SANGHMITRA KANDPAL
50
ML.

Assignment – 1

Q.) What do you ——— ——— ——— ——— ?
with examples.

Asymptotic notations are mathematical tools used to describe the limiting behavior of functions as their i/p size approaches infinity. They are used characterize the efficiency of algorithms.

(1.) **Big O notation** —
represents the upper bound of a fn; provides an upper limit on the growth rate of a fn.
eg — $O(n^2)$
worst case time complexity grows quadratically with i/p size.

(2.) **Omega (Ω)** —
lower bound of a fn; provides lower limit on the growth rate of a fn.
eg — $\Omega(n)$.
worst-case TC grows linearly with i/p size.

(3.) **Theta (Θ)** —
represents both upper & lower bound, provides a tight bound on the growth rate of a fn.
eg — $\Theta(n)$
worst-case TC grows linearly.

(4.) **small o notation** —
represents an upper bound that is not asymptotically tight.
eg — $o(n^2)$
W.C TC grows slower than quadratically.

(5) Little ω notation —
lower bound that is not asymptotically tight.
eg — $\omega(n)$

2.) The complexity —
for (i=1 to n) { i = i*2 }
So,
```
for(i=1; i<=n; i=i*2)
{
    ---- (loop)
}
```
$TC = O(\log n)$ Ans.

3.) $T(n) = ?$
$$T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$$

Sol ⇒ $T(n) = 3T(n-1)$
now sub. $T(n-1)$
$T(n) = 3 \cdot 3T(n-2)$
$= 3^2 T(n-2)$

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

After K steps —
$T(n) = 3^k T(n-2)$ continue till $n-k=0$; so;
$n-k=0$
$\boxed{K=n}$
$T(n) = 3^n T(0)$
$\boxed{T(n) = 3^n}$ Ans.

4) $T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n > 0 \\ 1 & \text{otherwise.} \end{cases}$

Sol $\Rightarrow$ $T(n) = 2T(n-1) - 1.$

sub. $T(n-1)$ —

$$T(n) = 2\left(2T(n-2) - 1\right) - 1$$
$$= 2^2 \cdot T(n-2) - 2 - 1$$

continue to sub. —

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

After K steps —

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

$$n - k = 0$$

$$\boxed{n = k}$$

so putting $K = n$;

$$T(n) = 2^n T(0) - \left(2^{n-1} + 2^{n-2} + \cdots 2 + 1\right)$$

$$T(n) = 2^n - \left(2^{n-1} + 2^{n-2} + \cdots 2 + 1\right)$$ Ans.

5)
```
int i=1; s=1;
while (s<=n)
{
  i++;
  s= s+i;
  printf ("#");
}
```

— $(1+2+3+ \cdots i)$

— $\left[(i+1)/2 > n \right]$

$$T(n) = O(i^2)$$
$$\Rightarrow O(n^2)$$ Ans.

6.) void function (int n)

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

loop reverse till $\sqrt{n}$ times because —

$i = \sqrt{n}$; $i \times i = n$;

Time complexity = $O(n)$ Ans.

7.) void function (int n)

```
{
  int i, j, k, count=0;
  for (i=n/2; i<=n; i++)            — (n/2 times)
  {
    for (j=1; j<=n; j=j*2)          — (log_2 n times)
    {
      for (k=1; k<=n; k=k*2)        — (log_2 n times)
        count++;
    }
  }
}.
```

So, TC —

$\Rightarrow O(n) \times O(\log n) \times O\log(n)$

$\Rightarrow O(\log^2 n)$ Ans.

8) function(int n)
{
if (n==1)
return;
for (i= 1; i<=n; i++) ——— (n times)
{
for(j=1; j<=n; j++) ——— (n times)
{
printf("*");
}
}
function(n-3); ——— recursive call so (n-3)
}.

TC ——

$= O(n \times n \times (n-3))$

$= O(n/3)$

$= O(n^2 \times n/3) \Rightarrow O(n^3)$ Ans.

9.) void function(int n)
{
for (i=1; i<=n; i++) ——— (n times)
{
for (j=1; j<=n; j=j+i) ——— (n/i times)
{
printf("*");
}
}
}.

TC —

$\Rightarrow O(n) \times O(n/i)$

loop reverse $i = 1$ to n.

$\therefore O(n^2)$ Ans.

10.) Asymptotic relationship b/w n^k & c^n —

n^k grows polynomially with n, while c^n grows exponentially. The exp. growth dominates as n increases.

Specifically, c^n grows faster than n^k for sufficiently larger values of n.

$$c^n > n^k$$

Taking the log base c on both sides —

$$n > \log_c(n^k) = k \cdot \log_c(n)$$

now,

$$n > \frac{k}{\log_c(k)}$$

now, $c > \frac{k}{\log_c(k)}$ & $n_0 = \frac{k}{\log_c(k)}$,

- c^n grows faster than n^k.
- c must be greater than $\frac{k}{\log_c(k)}$ & relationship holds when n is sufficiently large. Ans.

—x—x—x—x—