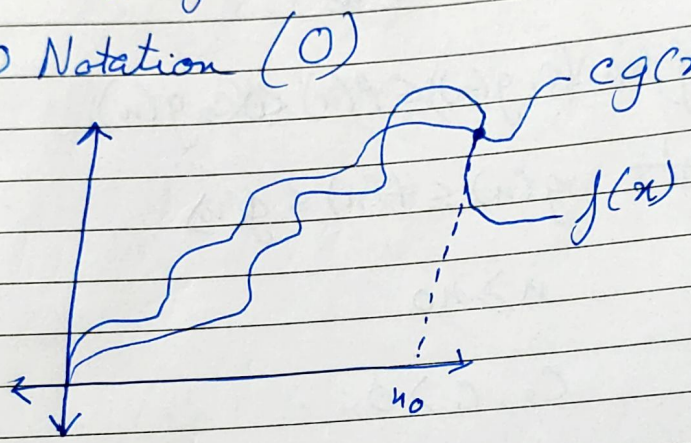
Date
Page

Q-1 Ans

Asymptotic Notation may be defined as the notation used for defining time complexity of the programs where input is very large.

Different type of Notation are.

1. Big O Notation (O)



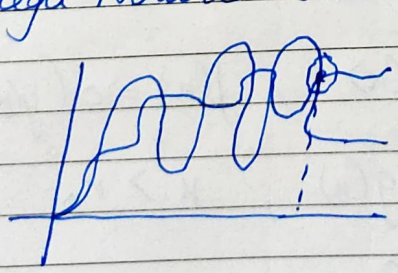
$f(n) = O(g(n))$

iff $f(n) \leq cg(n)$

$\forall\ n_0 \leq n,\ c > 0$

~~2. Big Theta Notation (Θ)~~

2. Omega Notation (Ω)



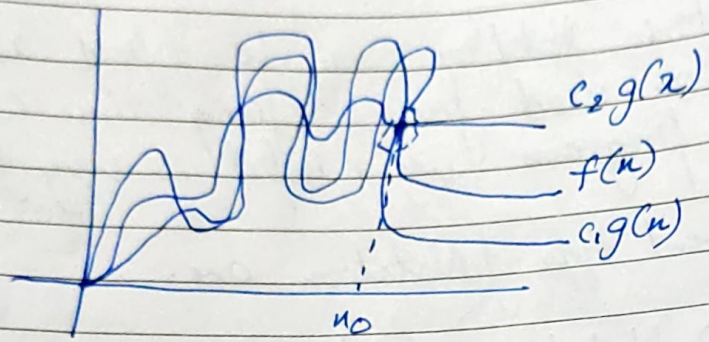
iff $f(n) \geq g(n)$

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

classnotes

Date
Page

iii. Theta Notation (Θ)



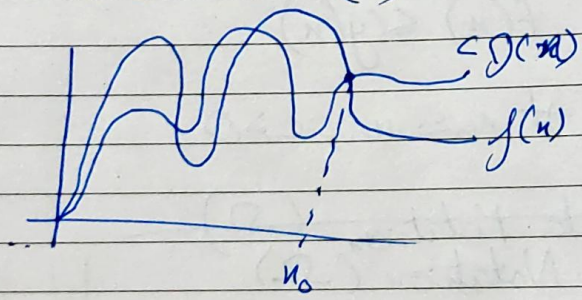
iff, $\Theta(c_1 g(n)) \leq f(n) \leq \Theta(c_2 g(n))$

$\therefore$ $g(n) \leq f(n) \leq g(n)$

$n \geq n_0$

$c_2, c_1 > 0$

iv. Small o Notation. (o)



