SHREYA SINGH RAGHUVANSHI

61

CST_SPL1

TCS409

Tutorial - 1

20022399

2017038

TUTORIAL-1

Q1- What do you understand by Asymptotic Notations. Define different Asymptotic notation with examples.

Ans - Asymptotic notations are used to describe the asymptotic running time of an algorithm

It is defined in terms of functions whose domains are the set of natural numbers $N = \{0, 1, 2 \dots\}$

These are convenient for describing the worst-case running-time function $T(n)$.

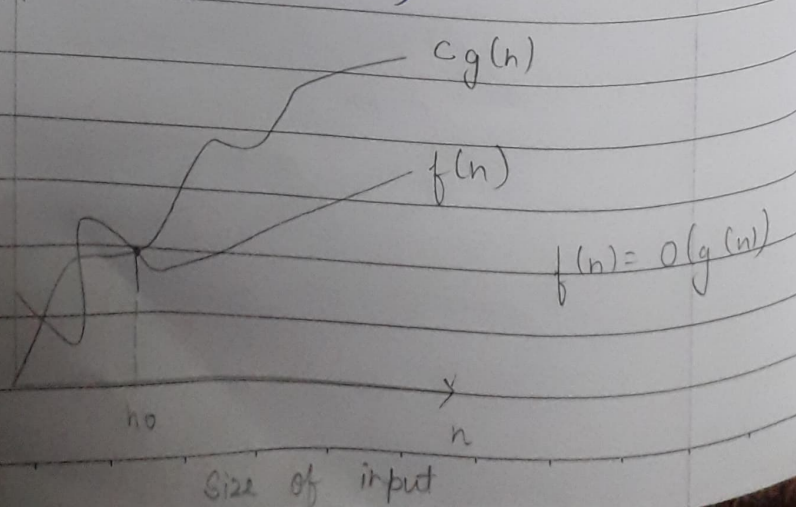
These help us finding the complexity of an algorithm when i/p is very large.

Different asymptotic notations are as follows:

1. Big O(o)

It defines an upper bound of an algorithm it bounds a function only from above.

eg. Insertion Sort : $O(n^2)$



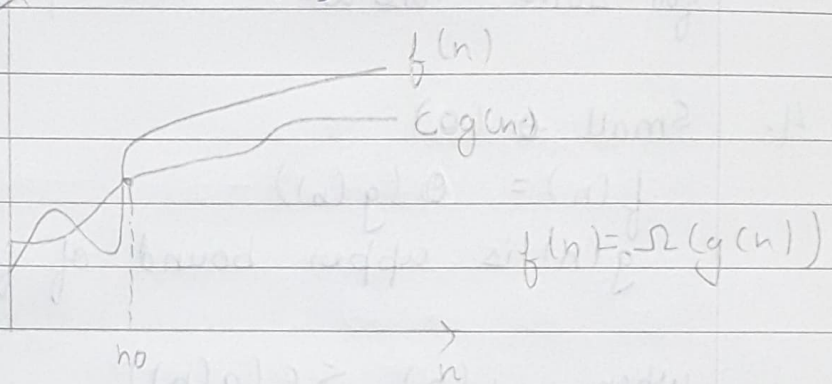
$f(n) = O(g(n))$ [tight upper bound]

if and only if $f(n) \leq c\,g(n)$
$\forall\ n \geq n_0$

for some constant, $c > 0$.

2. Big Omega (Ω)

It provides an asymptotic lower bound.
It is useful when we have a lower bound on the time complexity of an algorithm.
eg. Insertion sort: $\Omega(n)$



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

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

if and only if $f(n) \geq c\,g(n)$
$\forall\ n \geq n_0$

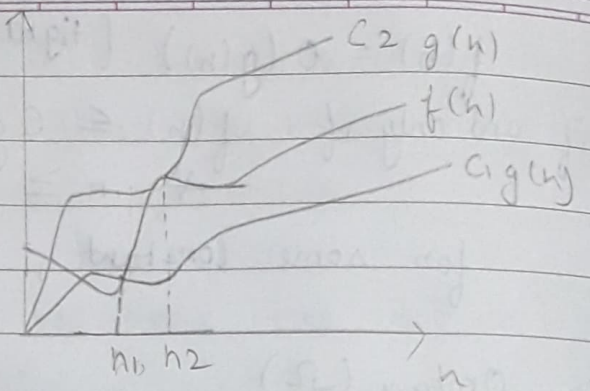
for some constant, $c > 0$

3. Theta (Θ)

It bounds a function from above & below, so it defines exact asymptotic behaviour

