Name - Gaurav Bhandari Sec - F Roll no. - 58

U. Roll no. - 2016747 ID - 20021182

Design & Analysis Of Algorithm
Tutorial - 1

1. Asymptotic notations are the mathematical notations 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.

$$\boxed{f(n) \leq C.g(n)} \quad \forall \ n \geq n_0 \ \& \text{ some constant } c > 0$$

eg. $O(1)$, $O(n)$, $O(\log n)$

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

The complexity of above example is $O(n)$

ii) Omega (Ω)

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

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

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

for Binary Search, the time complexity will be $\Omega(1)$

iii) Theta (θ)

This notation describes both tight upper bound & tight lower bound of an algorithm, so it defines exact asymtotic behaviour. In real case scenario the algorithm not always run on best & worst ~~casess~~ cases, the avg running time lies b/w best & worst and can be represented by 'θ' notation

$$\boxed{C_1\, g(n) \leq f(n) \leq C_2\, g(n)}$$

$\forall \ n \geq \max(n_1, n_2)$

& some constant $C_1 > 0$ & $C_2 > 0$.

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

$\Rightarrow$ $i = 1, 2, \cancel{3}, 4, 8 \ldots\ldots, n$

$a = 1, \quad r = 2$

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

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

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

$$2n = 2^k$$

$$\Rightarrow \quad \log_2(2n) = k \log_2 2$$

$$\log_2 2 + \log_2 n = k$$

$$k = \log n + 1$$

$\therefore$ Time Complexity $= O(\log n)$

3. $T(n) = 3T(n-1)$ —— ①

$T(1) = 1$

put $n = n-1$ in eq ①

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

putting the value of $T(n-1)$ in eq ①

$$\Rightarrow T(n) = 9T(n-2) \quad \text{—— ②}$$

put $n = n-2$ in eq ①

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

putting the value of $T(n-2)$ in eq ②

$\Rightarrow T(n) = \cancel{3T}\ 27T(n-3)$ —— ③

put $n = n-3$ in eq ①

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

putting the value of $T(n-3)$ in eq ③

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

For any constant k

$T(n) = 3^k . T(n-k)$ —— ④

let $n-k = 1$

$k = n-1$

putting value of k in eq ④

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

$\because T(1) = 1$

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

$\Rightarrow \boxed{O(3^n)}$

4. $T(n) = 2T(n-1) - 1$ —— ①

$T(1) = 1$

put $n = n-1$ in ~~eg~~ eq. ①

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

putting value of $T(n-1)$ in eq ①

$\Rightarrow T(n) = \cancel{4(T} \; 4T(n-2) - 3$ —— ②

put $n = n-2$ in eq ①

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

putting value of $T(n-2)$ in eq ②

$\Rightarrow T(n) = 8T(n-3) - \cancel{5}\,7$ —— ③

put $n = n-3$ in. eq ①

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

putting value of $T(n-3)$ in eq ③

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

For any constant k

$T(n) = 2^k . T(n-k) - (2^k - 1)$ —— ④

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

putting value of k in eq ④

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

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

$= 1$

$T(n) = 1$

5.

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

After 1^{st} iteration

$$S = S+1;$$

After 2^{nd} iteration

$$S = S+1+2$$

Let the loop goes for 'k' iteration

$$\Rightarrow 1+2+3+ \ldots\ldots\ldots +k \leq n$$

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

$$\text{or} \quad \frac{k^2+k}{2} \cancel{<} = n$$

ignoring constant & lower order terms

$$\Rightarrow k^2 = n$$

$$k = \sqrt{n}$$

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

6)

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

Let loop will iterate for k times

$$\Rightarrow k^2 <= n$$

$$k = \sqrt{n}$$

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

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 loop, for(k=1; k<=n; k=k*2)

time complexity $= O(\log n)$

Similarly for loop, for(j=1; j<=n; j=j*2)

time complexity $= O(\log n)$

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

The outer most loop $\rightarrow O(n)$

$\therefore \Rightarrow \boxed{O(n \log^2 n)}$

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

for both the loops

Time complexity $= O(n^2)$

& for the function calling

Time complexity $= O(n)$

$\therefore$ Total time complexity $= \boxed{O(n^3)}$

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

for ^ loop,

Time complexity = $O(\log n)$

for outer loop,

Time complexity = $O(n)$

$\boxed{\therefore \text{Total complexity} = O(n \log n)}$

10. The asymptotic notation between n^k & c^n is

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

$$\therefore n^k \leq c_1 (c^n)$$

$$n^k = c_1 . c^n$$

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 notation holds.