PRAKRITI RAWAT
CST SPL 1
Roll no: 11

Date

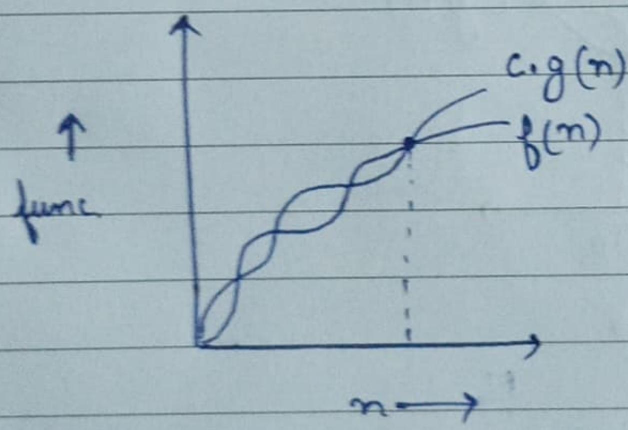
1) Asymptotic Notation.

these notations are used to tell the complexity of an algorithm, when input is very large. These are mathematical notations used to describe running time of an algorithm when the input tends towards a particular value or a limiting value.

- different Asymptotic Notations.

(i) Big-Oh (O) :-

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



g(n) is "tight" upper bound. $f(n) = O(g(n))$ iff

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

$\forall$ $n \geq n_0$ and some constant, $c > 0$

eg:

```
for(i=1; i<=n; i++)
  { print(i);          ———— O(1)
  }
```

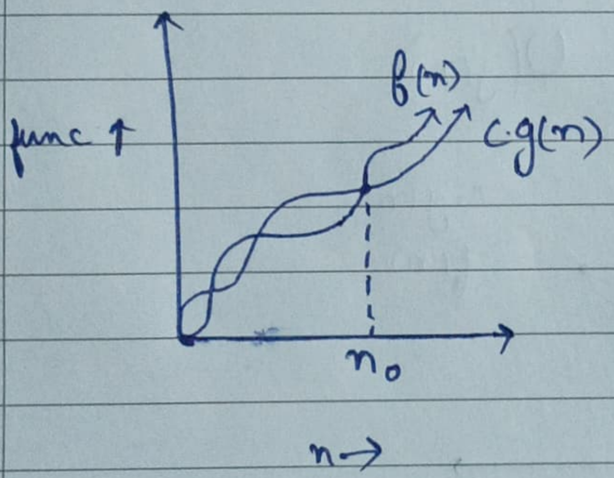
i
1
2
3
⋮

$T(n) = O(n)$

n times
↳ $O(n)$

(ii) Big Omega (Ω):

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



g(n) is 'tight' lowerbound.

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

if,

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

$\forall$ $n \geq n_0$, and some constant $c > 0$

eg: $f(n) = 2n^2 + 3n + 5$ $\qquad g(n) = n^2$

$\Rightarrow \because 0 \le c \cdot g(n) \le f(n)$

$\Rightarrow 0 \le c \cdot n^2 \le 2n^2 + 3n + 5.$

$\Rightarrow c \le 2 + \frac{3}{n} + \frac{5}{n^2}$

On putting $n = \infty$, $\Rightarrow \frac{3}{n} \to \infty$, $\frac{5}{n^2} \to \infty$

$\Rightarrow c = 2,$

$\Rightarrow 2n^2 \le 2n^2 + 3n + 5.$

On putting $n = 1$,

$2 \le 2 + 3 + 5$

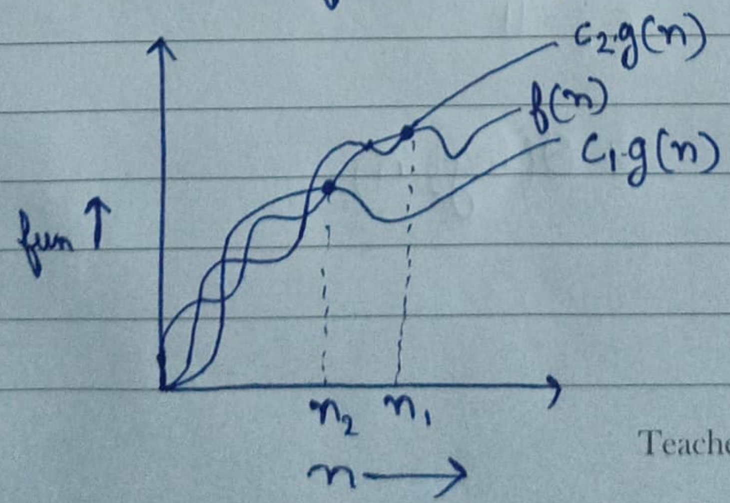
$2 \le 10$ (True)

