Tutorial - 1 (DAA)

Ans-1: Asymptotic Notation: Asymptotic Notation are the mathematical notations used to describe the remaining time of an algorithm.

Different types of Asymptotic Notations

1) Big-oh- 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 $f(n) > c * g(n)$

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

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

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 GP

$a_n = ar^{n-1}$

$n = ar^{k-1}$

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

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

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

$k = \log n + 1$

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

$O(\log n)$

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

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

$T(1) = 3 \times 1$

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

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

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

$= 3^n$

$= O(3^n)$

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

$T(0) = 1$

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

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

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

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

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

$\Rightarrow 2 - 1 = 1$

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

Ans-5)
```
int i=1, S=1
while (S<=n)
{
  i++;
  S = S+i;
  Printf ("#");
}
```

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

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

$i=1$
$i=2$
$i=3$
$i=4$
⋮
$i=k$

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, $i++$

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

- 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 $= O(n \times \log n \times \log n) = O(n \log^2 n)$

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)
```

$T(n) = T(n-3) + n^2$ $\quad T(1) = 1$

→ $T(1) = 1$

→ $T(4) = T(4-3) + 4^2$

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

→ $T(7) = T(7-3) + 7^2$

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

→ $T(10) = T(10-3) + 10^2$

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

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

also for terms like $T(2), T(3), T(5)$ $\quad = O(n^3)$

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 ("*");
    }
  }
}
```

$i=1$ — $j = 1$ 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: $f(n) = n^K$ $\quad f_2(n) = c^n$

$K >= 1, c > 1$

Asymptotic relationship between f_1 and f_2

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

is $n^K \leq G * c^n$

[G is some constt]