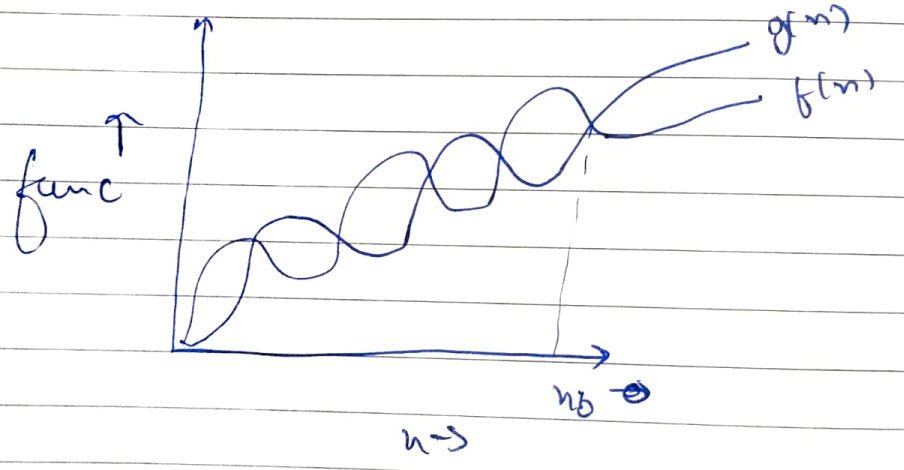
# Assignment - 1

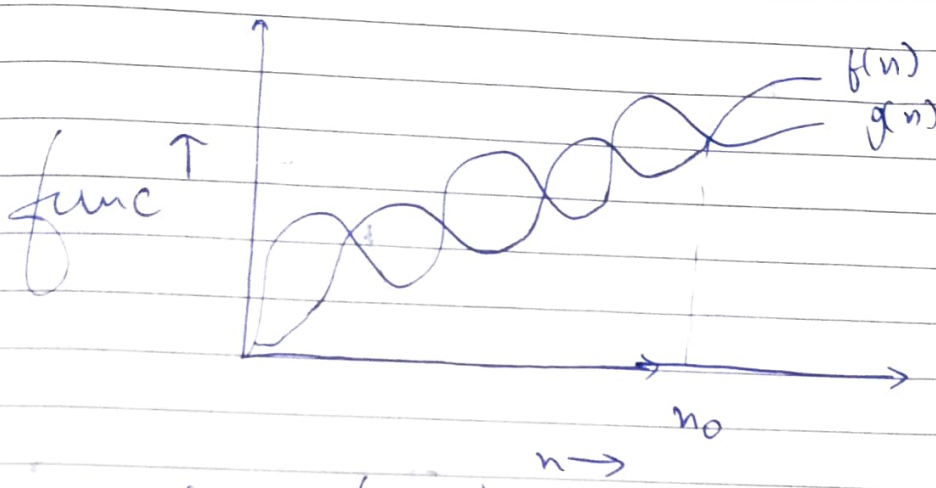
1. Asymptotic notation is used to describe the behaviour of a function as it's input grows without bound. It is commonly used in the analysis of algorithm to describe their time and space complexity when the input is very large.

