23-02-2024

DAA

Assignment - 1

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

(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)

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

iff $f(n) = (g(n))$

$\forall\, n = no$

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).

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

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

$\forall\, n = no$

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 $f(n) < g(n)$

$\forall\, n > no$

(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:

~~$f(n) \in \omega(g(n))$~~

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

$$T(n) = 3T(n-1)$$
$$= 3 \times 3 \times 3 \times 3 \times \cdots \cdots \times 3$$
$$= 3^n$$
$$T(n) = O(3^n)$$

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

$T(0) = 1$

for $T = 1$

$$T(1) = 2T(n-1) - 1$$
$$= 2T(1-1) - 1$$
$$= 2T(0) - 1$$
$$= 2 - 1$$
$$= 1$$

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

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

⋮

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

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

(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.

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

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

$O(x^{\wedge}2) <= n$

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

(6)

(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++;
}
```

6.
```
void function (int n)
{
    int i, count = 0;
    for (i = 1; i* i<=n; i++)
        count++
}
```