NAME : Aastha Verma

SECTION : I

ROLL NO.: 12

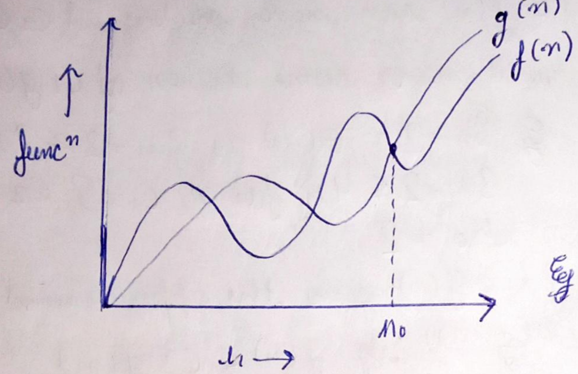
[Tutorial - 1]

Q:1:- What do you mean by Asymptotic notations? Define different type of notations along with example.

Ans:- Asymptotic Notations: Means tending to infinity. They are used to tell the complexity when input is very large.

→ Different types of asymptotic notations :

1. Big Oh (O) Notation : $f(n) = O(g(n))$



if $f(n) \leq g(n) \times C \; \forall \; n \geq n_0$

for some constant, $C > 0$

g(n) is 'tight' upperbound of f(n).

Eg. $f(n) = n^2 + n$

$g(n) = n^3$

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

$n^2 + n = O(n^3)$

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

2. Big Omega (Ω) : $f(n) = \Omega(g(n))$ means g(n) is 'tight' lowerbound of f(n) i.e. f(n) can go beyond g(n)

i.e. $f(n) = \Omega\, g(n)$ if and only if

$f(n) \geq C \cdot g(n) \; \forall \; n \geq n_0$ &

C = constant > 0

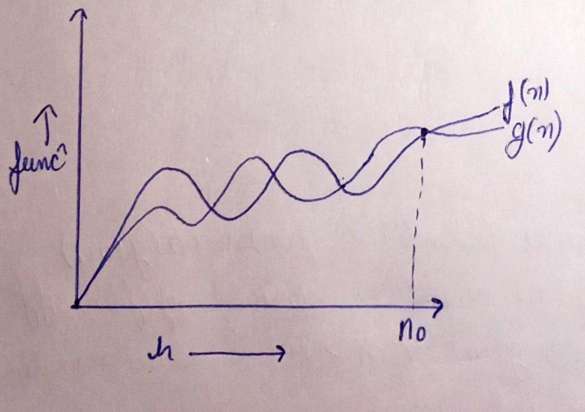
Eg $f(n) = n^3 + 4n^2$

$g(n) = n^2$

i.e. $f(n) \geq C * g(n)$

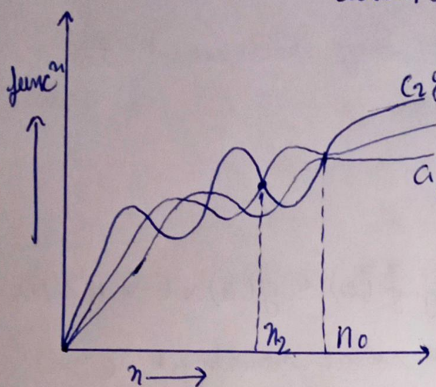
$n^3 + 4n^2 = \Omega(n^2)$

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



3. Big Theta (θ). When $f(n) = \theta(g(n))$ gives the tight upperbound and lower bound both

i.e. $f(n) = \theta(g(n))$ iff.



$c_1 * g(n_1) \le f(n) \le c_2 * g(n_2)$

$\forall\ n \ge max(n_1, n_2)$, some constant

$c_1 > 0$ & $c_2 > 0$

i.e. $f(n)$ can never go beyond $c_2 g(n)$ & will never come down of $c_1 g(n)$.

Eg $3n+2 = \theta(n)$ as $3n+2 \ge 3n$ &

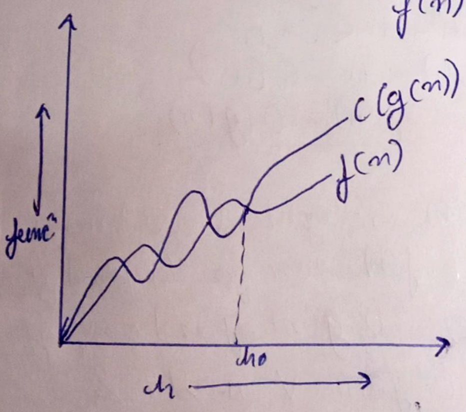
$3n+2 \le 4n$ for n, $c_1 = 3$, $c_2 = 4$ &

$n_0 = 2$

4. Small Oh (o): When $f(n) = o\,g(n)$ gives the upper bound i.e.

$f(n) = o(g(n))$ iff $f(n) < c * g(n)$

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



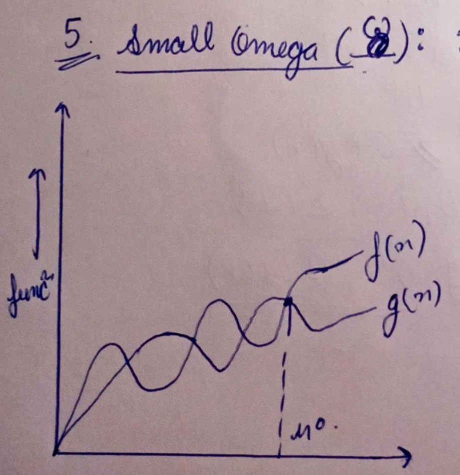
Eg. $f(n) = n^2$; $g(n) = n^3$

$f(n) < c * g(n)$

$n^2 = o(n^3)$

5. Small Omega (ω): It gives lower bound i.e. $f(n) = \omega(g(n))$ where $g(n)$ is lower bound of $f(n)$ iff

$f(n) > c * g(n)$ $\forall\ n > n_0$ & some constant $c > 0$



Q2: What should be time complexity of:

```
for (int i = 1 to n)
{   i = i * 2;   → O(1)
}.
```

⇒ for $i \Rightarrow 1, 2, 4, 6, 8 \ldots$ n times

i.e. series is a G.P.