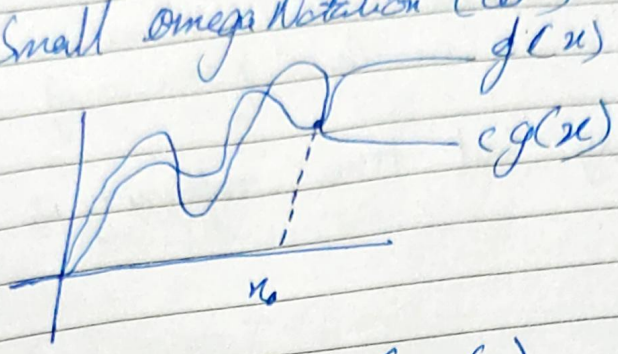
iff, $f(n) < g(n)$, $f(n) = o(g(n))$

$\therefore$ $f(n) < g(n)$, $n > n_0$

$c > 0$

Date
Page

v. Small omega Notation (ω)



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

iff, $f(n) > cg(n)$, ~~$f(n) = 0$~~

$f(n) > g(n)$, $n > n_0$, $c > 0$.

Q-2

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

for $n = 2$ loop turns = 1

$n = 3$ ——— = 1

$n = 4$ ——— = 2

$n = 5$ ——— = 2

$n = 6$ ——— = 2

$n = 7$ ——— = 2

$n = 8$ ——— = 3

⋮

$n = \log n$ ——— = n

Date

Page

We can see that loops iteration increases when the n is in the power of 2's only

we can say that time complexity is

$\log_2 n$

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

$T(0) = 1$

$3(T(n-1)) = ?$

for $T(1)$

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

$= 3 \times 1$

for $T(2)$

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

$= 3\,T(1)$

$= 3\,T(0)$

$= 3 * 3 \times 1$

$T(3) = 3\,T(3-1)$

$= 3\,T(2)$

$= 3 \times 3 \times 3 \times 1$

Date
Page

$T(4) = 3(T(4-1))$
$= 3(T(3))$
$= 3 \times 3 \times 3 \times 3 \times 1$

for T(n) generalizing.

$T(n) = 3T(n-1)$
$= 3 \times 3 \times 3 \times 3 \quad \text{---} \quad \times n$

$= 3^n$

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

4. $T(n) = \{ 2T(n-1) - 1 \text{ if } n > 0, \text{ else } 1 \}$

$T(0) = 1$

for $T = 1$

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

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

$T(3) = 2T(3-1) - 1$
$= 2T(2) - 1$
$= 2(2-1) - 1$
$= 4 - 2 - 1$
1
1
1

$T(n) = 2T(n-1) - 1$
$= 2n - (2n-2)$
$\text{----} \quad 4 - 2 - 1$

after solving we would get

$\boxed{T(n) = O(1)}$

classnotes

Date
Page

5. int i = 1, S = 1

while (S <= n) — ?

{

i++; — O(1)

S = S + i; — O(1)

print ("#"); — O(1)

}

S will run when it is less than n

S = S + i & i is increasing 1 time in each iteration we can say that S = sum of first n natural numbers.

Let sum of first m integer be

$$\frac{m}{2}(m+1)$$

for equation to satisfy

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

Solving for m

$$m^2 + m > 2n$$

$$m^2 + m - 2n > 0$$

Date
Page

$$n^2 + \frac{1}{4}n + \frac{3}{4}n + 2n \geq 0$$

$$n\left(n + \frac{1}{4}\right)n$$

$$n^2 + n > 2n$$

$$n^2 +$$

$$n \times (n+1) > 2n$$

$$n^2 + n > 2n$$] removing lower orders

$$n^2 > n$$

$$n > \sqrt{n}$$

we can say that T.C of the program would be $O(\sqrt{n})$. Operation of T.C $O(1)$ is not considered as they are lower order.

classnotes

Date
Page

6.

