Anjali Mishra
20174651
CST

Tutorial-1

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

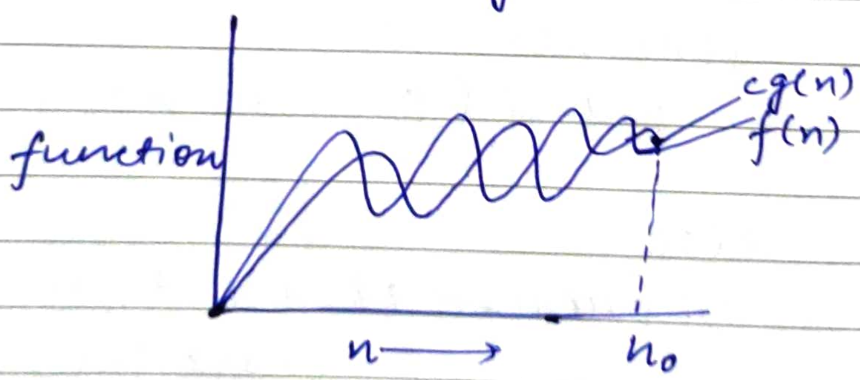
Asymptotic Notations

They 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.

Different asymptotic notations –

i) Big O(n)

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



size of input

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

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

$\forall\ n \geq n_0$

for some constants, $c > 0$

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

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

$g(n) = n^3$

$n^2 + n \le c n^3$

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

ii) Big Omega (Ω)

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

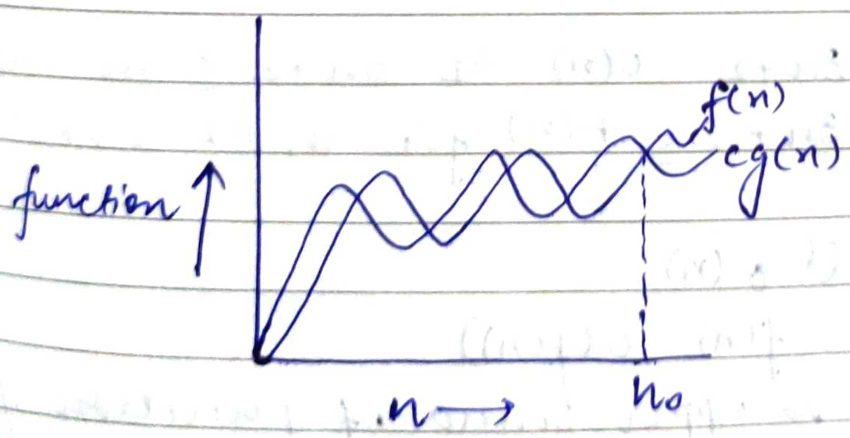
$g(n)$ is "tight" lower bound of function $f(n)$

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

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

$\forall\ n \ge n_0$

for some constant $c > 0$



ex: $f(n) = n^3 + 4n^2$

$g(n) = n^2$

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

iii) Big Theta (θ).

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

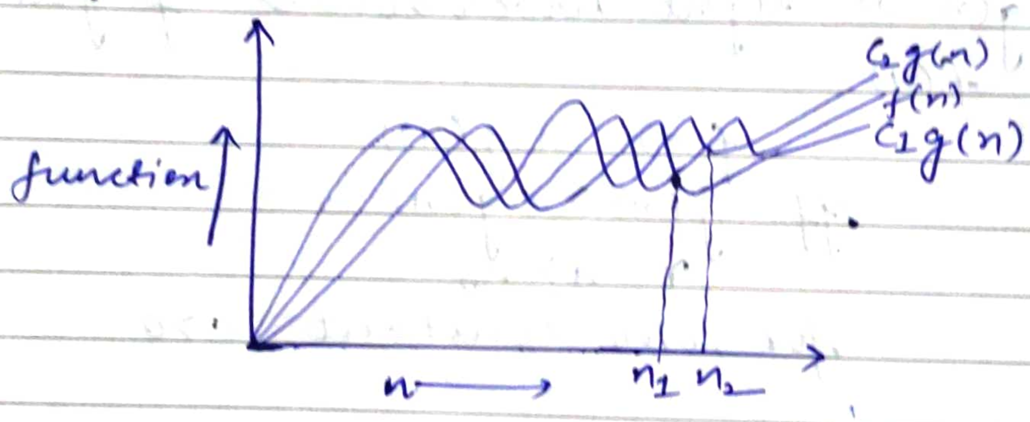
g(n) is both "tight" upper and "lower" bound of function of f(n).

