

## Tutorial-1

Q1. What do you understand by the Asymptotic Notations. Define Different Asymptotic Notation with example.

→ Asymptotic Notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value/limiting value.

Using these analysis we can very well conclude best case, average case, and worst case of an algorithm.

If is input bound, if there is no i/p it will work in constant time.

### Types -

① O Notation → Big O Notation represents the upper bound of the running time of an algorithm.

→ It gives the worst case complexity of an algorithm.

→ It tells us the no of operation an algorithm will make.

→ It tells us how fast an algorithm grows and let us compare it with others.



**Omega Notation** - It represents the lower bound of the running time of an algorithm.  
 → It provides the best case complexity.

Example - If  $T_c = 3n^2 + 5n$  for any algorithm, according to

→ It always indicates the minimum time required for any algorithm

**Theta Notation** - It represents the upper and lower bound of the running time.

→ It is used for analyzing the average case complexity.

Example :  $T_c = 3n^2 + 5n$

$\therefore \Theta(n^2) = \Theta(n^2)$

→ Ignoring constant and remaining part

→  $k_1 n^2 < n < k_2 n^2$

$k_1, k_2 = \text{constants.}$

Q2. What should be the TC of  
 for ( $i = 1$  to  $n$ )  
 {  $i = i + 2$ ;  
 }

For  $i = 1$ :

$i = 1, 2, 4, 8, \dots, n$

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

$$n = 2^k$$

$$k = \log n$$

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

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

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

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

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

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

...

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

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

$$= 3^n \cdot 1$$

$$= 3^n$$

(if  $n \leq 0$  then 1)

$$\rightarrow O(3^n)$$

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

$$T(n) = 1$$

$$\text{if } n \leq 0$$

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

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

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



$$\begin{aligned}
 &= 2^2(2T(n-3)-1)-2-1 \\
 &= 2^3(2T(n-3)-2^2-2^1-2^0) \\
 &\vdots \\
 &= 2^n T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \dots - 2^0 \\
 &= 2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} \dots - 2^0 \\
 &= 2^n - (2^n - 1) \\
 &= 2^n - 2^n + 1 \\
 &= 1
 \end{aligned}$$

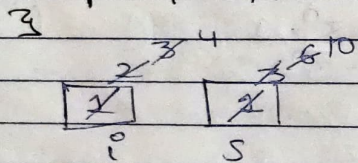
$\rightarrow O(1)$

5. What should be the TC of -;

```

int s=1, i=1;
while(s <= n) {
    i++;
    s = s+i;
    printf("%d\n", i);
}

```



$$s = 1 + 3 + 6 + 10 + \dots \text{upto } n$$

$$\frac{k(k+1)(k+2)}{6} = n$$

$$O(k^3) = n$$

$$\begin{aligned}
 k &= \sqrt[3]{n} \\
 &= O(\sqrt[3]{n})
 \end{aligned}$$

Q6 void function (int n)

```

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

```

3

i	count
<del>1</del>	<del>0</del>
2	1
<del>3</del>	<del>2</del>
4	<del>3</del>

n=10

$$4 \times 10 = 40$$

$$n=20 = 4$$

$$n=25 = 5$$

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

Q7. void function (int n) {

```

    int i, j, k, count=0;
    for(i=1; i*i <= n; i++)
        for(j=1; j*j <= n; j=j*2)
            for(k=1; k <= n; k=k*2)
                count++;
}

```

3

i will execute  $\sqrt{n}$  times =  $\sqrt{n}$  times  
 since  $j = j \times 2$ , and  $k = k \times 2$ , both are increasing exponentially upto n  
 so

$$\begin{aligned}
 TC &= O(\log n \times \log n \times \log n) \\
 &= O(n \log^2 n)
 \end{aligned}$$



Q8. function(int n) {

if (n == 1)

return

for (i = 1; i < n; i++) {

for (j = 1; j < n; j++) {

printf(" + ");

}

}

function(n-3)

}

i-loop n times =  $O(n^2)$

j-loop n times

Q9. void function (int n) {

for (i = 1 to n) {

for (j = 1; j < n; j = j + i)

printf(" + ");

}

}

Inner loop is dependent upon i

i = 1  $\Rightarrow$  j = 1..2..3..4..5..6..7..8...n

i = 2  $\Rightarrow$  j = 1..3..5..7..9..11..13...n/2

i = 3  $\Rightarrow$  j = 1..4..7..10..13... n/3

$T(n) = n + n/2 + n/3 + \dots$

$O(n \log n)$