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



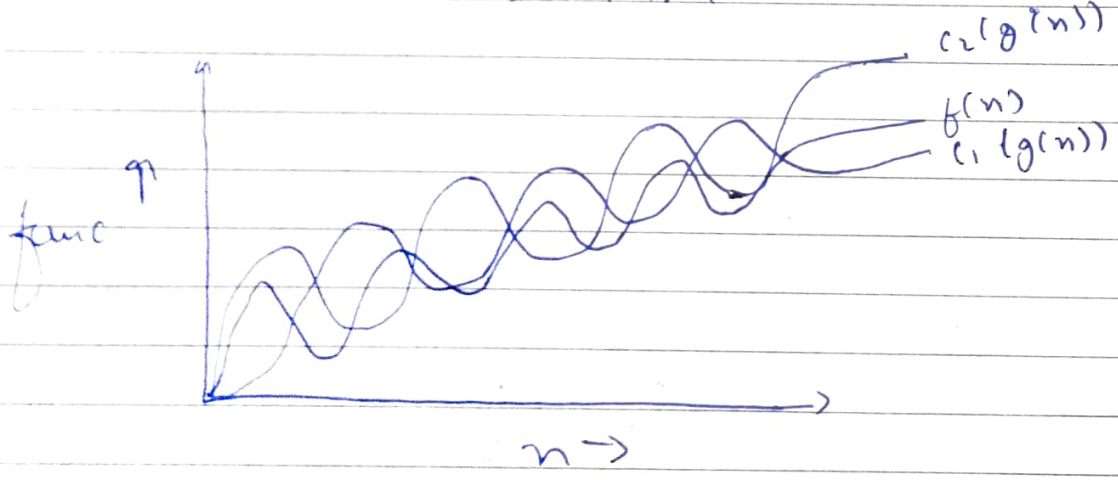

$f(n) = O(g(n))$
iff $f(n) \leq g(n)$
$\forall\ n \geq n_0$, some constant $> 0$

(2) Big Omega(Ω): It represents the lower bound of a function. It is used to describe the best case scenario for an algo. It provides a lower bound on the growth rate of function



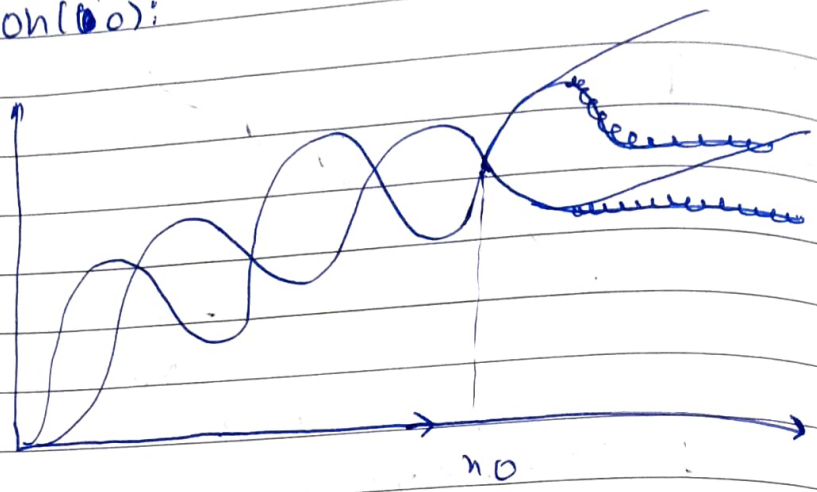

$f(n) = \Omega(g(n))$
iff $f(n) \geq g(n)$
$\forall\ n \geq n_0$ and some constant $c > 0$

(3) Theta ($\Theta$) notation:- It represents both the upper bound and lower bounds of a function. It is used to describe the right bounds of an algo. It provides an exact bound on the growth rate of the function. It tells exact time.



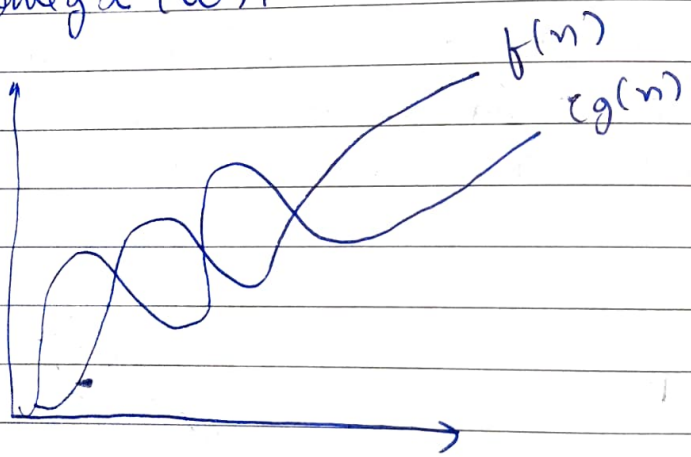

$f(n) = \Theta(g(n))$
iff $c_1(g(n)) \leq f_n \leq c_2(g(n))$
$\forall\ n > \max(n_2, n_1)$

(iv) Small oh (o):




$f(n) = o(g(n))$
iff $f(n) < g(n)$
$\forall\ n > n_0\ \forall\ c > 0$

(v) Small Omega (ω):




