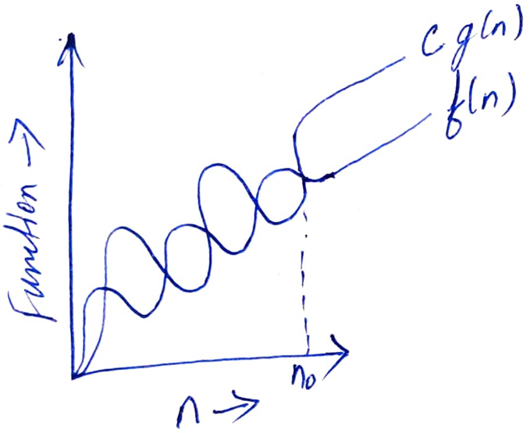
DAA ASSIGNMENT - 1

Naman Deol 4th Sem
CE - 27

Q.1 Asymptotic notation is used to describe the behaviour of a function as its input grows without bound. It is commonly used in the analysis of algorithms to describe their time and space complexity when the input is very large.
Asymptotic means → tending to infinity.

1) Big O (O): It represents the upper bound of a function. It is used to describe the worst case scenario for an algorithm. It represents the maximum time an algorithm will take to complete for any given input size.

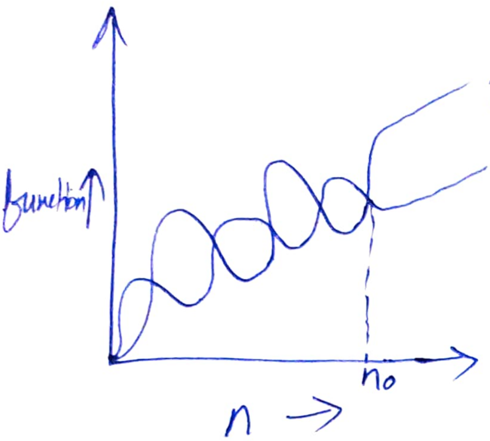


$f(n) = O(g(n))$
iff $f(n) \leq c \cdot g(n)$
$\forall\ n \geq n_0$, some constant $c > 0$.

* $g(n)$ is tight upper bound of $f(n)$.

Example:- $O(n)$ represents a function whose running time grows linearly with the input size n.

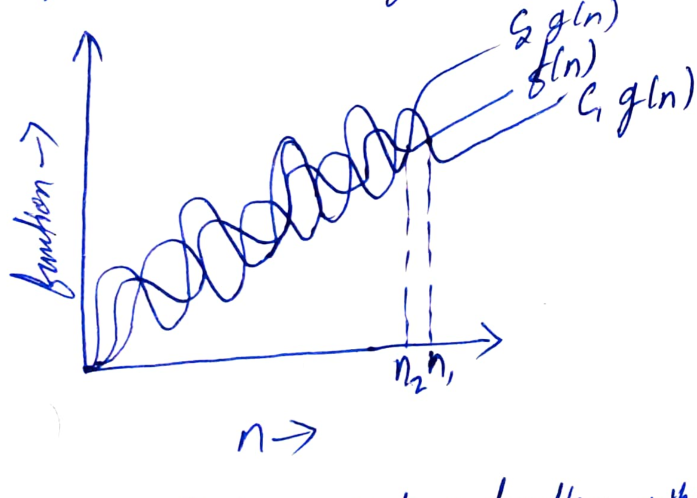
2) <u>Omega notation (Ω)</u>: It represents the lower bound of a function. It is used to describe the best case scenario for an algorithm. It provides a lower bound on the growth rate of the function.



$f(n) = \Omega(g(n))$
iff $f(n) \geq c\, g(n)$
$\forall\ n \geq n_0$ and some constant $c > 0$.
* $g(n)$ is tight lower bound of $f(n)$.

Example:- $\Omega(n)$ represents a function whose running time grows at least as fast as linearly with the input size n.

3) <u>Theta notation (θ)</u>: It represents both the upper and lower bounds of a function. It is used to describe the tight bound of an algorithm. It provides an exact bound on the growth rate of the function. It tells the exact time.



