Sec-H, 37

Tutorial-1

Ans-1- Asymptotic notation → Asymptotic Notation are the mathematical notation used to describe the running time of an algorithm.

Different types of Asymptotic Notation:-

1. Big-O Notation (O) → It represents upper Bound of algorithm. $f(n) = O(g(n))$ if $f(n) \leq c * g(n)$

2. Omega Notation (Ω) ⇒ It represents lower bound of Algorithm. $f(n) = \Omega(g(n))$ if $f(n) \geq c * g(n)$

3. Theta Notation (Θ) → It represents upper and lower bound of algorithm.

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

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

i = 1
i = 2
i = 4
i = 8
i = 16
i = n

It is forming up

$a_n = ar^{n-1}$

$n = ar^{k-1}$

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

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

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

$\boxed{k = \log n + 1}$

$\begin{pmatrix} a_n = n \\ r = 2 \\ a = 1 \end{pmatrix}$

$O(\log n)$

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

$T(1) = 3T(0)$ $[T(0)=1]$

$T(1) = 3\times1$

$T(2) = 3T(1) = 3\times3\times1$

$T(3) = 3\times T(2) = 3\times3\times3$

$\vdots$

$T(n) = 3\times3\times3 \ldots$

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

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

$T(0) = 1$

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

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

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

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

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

$= 2-1 = 1$

$\vdots$

$T(n) = 1$ $O(1)$

Ans- 5 -

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

$i = 1$ $\quad$ $S = 1$

$i = 2$ $\quad$ $S = 1 + 2$

$i = 3$ $\quad$ $S = 1 + 2\ 2 + 3$

$i = 4$ $\quad$ $S = 1 + 2 + 3 + 4$

$\vdots$ $\quad$ $\vdots$

Loop ends when $S > n$

$1 + 2 + 3 + 4 - - - k > n$

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

$$k^2 > n$$

$$k > \sqrt{n}$$

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

Ans-6- Void function (int n)

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

$i = 1$

$i = 2$

$i = 3$

$i = 4$

$\vdots$

$i = K$

Loop ends when $i * i > n$

$$K * K > n$$

$$K^2 > n$$

$$K > \sqrt{n}$$

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

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 - $i = \frac{n}{2}$ to n, i++

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

- 2nd Nested Loop - j = 1 to n, j = j * 2

$j = 1$

$j = 2$ $= O(\log n)$

$j = 4$

⋮

$j = n$

- 3rd Nested loop - K = 1 to n, K = K * 2

$K = 1$

$K = 2$

$K = 4$ $= O(\log n)$

Total complexity $= O(n \times \log n \times \log n) = O(n \log^2 n)$

Ans - 8 - function (int n)

```
{
    if (n == 1) return;   — 1
    for (int i=1 to n)
    {
        for (int j=1 to n)   — n^2
        {   Printf ("*");
        }
    }
    } function (n-3)   — T (n-3)
}
```

$$\boxed{T(n) = T(n-3) + n^2} \qquad T(1) = 1$$

$\rightarrow T(1) = 1$

$\rightarrow T(4) = T(4-3) + 4^2$

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

$\rightarrow T(7) = T(7-3) + 7^2$

$= 1^2 + 4^2 + 7^2$

$\rightarrow T(10) = T(10-3) + 10^2$

$= 1^2 + 4^2 + 7^2 + 10^2$

So, $T(n) = 1^2 + 4^2 + 7^2 + 10^2 - - - - n^2 = \frac{n(n+1)(2n+1)}{6}$

$= O(n^3)$

also for terms like T(2), T(3), T(5)

So, $T(n) = O(n^3)$

Ans-9- Void function (int n)

```
{
    for (int i=1 to n) — n
    {
        for (j=1 ; j<=n ; j =j+1) — n
        {
            Printf ("*");
        }
    }
}
```

$i=1$ — $i=1$ to n

$i=2$ — $j=1$ to n

$i=3$ — $j=1$ to n

$i=4$ — $j=1$ to n

So, for i upto n it will take n^2

So, $T(n) = O(n^2)$

Ans-10- $f(n) = n^K$, $f_2(n) = C^n$

$K >= 1$, $C > 1$

Asymptotic relationship between f_1 and f_2 is Big O i.e, $f_1(n) = O(f_2(0)) = O(C^n)$

is $n^K \leq C_1 * C^n$ [C_1 is some constant]