Design and Analysis of Algorithm

Tutorial - 1

Answer - 1) **Asymtatic Natation** - Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the inputs tends towards a particular value or a limiting value.

(1) **Big-O Notation** - It represents the upper bound of the running time of an algorithm.

$f(n) = O(g(n))$ such that $f(n) \leq c g(n)$

(2) **Omega notation** - It represents the lower bound.

$f(n) = \Omega(g(n))$ such that $c g(n) \leq f(n)$

(3) **Theta notation** - It represent the upper and lower bound.

$f(n) = \Theta(g(n))$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$

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

It will run till

$i = n$

$2^k = n$

$\boxed{k = \log_2 n}$

i
1
2
2^2
2^3
2^4
⋮
2^k

Hence

$T.C. = O(\log_2 n)$

Answer-3)

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

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

$T(1) = 3\times 1$

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

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

$\vdots$

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

$= 3^n$

$= O(3^n)$.

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

$T(0) = 1$

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

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

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

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

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

$= 2-1 = 1$

$\vdots$

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

Ans- 5)

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

$i=1$ $\quad s=1$

$i=2$ $\quad s=1+2$

$i=3$ $\quad s=1+2+3$

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

$\vdots$

k

loop ends. $s>n$

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

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

$$k^2>n$$

$$k>\sqrt{n}$$

$$=O(\sqrt{n})$$

Ans-6)

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

$i=1$

$i=2$

$i=3$

$\vdots$

$i=k$

Condition terminates at.

$$i*i>n$$

$$k*k>n$$

$$k^2>n$$

$$k>\sqrt{n}$$

$$=O(\sqrt{n})$$

Answer-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 ++
}
```

→ loop-1 $i = \frac{n}{2}$ to n

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

→ loop-2 (nested) $j=1$ to n, $j=j*2$

$j=1$
$j=2$
$j=4$
$j=n$

$= O(\log_2 n)$

→ loop-3 (nested)

$= O(\log_2 n)$

Total Complexity $= O(n \times \log_2 n \times \log_2 n) = O(n \log_2^2 n)$

Ans-8)

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

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

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

$= 1^2 + 4^2$

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

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

$T(10) = 1^2 + 4^2 + 7^2 + 10^2$

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

Also for term 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=j+1)    — n
      { printf("*");
      }
  }
}
```

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

Ans-10)

$f_1(n) = n^k$ $f_2(n) = c^n$.

$k >= 1$, $c > 1$

Asymptotic relationship between f_1 and f_2 is Big-Oh i.e.

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

is $n^k \leq C_1 * c^n$ [C_1 is some constant]