23-02-2024 DAA

Name:- Harshita Bhatt
Section:- ML
Roll No.:- 30
Semester:- IV

Assignment - 1

① Asymptotic notations is used to describe the running time of an algorithm- how much time an algorithm takes with a given input, n. The types of notations are:-

a) Big O notation (O)
Denotes the [g(n)] tigth upper bound of f(n)

$$f(n) = O\ g(n)$$
$$\text{iff}\ f(n) = (g(n))$$
$$\forall n = n_0$$

for some constant $c > 0$

Example:-
$f(n) = 2n^2+3n+1$ is $O(n^2)$

b) Big Omega notation (Ω)
Denotes the [g(n)] tight lower bound of f(n).

$$f(n) = \Omega\ g(n)$$
$$f(n) = g(n)$$
$$\forall n = n_0$$

for some constant $c > 0$.

Example:-
$f(n) = n^2$ is $\Omega(n)$

c) Theta (θ) Notation

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

Theta gives both tight upper bound and tight lower bound.

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

$$f(n) = O(g(n)) \text{ and } \Omega(g(n))$$

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

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

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

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

Example:-

$f(n) = n^2$ is $\theta(n^2)$

d) Small O notation (o)

Denotes the [g(n)] upper bound of f(n)

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

iff $f(n) < g(n)$

$\forall\, n > n_0$

(for all) $c > 0$

Example:

$f(n) = n$ is $o(n^2)$

e) Small Omega (ω) notation

Denotes the strict lower bound of a functions growth rate.

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

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

$f(n) > g(n)$

iff $f(n) > g(n)$

$\forall\ n > n_0$

for all $\forall\ c > 0$

Example:

$\cancel{f(n) = \omega(g(n))}$

$f(n) = n^2$ is $\omega(n)$

(2) for (i = 1 to n)

```
{
    i = i * 2;
}
```

$\Rightarrow$ for (i=1; i<=n; i = i * 2)

$$\underbrace{1,\ 2,\ 4,\ 8,\ 16 \ldots\ldots}_{\text{G.P.}}$$

$n = ar^{k-1}$

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

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

$2n = 2^k$

$k = \log_2(2n)$

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

$K = 1 + \log_2 n$

Time complexity $= O(\log_2 n)$

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

$T(0) = 1$

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

for $T(1)$

$T(1) = 3T(0)$

$= 3 \times 1$

for $T(2)$

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

$= 3T(1)$

$= 3T(0)$

$= 3 * 3 \times 1$

for $T(3)$

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

$= 3T(2)$

$= 3 \times 3 \times 3 \times 1$

for $T(4)$

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

$= 3T(3)$

$= 3 \times 3 \times 3 \times 3 \times 1$

for $T(n)$

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

$= 3 \times 3 \times 3 \times 3 \times \cdots \times 3$

$= 3^n$

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

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

$T(0) = 1$

for $T = 1$

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

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

$= 2T(0) - 1$

$= 2 - 1$

$= 1$

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

$= 2T(1) - 1$

$= 2 \times 1 - 1$

$= 2 - 1$

$= 1$

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

$= 2T(2) - 1$

$= 2 \times 1 - 1$

$= 2 - 1$

$= 1$

$\vdots$

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

$= 2n - (2n-2) \cdots 4 - 2 - 1$

$\therefore T(n) = O(1)$

(5) int i=1, s=1;

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

after first iteration :-

$s = s + 1$

after second iteration :-

$s = s + 1 + 2$

it goes on for x iterations.

$1 + 2 + \dots + x <= n$

$(x * (x+1))/2 <= n$

$o(x^2) <= n$

$x = o(\text{root}(n))$

(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 ++;
}
```

x	i	j	K
1	1	1	1
2	1	1+1 = 2	1+1+1+1 = 4
3	1+1	1+1 = 2	1+1+1+1 = 4
4	1+1	1+1+1 = 3	1+1+1+1+1+1+1+1 = 8
5	1+1	1+1+1 = 3	1+1+1+1+1+1+1+1 = 8
:	:	:	:
8	1+1+1	1+1+1+1 = 4	1+1+1 ... +1 = 16
	:	:	
	x	log x	

no. of K increases with power of 2^n, so we can say that it will iterate $\log n$ times and will K while i iterates n times.

$$O(n) = (n * \log_2 n)$$

(6)

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

for i = 1 iteration = 1

i = 2 iteration = 1

i = 3 " = 1

i = 4 " = 1

i = 5 " = 1

⋮

i = n iteration = 1

we can say that time complexity for the function will be O(1)

8) function (int n)

```
{
  if (n == 1)
     return;
  for (i = 1 to n)
  {
     for (y = 1 to n)
     {
        print (" + ");
     }
  }
```

for $x = 1 \rightarrow O(1)$

for $n \geq 2$ ⇣

i	j	$i * j$
1	1	1
2	2	4
3	3	9
4	4	16
⋮	⋮	⋮
n	n	n^2

Since i runs n time for j running $i*j$ we can say Time complexity $= n^2$

$$T.C. = O(n^2) \quad \forall\, n \geq 2$$

$$\text{for } x = 1, \quad T.C. = O(1)$$

(9) function (int n)

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

i	j
1	n
2	n/2
3	n/3
4	n/4
5	n/ 84
6	⋮
⋮	⋮
n	n/n = 1
n	log n

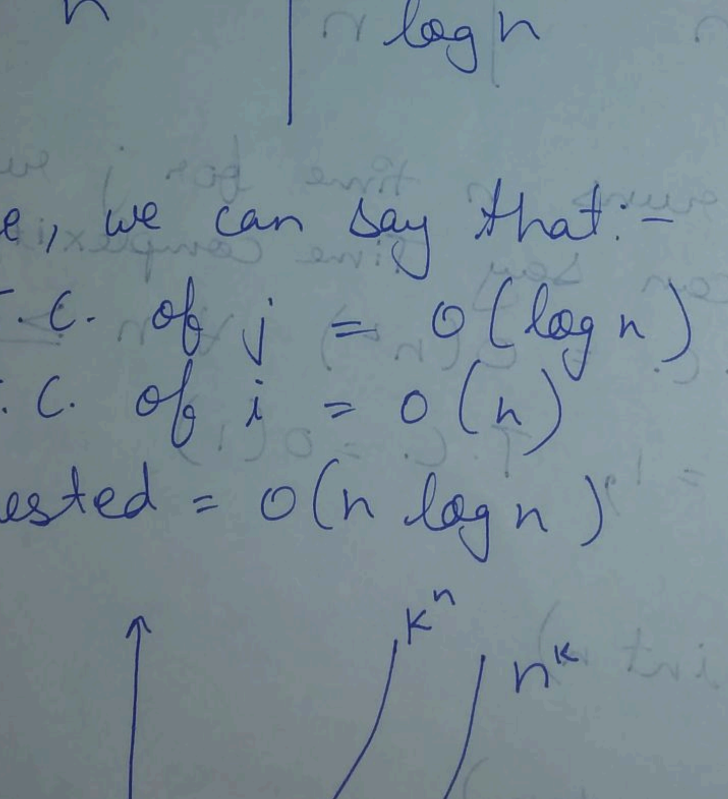
hence, we can say that:-

T.C. of j = O(log n)

T.C. of i = O(n)

nested = O(n log n)

(10)



For value of $n, k, c > 1$ all the value of

$$O(k^n) > O(n^k)$$

this is because (n) exponential time complexity is always greater than integer exponential.

for $n, k, c = 1$, $O(k^n) = O(n^k)$

and for ;

$n, k, c < 1$, the condition is false & the program worst iterate once.