Tutorial - 1 (DAA)

Name → Manan Sharma
Sec → H
ClassRollNo → 34
University R.N → 2016841

Soln1 → Asymptotic notation are the mathematical notation used to describe the running time of algorithm.

Different types of Notation are :

1 → **Big - O - notation** → It represents upper bound of algorithm
$f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$

2 → **Omega Notation** → It represents tight lower bound of algorithm
$f(n) = \Omega(g(n))$ iff $f(n) \geq c \cdot g(n)$

3 → **Theta Notation (Θ)** → It represents upper lower bound of an algorithm.
$f(n) = \Theta(g(n))$ iff $c_1 g(n) \leq f(n) \leq c_2 g(n)$

Soln2 →

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

i = 1
i = 2
i = 4
i = 8
i = n

It is forming a GP.

$a_n = a r^{n-1}$
$n = a r^{k-1}$
$n = 1 \times 2^{k-1}$
$\log n = \log 2^{k-1}$
$\log n = (k-1) \log 2$
$k = \log n + 1 \rightarrow O(\log n)$.

Soln3 → $T(n) = 3T(n-1)$ if $n>0$, otherwise 1

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

$T(1) = 3 \times 1$

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

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

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

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

Soln4 → $T(n) = 2T(n-1) - 1$ if $n>0$, otherwise 1

$T(0) = 1$

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

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

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

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

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

Soln5 →

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

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

$s > n$

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

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

$k^2 > n$

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

Soln 6 → void function (int n)

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

$i = 1$
$i = 2$
$i = 3$
$i = 4$
;
$i = K$

Loop ends when $i * i > n$

$K * K > n$

$K^2 > n$

$K > \sqrt{n}$ $\quad O(K) = \sqrt{n}$

Soln 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 → $i = n/2$ to n, $i++$

$= O(n/2) = O(n)$

→ 2nd nested loop → $j = 1$ to n, $j = j * 2$

$j = 1$
$j = 2$
$j = 4$ $\quad = O(\log n)$
$j = n$

→ 3rd nested loop → $k = 1$ to n, $k = k * 2$

$k = 1$
$k = 2$ $\Rightarrow O(\log n)$
$k = 4$

T.C $= O(n * \log n * \log n) = O(n \log^2 n)$

Soln8 →

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

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

$T(1)=1$, $T(4) = T(4-3)+4^2 \rightarrow T(1)+4^2 = 1^2+4^2$

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

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

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

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

Soln9 →

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

i=1 upto j=1 to n
i=2 upto j=1 to n
i=4 upto j=1 to n
i=3 upto j=1 to n

so, for i upto n it will take n^2 so $T(n) = O(n^2)$

Soln10 → $f_1(n) = n^k$, $f_2(n) = c^n$ $\quad k>=1, c>1$

Asymptotic relationship b/w f_1 & f_2

is Bigohie $f_1(n) = O(f_2(n)) = O(c^n)$

in $n^k \leq c * c^n$ [c is some constant]