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

for i = 1 — iteration = 1
i = 2 — = 4
i = 3 — = 9
i = 4 — = 16

for i = 1 — iteration = 1
i = 2 — iteration = 1
i = 3 — = 1
i = 4 — = 1
i = 5 — = 1
| |
i = n — = 1

we can say that T.C for the function would be O(1)

Date
Page

7.

```
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)
        {
            c++;
        }
    }
}
```

n	i	j	k
1	1	1	1
2	1 + ~~1~~ = 2	1 + 1 + ~~1~~ = 3	1 + 1 + 1 + 1 = 4
3	1 + 1	1 + 1 + ~~1~~ = 3	1 + 1 + 1 + 1 + ~~1~~ = 4
4	1 + 1 + ~~1~~	1 + 1 + 1 = 3	1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8
5	1 + 1 + ~~1~~	1 + 1 + 1 = 3	1 + 1 + 1 + 1 + 1 + 1 ---- = 8
1			
1			
8	1 + 1 + 1	1 + 1 + 1 + 1 = 4	1 + 1 + 1 + 1 + ---- + 1 = 16

$n \quad \log n$

no of k increases with power of 2^n so we can say that j will iterate $\log n$ times & k will ~~be~~ while i iterates n times.

$O(n) = (n * (\log_2 n))$

Date

Page

8 function (int n)
{
    if (n==1)
        return;
    for (i=1 to n)
    {
        for (j=1 to n)
        {
            print("*");
        }
    }

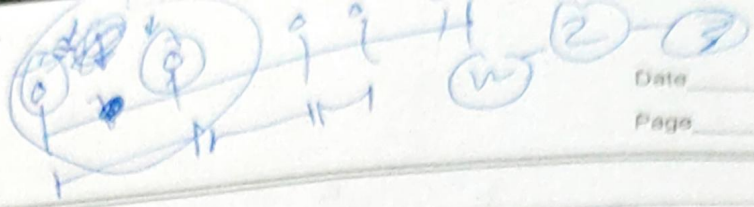
for, n = 1 ⇒ O(1)

for n ≥ 2 ⇒

i	j	i * j
1	1	1
2	2	4
3	3	9
4	4	16
¦	¦	¦
¦	¦	¦
n	n	n^2
n	n	n^2

] because of nest

Since ~~x~~ i runs n time for j running i x j
we can say T.C = n^3
or T.C = $O(n^3)$ ∀ n ≥ 2
for n = 1, T.C = O(1)



Date
Page

4.)

```
function (int n)
{
  for (i = 1 to n)
  {
    for (y = 1; y <= n; y = y + i)
    {
      print ("*")
    }
  }
}
```

i	y
1	n
2	n/2
3	n/3
4	n/4
5	n/4
6	⋮
⋮	⋮
⋮	⋮
n	n/n = 1
n	log n

Hence we can say tc of ~~that~~ is = O(log n)

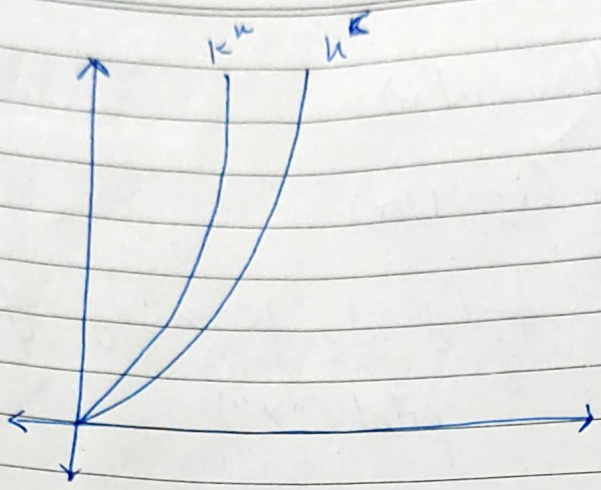
tc of i is = O(n)

nested = O(n * log n)

classnotes

Date
Page

10.



For ~~any~~ value of $n, k, c > 1$ all the value of

$$O(k^n) > O(n^k)$$

this is because (n) exponential time complexity is always greater than integer exponential.

for $n, k, c = 1$, $O(k^n) = O(n^k)$.

and for $n, k, c < 1$, the condition is false and the program wont iterate once.