Date.
Page No.

ASSIGNMENT -01

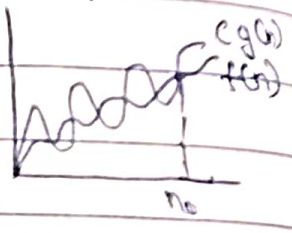
Q1 Asymptotic notations are used to tell the complexity of the algorithm when the input is very large.

(i) Big O Notation (O) →

$$f(n) = O(g(n))$$

It describes that function (g(n)) is tight upper bound of func f(n). iff $f(n) \leq c * g(n)$

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



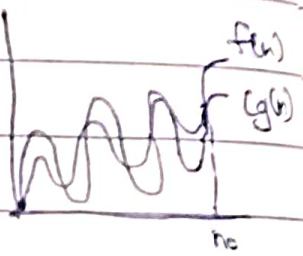
(ii) Big Omega (Ω) Notation →

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

It describes that function g(n) is tight lower bound of func(f(n)).

iff $f(n) \geq c * g(n)$

$\forall\ n \geq n_0$ and for some constant $c > 0$



(iii) Thota Notation(Θ) →

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

It describes that theta gives both tight upper bound and tight lower bound. $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

iff $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

$\forall\ n > \max(n_1, n_2)$ and some constant

$c_1 > 0$ && $c_2 > 0$

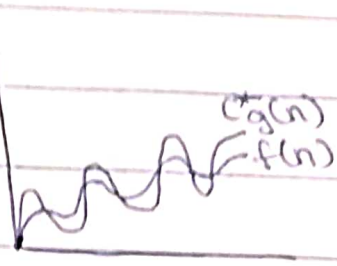
Date.
Page No.

(iv) Small (o) Notation →

$f(n) = o(g(n))$

It describes that $g(n)$ is upper bound of $f(n)$. iff $f(n) < c * g(n)$

$\forall\ n > n_0$ and $c > 0$

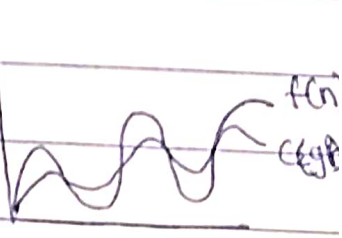


(v) Small omega (ω) Notation →

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

It describes that $g(n)$ is lower bound of func $f(n)$. iff $f(n) > c * g(n)$

$\forall\ n > n_0$ and $c > 0$



2)

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

i = 1, 2, 4, ..., n/2, n

∴ Acc. to G.P

$n = ar^{k-1}$

$n = 1.2^{k-1}$

$2^k = 2n$

Logging on both sides

$\log_2(2n) = k . \log_2 2$

$k = \log_2(2) + \log_2(n)$

$k = \log_2(n) + 1$

$k = \log_2 n$

∴ Time complexity $= O(\log_2 n)$

Date.
Page No.

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

$\therefore T(0) = 1$

$T(1) = 3T(0) = 3\times1$

$T(2) = 3T(1) = 3\times3\times1$

$T(3) = 3T(2) = 3(2\times3\times1) = 3\times3\times3\times1$

$\therefore T(n) = 3^n$

$\therefore$ Time complexity $= O(3^n)$ Ans

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

$\therefore T(0) = 1$

$T(1) = 2T(0) - 1 = 2.1 - 1 = 1$

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

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

$\therefore T(n) = 1$

$\therefore$ Time complexity $= O(1)$ Ans

5)

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

$i = 1, 2, 3, 4, \dots$

$s = 1, 3, 6, 10, \dots n$

$\therefore$ sum of first n natural no.

$$s = \frac{i(i+1)}{2}$$

Date.
Page No.

Also $s \leq n$

$\therefore \frac{i(i+1)}{2} \leq n$

$i^2 + i \leq 2n$

$i^2 < n$

$\therefore i < \sqrt{n}$

$\therefore$ Time complexity $= O(\sqrt{n})$

6)

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

$i=1$	$i*i \leq n$
$i=2$	$2*2 \leq n$
$i=3$	$3*3 \leq n$

$\therefore \quad i \leq \sqrt{n}$

$\therefore$ Time complexity $= O(\sqrt{n})$

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 ++ ;
}
```

Date.
Page No.

i	j	k	times
$n/2$	$1-n$	$1-n$	$(\log_2 n)^2$
$n/2+1$	$1-n$	$1-n$	$(\log_2 n)^2$
⋮			
n	$1-n$	$1-n$	$(\log_2 n)^2$
			$\frac{n}{2}(\log_2 n)^2$

$\therefore$ $\boxed{T.C = O(n(\log_2 n)^2)}$

8.)
```
function (int n){
    if (n==1) return;
    for( i = 1 to n){                    ——→ O(n)
        for (j = 1 to n){                ——→ O(n²)
            printf ("*");
        }
    }
    function(n-3);                       ——→ T(n-3)
}
```

$\therefore$ $T(n) = T(n-3) + O(n^2) + O(n)$

$\therefore$ $\boxed{T(n) = T(n-3) + O(n^2)}$ ~~tw~~

$\therefore$ $\boxed{T.C = O(n^2)}$ $\forall\ n \geq 2$

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

Date.
Page No.

i	fi	times
1	1-n	n
2	1-n	n/2
3	1-n	n/3
⋮		
n	1-n	1

n*log n

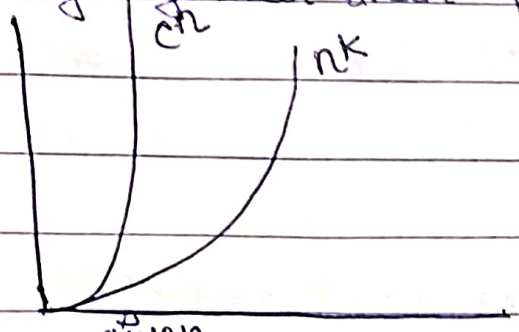
∴ Time complexity = O(n log n)

10] n^K and c^n

$k >= 1$ and $c > 1$

$O(c^n) > O(n^K)$

This is because exponential time complexity is always greater than polynomial time complexity



given

~~Given~~, then conditions does not give any value for c and n.