$\Rightarrow c = 2, \ n = n_0 = 1$

$0 \le 2n^2 \le 2n^2 + 3n + 5$

(iii) Big-Theta (θ):

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



g. $f(n) = 10\log_2 n + 4$ $\quad$ $g(n) = \log_2 n$

$\Rightarrow f(n) \le (c_2 \cdot g(n))$

$\Rightarrow 10\log_2 n + 4 \le 10\log_2 n + \log_2 n$

$10\log_2 n + 4 \le 11\log_2 n$

$c_2 = 11$

$4 \le 11\log_2 n - 10\log_2 n$

$4 \le \log_2 n$

$16 \le n$

here,

$\forall n \geq 16$

$n_2 = 16$

$c_2 = 11$

$f(n) \geq c_1 \cdot g(n)$

$10\log_2 n + 4 \geq 2\log_2 n$

$c_1 = 1$, $n > 0$

$\Rightarrow n_1 = 1$ $\quad$ $\Rightarrow n_0 = \max(n_1, n_2)$

$\Rightarrow n_0 = 16$

$\log_2 n \le 10\log_2 n + 4 \le 11\log_2 n$

$c_1 = 1$

$c_2 = 11$

$\Rightarrow \Theta(\log_2 n)$

Date

(IV) Small oh (o):

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

$g(n)$ is the upper bound of the function $f(n)$.

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

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

$\forall \; n > n_0$

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

(V) Small omega (ω):

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

$\forall \; n > n_0$

and $\forall \; c > 0$.

Q2. Time complexity of :→

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

→ $\underbrace{i = 1, 2, 4, 8, 16 \ldots n}_{k \text{ terms}}$

$a = 1$, $r = 2$

⇒ k^{th} term $\quad t_k = ar^{k-1}$

$$n = 1.2^{k-1}$$

$$= 2^{k-1}$$

take $\log_2$ both sides.

⇒ $\log_2 n = \log_2 2^{k-1}$

$$\log_2 n = (k-1)\log_2 2$$
$$\log_2 n = (k-1)$$
$$k = 1 + \log_2 n$$

$$T(n) = O(k)$$
$$= O(1 + \log_2 n)$$
$$= O(\log_2 n)$$

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

$\Rightarrow \because T(n) = 3T(n-1)$ — ①

put $n = n-1$ in eq^n ①.

$T(n-1) = 3T(n-2)$ — ②

put ② in eq^n ①

$T(n) = 3[3T(n-2)]$ — ③

put $n = (n-2)$ in eqn ①

$T(n-2) = 3T(n-3)$ — ④

put this in eq^n ③

$\Rightarrow T(n) = 9[3T(n-3)]$
$= 27\,T(n-3)$

Generalised form:

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

(put $n-k=0$) $\Rightarrow T(n) = 3^n T(0)$

$T(n) = 3^n$

$\Rightarrow O(3^n)$

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

$\Rightarrow T(n) = 2T(n-1) - 1$ —— ①

put $n-1$ in eqⁿ ①

$\rightarrow T(n-1) = 2T(n-2) - 1$ —— ②

put this in eqⁿ ①

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

$T(n) = 4T(n-2) - 2 - 1$ —— ③

put $n = n-2$ in eqⁿ ①

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

put this value in eqⁿ ③

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

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

$\Rightarrow$ generalised form:

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

put $n - k = 0$

$\Rightarrow n = k$, $T(0) = 1$ (given)

$$\Rightarrow T(n) = 2^n T(0) - 2^{n-1} - 2^{n-2} \ldots - 1$$

$$= 2^n - 2^{n-1} - 2^{n-2} \ldots - 1$$

$$= 2^n - \underbrace{(2^{n-1} + 2^{n-2} + \ldots + 1)}_{k \text{ terms.}}$$

$\Rightarrow a = 2^{n-1}$, $r = 1/2$

$$\text{Sum of GP} = \frac{2^{n-1}\left(1 - (1/2)^{n-1}\right)}{1 - \frac{1}{2}} = 2^n - 2$$

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

$$O(2) \Rightarrow O(1)$$

5.