So, $a = 1$, $r = 2$

Now, k^{th} term:- $t_k = a r^{k-1}$

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

$n = 2^{k-1}$

taking log both sides

$\log_2 n = \log_2 2^{k-1}$

$\log_2 n = (k-1)\log_2 2$

$\log_2 n = k - 1 \Rightarrow k = 1 + \log_2 n$ [$\because \log_2 2 = 1$]

$\therefore$ Time complexity $T(n) = O(k)$

$= O(1 + \log_2 n)$

$= \underline{O(\log_2 n)}$

Q3. $T(n) = \{3T(n-1)$ if $n > 0$, otherwise $1\}$

⇒ $T(n) = 3T(n-1)$ —— (1)

put $n = (n-1)$ in eq^n (1)

$T(n-1) = 3T(n-2)$ —— (2)

put eq (2) in (1)

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

$T(n) = 9T(n-2)$ —— (3)

put $n = n-2$ in eq^n (1)

$T(n-2) = 3T(n-3)$ —— (4)

put (4) in (3)

$T(n) = 27T(n-3)$ —— (5)

On generalising eqⁿ 5

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

put $n-k=0$

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

$$= 3^k \quad (\because T(0) = 1)$$

$$\therefore \underline{T(n) = O(3^n)}$$

Q4: $T(n) = \{2T(n-1) - 1$ if $n > 0$, otherwise $1\}$

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

put $n = n-1$ in eqⁿ (1)

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

$$T(n-1) = 2T(n-2) - 1 \quad \text{—— (2)}$$

put eqⁿ (2) in (1)

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

$$T(n) = 4T(n-2) - 3 \quad \text{—— (3)}$$

put $n = n-2$ in eqⁿ (1)

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

put value of $T(n-2)$ in eqⁿ (3)

~~T (n-2) = 2T(~~

$$T(n) = 4[2T(n-3) - 1] - 2 - 1$$

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

On generalising

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

put $n-k = 0 \Rightarrow n = k$, $T(0) = 1$ (given)

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

$$= 2^n - \underbrace{[2^{n-1} + 2^{n-2} \cdots\cdots + 1]}_{k \text{ Terms}}$$

$\Rightarrow a = 2^{n-1}$, $r = 1/2$

Sum of GP = $\dfrac{2^{n-1}\left[1-(1/2)^{n-1}\right]}{1-1/2} = 2^n - 2$

$\Rightarrow T(n) = 2^n - [2^n - 2] = 2$

$= O(2)$

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

Q 5: What should be the time complexity of –

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

Ans:

i	s
1	1
2	3
3	6
4	10
⋮	⋮
n	n
	k times

$S = \underbrace{1, 3, 6, 10, 15, \ldots n}_{k \text{ terms}}$

k^{th} term $\Rightarrow T_k = 1+2+3+4+\ldots+k$

$T_k = 1/2\, k(k+1)$

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

loop runs k times

Time complexity $= O(k^2) <= n$

$k = O(\sqrt{n})$.

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

Q 6: Time complexity of: void f(int n)

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

⇒ As $i^2 = n$

$i = \sqrt{n}$

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

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

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

$T(n) = O(n)$ Ans

Q 7: Time complexity of void f(int n)

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

⇒ Since, for $k = n^2$

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

∴ series is in G.P.

So, $a = 1$, $r = 2$

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

$n = 2^k - 1 \Rightarrow n+1 = 2^k$

$\log_2(n) = k$

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

T.C. ⇒ $O(n \log n * \log n)$

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

Q 8: Time complexity of void func (int n)

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

Ans: for (i=1 to n)

we get j = n times every turn

$\therefore i * j = n^2$

k^{th}, Now, $T(n) = n^2 + T(n-3)$;

$T(n-3) = (n-3)^2 + T(n-6)$;

$T(n-6) = (n-6)^2 + T(n-9)$;

$\& \ T(1) = 1$

$\Rightarrow T(n) = n^2 + (n-3)^2 + (n-6)^2 + \ldots + 1$

let $n-3k = 1$

$k = (n-1)/3$ Total terms $= k+1$

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \ldots + 1$

$T(n) \simeq kn^2$

$T(n) \simeq (k-1)/3 * n^2$

So, $T(n) = O(n^3)$

Q. 9: Time complexity of: void func (int n) {

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

Ans: for $i = 1 \rightarrow j = 1, 2, 3, 4, \ldots n = n$

for $i = 2 \rightarrow j = 1, 3, 5, 7, \ldots n = n/2$

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

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

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

$$\sum_{j=n}^{1} n[1 + 1/2 + 1/3 + \ldots + 1/n]$$

$$\sum_{j=n}^{1} n \log n$$

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

Q10:- For the functions, n^k & c^n, what is the asymptotic relation relationship b/w these functions. Assume that $k >= 1$ & $c > 1$ are constants. Find out the value of c and no for which relation holds.

Ans: As given n^k and c^n

relation b/w n^k and c^n is $\boxed{n^k = O(c^n)}$

as $n^k \leq a.c^n \;\forall n \geq n_0$ for a constant $a > 0$

for $n_0 = 1$

$c = 2$

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

$\therefore \boxed{n_0 = 1 \;\&\; c = 2}$