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

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

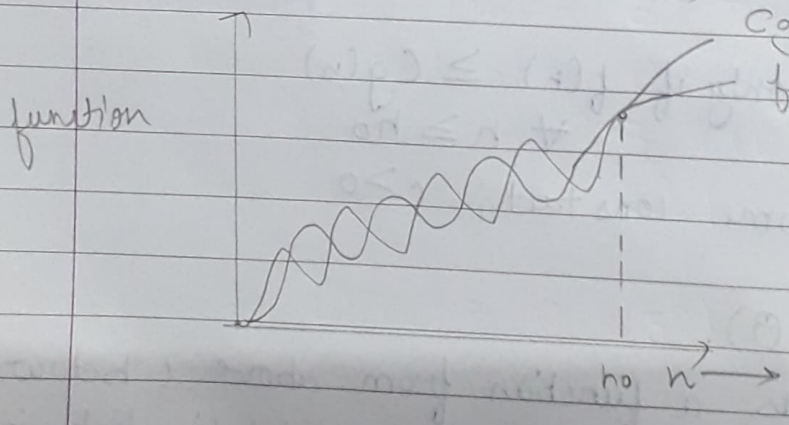


$f(n) = \Theta(g(n))$
iff $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$
$\forall\ n \geq (n_1, n_2)$
for some constant $c_1 > 0$, $c_2 > 0$.

4. Small o (o)

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

when $f(n) < c(g(n))$
$\forall\ n > n_0$
and $\forall$ constants, $c > 0$



5. Small omega (ω)

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

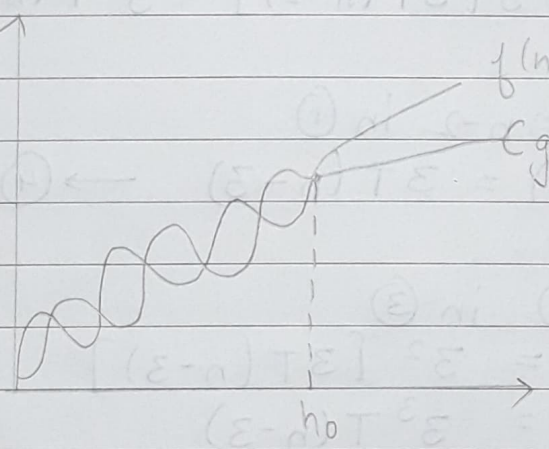
$g(n)$ is lower bound of function $f(n)$.

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

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

$\forall\ n > n_0$

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



Q2- What should be the time complexity of :

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

Sol- $i = \overbrace{1, 2, 4, 8, \dots, n}^{\text{Let } k \text{ terms}}$ // G.P.

$a_k = a\, r^{k-1}$

$a = 1$, $r = 2$, $a_k = n$

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

$n = 2^{k-1}$

$2n = 2^k$

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

$k = 1 + \log_2 n$

$T(n) = O(\log_2 n + 1) = \boxed{O(\log_2 n)}$

Q3- $T(n) = \begin{cases} 3T(n-1) & \text{if } n > 0, \\ 1 & \text{otherwise} \end{cases}$ // $T(0) = 1$

Sol- By backward substitution

Put $n = n-1$ in $T(n) = 3T(n-1)$ → (1)

$T(n-1) = 3T(n-2)$ → (2)

Put (2) in (1)

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

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

$T(n-2) = 3T(n-3)$ → (4)

Put (4) in (3)

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

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

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

Let $n-k = 0$

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

$= 3^n . 1$

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

Q4- $T(n) = \begin{cases} 2T(n-1)-1 & \text{if } n>0, \\ \text{otherwise } 1 & \text{// } T(0)=1 \end{cases}$

Sol Put $n=n-1$ in $T(n) = 2T(n-1)-1 \rightarrow (1)$

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

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

Put (2) in (1)

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

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

Put $n = n-2$ in $T(n) = 2T(n-1)-1 \rightarrow (1)$

$T(n-2) = 2T(n-3)-1 \longrightarrow (4)$

Put (4) in (3)

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

$= 2^3 T(n-3) - 2^2 - 2^1 - 2^0 \rightarrow (5)$

Generalising

$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - 2^{k-3} \cdots 2^0.$

Let $n-k = 0$

$n = k$

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

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

as $T(0)=1$

$$T(n) = 2^n - 2^{n-1} - 2^{n-2} \quad \cdots \quad 2^0$$
$$= 2^n - [1 + 2^1 + 2^2 \cdots 2^{n-1}]$$
$$= 2^n - \left[\frac{1 \times (2^n - 1)}{2 - 1}\right]$$

