TUTORIAL - 1

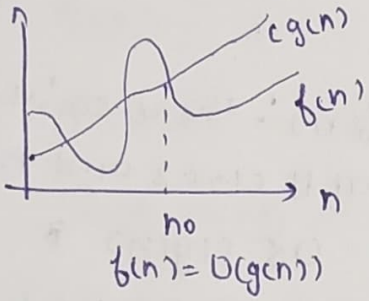
Q1 What do you mean/understand by Asymptotic notations. Define different Asymptotic notations with example.

Ans Asymptotic notation are those notations that describing the limiting behaviour of a function OR we can also say that these are used to tell the complexity of an algo when the i/p is very large.

There are mainly three asymptotic notations

1. Big O notation
2. Omega notation
3. Theta notation

Big-O notation - It represents the upper bound of the running time of a algorithm. Thus it gives the worst case complexity of analgorithm.



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

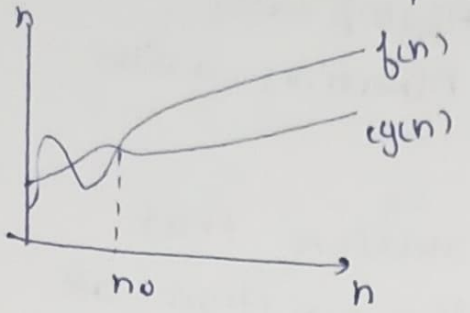
$O(g(n)) = \{f(n)\}$: there exists +ve constants c and no such that $0 \le f(n) \le cg(n)$ $\forall$ $n \ge n_0$

Eg

```
for(i=1; i<=n; i++)
{
    sum+=i;
}
```

O(n)

Big Omega Notation - It represents the lower bound of the running time of an algorithm. Thus it provides the best case complexity of an algorithm.



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

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

$\Omega(g(n)) = \{ f(n)$: there exists +ve constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ $\forall\ n \geq n_0 \}$

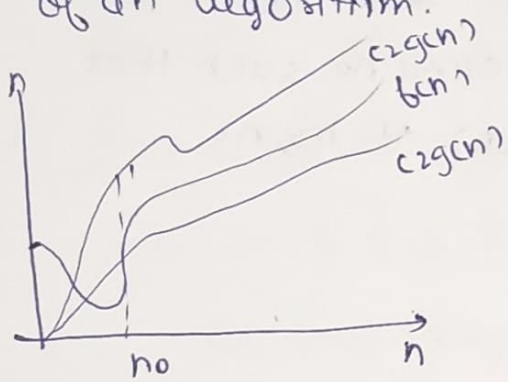
Ex) $f(n) = 2n + 3$

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

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

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

Theta Notation - Theta notation encloses the function from above and below. since it represent the upper & the lower bound of the running time of an algorithm it is used for analyzing the average-case complexity of an algorithm.



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

$\Theta(g(n)) = \{ f(n)$: there exists +ve constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$

Q2) what should be the time complexity of

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

A

$i = 1, 2, 4, 8, 16 \ldots$

$i = 2^0, 2^1, 2^2, 2^3 \ldots 2^k$

$2^k = n$

taking log both sides

$k \log_2 2 = \log n$

$k = \log n$

Time complexity $= O(\log n)$ ✓

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

A

$T(n) = 3T(n-1)$ — (1)

$T(n-1) = 3T(n-2)$ — (2)

Put eq (2) in eq (1)

$T(n) = 3 \cdot 3T(n-2)$ — (3)

$T(n-2) = 3T(n-3)$ — (4)

Put eq (4) in eq (3)

$T(n) = 3 \cdot 3 \cdot 3 \cdot T(n-3)$ — (5)

On comparing eq 1, 3, 5

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

$n-k=0$

$n=k$

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

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

$= 3^n \cdot 1$

$= 3^n$

Time complexity $= O(3^n)$

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

$T(n) = 2T(n-1) - 1$ — (1)

$T(n-1) = 2T(n-2) - 1$ — (2)

Put eq (2) in eq (1)

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

$T(n) = 2 \cdot 2 \cdot T(n-2) - 2 - 1$ — (3)

$T(n-2) = 2T(n-3) - 1$ — (4)

Put eq (4) in eq (3)

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

$= 2 \cdot 2 \cdot 2 \cdot T(n-3) - 2 \cdot 2 - 2 - 1$ — (5)

On comparing eq 1, 3, 5

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

$n-1k=0$

$n=1k$

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

$2^{n-1} - 2^{n-2} - \cdots - 1 = 2^n - 1$

So

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

$= 2^n - 2^n + 1$

$= 1$

Time complexity = $O(1)$

Q5) What should be time complexity of-

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

Ans

i	s	
1	1	→ Initially
2	3	1+2
3	6	1+2+3
4	10	1+2+3+4
5	15	1+2+3+4+5
⋮	⋮	

$1+2+3+4+\cdots+k$

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

$k^2 = n$

Time complexity → $\log(\sqrt{n})$

Q6) Time complexity of :

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

Ans

$i * i <= n$

$i^2 <= n$

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

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

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

$$= \frac{\sqrt{n} * \sqrt{n} + \sqrt{n}}{2}$$

$$= \frac{n\sqrt{n}}{2}$$

Time complexity = $O(n)$ Ans.

Q7) 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++;
}
```

Ans) Outer loop complexity → O(n)
Inner loop complexity → O(log n)
Last loop complexity → O(log n)

Total complexity = O(n log n · log n)
= O(n (log n)²) Ans

Q8) Time complexity of -

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

Ans

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

By master's method.

$a = 1, \; b = 3, \; f(n) = n^2$

$c = \log_b a$

$c = \log_3 1$

$c = 0 \qquad n^0 = 1$

$f(n) > n^0$

$n^2 > 1$

So Time complexity $T(n) = O(n^2)$

Q4] Time complexity of -

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

Ans

for $i=1 \Rightarrow j = 1, 2, 3 \dots n$

for $i=2 \Rightarrow j = 1, 3, 5, \dots n$

for $i=3 \Rightarrow j = 1, 4, 7 \dots n$

⋮

for $i=n \Rightarrow j = 1$

for inner loop

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

$$n + \frac{n}{2} + \frac{n}{3} + \text{---} + \frac{n}{n}$$

$$n\left[1 + \frac{1}{2} + \frac{1}{3} + \text{---} + \frac{1}{n}\right]$$

$n \log(n)$ { $1 + \frac{1}{2} + \frac{1}{3} + \text{---} \frac{1}{n}$ this is an HP we can also write it like $\int_1^n \frac{1}{x}\,dx = \log n$ }

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

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

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

Sol

As given n^k & c^n

relation b/w n^k & c^n is

$n^k \rightarrow O(c^n)$ as $n^k \leq c^n$

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

for $n_0 = 1$ $c = 2$

$1^k \leq a 2^1$

$n_0 = 1$ & $c = 2$