Name - Palak Tayal
Section - F
Roll no - 18

Design & Analysis of Algorithms

Tutorial-1

Que1. Asymptotic notations are the mathematical notations used to descride the complexity (i.e, running time) of an algorithm when the input tends towards a particular value or a limiting value.

Different types of Asymptotic Notations:-

i.) Big-O [O]

It specifically describes worst case scenario. It represents the "thight" upper bound running time complexity of an algorithm.

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

E.g. 1.

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

Complexity - $O(n)$

E.g. 2.

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

Complexity = $O(\log_2 n)$

ii.) Omega (Ω)

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

$$\boxed{f(n) \geq c \cdot g(n)} \quad \forall\ n \geq n_0\ ,\ c > 0$$

E.g. for Binary Search,

complexity = $\Omega(1)$

iii) Theta (Θ)

This notation describes both tight upper & 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 avg running time lies b/w best & worst and can be represented by 'Θ' notation.

$$\boxed{c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)}$$

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

& $c_1 > 0,\ c_2 > 0$

Que 2. Complexity of :

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

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

$a = 1 \quad , \quad r = 2$

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

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

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

$$2n = 2^k$$

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

$$1 + \log_2 n = k$$

$$k = \log_2(n) + 1$$

Complexity = $O(\log n)$

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

$T(1) = 1$

let,

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

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

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

put value of $T(n-1)$ in eq^n (1)

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

$$T(n) = 3^2 T(n-2) \quad ---- \quad (11)$$

put $n = n-2$ in eqⁿ (i)

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

put value of $T(n-2)$ in eqⁿ (ii)

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

$$T(n) = 3^3(T(n-3)] \quad \ldots \text{(iii)}$$

from eqⁿ (i), (ii) & (iii)

$$T(n) = 3^k [T(n-k)] \quad \ldots \text{(iv)}$$

$$T(1) = 1$$

$$n-k = 1$$

$$k = n-1$$

put value of k in eqⁿ (v)

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

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

$$\boxed{\text{Complexity} = O(3^n)}$$

Que4

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

let,

$$T(n) = 2T(n-1) - 1 \quad \ldots\ldots \text{(I)}$$

put $n = n-1$ in eq^n (I)

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

put value of $T(n-1)$ in eq^n (I)

$$T(n) = 2[2T(n-2) - 1] - 1$$
$$T(n) = 2^2 T(n-2) - 2 - 1 \quad \ldots\ldots \text{(II)}$$

put $n = n-2$ in eq^n (I)

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

put value of $T(n-2)$ in eq^n (II)

$$T(n) = 2^2[2T(n-3) - 1] - 2 - 1$$
$$T(n) = 2^3 T(n-3) - 2^2 - 2^1 - 1 \quad \ldots\ldots \text{(III)}$$

from eq^n (I), (II) & (III)

$$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} \ldots\ldots - 2^{k-k} \quad \ldots \text{(IV)}$$

$$n - k = 1$$
$$k = n-1$$

put value of k in eq^n (IV)

$$T(n) = 2^{n-1} T(1) - 2^{n-2} - 2^{n-3} \ldots\ldots - 2^{0}$$

$$T(n) = 2^n \left[\frac{1}{2} - \frac{1}{2^2} - \frac{1}{2^3} - \ldots - \frac{1}{2^n}\right]$$

$$T(n) = 2^{n-1} - \left[2^n \left(\frac{1}{2^2} + \frac{1}{2^3} + \ldots + \frac{1}{2^n}\right)\right]$$

$$T_n = 2^{n-1} - \left[2^n \cdot \frac{1}{4} \times \frac{1-(1/4)^k}{(1-1/4)}\right]$$

$$= 2^{n-1} - 2^n\left[\frac{1}{\cancel{4}} \times \frac{\cancel{4}}{3} \times 1 - \frac{1}{(4)^k}\right]$$

$$= 2^{n-1} - 2^n\left[\frac{2^{2k}-1}{(2)^{2k}}\right]$$

$$= 2^{n-1} - \frac{2^n(2^{2n-2}-1)}{2^{(2n-2)}}$$

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

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

$$\boxed{T(n) = O(1)}$$

Ques 5

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

After 1st iteration,

$s = s+1$

2nd,

$s = s+1+2$

let the loop goes for 'k' iterations

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

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

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

by ignoring lower order terms

$k^2 = n$

$k = \sqrt{n}$

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

Ans6

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

loop will iterate for k times

$k^2 <= n$

$k = \sqrt{n}$

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

Ans7

```
void func(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 loop k, Complexity = $O(\log_2 n)$

for loop j, Complexity = $O(\log_2 n)$

for loop i, Complexity = $O(n)$

total complexity, $O(\log n * \log n * n)$

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

Que8. func(int n)

```
{
  if(n==1)
    return;
  for(i=1 to n) //O(n)
  {
    for(j=1 to n)  //O(n)
    {
      printf("*");
    }
  }
  func(n-3); //O(n)
}
```

for both loops,

comp = $O(n^2)$

for f^n calling,

complexity = $O(n)$

total complexity = $\boxed{O(n^3)}$

Que9

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

for loop j,

complexity = $O(\log(n))$

for loop i,

complexity = $O(n)$

Total,

$= \boxed{O(n.\log n)}$

Que 10

The asymptotic notation between n^k & c^n is;

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

$$n^k \le C_1 \cdot (c^n)$$

$$n^k = C_1 \cdot c^n$$

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

$$2^2 = C_1 \cdot 2^2$$

$$\boxed{C_1 = 1}$$