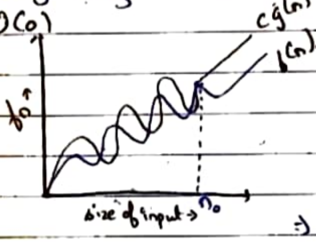
Ques: Asymptotic Notations
↳ Tending to infinity

They help you find the complexity an algorith when input is very large.

1) Big O(O)



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

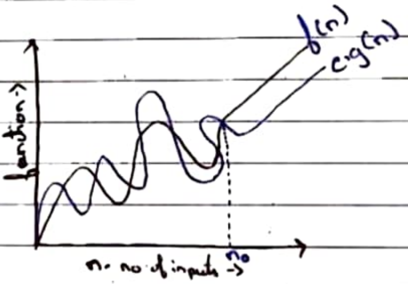
iff $f(n) \leq c \cdot g(n)$.

$\forall n \geq n_0$

for some constant $c > 0$

⇒ g(n) is 'tight' upper bound of f(n)

ii) Big Omega (Ω)



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

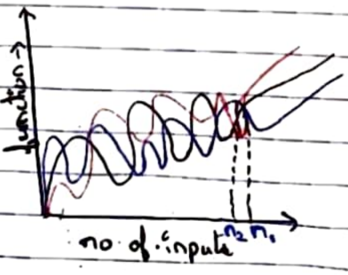
g(n) is 'tight' lower bound of f(n)

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

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

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

iii) Theta (Θ)



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

g(n) is both 'tight' upper & lower bound of fn f(n)

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

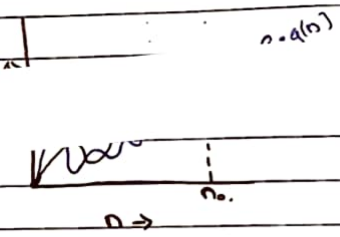
iff

$c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$

$\forall n \geq \max(n_1, n_2)$

for some constant $c_1 > 0$ & $c_2 > 0$

Small o(O)



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

g(n) is upper bound

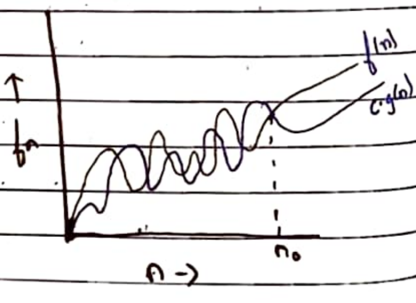
$f(n) = o(g(n))$

when $f(n) < c \cdot g(n)$

$\forall n > n_0$

& $\forall c > 0$

small omega (ω)



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

g(n) is lower bound of fn f(n)

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

when $f(n) > c \cdot g(n)$

$\forall n > n_0$

& $\forall c > 0$

Ques-2 What should be time complexity of
for(i=1 to n) {i = i*2}

for(i=1 to n) // i = 1, 2, 4, 8 .... n
{ i=i*2} // O(1)

$\Rightarrow \sum_{i=1} 1+2+4+8+ \dots n$

GP kth value $\Rightarrow T_k = ar^{k-1}$

$\Rightarrow 1 \times 2^{k-1}$

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

$\Rightarrow 2n = 2^k$

$\Rightarrow \log 2n = k \log 2$

$\Rightarrow \log_2 + \log n = k \log 2$

$\Rightarrow \log n + 1 = k$

$\Rightarrow O(k) = O(1 + \log n)$
$= O(\log n)$

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

$T(n) = 3T(n-1)$ — ①

put $n = n-1$
$T(n-1) = 3\{T(n-2)$ — ②
from 1 & 2

$\Rightarrow T(n) = 3(3T(n-2))$
$= 9T(n-2)$ — ③

putting $n = n-2$ in ①

$T(n) = 3(T(n-3))$ - ④

$\Rightarrow T(n) = 27(T(n-3))$

$\Rightarrow T(n) = 3^k (T(n-k))$

putting $n-k=0$

$\Rightarrow n = k$.

