DAA, Pranshu Soti, 15, ESPSPL-2

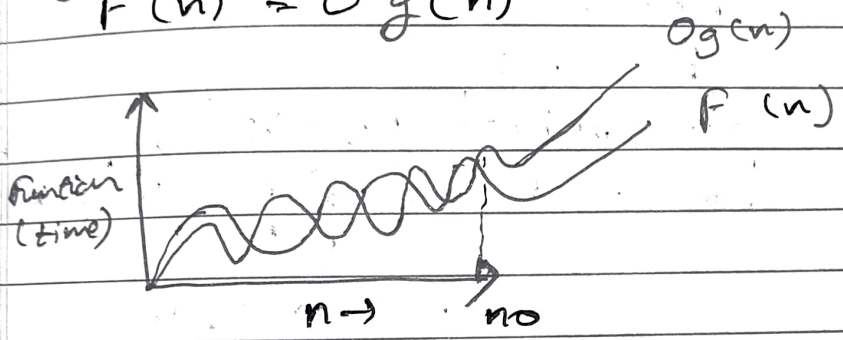
Date

Tutorial-1

Asymptotic notation is used to describe the running time of an algorithm. How much time an algorithm takes with a given input n. There are three different notation : big O, big theta (Θ), big Omega (Ω)

Big O

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



size of Input / size of element

iff $f(n) \leq g(n)$

$\forall\, n \geq n_0$

for some constant, $c > 0$

· g(n) is "tight" upper bound of F(n)

Big Omega (Ω)

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

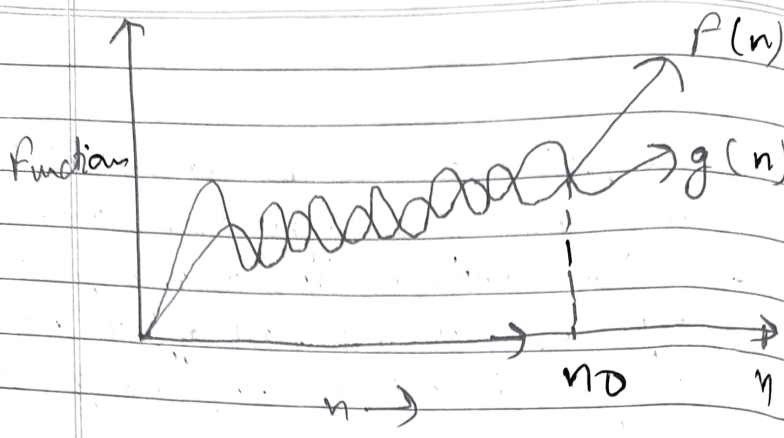
g(n) is "tight" lower bound of function f(n).

$F(n) = \Omega\, g(n)$

iff : $F(n) \geq g(n)$

$\forall\, n \geq n_0$

for some constant, $c > 0$



3=) Theta(Θ)

