Design and Analysis of Algorithms
(TCS - 409)

Tutorial - 1

1. Asymptotic Notations

Those notations which help us to calculate order of growth when input size is very large.

Different Asymptotic notations are →

1. Big oh (O)

This is also known as upper bounding function.

given two functions

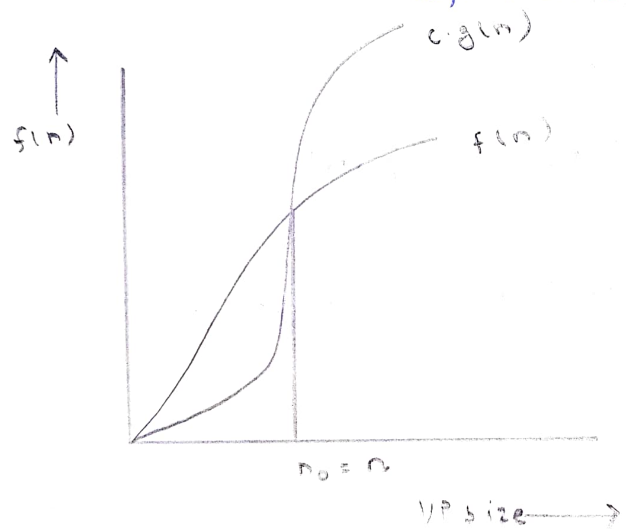
$f(n)$ and $g(n)$

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

iff $0 \le f(n) \le c \cdot g(n)$

if $n \ge n_0$, $c > 0$

→



2. Big omega (Ω)

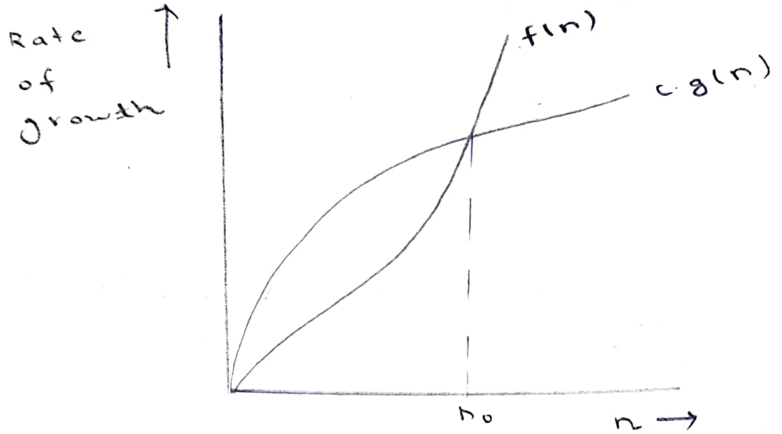
This is also known as lower bounding function.

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

iff

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

for all $n \geq n_0$ $c > 0$



3. **Theta (Θ)**

Given two functions

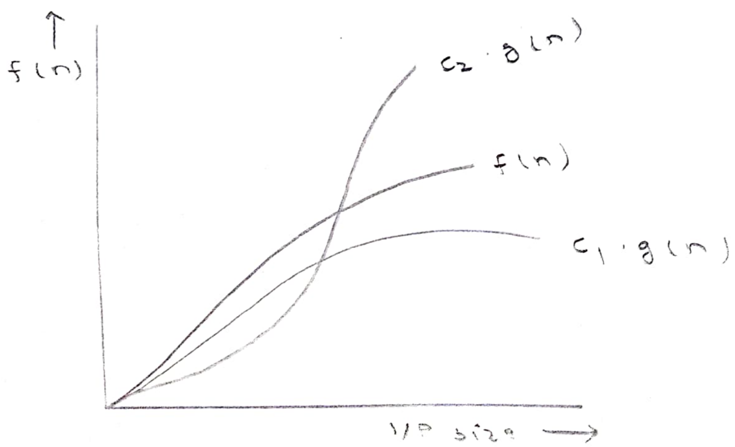
$f(n)$ and $g(n)$

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

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

$c_1, c_2 > 0$

$n_1, n_2 > 0$, $\forall$ $n > \max(n_1, n_2)$



This helps us to find whether U.B or L.B of a given func. or algorithm is same.

4. small oh

gives upper bound that cannot be tight.

