Tanishka
Chauhan
ML/63

Design and Analysis of Algorithms

ASSIGNMENT- 01

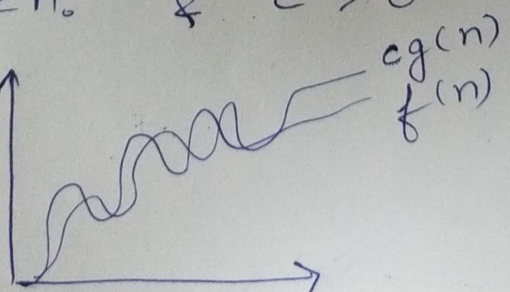
Ques1- Asymptotic notations are mathematical tools used to describe the behaviour of functions as their input grow towards infinity. Used in algorithm analysis to describe efficiency of algorithms in terms of time & space Complexity.

1. Big O notation (O) :-

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

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

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

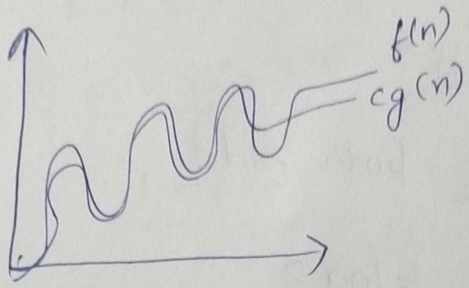


2. Big Ω notation :-

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

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

$f(n) = \Omega(g(n))$ iff $f(n) \geq c\,g(n)$
$\forall\, n \geq n_0$ & $c > 0$.



3. Theata (Θ) notation

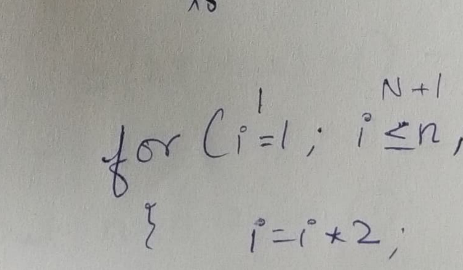
$f(n) = \Theta g(n)$

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

$\Omega(g(n)) \leq \Theta(g(n)) \leq O(g(n))$

$c_1 g(n) \leq f(n) \leq c_2 g(n) \cdot \forall\, n > \max(n_1, n_2)$

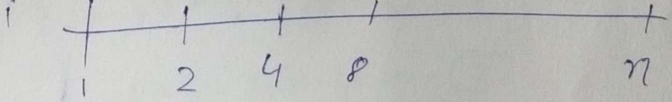
& $c_1, c_2 > 0$.



Ques-2

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

(1 ... N+1)



K = no. of steps

G.P. ~~AP~~.

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

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

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

~~$2n = 2^k$~~

~~$\log n + \log 2 = k \log 2$~~

$\log n$ ~~$+ \log 2$~~ $(1$

$\cancel{\frac{\log n}{\log 2} + 1} = k$

$$2n = 2^k$$

taking $\log_2$ both sides

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

$$\log_2 n + 1 = k \quad (\because \text{constants are ignored})$$

So; time Complexity is $O(\log_2 n)$

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

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

$$\boxed{T(1) = 1}$$

Put $n = n-1$

$\cancel{T(n-1) = 3T}$

$$T(2) = 3T(2-1)$$
$$= 3T(1)$$
$$= 3 \times 1 = 3$$

$$T(3) = 3T(3-1)$$
$$= 3T(2)$$
$$= 3 \times 3 = 9$$

$$T(4) = 3T(4-1)$$
$$= 3T(3)$$

$\because 3 \times 9 = 27$

$=$

$T(n) = 1 + 3 + 9 + 27 \ldots - n$

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

$= \frac{1 \times (1 - 3^n)}{1 - 3}$

$T_n = \frac{3^n - 1}{2}$

~~Ttr~~ $O(3^n)$

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

$T(n) = 2T(n-1) - 1$ — ①

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

$\boxed{T(1) = 1}$

put $n = n-1$

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

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

put $T(n-1)$ in ①

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

$T(n) = 4T(n-2) - 3$ — ②

put $n = n-2$ in ①

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

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

put $T(n-2)$ in ②

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

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

$T(n) = 2^k T(n-k) - 2^k + 1$

$n - k = 1$

$n - 1 = k$

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

$= \cancel{2^{n-1}} - \cancel{2^{n-1}} + 1$

$T(n) = 1$

$O(1)$

Ques-5 -

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

$\cancel{O(N)}$ $O(\sqrt{N})$

Ques-~~6~~7

```
void function(int n) {
    int i, 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++
}
```

$\cancel{O(\sqrt{N})}$ $O(n(\log n)^2)$

Ques-6. void function (int n) {

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

$O(\sqrt{N})$

Ques-8. function (int n) {

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

$O(n^2)$

Ques-9. void function (int n) {

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

$O(n\log n)$

Ques 10

- for n^k, the f^n grows polynomially with the exponent k.
- For c^n, the f^n grows exponentially with base c.

As $n \to \infty$, the exponential f^n c^n grows much faster than any n^k.

c^n is $O(c^n)$

n^k is $\Omega(n^k)$

$$c^n \geq n^k \quad —① $$

$$n \geq (\log_c n)^k \quad —② $$

for eq ① $c \geq n^{\frac{k}{n}}$ as $n \to \infty$ $c \geq 1$

for eq ② n_0 should be chosen acc. based on values of k & c.