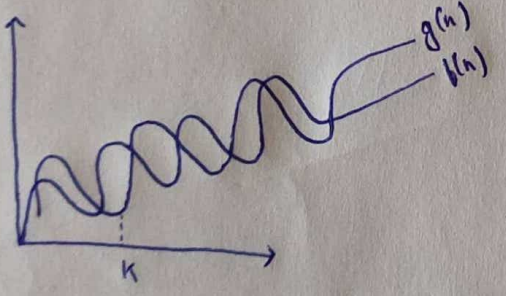
Assignment No: 1

① These notations are used to tell the complexity of an algorithm when the input is very large.

It describe the algorithm efficiency and performance in a meaningful way. It describes the behaviour of time or space complexity for large instance characteristics.

These have 5 types.

i) Big Oh Notation (O): The function $f(n) = O(g(n))$. if and only if there exists a +ve constant c and k such that $f(n) \leq c * g(n)$, for all n, $n \geq k$



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

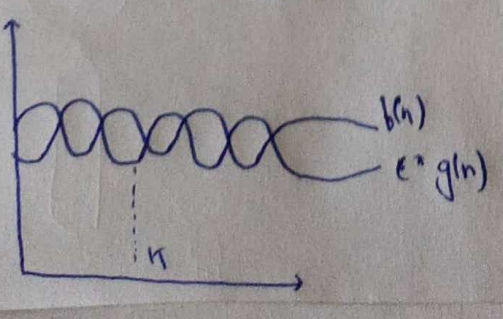
iff

$f(n) \leq c * g(n)$

$\forall\ n \geq n_0$

so constant $c > 0$

ii) Big Omega Notation: The function $f(n) = \Omega(g(n))$, iff there exists a +ve constant c and k such that $f(n) \geq c * g(n)$ for all n, $n \geq k$



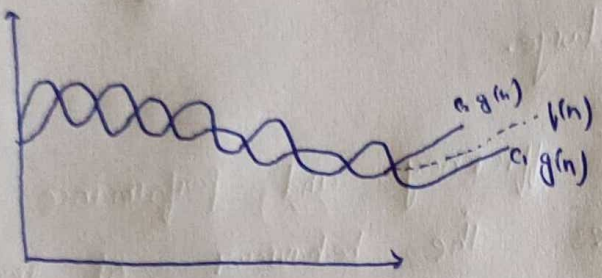
$f(n) = \Omega(g(n))$

iff.

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

iii) Big Theta Notation:- The function $f(n) = \Theta(g(n))$ iff there exists a +ve constant c_1, c_2 and k such that

$c_1 * g(n) < f(n) < c_2\, g(n)$ for all n,



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

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

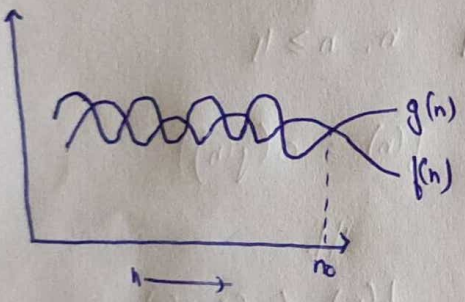
$\forall$ n, max n_1, n_2

iv) Small - oh (o):-

$f(n) = o(g(n))$ $\quad f(n) \leq cg(n)$

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

$n = o(n^2)$



v) Small - Omega :-

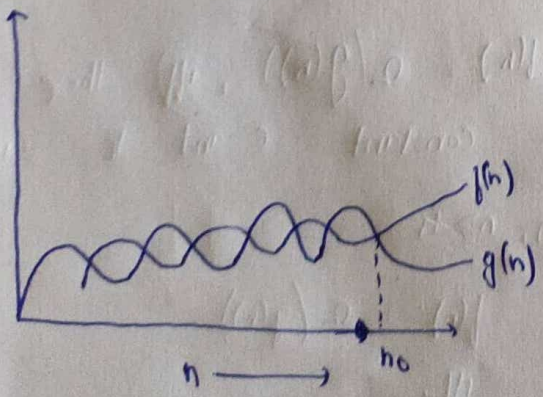
lower bound

$f(n) - \omega\, g(n)$

$f(n) > c.g(n)$

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

$n^2 = \omega(n)$



(2) Time Complexity of a loop means no. of times it has run

i	1	2	4	8	16	32	- - - .	2^k
value	2	2^2	2^3	2^4	2^5	2^1	. . . -	n

i= 1, 2, 4, 8, 16, 32 . . . . . 2^k this means k times

i.e $2^k = n$

$k \log_2 2 = \log_2 n$ $[\log_2 2 = 1]$

T.C = O(logn)

(3)

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

By forward substitution

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

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

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

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

$T(n) = 3^n$

$\therefore$ **T.C = O(3^n)**

(4) $T(n) = \begin{cases} 2T(n-1)-1 & , n > 0 \end{cases}$

By forward substitution

$T(0) = 1$

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

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

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

$2^n - [2^{n-1} - 2^{n-2} - 2^{n-1}, \ldots\ldots - 2^1 - 2^0)$

$= 2^n - [2^n - 1]$

$= 1$

$\boxed{T.C = O(1)}$

(5) The value of i increases by one for each iteration The value contained in 's' ed the ith iteration is the sum of the first 'i' tve integers. If k is the total no. of iterations.

taken by an program then while loop terminate of

$1 + 2 + 3 + \ldots\ldots + k \Rightarrow [k(k+1)/2] > n$

so, $k = O(\sqrt{n})$

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

(6) $O(n) = T.(c)$

(7)

```
Void function (int n)
{
    int i, j, k, count = 0;
    for (i=n/2; i<=n; i++)          O(n)
        for (j=1; j<=n; j*2)        O(log n)
            for (k=1; k<=2; k*2)    O(log n)
                count++;
}
```

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

(8)

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

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

(9)

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

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

(10) For the function, n^k and c^n, what is the asymtotic notation b/w these function

Assume that $k >= 1$ & $c > 1$ are constant

Find out the value of c and no for which relation holds

n^k is $O(c^n)$

As c^n is the upper bound.