Tutorial - 1 (DAA)

Ans 1) **Asymptotic Notation:** Asymptotic Notation are the mathematical notations used to describe the running time of an algorithm.

Different forms 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.

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

Ans 2)

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

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

$a_n = n$, $r = 2$, $a = 1$

It is forming GP

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

$n = ar^{k-1}$

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

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

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

$O(\log n)$

Ans3]

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

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

$T(1) = 3 \times 1$

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

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

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

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

Ans4]

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

$T(0) = 1$

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

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

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

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

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

$= 2 - 1 = 1$

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

Ans5]

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

$\pm$

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

Loop ends when $S > n$

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

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

$$k^2 > n$$

$$k > \sqrt{n}$$

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

Ans-6

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

$i = 1$
$i = 2$
$i = 3$
$i = 4$

Loop ends when $i * i > n$

$k * k > n$ $\quad i = k$

$k^2 > n$

$k = \sqrt{n}$

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

Ans7

```
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 nested Loop :

$j = 1$ to n, $j = j * 2$

$j = 1$
$j = 2$
$j = 4$
$j = n$

$= O(\log n)$

3rd nested Loop :-

$k = 1$ to n, $k = k * 2$

$k = 1$
$k = 2$
$k = 4$

$= O(\log n)$

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

Ans6

```
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)
```

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

$T(1) = 1$

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 — j = 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, $T(n) = O(n^2)$.

Ans10 $f(n) = n^k$ $f_2(n) = c^n$

$k >= 1, c > 1$

Asymptotic relationship between

is Bigo i.e $f_1(n) = O(f_2(n)) = O(c^n)$

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