# Tutorial-1

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

→ Asymptotic notations 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.

There are three types of Asymptotic notation :-

(i) **Big-O** →

* This notation defines an upper bound of an algorithm.
* The function $f(n) = O(g(n))$ if and only if $f_n <= c.g(n)$ for all $n >= n_0$ where $c$ and $n_0$ are constants
* Here $g(n)$ is known as upper bound on values of $f(n)$.
* E.g. $f(n) = 3n+3$, $g(n) = 4n$.

(ii) **Big-Ω** →

* Ω notation provides an asymptotic lower bound.
* The function $f(n) = \Omega\, g(n)$ if $f(n) >= c.g(n)$ for all $n >= n_0$ where $c$ and $n_0$ are constants.
* Here, $g(n)$ is known as lower bound on values of $f(n)$.
* E.g. $f(n) = 3n+2$ and $g(n) = 3n$.

(iii) **Big. θ** →

* The theta notation bounds a function from above and below, so it defines exact asymptotic behaviour. Hence, it is also known as tightly bound.
* The function $f(n) = \theta(g(n))$ if $C_1.g(n) <= f(n) <= C_2.g(n)$ for all $n >= n_0$ where $C_1$, $C_2$ and $n_0$ are constants.
* E.g. $f(n) = 3n+2$, $g(n) = n$, $C_1 = 3$ and $C_2 = 4$

② What should be time Complexity of -

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

→ $O(n\log n)$

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

Using Substitution

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

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

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

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

----

----

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

$= 3^n T(0)$

$= 3^n$

Complexity will be $O(3^n)$

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

Using Substitution

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

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

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

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

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

----

----

$= 2^nT(n-n)-2^{n-1}-2^{n-2}-2^{n-3}$

$\ldots -2^2-2^1-2^0$

$= 2^n - 2^{n-1} - 2^{n-2} - 2^{n-3}$

$\ldots - 2^2 - 2^1 - 2^0$

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

[Note: $2^{n-1} + 2^{n-2} + \dots + 2^0 = 2^n - 1$]

$T(n) = 1$

Time Complexity is $O(1)$.

(5) What should be time complexity of -

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

→ Let the loop execute $x$ times. Now, the loop will execute as long as s is less than n.

After 1st iteration:

$S = S+1$

After 2nd iteration:

$S = S+1+2$

As it goes for $x$ iterations,

$1+2 \dots + x <= n$

$\Rightarrow (x*(x+1))/2 <= n$

$\Rightarrow O(x^2) <= n$

$\Rightarrow x = O(\sqrt{n})$

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

⑥ Time complexity of -

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

→ Let 'K' be max + value such that

$K^2 \le n$

$K = \sqrt{n}$

$i^2 \le n$

$\therefore \sum i \Rightarrow 1+1+ \ldots K$ times

$i = 1$

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

(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++
}
```

→ For $k = k*2$

$K \Rightarrow \overbrace{1, 2, 4, 8 \ldots n}^{k}$

$a = 1, \quad r = 2$

G.P., $K \Rightarrow \dfrac{a(r^n - 1)}{r-1} \Rightarrow \dfrac{1(2^n - 1)}{1}$

$n = 2^k$

$k = \log n$

| i | j | k |
|---|---|---|
| $n/2$ | $\log n$ | $\log n + \log n$ |
| $(n+2)/2$ | $\log n$ | $\log n + \log n$ |
| ⋮ | ⋮ | ⋮ |
| $n$ | $\log n$ | $\log n * \log n$ |

$n * \log n * \log n$

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

(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);
}
```

→ for :- for (i=1 to n)

we ger j = n times every turn

$\therefore i \times j = n^2$

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

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

$T(n-6) = (n-6)^2 + T(n-5),$

K times

$\vdots$

$T(1) = 1;$

Now subs. each value in T(n)

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

Let

$(n-3k) = 1$

$k = (n-1)/3$

total terms $= k+1$

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

$T(n) \approx n^2 + n^2 + n^2 \; \ldots$ ($k$ times $+1$)

$T(n) \approx k n^2$

$T(n) \approx \frac{(n-1)}{3} \times n^2$

$\because \; T(n) = O(n^3)$

Scanned by TapScanner

⑨ Time Complexity of -

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

→ for:- $i=1$ $\quad j = 1+2 \cdots (n \geq j+i)$

$i=2$ $\quad j = 1+3+5 \cdots$ "

$i=3$ $\quad j = 1+4+7 \cdots$ "

$\vdots$

$M^{th}$ term of AP is

$$T(m) = a + d \times m$$

$$T(m) = 1 + d \times m$$

$$(n-1)/d = m$$

$\therefore$ for

| | |
|---|---|
| $i=1$ | $(n-1)/1$ times |
| $i=2$ | $(n-1)/2$ times |
| $i=3$ | $(n-1)/3$ times |
| $i=n-1$ | $1$ |

we get

$$T(n) = i_1 j_1 + i_2 j_2 \cdots i_{n-1}\ j_{n-1}$$

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

$$= n + n/2 + n/3 \cdots + \frac{n}{n-1} - n \times 1 +$$

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

$$= n \times \log n - n + 1$$

Since $\int \frac{1}{x} = \log x$

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

(10) For the functions, n^k and $c^n$, what is the asymptotic relationship between these functions? Assume that $k >= 1$ and $c > 1$ are constants. Find out the value of c and no for which relation holds.

→ as given $n^k$ and $c^n$

relation between $n^k$ & $c^n$ is

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

$n^k \leq a(c^n)$

$\forall\ n \geq n_0$ &

constant, $a > 0$

for $n_0 = 1$

$c = 2$

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

$\Rightarrow n_0 = 1$ and $c = 2$