$f(n) = \omega(g(n))$
iff $f(n) > cg(n)\ \forall$
$n > n_0\ \forall\ c > 0$

2

```
for (i=1 ; i<=n ; i=i*2)
{
    // Some Statements
}
```

i =

1

1*2

1*2*2

⋮

k times

$2^k$

$2^k = n$

Taking log on both sides

$\log (2)^k = \log n$

$k \log_2 2 = \log_2 n$

$k = \log_2 n$

Time complexity = $O(\log_2 n)$

4

3

$$T(n) = \begin{cases} 3T(n-1) & n>0 \\ 1 & n=0 \end{cases}$$

$T(n) = 3T(n-1)$ — ①

$T(n-1) = 3T(n-2)$ — ②

Putting value of ② in ①

$T(n) = 3[3T(n-2)] = 3^2 T(n-2)$ — ③

$T(n-2) = 3T(n-3)$ — (4)

Putting value in eqn (3)

$T(n) = 3^2 [3T(n-3)]$

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

⋮ K terms

⋮

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

Assume $n-k = 0$

$n = k$

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

$= 3^n T(0)$

$= 3^n \times 1$

$= 3^n$

$= O(3^n)$

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

$T(n) = 2T(n-1)-1$ — (1)

$T(n-1) = 2T(n-2)-1$ — (2)

Putting value of (2) in (1)

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

$T(n) = 2^2 T(n-2) - 2 - 1$ — (3)

$T(n-2) = 2T(n-3) - 1$ — (4)

Putting value of (4) in (3)

$T(n) = 2^2 [2T(n-3) - 1] - 2 - 1$

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

⋮ k times

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

Assume $n-k = 0$

$n = k$

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

$= 2^n \times 1 - n$

$= 2^n - n$

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

$T(n) = 2^n T(n-n) - [1 + 2 + 2^2 + \cdots 2^{n-2} + 2^{n-1}]$

$= 2^n T(0) - \left[\frac{1(2^n - 1)}{2-1}\right]$

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

$= 2^n - 2^n + 1$

$= 1$

$= O(1)$

5

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

| i | S |
|---|---|
| 1 | 1 |
| 2 | 1+2 |
| 3 | 1+2+3 |
| 4 | 1+2+3 |
| | : K times |
| | 1+2+3 ... K |

$1+2+3+\ldots+K = n$

$\frac{K(K-1)}{2} = n$

$K^2 = n$ (Ignoring Lower Order terms)

$K = \sqrt{n}$

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

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

| i |
|---|
| $1^2$ |
| $2^2$ |
| $3^2$ |
| : K times |
| $K^2$ |

$K^2 = n$

$K = \sqrt{n}$

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 ++

}

| i | No of times |
|---|---|
| 1 | $(\log n)^2$ |
| 2 | $(\log n)^2$ |
| : $n/2$ times | |
| $n/2$ | $\frac{n}{2} (\log n)^2$ |

Complexity = $O\left(\frac{n(\log n)^2}{2}\right)$

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

8.

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

| i | j |
|---|---|
| 1 | $1+2+3\cdots n = \frac{n(n-1)}{2}$ |
| 2 | $\frac{n(n-1)}{2}$ |
| : | |
| n | $n \times \frac{n(n-1)}{2}$ |

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

$= \frac{n^3-n}{2}$

Complexity = $O(n^3)$

9

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

| i | j |
|---|---|
| 1 | $1+2+3+\cdots+n$ <br> $\frac{n(n-1)}{2}$ |
| 2 | $1+3+5+\cdots+n$ <br> $= \frac{n(n-1)}{2}$ |
| 3 | $1+4+7+\cdots+n$ <br> $= \frac{n(n-1)}{2}$ |
| ⋮ | |
| n | $= \frac{n(n-1)}{2}$ |

$$\text{Total time} = \frac{n(n-1)}{2} + \frac{n(n-1)}{2} + \frac{n(n-1)}{2} + \cdots + \frac{n(n-1)}{2}$$

$$= \frac{n(n-1)}{2}\left[1+2+3+\cdots n\right]$$

$$= \frac{n(n-1)}{2} \cdot \frac{n(n-1)}{2}$$

$$= \frac{n^2(n-1)^2}{4}$$

$$= \frac{n^2(n^2+1-2n)}{4}$$

Complexity = $O(n^4)$