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

iff

$$c_1 g(n) \le f(n) \le c_2 g(n)$$
$$\forall\ n \ge \max(n_1, n_2)$$

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



Ex:

$3n+2 = \theta(n)$ as $3n+2 \ge 3n$ and $3n+2 \le 4(n)$ for n, $k_1 = 3$, $k_2 = 4$ & n_0

iv) Small o(n)

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

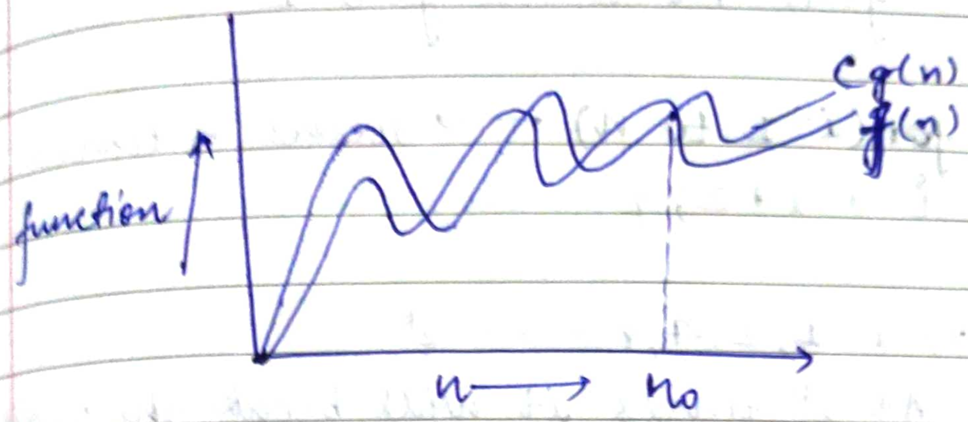
g(n) is upper bound of function f(n)

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

when, $f(n) < c\ g(n)$

$$\forall\ n > n_0$$

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



Ex: $f(n) = n^2$
$g(n) = n^3$
$n^2 = O(n^3)$

1) small omega (ω)

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

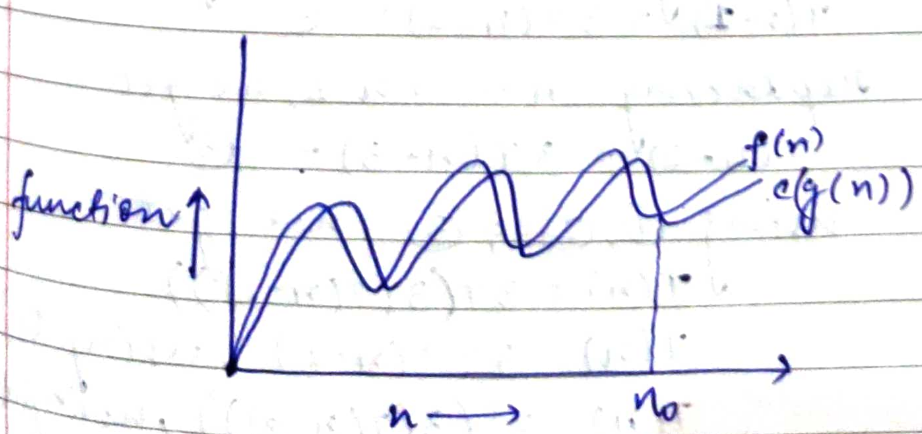
g(n) is lower bound of f(n)