$f(n) = \theta(g(n))$
iff $c_1 g(n) \leq f n \leq c_2 g(n)$
$\forall\ n > max(n_2, n_1)$
and some constant $c_1 > 0$ & $c_2 > 0$

Example:- $\theta(n)$ represents a function whose running time grows linearly with the input size n.

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

$1, 2, 4, 8, 16, 32$ ------ k terms.

k_{th} term $= a \cdot r^{k-1}$ $\quad \{ r = \frac{4}{2} = 2 \}$ $a = 1$;

k_{th} term $\Rightarrow 1 \cdot 2^{k-1}$

$n = 2^{k-1}$

$n = \frac{2^k}{2^1} \Rightarrow 2n = 2^k$

Taking $\log_2$ both sides

$\log_2 2n = \log_2 2^k$

$\log_2 2n = k$ $\quad \{ \log_2 2 = 1 \}$

$k = \log_2 2 + \log_2 n$

$k = 1 + \log_2 n$

Complexity $\Rightarrow O(1 + \log_2 n)$, ignoring constants we have

Ans $= O(\log_2 n)$

$3 \Rightarrow$ $T(n) = 3T(n-1)$ — (1)

$T(0) = 1$ — (2)

Putting, $n = n-1$ in (1)

$T(n-1) = 3T(n-2)$ — (3)

Putting, $n = n-2$ in (1)

$T(n-2) = 3T(n-3)$ — (4)

Putting, $n = n-3$ in (1)

$T(n-3) = 3T(n-4)$ — (5)

Putting (5) in (4)

$T(n-2) = 9T(n-4)$ — (6)

Putting (6) in (3)

$T(n-1) = 3^3 T(n-4)$ — (7)

Putting (7) in (1)

$T(n) = 3^4 T(n-4)$ — (8)

~~Assume~~ for k terms,

$T(n) = 3^k T(n-k)$ — (9)

Assume $n-k = 0$

$n = k$

$T(n) = 3^n . T(0)$

$T(n) = 3^n$

Complexity $\Rightarrow$ $O(3^n)$

4 =) $T(n) = 2T(n-1) - 1$ —— (1)
$T(0) = 1$ —— (2)

Putting $n = n-1$ in (1)

$T(n-1) = 2T(n-2) - 1$ — (3)

Putting $n = n-2$ in (1)

$T(n-2) = 2T(n-3) - 1$ — (4)

Putting $n = n-3$ in (1)

$T(n-3) = 2T(n-4) - 1$ — (5)

Putting (5) in (4)

$T(n-2) = 2 \cdot 2T(n-4) - 2 - 1$ — (6)

Putting (6) in (3)

$T(n-1) = 2 \cdot 2 \cdot 2T(n-4) - 2 \cdot 2 - 2 - 1$ — (7)

Putting (7) in (1)

$T(n) = 2^4 T(n-4) - 2^3 - 2^2 - 2^1 - 2^0$ — (8)

for k terms

$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - 2^{k-3} \ldots - 2^1 - 2^0$ — (9)

=) ~~$2^k T(n$~~ Assume $n-k=0$

$\Rightarrow 2^k \cdot T(0) - 2^{k-1} - 2^{k-2} - 2^{k-3} \ldots 2^0$

$\Rightarrow 2^k - 2^{k-1} - 2^{k-2} - 2^{k-3} \ldots - 2^1 - 2^0$

$\Rightarrow 2^k - (2^0 + 2^1 + \ldots 2^{k-3} + 2^{k-2} + 2^{k-1})$

$\Rightarrow 2^k - \left(\frac{1-2^k}{1-2}\right) \Rightarrow 2^k + 1 - 2^k$

$\Rightarrow 1$

Complexity = $O(1)$

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

print("#") will execute till s<=n and s is incremented by i ~~which~~ every time which also increases by 1 everytime

i = 1, 2, 3, 4, 5, 6, 7, ---

S = 1, 3, 6, 10, 15, 21, 28 ------ k^{th} term

loop will run till s<=n

S is the sum of k numbers (AP)

k^{th} term = $\frac{k*(k+1)}{2}$

$$n = \frac{k^2+k}{2}$$

$$2n = k^2+k$$

$$k(1+k) = 2n$$

$$k^2+k-2n=0$$

$$k = \left(-\frac{1}{2} \pm \sqrt{1+4*n}\right)/2$$

$$k = 2\sqrt{n}$$

$$k = \sqrt{n}$$

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

6=) void function(int n)