$\Rightarrow T(n) = 3^n [T(n-n)]$

$\Rightarrow T(n) = 3^n T(0)$

$\Rightarrow T(n) = 3^n \times 1$ $[T(0)=1]$

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

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

$T(n) = 2T(n-1)-1$ - ①

Let $n = n-1$

$\Rightarrow T(n-1) = 2T(n-2)-1$ - ②

$\Rightarrow$ from ① & ②

$\Rightarrow T(n) = 2[2T(n-2)-1]-1$

$\Rightarrow T(n) = 4T(n-2)-2-1$ - ③

let $n = n-2$

$\Rightarrow T(n-2) = 2T(n-3)-1$ - ④

from ③ & ④

$\Rightarrow T(n) = 4[2T(n-3)-1]-2-1$

$\Rightarrow T(n) = 8T(n-3)-4-2-1$

$\Rightarrow T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} \cdots$

$\Rightarrow GP = 2^{k-1} + 2^{k-2} + 2^{k-3} + \cdots$

$a = 2^{k-1}$

$r = 1/2$

$\Rightarrow S_k = \dfrac{a(1-r^n)}{1-r}$

$= \dfrac{2^{k-1}(1-(1/2)^n)}{1/2}$

$= 2^k(1-(1/2)^k)$

$= 2^k - 1$

Let $n-k=0$

$\Rightarrow n=k.$

$\Rightarrow T(n) = 2^n T(n-n) - (2^n - 1)$

$\Rightarrow T(n) = 2^n \cdot 1 - (2^n - 1)$

$\Rightarrow T(n) = 2^n - (2^n - 1)$

$\Rightarrow T(n) = O(1)$

Ques-5 What should be time complexity of

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

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

s = 1 + 3 + 6 + 10 + 15 + 21 - - - - n — (1)

Sum of s = 1 + 3 + 6 + 10 + - - - + T_n — (1)

also s = 1 + 3 + 6 + 10 + - - T_{n-1} + T_n — (2)

from (1) - (2)

$0 = 1 + 2 + 3 + 4 + \cdots n - T_n$

$\Rightarrow T_k = 1 + 2 + 3 + 4 + \cdots k$

$\Rightarrow T_k = \frac{1}{2} k(k+1)$

$\Rightarrow$ for k iterations.

$1 + 2 + 3 + \cdots + k <= n$

$\Rightarrow \frac{k(k+1)}{2} <= n$

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

$\Rightarrow O(k^2) <= n$

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

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

Ques 6 Time complexity of -

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

as $i^2 <= n$

$\Rightarrow i <= \sqrt{n}$

$i = 1, 2, 3, 4 \dots \sqrt{n}$

$\sum_{i=1}^{n} 1+2+3+4+ \dots + \sqrt{n}$

$\Rightarrow T(n) = \frac{\sqrt{n} \times (\sqrt{n}+1)}{2}$

$\Rightarrow T(n) = \frac{n \times \sqrt{n}}{2}$

$\Rightarrow T(n) = O(n)$

Ques-7 Time complexity of :-

```
void fn(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++ ;
}
```

for $k = k*2$

$k = 1, 2, 4, 8, \ldots n.$

$\Rightarrow$ GP $\Rightarrow$ $a = 1$, $r = 2$.

$S_n = \frac{a(r^n - 1)}{r - 1}$

$= \frac{1(2^k - 1)}{1}$

$n \Rightarrow 2^k$.

$\Rightarrow \log n = k$.

$\Rightarrow$

i	j	k
1	$\log n$	$\log n * \log n$
2	$\log n$	$\log n * \log n$
⋮	⋮	⋮
n	$\log n$	$\log n * \log n$

$\Rightarrow O(n * \log n * \log n)$

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

8) Time Complexity of

```
function (int n)
{ int (n==1)
    return;                       // O(1)
  for (i=1 to n)                  // i = 1, 2, 3, 4 --- n => O(n)
  { for (j=1 to n)                // j = 1, 2, 3, 4, --- n => O(n^2)
     { print ("*");
     }
  }
  function (n-3);                 T(n/3)
}
```

$\Rightarrow$ $T(n) = T(n/3) + n^2$

$\Rightarrow$ $a = 1$, $b = 3$, $f(n) = n^2$.

$c = \log_3 1 = 0$

$\Rightarrow$ $n^0 = 1 > (f(n) = n^2)$.

$\Rightarrow$ $T(n) = \Theta(n^2)$

Ques: Time complexity of -

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

for i=1 ⇒ j = 1, 2, 3, 4 ... n = n.
for i=2 ⇒ j = 1, 3, 5, ... n = n/2
for i=3 ⇒ j = 1, 4, 7, ... n = n/3
⋮
for i=n ⇒ j = 1 ...

$\Rightarrow \sum_{j=n} n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \cdots + 1$

$\Rightarrow \sum_{j=n} n[1 + \quad \frac{}{3} \quad \frac{}{4} + \cdots \frac{1}{n}]$

$\sum_{j=n} n[\log n]$

$\Rightarrow T(n) = [n \log n]$

$T(n) = O(n \log n)$

Q-10. for functions, n^k & c^n, what is the asymptotic relation between these functions?
assume that $k>=1$, & $c>1$ are constant.
Find out the value of c & n_0 for which relation holds

as given n^k & c^n

relation b/w n^k & c^n is

$$n^k = O(c^n).$$

as $n^k \leq ac^n$
$\forall n \geq n_0$ & some constant $a > 0$

for $n_0 = 1$
$c = 2$

$\Rightarrow 1^k \leq a2^1$

$\Rightarrow$ $n_0 = 1$ & $c = 2$