DESIGN AND ANALYSIS OF ALGORITHMS

TUTORIAL-I

Ques 1. What do you understand by Asymptotic notations. Define different asymptotic notation with examples.

Ans 1. Asymptotic Notations are mathematical tools to represent the time of complexity of algorithms for asymptotic analysis. Asymptotic Notations are used to describe the running time of algorithms.

Types of Asymptotic Notation :

1. Big-Oh Notation: (O) → It describes the upper bound, it tells about the maximum complexity an algorithm can have.

   Consider two functions f(n) and g(n).

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

   iff $f(n) \leq c\,g(n)$. , $n \geq n_0$ & $c > 0$

2. Big Omega (Ω) → It describes the strict lower bound of any algorithm.

   Consider two functions f(n) and g(n).

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

   iff $f(n) \geq c \cdot g(n)$, for $n \geq n_0$ & $c > 0$

3. Theta Notation (θ) → It describes the running time of an algorithm.

   Consider two functions f(n) and g(n)

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

   iff $c_1 g(n_1) \leq f(n) \leq c_2 \cdot g(n_1)$

   $n_1 \geq n_2$

   c_1 & $c_2 > 0$

(d) Small 'o' → It describes the ~~tight~~ upper bound of an algorithm.
Consider two functions $f(n)$ and $g(n)$

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

iff $f(n) < c \cdot g(n)$

for $n \geq n_0$, $c > 0$

(e) Small 'ω' → Let $f(n)$ and $g(n)$ be functions that map positive integers to positive real numbers.
then, $f(n) \in \omega\, g(n)$

if $c > 0$, $n_0 \geq 1$

Such that $f(n) > c \cdot g(n) \geq 0$ for every $n \geq n_0$.

It describes the lower bound that is not asymptotically tight.

Ques 2. What should be the time complexity of -

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

Ans 2. Since, $i = 1, 2, 4, 8, 16 \ldots\ldots 2^k$ times

So, $2^k = n$

$k = \log n$

So, time complexity : $T(n) = O(\log n)$

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

Ans 3. Applying substitution method,

$\because \; T(n) = 3T(n-1)$

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

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

$\ldots$

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

$= 3^n T(0)$

$= 3^n (1)$

Time complexity : $T(n) = O(3^n)$

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

Ans. Applying substitution method:

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

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

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

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

$= 2^3 T(n-3) - 2^2 - 2^1 - 2^0$

$\ldots$

$= 2^n T(n-n) - 2^{n-1} - 2^{n-2} - \ldots 2^2 - 2^1 - 2^0$

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

$= 1$

$T(n) = O(1)$

Ques 5. What should be the time complexity of -

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

Ans. Initially, $i=1$; $s=1$

1^{st} iteration : $i=2$; $s=3$

2^{nd} iteration : $i=3$; $s=6$ ...

So, $$s = \frac{k(k+1)}{2}$$

Also, terminating condition :

$$\frac{k^2+k}{2} > n$$

$$k > \sqrt{n}$$

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

Ques 6. Time complexity of -

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

Ans. Time complexity : $T(n) = O(n)$

Ques 7. Time complexity of -

```
void function (int n) {
    int i, j, k, count =0;
    for( i=n/2 ; i<=n ; i++)          --- (i)
        for( j=1 ; j<n ; j=j*2)        --- (ii)
            for( k=1 ; k<n ; k*2)      --- (iii)
                count ++ }
```

Ans. Loop 1 executes : $n/2$ times

loop 2 executes : $\log n$ times

loop 3 executes : $\log n$ ~~log~~ times

$$T(n) = O(n \log^2 n)$$

Ques8. Time complexity of –

```
function(int n) {
    if(n==1) return;          ...1
    for(i=1 to n) {
        for(j=1 to n) {       ...n2
            printf(" * ");
        }
    }
    function(n-3)             ... T(n-3)
}
```

Ans8. So, $T.C = T(n) = T(n-3) + n^2$

Here $T(1) = 1$

Put $n = 4$

$T(4) = T(1) + n^2$

$T(4) = n^2 = 16$

Put $n = 7$

$T(7) = T(7-3) + 7^2$

$= T(4) + 49$

$= 1^2 + 4^2 + 7^2$

So, $T(n) = 1^2 + 4^2 + 7^2 + 10^2 + \dots n^2$

$= \frac{n(n+1)(2n+1)}{6} = O(n^3)$

Ques9. Time complexity of :

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

Ans. So, for i upto n it will take n^2.

So, $T(n) = O(n^2)$

Q10. For the functions, n^k and c^n ... ?

Ans. Since, $f_1(n) = n^k$ and $f_2(n) = c^n$ here $k \geq 1$, $c > 1$.

So, $f_1(n) = O(f_2(n)) = O(c^n)$

is $n^k < G * c^n$

Here G = constant.

Ques8. Time complexity of –

```
function(int n) {
    if(n==1) return;            ... 1
    for(i=1 to n) {
        for(j=1 to n) {         ... n2
            printf("*");
        }
    }
    function(n-3)               ... T(n-3)
}
```

Ans8. So, $T.C = T(n) = T(n-3) + n^2$

Here $T(1) = 1$

Put $n = 4$

$T(4) = T(1) + n^2$

$T(4) = n^2 = 16$

Put $n = 7$

$T(7) = T(7-3) + 7^2$

$= T(4) + 49$

$= 1^2 + 4^2 + 7^2$

So, $T(n) = 1^2 + 4^2 + 7^2 + 10^2 + \dots n^2$

$= \frac{n(n+1)(2n+1)}{6} = O(n^3)$

Ques9. Time complexity of :

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

Ans. So, for i upto n it will take n^2.

So, $T(n) = O(n^2)$

Q10. For the functions, n^k and c^n ... ?

Ans. Since, $f_1(n) = n^k$ and $f_2(n) = c^n$ here $k \geq 1$, $c > 1$.

So, $f_1(n) = O(f_2(n)) = O(c^n)$

is $n^k < G * c^n$

Here G = constant.