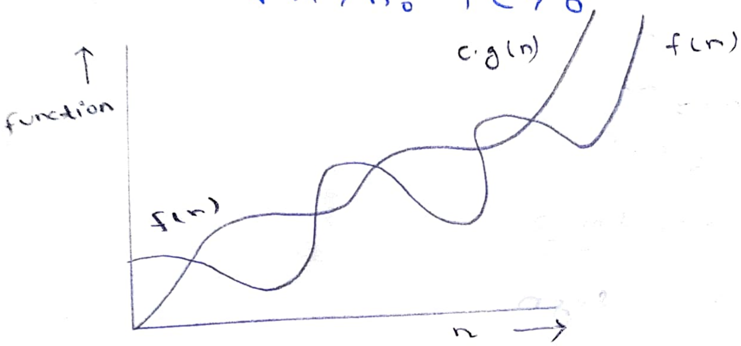
given two functions f(n) & g(n)

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

iff

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

$\forall n > n_0, c > 0$



5. small omega

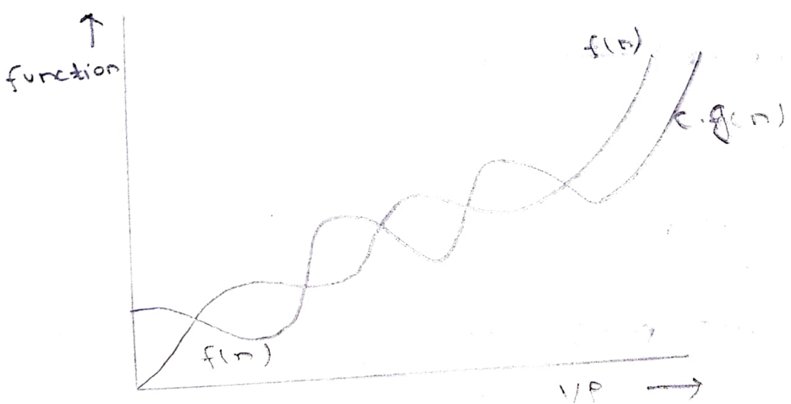
giveb lower bound that cannot be tight.

given two functions f(n) & g(n)

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

iff

$f(n) > c \cdot g(n)$ $\forall n > n_0, c > 0$



2. what should be time complexity of -

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

↳ O(log n)

since in this loop value of 'i' is incremented by 2 until it is less than or equal to n i.e

1, 2, 4, 8 . . . . n

$$a_n = a \cdot r^{n-1}$$

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

$$\log n = k-1 \log_2 2$$

$$k = 1 + \log n$$

$$\therefore O(\log n)$$

3. $T(n) = 3T(n-1)$ if $n > 0$, otherwise 1

$$T(n) = 3T(n-1) \text{ — } 1$$

$$n = n-1$$

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

$$T(n) = 9T(n-2) \text{ — } 2$$

$$n = n-2$$

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

$$T(n) = 27T(n-3) \text{ — } 3$$

⋮

$$T(n) = 3^n (T(n-n)) \quad (\because T(0) = 1)$$

$$= 3^n \quad \therefore \Theta(3^n)$$

4. $T(n) = 2T(n-1) - 1$ if $n > 0$, otherwise 1

$T(n) = 2T(n-1) - 1$ — ①

$n = n-1$

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

$T(n) = 4T(n-2) - 2 - 1$ — ②

$n = n-2$

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

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

$= 8T(n-3) - 4 - 3$

$T(n) = 8T(n-3) - 7$ — ③

$T(n) = 2^n T(n-n) - 2^n + 1$ $(\because T(0) = 1)$

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

$\therefore O(1)$

5. what should be time complexity of -

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

i	s
1	1
2	1 + 2
3	1 + 2 + 3
k	k

$1 + 2 + 3 + \dots + k = \frac{k(k+1)}{2}$

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

$k^2 > n$

$k > \sqrt{n}$

↳ $O(\sqrt{n})$

6. Time complexity of -

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

This will stop when

$i * i > n$

$i^2 > n$

$i > \sqrt{n}$ $O(\sqrt{n})$

```
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 ++;
}
```

$$\underbrace{\frac{n}{2}}_{i\ loop} \cdot \underbrace{\log n}_{j\ loop} \cdot \underbrace{\log n}_{k\ loop} = \frac{n}{2} \log^2 n$$

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

8. Time complexity of .

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

n^2

$T(n-3)$

$$T(n) = T(n-3) + cn^2$$

The inner loop works n^2 times & recursive call is made $n/3$ times

$\therefore O(n^3)$

9. Time complexity of -

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

This loop will work n/i times since j is incremented by i

O (n log n)

10. For the functions n^k and c^n what is the asymptotic relationship between these functions

Assume that k>=1 and c>1 are constants

n^k c^n

$>$ 2 $k>=1$

$c>1$

putting $k=1$

$c=2$

we get n^k, c^n as   where constant is 1

2^1 and 2^2 if $n=2$

$\Rightarrow$ since $2 < 2^2$ i.e $(2<4)$

if $n=1$, $c=2$, $k=1$

where const. is 1/2

we get n^k c^n as

$\downarrow$ $\downarrow$

$1^1=1$ $\quad 2^1=2$

$\therefore$ O (cn)   since $\Rightarrow 1 <= 2 \cdot \frac{1}{2}$ $1<=1$