Tutorial-1

Ans-1 Asymphotic Notation :- Asymphotic Notation are the mathematical notation used to describe the running time of an algorithm

Different types of Asymptotic Notations:-

1) Big-O Notation (O):- It represent lower bound of Algorithm.
$f(n) = \Omega(g(n))$ if $f(n) \geq C * g(n)$.

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 represends upper & 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

It is forming G.P

$an = ar^{n-1}$

$n = ar^{k-1}$

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

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

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

$k = \log n + 1$

$O(\log n)$

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

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

$T(1) = 3 \times 1$

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

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

⋮

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

$= O(3^n)$

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

$T(0) = 1$

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

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

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

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

$= 2 - 1 = 1$

⋮

$T(n) = 1$

$O(1)$

Ans-5

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

i=1 s=1

i=2 s=1+1

i=3 s=1+2+3

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

$k > n$

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

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

$$k^2 > n$$

$$k > \sqrt{n}$$

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

Ans 6

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

loop ends when $i*i > n$

$k*k > n$

$k^2 > n$

$k > \sqrt{n}$

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

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++

$= O(\frac{n}{2}) = O(n)$

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

$= O(\log n)$

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

$= O(\log n)$

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

Ans-3

```
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)
}
```

$T(n) = T(n-3) + n^2$

$T(1) = 1$

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

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

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

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

So, $T(n) = 1^2 + 4^2 + 7^2 + 10^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6} = O(n^3)$

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

Ans-9

```
void function(int n)
{
  for(int i=1 to n)        — n
  {
    for (int 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, $T(n) = O(n^2)$

Ans-10 $f(n) = n^k$, $f_2(n) = c^n$

$1<>=1$, $c>1$

Asymptoic relationship b/w f_1 & f_2 is Big O ie...

$f_1(n) = O(f_2(n)) = O(c^n)$ is $n^k \leq G * c^n$ [G is some constant].