Assignment :- 01

Ans 01:- Asymptotic Notation :- It is the mathematical notation used to describe the running time of an Algorithm.

Different types of Notation are:-

(i) Big O Notation (Big - O) → It represents the upper bound of the algorithm.

$f(n) = O(g(n))$ iff $f(n) \le c(g(n))$

(ii) Omega Notation (Ω) → It represents the lower bound of the algorithm.

$f(n) = \Omega(g(n))$ iff $f(n) \ge c(g(n))$

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

$f(n) = \Theta(g(n))$ iff. $c_1 g(n) \le f(n) \le c_2 g(n)$

Ans. 02)

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

$i = 1, 2, 3, 4, 5, 6, 7$

i value $= 2, 4, 8, 16, \ldots$

forms a GP; $a_n = a(2)^{n-1}$ $a = 1$

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

$\log n = \log 2^{k-1}$ — taking log both sides

$\log n = k-1$

$\boxed{k = \log n + 1} \implies \boxed{T(n) = O(\log n).}$ — Ans.

Ans 03)

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

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

$T(2) = 3T(1) = 3 \cdot 3T(0) = 3^2$

$T(k) = 3^k$

$T(n) = 3^n$

$\boxed{T(n) = O(3^n).}$ — Ans.

Ans 04) $T(n) = 2T(n-1) - 1$; if $n>0$; Otherwise 1

$T(0) = 1$

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

$T(2) = 2T(1) - 1 = 2(2(T(0) - 1)) - 1 = 2^2T(1) - 5$

$\vdots$

$T(n-k) = T(n-k) - 2^k ! \ldots 2^0$

Substituting $K=n-1$

$T(n) = 2^{n-1}T(1) - [2^0 + 2^1 + \ldots 2^{n-2}]$

$= 2^{n-1} \times 1 - [2^{n-1} - 1]$

$T(n) = 1$

→ Ans

Ans 05)

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

i = 1 2 3 ---- loop ends

s = 1 1+2 1+2+3 ---- while s > n

Hence;

$s > n$

$1+2+3+4+\ldots+K > n$

$\frac{K(K+1)}{2} > n$ $\Rightarrow K > \sqrt{n}$

$K^2 > n$

$\boxed{T(n) = O(\sqrt{n}).}$ — Ans

Ans 06) void function (int n)

{

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

count++;

i = 1, 4, 9, 16 —— till i <= n

Hence;

$K*K < n$

$\boxed{T(n) = O(\sqrt{n})}$ — Ans

Ans. 7)

```
void function (int n)
{
 int i, count=0;
 for (int i=n/2; i<=n; i++)
 {
 for (j=1; j<=n; j*2=j)
  {
   for (K=1; K<=n; K=K*2)
     count++;
  }
 }
```

1st loop:-
$i = n/2$ to n; i++
$T(i) = O(n)$

2nd loop:-
$j = 1$ to n; $j*2$
$T(j) = O(\log n)$

3rd loop:-
$k = 1$ to n; $k*2$
$T(k) = O(\log n)$

Hence:-
$T(n) = T(i) \times T(j) \times T(k)$
$= O(n) \times O(\log n) \times O(\log n)$
$\boxed{T(n) = O(n \log^2 n).}$ — Ans.

Ans 8)

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

{ Hence;
Relⁿ:- $T(n) = T(n-3) + n^2$; $T(1) = 1$
$T(4) = T(1) + (4)^2 = 1^2 + 4^2$
$T(7) = T(4) + n^2 = 1^2 + 4^2 + 7^2$
⋮
$T(n) = 1^2 + 4^2 + 7^2 + \dots n^2$ $\boxed{T(n) = O(n^3).}$ — Ans
$= \frac{n(n+1)(2n+1)}{6} = \boxed{n^3}$ - nearest

Ans. 9)

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

n — i = 1   2 . . . n
n — j = 1 to 1   1 to 2 . . . 1 to n

Hence;

$\boxed{T(n) = O(n^2).}$ — Ans

Ans 10)

$f_1(n) = n^k$; $k \geq 1$
$f_2(n) = c^n$. $c > 1$

Asymptotic relⁿ b/w f_1 & f_2 :-

Big-O ⟶ $f_1(n) = O(f_2(n)) = O(c^n)$

and $n^k \leq G * c^n$ [G is some constant].