Tutorial I

Q1 Asymptotic notation : Asymptotic notation are function used to study space and time complexity of the algorithm these notation can be used to study and campare growth of algorithms ~~in~~

There are fine type of Asymptotic notation:

1. Big 'Oh' - (O).
2. little / small 'Oh' - (o).
3. Big Omega (Ω).
4. Small / Little Omega (ω).
5. Theta (Θ).

Big 'Oh' : Big 'oh' notation is used in understanding complexity of algorithm by giving its upper bound $f(n) = n$; $g(n) = n^2$.

$f(n) = O$ (function) $\quad f(n) = O(g(n))$.

if for $n > n_0$ for which ~~∀~~ theer exist constant c

: $\forall n > n_0 \quad f(n) \leq C(gn)$

~~Big~~ $\overset{\Omega}{\cancel{\theta}}$ = Big Omega (Ω) Big Omega (Ω) notation is used to understand complexity of algorithm by giving its lower bound

$f(n) = n$; $g(n) = n^2$

$f(n) = \Omega$ (function) $\quad f(\overset{n}{\cancel{cn}}) = \Omega(g(n))$.

if for $n > n_0$ for which theer exist constant c

$\forall \; n > n_0 \quad f(n) \geq c(g(n))$

little 'oh' : little 'oh' notation is used understanding complexity of algorithm by giving upper bound only

$f(n) = o$ (function) $\quad f(n) = n$; $g(n) = n^2$.

$f(n) = \overset{o}{\cancel{\Omega}}(g(n)$

if $n > n_0$ for which theer exist constant c $\forall \; n > n_0$

$f(n) > c(g(n))$.

little omega(ω) = ~~little~~ little omega notation based on understanding of growth of algorithm ~~in~~ which we use only Lower bound.

i.e $\forall n > n_0$ and for some constant c $f(n) = n$; ~~f~~ $g(n) = $ ~~n^2~~ n^4.

$- f(n) \geq c(g(n))$ — then $f(n) = \Omega(g(n))$

Theta (θ) = Theta (θ) notation based on understanding to give range of the growth of the function.

$f(n) = n$ then $f(n) = \theta(n)$ will exist if for any two Constants ~~constants~~ c_1 and c_2 $\exists$ relation that $\forall n_0 > n_0$ $c_1 n \leq f(n) \leq c_2 n$:

(2.) Total steps for = $(n+1) + n$

$= 2n+1$

$O(n)$

(3.) ~~1T~~ $T(n) = 3T(n-1)$ if $n > 0$

1 ~~or~~ otherwise ~~$n=0$~~

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

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

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

$n-k=0$

$n=k$

$T(n) = 3^n (T(0))$

$T(n) = 3^n$

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

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

1 otherwise

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

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

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

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

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

$-$ ~~3~~ $2 - 1$

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

$-2-1$

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

$T(n) = 8(2T(n-4) - 1)$

$= 16T(n-4) - 8 - 4 - 2 - 1$

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

$- 2^{k-1} - 2^{k-2} \ldots 1$

Q5:

Solution:

```
int i=1, s=1;
while (s<=n) √n-2 +1
{
      i++; → K   s=s+i; → K

   print("#"); → K

}
```

$$\sqrt{n} + \overset{\sqrt{n}}{\cancel{k}} + \overset{\sqrt{n}}{\cancel{k}} + \overset{\sqrt{n}}{\cancel{k}} + 1$$

$O(\sqrt{n})$ - Ans.

i	s
1	$s = 1 + 1$
2	$s = 1 + 1 + 2$
⋮	⋮

$s = 1 + 1 + 2 + \dots + k$

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

$k^2 + 2 = n$

$k^2 = n - 2$

$k = \sqrt{n-2}$

Q6:

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

$1*1$

$2*2$

$3*3$

⋮

k^2

$k^2 = n$

$k = \sqrt{n}$

$O(\sqrt{n})$ Ans

Q7 :-

```
~~n/2 +~~   void function ( int n)
  {
     f int i, j, k, count =0;
     for ( i = n/2 ; i<=n ; i++)  (n/2 +1)
                                          +1)
                                 j=        ↑
                                 ↑
        {   for (j = 1 ; j <= n ; j *2) (log(n)(n/2 + 1) ~~+1~~
                                          ( log(n)+1)(
                                            ↑
            {  for( k= 1; k<=n; k= k*2) ~~( log(n))~~( log(n))
                                                 (n/2)

                {   count ++;  log(n) log(n) n/2

                }
            }
        }
  }
```

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

Q8 :-

```
function (int n)        →  Recurrence
                           Relation
  {   if (n==1) return;     T(n)= T(n/3)+ n^2
                            ~~n lo~~ n^log3 1
    for (i=1 to n)
                              n^0 = 1
       { for (j=1 to n)       n^2 > 1
                +
         printf ("~~*~~");     T(n)= Θ(~~+~~n^2) A
       }  function (n-3);                  Ans
  }
```

$T(n) = T(n/3) + n^2$

$n^{\log_3 1}$

$n^0 = 1$

$n^2 > 1$

$T(n) = \Theta(n^2)$ Ans

Q9 :-

$O(n\sqrt{n})$

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

Ques^n 10 :-

Solution :-
~~Relate~~ Relation between c^~~2~~n and n^k is

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

as ~~n^k~~ n=1
c=2

1^k ~~=~~ $\leq 2^n$

where n= 1

Hence

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