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

$g(n)$ is both 'tight' upper and lower bound of function $f(n)$

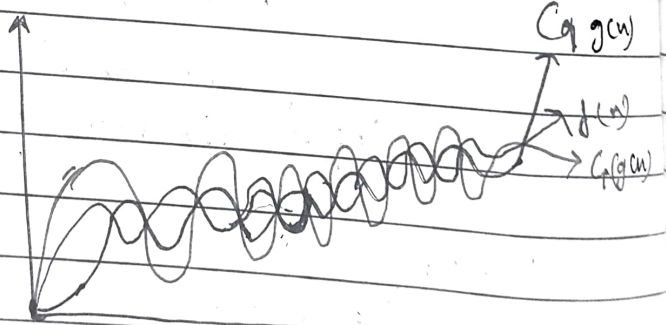
$f(n) = \Theta(g(n))$

iff

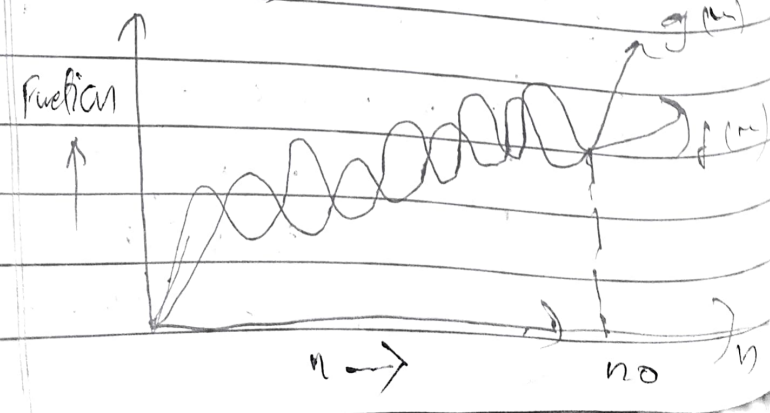
$i\,g(n) \leq f(n) \leq c_2 g(n)$

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

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



4. Small O(Θ)



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

$g(n)$ is upper boband of Fuctin

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

when

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

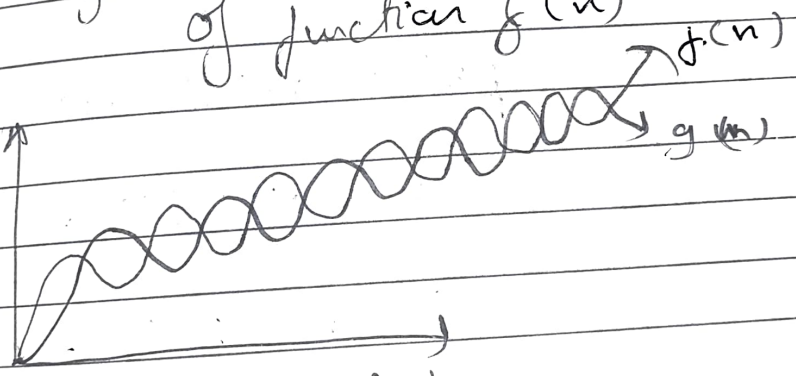
$\forall\ n > n_0$

and $\forall$ constant, $c > 0$

=) Small omega (ω)

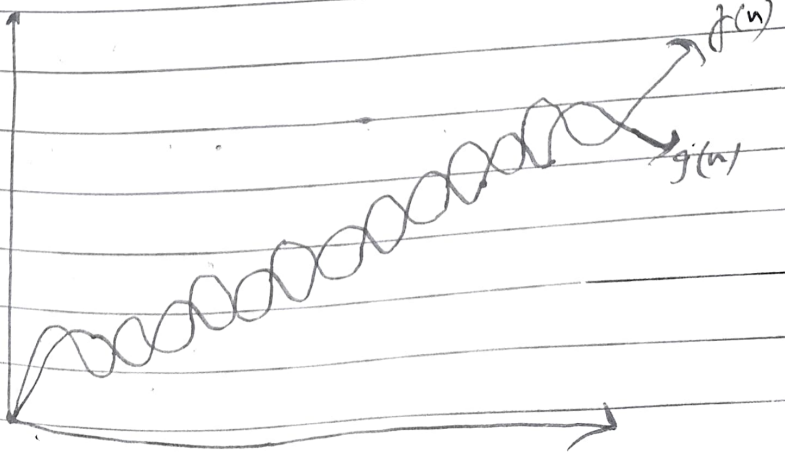
$f(n) = \omega\, g(n)$

$g(n)$ is lower bound of junction $f(n)$



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

$g(n)$ is lower bound of junction $f(n)$



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

$\forall\ f(n) > c\, g(n)$

$\forall\ n > n_0$

$\forall$ constant, $c > 0$

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

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

$2^k = n$

$\log 2^k = \log n$

$k \log_2 2 = \log n$

$k = \log n$

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

put $n = n-1$ in eqn (2).

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

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

$= 9[T(n-2)]$ - (2)

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

$T(n-2) = \{ 3T(n-3) \}$ - (4)

put eqn (4) in (2)

$T(n) = 9\{ 3T(n-3) \}$

$27[T(n-3)]$ - (5)

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

$3^n [T(0)]$ - (6)

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

Put $n = n-1$ in eqn (1)

$T(n-1) = 2[T(n-2)-1]$ — (3)

$T(n) = 2[2T(n-2)-1]$ —

$4T(n-2) - 2$ — (4)

Put $n = n-2$

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

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

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

$= 8T(n-3)-7$

$= 2^K T(n-K) - 2^{K-1} - 2^{K-2} - 2^{K}$

$n-K = 1$

$n-1 = K$

$= 2^{n-1} - 2^{n-2} - 2^{n-3} - 2^{n-4}$

2^{n-1} ∴ $T(n) = O(1)$

(5) What is time complexity of

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

$1 + 3 + 6 + 10 + \cdots - k = n$

$\frac{k(k+1)(k+2)}{6} = n$

$O(k^3) = n$

$k = \sqrt[3]{n}$

$O(\sqrt{n})$ Ans

(6) Calc. time complexity

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

$1 + 4 + 25 + 67 + \cdots n$

$a = 1$

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

Q6

```
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 = j*2)    -> O(log n)
        {
            for(k = 1; k <= n; k = k*2) -> O(log n)
            {
                count++;
            }
        }
    }
}
```

Time complexity $= O(n) * O(\log n * \log n)$

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

Q7 Time complexity of

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

The time complexity of above function is $O(\sqrt{n})$

Q8

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

3:

```
{
function(n-3);
}
```

i = 1, 2, 3, 4, ... n => O(n)

j = 1, 2, 3, 4, ... n => O(n)

Time complexity = O(n × n)

= O(n²) Ans

9

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

~~T(n) = O(n×n)~~

~~O(n²)~~

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

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

$$\sum_{j=n} n \left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} - - - \frac{1}{n}\right]$$

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

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

$T(n) = O(n \lg n)$ Ans