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

when $f(n) > c\,g(n)$

$\forall\ n > n_0$

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



$f(n) = 4n+6 \quad g(n) = 1$

2. for(i=1 to n) $\longrightarrow$ runs 'n' times
{ i=i*2; }

$\Rightarrow i = 1, 2, 4, 8 \text{-----} 2^k$
At 2^k times it will break the condition
So, $2^k = n$
Taking, $\log_2$ both side, we get
$k = \log_2 n$

$$\boxed{\text{Time complexity} = O(\log_2 n)}$$

3. $$T(n) = \begin{cases} 3T(n-1) & ; n>0 \\ 1 & ; \text{otherwise} \end{cases}$$

$T(n) = 3T(n-1) \longrightarrow (1)$
replacing $n \longrightarrow n-1$, we get
$T(n-1) = 3T(n-2) \longrightarrow (2)$
replacing $n \longrightarrow n-1$, we get
$T(n-2) = 3T(n-3) \longrightarrow (3)$
using (1), (2) & (3), we get
$T(n) = 3T(3(T(n-2)))$
$T(n) = 3^2 T(n-2)$, using (2)
$T(n) = 3^2(3(T(n-3)))$, using (3)
$T(n) = 3^3 T(n-3)$

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

Now, we know that $T(1) = 1$

so, $n-k = 1$

$k = n-1$

On putting, $k = n-1$, we get

$T(n) = 3^{n-1} T(1)$

$T(n) = 3^{n-1}$

$\boxed{\text{Time complexity} = O(3^n)}$

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

$T(n) = 2T(n-1) - 1$ ---- ①

replacing n with n-1

$T(n-1) = 2T(n-2) - 1$ —— ②

replacing n with n-1

$T(n-2) = 2T(n-3) - 1$ —— ③

from ①, ② & ③, we get

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

$T(n) = 2^2 T(n-2) - 2 - 1$, using ②

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

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

⋮

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

we know that,

$T(0) = 1$

$n-k = 0$

$\boxed{k=n}$

On putting $k=n$, we get

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

$= \underbrace{2^n + 2^{n-1} + 2^{n-2} + \dots + 2 + 1}_{G.P.}$

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

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

$\boxed{\text{Time complexity} = O(2^n)}$

Soln 5

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

s depends on i, so we make cases

At, $i = 1$	2	3	4	------- n
$s = 1$	3	6	10	------- k

at $i = n$, $s = k$ & k breaks the while condition. so,
we can clearly see that 's' is just of ~~some~~ sum of 'n' natural no., so

$$\frac{k(k+1)}{2} > n$$

$$\frac{k^2 + k}{2} > n$$ → dominating power

$\Rightarrow \quad k^2 = n$

$k = \sqrt{n}$

Time complexity = $O(\sqrt{n})$

~~Time complexity =~~

Solu 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++;
}
```

All are independent loops,
so

Time complexity $= \frac{n}{2} * \log_2 n * \log_2 n = O(n\log^2 n)$

Solu 8

```
function(int n)
{ if (n==1) return;

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

    function(n-3);
}
```

$$T(n) = T(n-3) + n^2 \quad \text{——— } ①$$

let $n = n-3$

$$T(n-3) = T(n-6) + n^2 \quad \text{——— } ②$$

$$T(n) = T(n-6) + n^2 + n^2 \quad \text{——— } ③$$

$\vdots$

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

let $n - 3k = 1$

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

$\therefore$ Time complexity $= O(n^2)$

Date / /

9. Time complexity of

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

for $i=1 \longrightarrow j=1$ to $n \longrightarrow n$ times

$i=2 \longrightarrow j=1$ to $n \longrightarrow n/2$ times

$i=3 \longrightarrow j=1$ to $n \longrightarrow n/3$ times

$\vdots$

$i=n \longrightarrow j=1$ to $n \longrightarrow 1$ times

So, total $= n + n/2 + n/3 + \cdots + 1$

$= n(1 + 1/2 + 1/3 + \cdots + 1/n)$

$= n\log n$

Time complexity $= O(n\log n)$

10. Given: n^k and c^k

Relation between n^k and c^k

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

As, $n^k \leq a c^n$

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

for $n_0 > 1$

$c > 2$

$1^k \leq a2$

$n_0 = 1$ & $c = 2$