## Tutorial - 01

Unzila
Sec - I

**Ans 1 →** Asymptotic Notation :-

Mathematical Notation used to describe the running time of an algorithm.

Types of Asymptotic Notation -

1. Big-O Notation : It represent upper bound of Algorithm.

$$f(n) = O(g(n)) \text{ if } f(n) \le c\,g(n)$$

2. Omega Notation : It represent lower bound of Algorithm

$$f(n) = \Omega(g(n)) \text{ if } f(n) \ge c\,g(n)$$

3. Theta Notation : It represents upper & lower bound of Algorithm.

$$f(n) = \Theta(g(n)) \text{ if } c_1 g(n) \le f(n) \le c_2\, g(n)$$

**Ans 2 →**

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

$i = 1, 2, 4, 8, 16, \ldots n$ (It is forming GP)

$$a_n = ar^{n-1}$$

$$n = ar^{k-1} \qquad \because k = \log n + 1$$

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

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

$$\log n = (k-1) \log 2$$

Time Complexity = $O(\log n)$

Ay.

**Ans 3 →**

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

$T(1) = 3T(0)$ $\quad T(0) = 1$

$T(1) = 3 \times 1$

$T(2) = 3T(1) = 3 \times 3 \times 1$

$T(3) = 3T(2) = 3 \times 3 \times 3 \times 1$

$\vdots$

$T(n) = 3 \times 3 \times 3 \ldots \times n = 3^n$

Time Complexity = $O(3^n)$

Ay.

Ans 4 - $T(n) = 2T(n-1) - 1$ if $n>0$, otherwise 1

$T(0) = 1$

$T(1) = 2T(0) - 1$

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

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

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

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

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

$\vdots$

$T(1) = 1$

Time Complexity $= O(1)$

Ay.

Ans 5 -

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

$i = 1$ $s = 1$

$i = 2$ $s = 1+2$

$i = 3$ $s = 1+2+3$

$i = 4$ $s = 1+2+3+4$

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

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

$k^2 > n$

$k > \sqrt{n}$

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

Ay.

Ans 6 -

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

}
```

Loop ends when $i * i > n$

$k * k > n$

$k^2 > n$

$k > \sqrt{n}$

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

Ay.

Ans 7, 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 ++;
            }
        }
    }
}
```

1st loop : $i = n/2$ to $n$ ; $i{+}{+}$

$= O(n/2) = O(n)$

2nd loop : $j = 1$ to $n$, $j = j*2$

$= O(\log n)$

3rd loop: $k = 1$ to $n$, $k = k*2$

$= O(\log n)$

Time Complexity $= O(n * \log n * \log n) = O(n \log^2 n)$

Ay.

Ans 8, function (int n)

```
{
    if (n==1) return;            —— O(1)
    for (i=1 to n){
        for (j=1 to n){          —— O(n²)
            prinf ("*");
        }
    } function (n-3);            —— T(n-3)
}
```

$T(n) = T(n-3) + n^2 \quad , \quad T(1) = 1$

$T(4) = T(4-3) + 4^2 = 1 + 4^2$

$T(7) = T(7-3) + 7^2 = 1 + 4^2 + 7^2$

$T(10) = T(10-3) + 10^2 = 1 + 4^2 + 7^2 + 10^2$

So,

$$T(n) = 1^2 + 4^2 + 7^2 + 10^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6} = n^3$$

Time Complexity $= O(n^3)$

Ay.

Ans 9 →

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

for $1^{st}$ loop :- $O(n)$

for $2^{nd}$ loop :- $O(n \times n) = O(n^2)$

Time Complexity $= O(n^2)$

Ay.

Ans 10 →

$f_1(n) = n^k$

$f_2(n) = c^n \qquad k >= 1, \quad c > 1$

Asymptotic Notation b/w $f_1$ & $f_2$ is Big O ie

$f_1(n) = O(f_2(n))$

$= O(c^n) \qquad n^k \leq c_1 \cdot c^n$

$c_1$ is some constant.

Ay.