Tutorial - 1 (DAA)

Ans 1 → **Asymtotic notations**: Asymtotic notation are the mathematical notations used to describe the running time of an algorithm.

Different types of notations are.

1) **Big - Oh (O)** – It represents upper bound of an operation
$f(n) = O(g(n))$ if $f(n) \le c * g(n)$

2) **Big Omega (Ω)** : represents lower bound of an algorithm
$f(n) \le \Omega(g(n))$ if $f(n) \ge c * g(n)$

3.) **Theta** (Θ): It represents upper & lower bound of algorithm
$f(n) = \Theta(g(n))$ if $c_1 g(n) \le f(n) \le c_2 g(n)$.

Ans 2) → for (i=1 to n){
i = i*2
}

$i = 1, 2, 4, 8, 16 \dots n$.

It is forming G.P

$a_n = ar^{n-1}$

$n = ar^{k-1}$ $\left(\begin{array}{c} a_n = n \\ r = 2 \\ a = 1 \end{array}\right)$

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

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

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

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

$O(\log n)$

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

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

$T(1) = 3 \times 1$

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

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

$\vdots$

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

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

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

$T(0) = 1$

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

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

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

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

Ans 5)

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

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

loop ends when $s > n$

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

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

$k^2 > n$ $\qquad = O(\sqrt{n})$

$k > \sqrt{n}$

Ans 7) void function(int n) {
int i, j, k; count = 0;
for (int 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 ; $j = j * 2$

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

3rd nested loop - $k = 1$ to n, $k = k * 2$
$k = 1$
$k = 2$
$k = 4$ $= O(\log n)$

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

Ans 8) function(int n) {
if (n == 1) return; — 1
for (int i = 1 to n) — n^2
{ for (int j = 1 to n)
printf("*");
}
function (n - 3) — $T(n-3)$

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

$T(1) = 1$

$T(1) = 1$

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

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

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

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

for terms like $T(2), T(3), T(5)$ $O(n^3)$

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++) — 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'll take n^2

So, $T(n) = O(n^2)$

Ans 10) $f(n) = n^k$ $f_2(n) = c^n$ $k >= 1, c > 1$

Asymtotic relationship b/w f_1 & 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 const.]