Name- Nidhi Rathore Tutorial 1

Section- D

Roll No - 2014422

Q1. What do you understand by Asymptotic notations. Define with examples?.

Ans1. **Asymptotic Notations** - are methods/languages using which we can define the running time of the algorithm based on input size.

1) Big-oh Notation (O)-

Big-Oh - gives an upper bound of a function f(n) to within a constant factor.

$f(n) = O(g(n))$ such that $f(n) \leq c \cdot g(n)$ $\forall$ $n \geq n_0$ some constant $c > 0$.

Eg- $f(n) = n!_0 = n.(n-1)(n-2) \cdots 1$

$= 1 \times 2 \times \cdots \times n \leq n \times n \times n \cdots$

$n!_0 \leq n^n$

$f(n) = O(n^n)$

2) Big Omega Notation Ω - gives lower bound for a function f(n) to within a constant factor.

$f(n) = \Omega g(n)$

if $\Omega(g(n))$

$\forall$ $n \geq n_0$ & some constant $c > 0$.

Eg- $f(n) = n!_0$

$1 \times 1 \times 1 \cdots \times 1 \leq 1 \times 2 \times 3 \cdots \times n$.

$1 \leq n!_0$

$f(n) = \Omega(1)$

3) Theta Notation –

Theta gives both tight upper and lower bound for a function.

$f(n) = \theta g(n)$ iff $c_1(g(n)) \leq f(n) \leq c_2(g(n))$

$\forall \; n \geq \max(n_1, n_2)$

$c_1, c_2 > 0$.

Ans2. for (i = 1 to n) { i = i * 2 }

Suppose loop ends in k steps

$n = 1 \times 2 + 2 \times 2 + 4 \times 2 + \ldots$

$n = 2^1 + 2^2 + 2^3 \ldots 2^k$

Sum of GP $= \frac{a(r^k - 1)}{r - 1}$

$a = 2$

$r = 2$

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

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

$\frac{n+1}{2} = 2^k$

$\log_2\left(\frac{n+1}{2}\right) = \log_2 2^k$

$\log_2\left(\frac{n+2}{2}\right) = k$.

$k = O\left(\log_2\left(\frac{n+2}{2}\right)\right) = O(\log n)$

$\boxed{T.C = O(\log n)}$

Ans3. $T(n) = \begin{cases} 3T(n-1), & n > 0 \\ 1, & n \leq 0 \end{cases}$

$T(n) = 3T(n-1)$ — ① .

$T(0) = 1$.

putting $n = n-1$ in ①

$T(n-1) = 3T(n-2)$ — ② .

putting value of $T(n-1)$ in ① –

$T(n) = 3 \cdot 3T(n-2)$ — ③ .

Putting $n = n-2$ in ① .

$T(n-2) = 3T(n-3)$ — ④ .

Putting value of $T(n-2)$ in ③ –

$T(n) = 3 \cdot 3 \cdot 3\, T(n-3)$ — (5). (let k steps)

!

$T(n) = 3^k : T(n-k)$ — (6).

$T(0) = 1$.

$n-k = 1$,

$k = n-1$. — (7)

putting in (6) -

$T(n) = 3^{n-1}\, T(n-k+1)$

$T(n) = 3^{n-1}\, T(1)$ — (8).

put $n=1$ in (1).

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

$= 3$

equation (8).

$T(n) = 3^{n-1} \cdot 3 = 3^n$

T.C. $= O(3^n)$

Ans 4. $T(n) = 2T(n-1) - 1$

$T(0) = 1$.

Using Backward substitution —

$T(n) = 2T(n-1) - 1$ — (1).

putting $n = n-1$

$T(n-1) = 2T(n-2) - 1$ — (2)

putting value of $n-1$ in (1)

$T(n) = 2 \cdot (2T(n-2) - 1) - 1$

$T(n) = 2 \cdot 2T(n-2) - 2 - 1$ — (3).

putting $n = n-2$ in (1) -

$T(n-2) = 2T(n-3) - 1$ — (4).

putting $T(n-2)$ value in (3) -

$T(n) = 2 \cdot 2 \cdot 2T(n-3) - 4 - 2 - 1$

if k is steps.

$T(n) = 2^k \cdot T(n-k) - k$ — (5)

$(1+2 \ldots 2^k$

$T(0) = 1$ — (5)

$n-k = 1$

$k = n-1$

putting in (5) -

$T(n) = 2^{n-1} T(n-n+1) - (1+2+4 \cdots 2^{n-1-1})$

$T(n) = 2^{n-1} T(1) - (1+2+\cdots 2^{n-2})$

$T(n) = 2^{n-1} T(1) - \frac{(2^{n-1}-1)}{2-1}$

$T(n) = 2^{n-1} T(1) - (2^{n-1}-1)$ — (6).

$n=1$ in equation (1) -

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

$T(1) = 1$

$T(n) = (2^{n-1} - 2^{n-1} + 1)$

$T(n) = O(1)$

Ans 5.

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

Suppose loop end at k^{th} step.

s after 1st iteration = 1+2

" " 2nd " = 1+2+3

.

$1+1+2+3 \ldots k = n$.

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

$k^2 + k = 2(n-1)$

eliminating power of k and constants.

$k^2 = n$

$k = \sqrt{n}$

$T.C = O(\sqrt{n})$

Ans 6.

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

loop ends $i^2 > n$

$i > \sqrt{n}$

T.C = $O(\sqrt{n})$

Ans 7.

```
void fun (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++;
}
```

T.C loop 1 = $O(n/2) = O(n)$

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

T.C loop 3 = $O(\log n)$

Ans 8.

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

T.C = $O(n^2)$

Ans 9 - void fun(int n)

```
{for (i = 1 to n)
 {
  for(j=1; j<=n; j=j*1)      } O(n²)
    printf("*");
 }
}
```

T.C = $O(n^2)$

Ans 10. n^k is $O(c^n)$

$n^k = O(c^n)$.