Singh

(5) Small Omega (ω):-

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

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

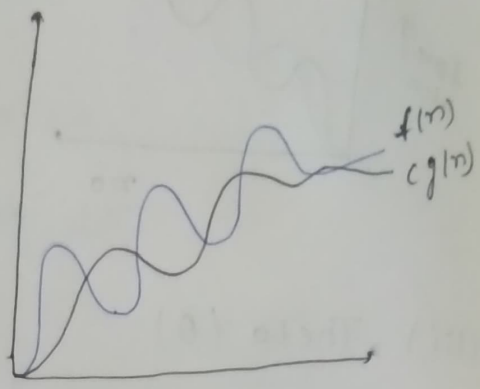
$f(n) = \omega(g(n))$

When

$f(n) > cg(n)$

$\forall\ n > n_0$ and

$\forall$ constants, $c > 0$



Que (2) What should be time complexity of

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

→

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

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

GP k^{th} value $\Rightarrow T_k = ar^{k-1}$

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

$\Rightarrow n = 2^k$

$\Rightarrow 2n = 2^k$

$\Rightarrow \log 2n = k \log 2$

Shikha Singh

$\Rightarrow \log_2 + \log n = K \log 2$

$\Rightarrow \log n+1 = K$

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

$= \underline{O(\log n)}$

upto n times

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

$\Rightarrow \underline{O(\log_2 n+1)}$

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

→ $T(n) = 3T(n-1)$ — (1)

put $n = n-1$

$T(n-1) = 3T(n-2)$ — (2)

from (1) & (2)

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

$= 9T(n-2)$ — (3)

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

$T(n) = 3(T(n-3))$ — (4)

$T(n) = 27(T(n-3))$

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

Putting $n-K = 0$

$\Rightarrow n = K$

Shikha Singh

$\Rightarrow T(n) = 3^n [T(n-n)]$

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

$\Rightarrow T(n) = 3^n \times 1 \qquad [T(0) = 1]$

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

Que (4) $T(n) = \{2T(n-1) - 1$ if $n > 0$, otherwise $1\}$

→

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

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

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

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

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

$= 2^n T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots 2^2 - 2^1 - 2^0$

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

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

$T(n) = O(1)$

Note:- $2^{n-1} + 2^{n-2} + \ldots + 2^0 = 2^n - 1$

Que (5) What should be 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 ___ n

Sum of S = $1 + 3 + 6 + 10 + \dots + T_n$ — (1)

also S = $1 + 3 + 6 + 10 + \dots T_n + T_n$ — (2)

from (1) − (2)

$0 = 1 + 2 + 3 + 4 + \dots - n - T_n$

$T_k = 1 + 2 + 3 + 4 + \dots k$

$T_k = \frac{1}{2} ok(k+1)$

for k iterations

$1 + 2 + 3 + \dots + k <= n$

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

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

$O(k^2) <= n$

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

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

Que (6) Time complexity of -

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

→ Here for ~~H~~th term

$$k \times H \leq \sqrt{n}$$

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

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

Que (7) Time complexity of —

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

Shikha Singh

for $k = k * 2$

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

GP $\Rightarrow \quad a = 1 \qquad r = 2$

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

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

$$\Rightarrow n = 2^k$$

$$\Rightarrow \log n = k$$

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

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

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

Que (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);
}
```

→ $i = 1, 2, 3, 4, \ldots n \Rightarrow O(n)$

$j = 1, 2, 3, 4 \ldots n \Rightarrow O(n^2)$

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

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

In standard form : $T(n) = aT(n-b) + O(n^k)$

$a = 1$

$b = 3$

$k = 2$

here $a = 1$

therefore $T(n) = O(n^{k+1})$

$= O(n^3)$

Que (9) Time complexity of -

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

→ for i = 1, j = 1, 2, 3 ---- n

for i = 2, j = 1, 3, 5 ---- n/2

for i = 3, j = 1, 4, 7 ---- n/3

i = n, j = 1 + 1 + ---- 1

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

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

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

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

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

Que (10) For the function, n^K and c^n, what is the asymptotic relation between these function?

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

⇒ as given n^K and c^n

relation between n^K & c^n is

$$n^K = O(c^n)$$

as $n^K \leq a c^n$

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

for $n_0 = 1$, $c = 2$

$\Rightarrow 1^K \leq a\, 2^1$

$n_0 = 1 \quad c = 2$