

## 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  $f(n) \leq c \cdot g(n)$  for all  $n \geq n_0$  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  $f(n) \geq c \cdot g(n)$  for all  $n \geq n_0$  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  $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$  for all  $n \geq n_0$  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; }  
 $\rightarrow O(n \log n)$

③  $T(n) = \begin{cases} 3T(n-1) & \text{if } n > 0, \text{ otherwise } 1 \end{cases}$   
 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  $T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n > 0, \text{ otherwise } 1 \end{cases}$   
 Using Substitution

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

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

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

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

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

.....

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

$$- 2^2 - 2^1 - 2^0$$

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

$$- 2^2 - 2^1 - 2^0$$

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



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

$$T(n) = 1$$

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,

$$1 + 2 + \dots + x \leq n$$

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

$$\Rightarrow O(x^2) \leq n$$

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

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

$$k^2 \leq n$$

$$k = \sqrt{n}$$

$$i^2 \leq n$$

$$\therefore \sum i \Rightarrow 1+1+ \dots k \text{ times}$$

$$i=1$$

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



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

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

a = 1, r = 2

G.P.  $k \Rightarrow \frac{a(r^n - 1)}{r - 1} \Rightarrow \frac{1(2^n - 1)}{1}$

$n = 2^k$

$k = \log n$

i  
n/2  
(n+2)/2  
⋮  
n

$n + \log n + \log n$

j  
 $\log n$   
 $\log n$   
⋮  
 $\log n$

k  
 $\log n + \log n$   
 $\log n + \log n$   
⋮  
 $\log n * \log n$

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



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 get j = n times every turn

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

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

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

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

⋮

$$T(1) = 1;$$

Now subs. each value in  $T(n)$

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

} K times



Let

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

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

$$\text{total terms} = k+1$$

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

$$T(n) \approx n^2 + n^2 + n^2 \dots (k \text{ times} + 1)$$

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

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

$$\therefore T(n) = O(n^3)$$



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  
       i = 2  
       i = 3  
       ⋮  
       ⋮

j = 1 + 2 --- (n ≥ j + i)  
      j = 1 + 3 + 5 --- "  
      j = 1 + 4 + 7 --- "

M<sup>th</sup> term of AP is

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

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

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

∴ 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

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



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

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

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

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

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

→

As given  $n^k$  and  $c^n$

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

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

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

$$\forall n \geq n_0 \text{ \& } \text{constant, } a > 0$$

$$\text{constant, } a > 0$$

$$\text{for } n_0 = 1$$

$$c = 2$$

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

$$\Rightarrow n_0 = 1 \text{ and } c = 2$$