Akanksha Dubey Tutorial - 1

Sec - C

DAA

Roll No - 2016600 (42)

Date.

Page No.

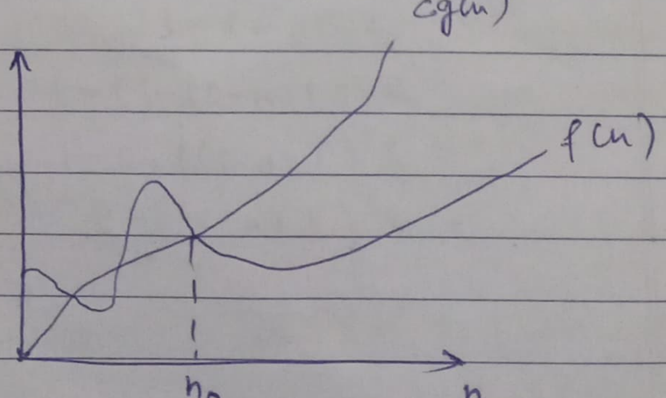
1. What do you understand by Asymptotic notation? Define different types of Asymptotic notation with examples?

Ans:- Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

(a) B-O notation

It represents the upper bound of the running time of an algorithm.
Thus it gives the worst case complexity of an algorithm.

$O(g(n)) = \{ f(n)$: there exist +ve constant c and n_0 such that $0 \le f(n) \le c g(n)$ for all $n \ge n_0 \}$



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

b. Omega Notatio

It represents

time of an alg

Thus it provid

algorithm.

$\Omega(g(n)) =$

(c) Theta Nota

Encloses th

represents t

time of an

average - ca

$\theta(g(n))$

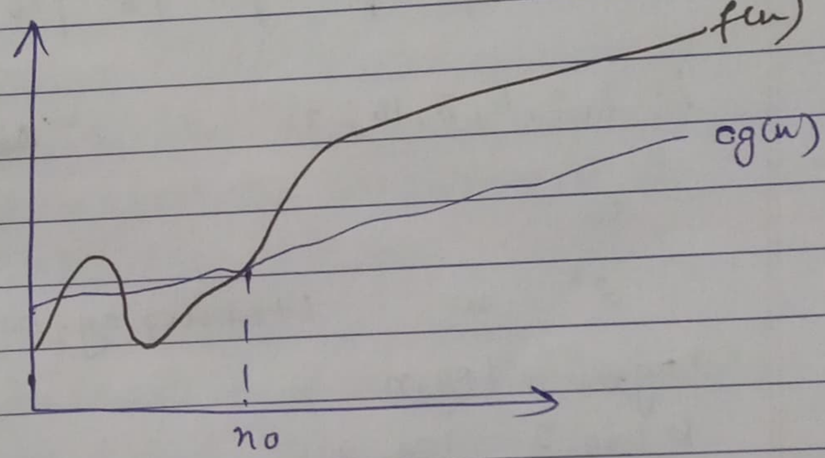
Date.
Page No.

b. Omega Notation (Ω-notation)

It represents the lower bound of the running time of an algorithm.

Thus it provides the best case complexity of an algorithm.

$\Omega(g(n)) = \{ f(n)$: there exist the constants c & n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0 \}$

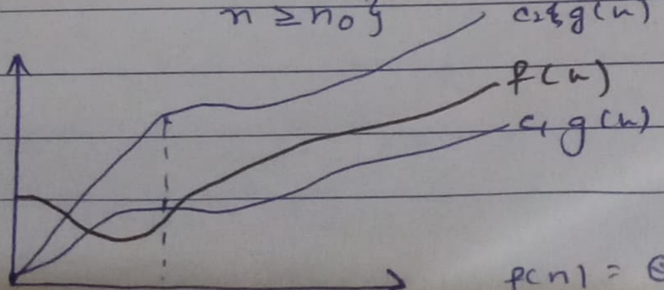


$f(n) = \Omega(g(n))$

(c) Theta Notation (Θ-notation)

Encloses the function from above & below. It represents the upper and lower bound of the running time of an algorithm it is used for analyzing the average-case complexity of an algorithm.