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

5. What should be the time complexity of

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

Sol-

i	s	
1	1	1
2	1+2	3
3	1+2+3	6

The value in s at ith iteration is the sum of first i positive integers.
So, to break out of loop
$s > n$

i.e., if k is total iterations
Then Sum of k terms $> n$

i.e., $1+2+3+\cdots+k \geq n$

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

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

$$k^2 \geq n$$

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

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

6- Time complexity of

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

Sol- $i = 1, 2, 3, \cdots n$

$i^2 = 1, 4, 8, \cdots n$

$i^2 <= n$ or $i <= \sqrt{n}$

For kth term: $t_k = a + (k-1)d$

$a = 1 \quad d = 1$

$t_k <= \sqrt{n}$

$\sqrt{n} = 1 + (k-1)$,

$\sqrt{n} = k$

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

7. Time complexity of

```
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++;
}
```

Sol-

i	j	k
$\frac{n}{2}$	$\log n$	$\log n$
$\frac{n}{2}+1$	$\log n$	$\log n$
$\frac{n}{2}+2$	$\log n$	$\log n$
$\vdots$	$\vdots$	$\vdots$
n	$\log n$	$\log n$

$\frac{n+1}{2}$ times

$$O(i * j * k) = O\left(\left(\frac{n+1}{2}\right) * \log n * \log n\right)$$

$= O\left(\left(\frac{n+1}{2}\right) * (\log_2 n)^2\right)$

$T(n) = O(n(\log_2 n)^2)$.

8. Time complexity of

```
function (int n)
{
    if (n == 1)
        return;
    for (i=1 to n)                    // n
    {
        for (j=1 to n)                // n
        {
            printf ("*");
        }
    }                                 // n^2
    function (n-3);
}
```

Sol $T(n) = T(n-3) + n^2 \longrightarrow (1)$

$T(1) = 1 \longrightarrow (2)$

Put $n = n-3$ in (1)

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

Put (3) in (1)

$T(n) = T(n-6) + (n-3)^2 + n^2 \longrightarrow (4)$

Put $n = n-6$ in (1)

$T(n-6) = T(n-9) + (n-6)^2 \longrightarrow (5)$

Put (5) in (4)

$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n$

Generalize

$T(n) = T(n-3k) + (n-3(k-1))^2 + (n-3(k-2))^2 + \cdots - n^2$

Let $n - 3k = 1$

$\frac{n-1}{3} = k$

$T(n) = T(1) + \left(n-3\left(\frac{n-1}{3}-1\right)\right)^2 + \left(n-3\left(\frac{n-1}{3}-2\right)\right)^2 \cdots n^2$

$= T(1) + (n-[(n-1)-3])^2 + (n-[(n-1)-6])^2 + (n-[n-1-9])^2 + \cdots n^2$

$= 1 + (3+1)^2 + (6+1)^2 + \cdots n^2$

$= 1 + 4^2 + 6^2 + \cdots n^2$

$= n^2 + \cdots 1$

$\boxed{T(n) = O(n^2)}$

9.
```
void function (int n)
{
    for (i=1 to n)
    {
        for (j=1 ; j<=n ; j=j+i)
            printf ("*");
    }
}
```

Lt

i	j
1	n times
2	1+3+5---+n times

$a_n = a + (k-1)d$

$a = 1 \quad , \quad d = 2$

$n = 1 + (k-1).2$

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

$k = \frac{n-1}{2} + 1$

$k = \frac{n+1}{2}$; No. of terms

n^2

for $i = 2$ $\quad j = \frac{n+1}{2}$ times

for $i = 3$ $\quad j = 1 + 4 + 7 + \ldots$ n times

$n = 1 + (k-1)d$

$n = 1 + (k-1)3$

$\frac{n-1}{3} + 1 = k$

$k = \frac{n+2}{3}$; No. of terms

Generalising

for $i = n$ $\quad j = \frac{n + k - 1}{k}$ times

$$T(n) = n + \frac{n+1}{2} + \frac{n+2}{3} + \cdots + \frac{n+k-1}{k}$$

(n terms)

General term $= \frac{n+k-1}{k}$

$$\Rightarrow \sum_{i=1}^{n} \frac{n+k-1}{k} \Rightarrow \frac{\sum_{i=1}^{n} n + \sum_{i=1}^{n} k - \Sigma 1}{k}$$

$$\Rightarrow \frac{\frac{n(n+1)}{2} + nk - n}{k}$$

$$\Rightarrow \frac{\frac{n^2+n}{2} + nk - n}{k}$$

$$\Rightarrow \frac{\frac{n^2}{2} + \frac{n}{2} + nk - n}{k}$$

After removing constant terms

$$\boxed{T(n) = O(n^2)}$$

Q10- For the functions, n^k & c^n, what is the asymptotic relationship b/w these functions?

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

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

as $n^k \leq a \cdot c^n$

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

for $n_0 = 1$

$c = 2$

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

$n_0 = 1$ & $c = 2$