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

=) $1, 2^2, 3^2, 4^2, 5^2 \ldots\ldots k^{th}$

k^{th} term $= K*K$

k^{th} term $<= n$

$K*K <= n$

$K^2 = n$

$K = \sqrt{n}$

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

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++;
            }
        }
    }
}
```

Time complexity of Inner most loop;

$K=1$ to n, $k = k*2$

$1, 2, 4, 8, 16, \ldots\ k^{th}$ term

k^{th} term $= 2^{K-1}$

$n = \frac{2^K}{2}$ =) $2n = 2^K$ =)

Taking $\log_2$ both sids

$\log_2 2n = \log_2 2^k$

$\log_2 2n = k$

$k = \log_2 2 + \log_2 n$

$\boxed{k = 1 + \log_2 n}$

It means for each value of j, this loop ~~iterates~~ runs $1 + \log_2 n$ times.

Complexity of middle loop,

$j = 1$ to n ; $j = j \times 2$

1, 2, 4, 8, 16 - - k^{th} term

$\Rightarrow (1 + \log_2 n)$

It means for each value of i, this loop runs $(1 + \log_2 n)$ times

Complexity of Outer most loop :

$i = n/2$ to n ; $i{+}{+}$

~~$\frac{n}{2}, \frac{n+1}{2}, \frac{n+2}{2}, \ldots \frac{n+k}{2}$~~

~~k^{th} term $= \frac{n+n}{2} \Rightarrow k = \frac{2n}{2}$~~

~~k^{th} term $= \frac{n+k}{2}$~~

$n =$ ~~$\frac{n+k}{2}$~~

$n/2, \frac{n}{2}+1, \frac{n}{2}+2, \frac{n}{2}+3, \ldots k^{th}$ term

k^{th} term $= \frac{n}{2} + k$

$n = \frac{n}{2} + k \Rightarrow k = n - \frac{n}{2}$

$k = \frac{n}{2}$

this loop will run $\frac{n}{2}$ times

i	j	k
n/2	1 → n	$1 + \log_2 n$
n/2 + 1	1 → n	$1 + \log_2 n$
n/2 + 2	1 → n	
n	1 → n	

Total complexity $= \frac{n}{2} * (1 + \log_2 n) * (1 + \log_2 n)$

$\Rightarrow \frac{n}{2} + \frac{n}{2}\log_2 n + \frac{n}{2}(\log_2 n)^2 + \frac{n}{2}\log_2 n$

$\Rightarrow O(n (\log_2 n)^2)$

8 =) function (int n)

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

i	j
1	1 → n
2	1 → n
3	1 → n
4	1 → n
⋮	
n	
n times	n times

=) n * n times

~~printf will execute n *~~

for function (n-3)

$n, n-3, n-6, n-9$ ------ k^{th} term

$n, n-1\cdot3, n-2\cdot3, n-3\cdot3,$ --- k^{th} term

k^{th} term $= n-(k-1)\cdot3 = n-3k-3$

$1 = n-3k-3$

$\Rightarrow n-3k-3-1=0 \Rightarrow n-3k-4=0 \Rightarrow k=\frac{n-4}{3}$

Inner most loop will execute $= n * n * \frac{n-4}{3}$ times $= \frac{n^3-4n^2}{3}$

Complexity $= O(n^3)$ ignore constants and smaller values.

9 =)

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

Outer loop will run n times (i).

~~Aft~~ for $i=1$, j will run n times

$i=2$, j will run $n/2$ times

$i=3$, j will run $n/3$ times

$i=n$, j will run n/n times.

Inner loop will run =) $\left(n + \frac{n}{2} + \frac{n}{3} + \text{---} \frac{n}{n-1} + \frac{n}{n}\right)$ times

$$\Rightarrow n\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \text{-----} + \frac{1}{n}\right)$$

$$\Rightarrow n \cdot \log n.$$

sum of $\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \text{---} + \frac{1}{n}\right)$ is $\log n$

complexity =) $\underline{O(n \cdot \log n)}$