DESIGN & ANALYSIS OF ALGORITHM

TUTORIAL-1

1. Asymptotic notations are the mathematical notations and are used to describe the complexity (i.e. running time) of an algorithm when the input tends towards a particular value or a limiting value.

Different type of Asymptotic Notations:

(i) Big-O (O)

Big-O notation specifically describes worst case scenario. It represents the tight upper bound running time complexity of an algorithm.

$f(n) \leq c.g(n) \quad \forall n \geq n_0$
and some constant $c > 0$

Eg:- O(1), O(n), O(log n)

```
for(int i= 1; i<=n; i++)
  {
    sum = sum+i;
  }
```

the complexity of above example is O(n)

(ii) Omega (Ω)

Omega notation specifically describes least case scenario. It represents the tight lower bound running time complexity of an algorithm.

$f(n) \geq c.g(n) \quad \forall n \geq n_0$
& some constant $c > 0$

Eg $\Omega(1)$, $\Omega(\log n)$, etc.

For binary search, the complexity will be $\Omega(1)$.

(iii) Theta (Θ)

This notation describes both tight upper bound and tight lower bound of an algorithm, so it defines exact asymptotic behaviour. In real case scenario the algorithm not always run on best and worst cases, the average running time lies between best and worst cases and can be represented by 'Θ' notation.

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad ; \forall n \geq \max(n_1, n_2)$$

& some constant $c_1 > 0$ &
$c_2 > 0$

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

$\Rightarrow i = 1, 2, 4, 8, \ldots, n$

$a = 1, r = 2$

k^{th} term of GP, $t_k = a * r^{k-1}$

$n = 1 * (2)^{k-1}$

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

$2n = 2^k$

$\log_2 2n = K . \log_2 2$

$\log_2 2 + \log n = K$

$1 + \log(n) = K$

3. $T(n) = 3T(n-1)$ — (i)

$T(1) = 1$

put $n = n-1$ in eq (i)

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

putting the value of $T(n-1)$ in eq (i)

$\Rightarrow T(n) = 9T(n-2)$ — (ii)

put $n = n-2$ in eq (i)

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

putting the value of $T(n-2)$ in eq (ii)

$\Rightarrow T(n) = 27T(n-3)$ —— (iii)

put $n = n-3$ in eq (i)

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

putting the value of $T(n-3)$ in eq (iii)

$\Rightarrow T(n) = 81T(n-4)$

for every constant K

$T(n) = 3^K . T(n-K)$ —— (iv)

let $n-K = 1$

$K = n-1$

putting value of K in eq (iv)

$T(n) = 3^{n-1} . T(1)$

$\because T(1) = 1$

$\Rightarrow T(n) = 3^{n-1}$

$\Rightarrow O(3^n)$

4. $T(n) = 2T(n-1) - 1$ ——(i)

$T(1) = 1$

put $n = n-1$ in eq (i)

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

putting value of $T(n-1)$ in eq (i)

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

put $n = n-2$ in eq (i)

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

putting value of $T(n-2)$ in eq (ii)

$\Rightarrow T(n) = 8T(n-3) - 7$ —— (iii)

put $n = n-3$ in eq (i)

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

putting value of $T(n-3)$ in eq. (iii)

$\Rightarrow T(n) = 16T(n-4) - 15$

for any constant K

$T(n) = 2^K . T(n-K) - (2^K - 1)$ ——(iv)

Let $n-K = 1$

$K = n-1$

putting value of K in eq (iv)

$T(n) = 2^{n-1} . T(1) - (2^{n-1} - 1)$

$= 2^{n-1} - 2^{n-1} + 1$

$= 1$

$T(n) = 1$

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

After 1st iteration
$S = S + 1$

After 2nd iteration
$S = S + 1 + 2$

Let the loop go for 'K' iteration

$\Rightarrow 1 + 2 + \dots + K \leq n$

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

or $\frac{K^2 + K}{2} = n$

ignoring constants & lower order term

$\Rightarrow K^2 = n$

$K = \sqrt{n}$

$\therefore O(\sqrt{n})$

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

Let loop will iterate for K times

$\Rightarrow K^2 <= n$

$\Rightarrow K = \sqrt{n}$

$\therefore O(\sqrt{n})$

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 ++
}

For the last loop, time complexity = $O(\log n)$

Similarly for the second loop, time complexity = $O(\log n)$

$\therefore$ Total time complexity = $O(\log^2 n)$

The first loop's time complexity = ~~O log~~ $O(n)$

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

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

$(n-3), (n-6), (n-9) - - - - (1)$

$a = n-3$, $d = -3$

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

$3K = n - 1$

$K = \frac{n-1}{3} = O(n * n2)$

$= O(n^3)$

9. void function(int n){
for(i=1 to n){
for(j=1; j<=n; j=j+i)
printf("*")
}
}

for loop; for(j=1; j<=n; j=j+i)

Time complexity = O(log n)

for outer loop,

Time complexity = O(n)

∴ Total complexity = O(n log n)

10. The asymptotic relation between n^K & C^n is

$n^K = O(C^n)$

i.e. $n^K \leq C_1.(C^n)$

$\Rightarrow n^K = C_1.C^n$

put $n=2$, $K=2$ & $C=2$

$(2)^2 = C_1.(2)^2$

$4 = C_1.4$

$C_1 = 1$

∴ for $C_1 = 1$, the relation holds.