$\Theta(g(n)) = \{ f(n)$: there exist the constants c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$



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

Name - Akanksha Dubey
Section - C
Roll No. 2016600 (42)

Tutorial - 1
DAA

Date.
Page No.

2. What should be time complexity of -

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

for loop will run for following values of i

Pn power of 2	2^0	2^1	2^2	2^3	2^4	2^5	
i	1	2	4	8	16	32	..

i = 1, 2, 4, 8, 16, 32 .... 2^k ~~times~~ means k times

so

$2^k = n$ (taking $\log_2$ both side)

$\log_2 2^k = \log_2 n$

$k \log_2 2 = \log_2 n$

$k = \log_2 n$

where k is the time complexity of the program so the Time complexity of above program is

$\boxed{T(n) = O(\log_2 n)}$

Date.
Page No.

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

Solve using substitution

$$T(n) = 3T(n-1)$$
$$= 3(3T(n-2))$$
$$= 32T(n-2)$$
$$= 33T(n-3)$$
$$\cdots$$
$$\cdots$$
$$= 3nT(n-n)$$
$$= 3nT(0)$$
$$= 3n.$$

This shows that the complexity of this function is $O(3n)$.

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

solve by using substitution.

$$T(n) = 2T(n-1) - 1$$
$$= 2(2T(n-2)-1) - 1$$
$$= 2^2 T(n-2) - 2 - 1$$
$$= 2^2(2T(n-3)-1) - 2 - 1$$
$$= 2^3 T(n-3) - 2^2 - 2 - 1$$
$$\vdots$$
$$= 2^k T(n-k) - 2^{k-1} - 2^{k-2} - \cdots - 2^2 - 2^1 - 2^0$$

Let $T(1) = 1$

$n-k = 1 \Rightarrow k = n-1$

put $k = n-1$

$$T(n) = 2^{n-1} T(1) - [2^0 + 2^1 + 2^2 + \cdots 2^{n-3} + 2^{n-2}]$$
$$= 2^{n-1} \times 1 - [2^{n-1} - 1] = 2^{n-1} - 2^{n-1} + 1$$

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

Date.
Page No.

5. What is Time complexity of

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

We can say

$s_i = s_{i-1} + i$

if k is the total no. of iteration taken by program then while loop terminates if

$1+2+3+ \dots k = [k(k+1)/2] > n$

so

$k = O(\sqrt{n})$

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

6.

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

$TC = O(n*n)$

$\boxed{T(n) = O(n^2)}$

7. void fun
{ int
for (i=
fo

}

Time

8. functi
{ i

for

f
}

T.C

Date.
Page No.

7.
```
void function (int n)
{ int i, j, k, count = 0;
  for (i = n/2; i <= n; i++)          // Executes O(n/2) times
    for (j = 1; j <= n; j = j*2)      // Executes O(log n) times
      for (k = 1; k <= n; k = k*2)    // Executes O(log n) times
          count++;
}
```

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

$\boxed{T(n) = O(n \log^2 n)}$

8.
```
function (int n)
{ if (n == 1)
     return;
  for (i = 1 to n)          → n time
  { for (j = 1 to n)        — n time
     { printf("*");
     }
  }
  function (n-3);           — (n-3) times
}
```

$O(n * n * (n-3))$

$= O(n^3 - 3n^2)$

$\boxed{T.C = O(n^3)}$

Date.
Page No.

9. void fun(int n)

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

for loop for (j=1 to n ; j++)

$TC = O(\log n)$

for (i=1 to n)

$TC = O(n)$

$\therefore T.C = O(n \log n)$

10. Asymptotic relation b/w n^k & c^n is

$$\boxed{n^k = O(c^n)}$$

ie $n^k < c_1 . c^n$

$n^k = c_1 . c^n$

put $n=2$, $k=2$ & $c=2$

$(2)^2 = c_1 . (2)^2$

$4 = c_1 . 4$

$c_1 = 1$

$\therefore$ for $c_1 = 1$, the relation holds.