Tutorial :→ 1

1. Asymptotic notation used to analyse an algorithm when input is large.

2. Big O Notation :→ It represents the upper bound of the running time of an algorithm It gives the worst Case of complexity of an algorithm.

Ex: $F(n) = O(g(n))$ if $0 \leq F(n) \leq g[g(n)]$
if $n \geq n_0$ for $cg(n)$.

3. Small O notation : It is denoted by o It is used to describe an upperobound that cannot be tight.
Ex: $F(n) = o(g(n))$ if $F(n) < g(n) \forall n$

4. Small omega Notation : It is used to describe a lower bound of $f(n)$ and it is denoted by ω.
Ex:- $f(n) = \omega(g(n))$ if $f(n) > g(n)$

2.

```
For( i=1; to n , i = i * 2; )
for(i = 1; i<=n; i *= 2)
```

1, 2, 4 ---- n

$n = ar^{k-1}$

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

$2^k = 2n$

$\log_2 2^k = \log_2 2n$

$k = \log_2 2 + \log_2 n$

$k = 1 + \log_2 n$

$k = O(\log n)$

3. $T(n) = \{ 2T(n-1) - 1 \text{ if } n > 0 \text{ otherwise } 1 \}$

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

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

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

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

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

$\Rightarrow 2^n - 2^n + 1 = 1$

$\therefore T = O(1)$

4.
```
int i=1, s=1.
while (s<=n)
{ i++; s=s+i;
  Print("#");
}
```

$1 <= n$

$s = 1+2, \quad s = 3+3, \quad s = 6+4, \quad s = 3, \quad 6, \quad 10$

$1, 3, 6, 10 \text{ ——}$

$1 + 2 + 3 + \text{——} \; K$

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

$K = \frac{-1 \pm \sqrt{1+8n}}{2} \Rightarrow K = \frac{1}{2}\sqrt{1+8n}$

$K = \sqrt{n}$

$K = O(\sqrt{n}) \qquad T = O(\sqrt{n})$

5. Function (int n)
{ if (n == 1) return;
for (i = 1 to n) {
for (j = 1 to n) { printf ("*"); }
} Function (n-3);
}

1, 2, 3, ——— n,

$n = i + (K-1) \times 1$

$n = K$, $T = O(n)$

1, 2, 3 ——— n

$n = 1 + (K-1) n$

$n = k$, $T = O(n)$

$T = O(n^2)$

6. void function (int n)
{ for (i = 1 to n)
{ for (j = 1, j <= n; j = j + i)
{ Pf ("*"); } }

1, 2, 3 ——— n;

$n = 1 + (K-1) 1$ $n = k$

$T = O(n)$

1, 2, 3 — — — . n

$T = O(n)$

$T = O(n^2)$