$S = 1, 3, 6, 10, 15 \dots n$

S
1
3
6
10
15
⋮
n
k times

$\Rightarrow k^{th}$ term,

$$t_k = t_{k-1} + k$$

$$k = t_k - t_{k-1} \quad \text{— (1)}$$

$\Rightarrow k = n - t_{k-1}$

(loop runs k times)

$$\text{Time comp.} = O(1+1+1+n-t_{k-1})$$

but, $t_{k-1} = c$ (constant)

$$\text{T. Comp.} = O(3+n-c)$$
$$= \underline{\underline{O(n)}}$$

6 $i*i \Rightarrow \underbrace{1^2, 2^2, 3^2, 4^2, 5^2 \ldots n}_{k \text{ terms}}$

$i*i$
1^2
2^2
3^2
⋮
n

k^{th} term $\Rightarrow$

$$t_k = k^2$$
$$k^2 = n$$
$$k = n^{1/2}$$

Time complexity $= O(1+1+1+n^{1/2}+1)$
$= O(n^{1/2}) = O(\sqrt{n})$

7 Time complexity of:

```
void func (int n) {                          ——— O(1)
  int i,j,k, count=0;                        ——— O(1)
  for(i=n/2; i<=n; i++)
     for (j=1; j<=n; j=j*2)                  ——— log(n)
        for (k=1; k<=n; k=k*2)               ——— log(n)
           count++                           ——— O(1)
}
```

$i \Rightarrow \frac{n}{2}, \frac{n+2}{2}, \frac{n+4}{2}, \frac{n+6}{2} \ldots$ upto n

$\Rightarrow \frac{n+0\times2}{2}, \frac{n+(1*2)}{2}, \frac{n+2\times2}{2}, \frac{n+3\times2}{2} \ldots$ upto n

General form:

$$t_k = \frac{n + k*2}{2}$$

total terms $= k+1$

$$t_{k+1} = n$$

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

$$n + 2k + 2 = 2n$$

$$2k = n - 2$$

$$k = \frac{n}{2} - 1$$

i	j	k
$\frac{n}{2}$	$\log_2 n$ times	$(\log_2 n)^2$
$\frac{n+2}{2}$	$\log_2 n$ times	$(\log_2 n)^2$
$\frac{n+4}{6}$	$\log_2 n$ times	$(\log_2 n)^2$
⋮	⋮	⋮
n	$\log_2 n$ times	$(\log_2 n)^2$

$\left(\frac{n}{2} - 1\right)$ times $\rightarrow \left(\frac{n}{2} - 1\right)(\log_2 n)^2$

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

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

8 Time complexity of: →

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

for function call,

$$\underbrace{n, n-3, n-6, n-9 \cdots 1}_{k \text{ terms.}}$$

⇒ AP with $d = -3$

$\Rightarrow l = a + (k-1)d$

$1 = n + (k-1)(-3)$

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

$k = \frac{(n-1)+3}{3}$

$= \frac{n+2}{3}$

→ function gives a recursive call $\frac{n+2}{3}$ times

→ Time complexity $= \frac{(n+2)(n)(n)}{3}$

$= n^3$

$\therefore O(n^3)$.

(10) Assuming that $k >= 1$ and $c > 1$ are constraints. Find out the value of c and n_0 for which relation holds.

→ given n^k and c^n is

relation between n^k and c^n is

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

as $n^k \leq a c^n$

$\forall\ n \geq n_0$ and some constant $a > 0$

for $n_0 = 1$

$c = 2$

$\Rightarrow 1^k \leq a \cdot 2^1$

$\therefore n_0 = 1, \quad c = 2.$