NAME : SHIVAM SUNDRIYAL SEC : D ROLL NO : 37

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.

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

e.g. $O(1)$, $O(n)$, $O(\log n)$

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

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

(ii) Omega (Ω)

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

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

e.g. $\Omega(1)$, $\Omega(\log n)$, etc.

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

(iii) <u>Theta</u> (θ)

This notation describes both tight upper bound & tight lower bound of an algorithm, so it defines exact asymptotic behaviour. In real case scenario the algorithm not always run on best & worst 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 constt. $c_1 > 0$ & $c_2 > 0$

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

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

$a = 1$, $r = 2$

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

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

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

$2n = 2^k$

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

$\log_2 2 + \log_2 n = k$

$\Rightarrow k = \log(n) + 1$

$\therefore$ Time Complexity

$\Rightarrow \boxed{O(\log(n))}$

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)

$\rightarrow T(n) = 9T(n-2)$ —— (2)

put $n = n-2$ in eq. (1)

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

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

$\rightarrow T(n) = 27T(n-3)$ —— (3)

put $n = n-3$ in eq. (1)

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

putting the value of $T(n-3)$ in eq. (2)

$\rightarrow T(n) = 81\,T(n-4)$

for any constt. k

$T(n) = 3^k \cdot T(n-k)$ —— (4)

let $n-k = 1$

$\Rightarrow \cancel{n = k+1}$

$k = n-1$

putting value of k in eq. (4)

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

$\because T(1) = 1$

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

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

4.

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

$$T(1) = 1$$

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

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

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

$$\rightarrow T(n) = 4T(n-2) - 3 \quad \text{--- (2)}$$

put $n = n-2$ in eq. (1)

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

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

$$\rightarrow T(n) = 8T(n-3) - 7 \quad \text{--- (3)}$$

put $n = n-3$ in eq. (1)

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

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

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

for any constt. k

$$T(n) = 2^k \cdot T(n-k) - (2^k - 1) \quad \text{--- (4)}$$

let $n - k = 1$

$$\Rightarrow k = n - 1$$

putting value of k in eq. (4)

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

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

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

$$= 1$$

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

$\cancel{a = \frac{1}{2}}$ $\cancel{r = \frac{1}{2}}$

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

$$\Rightarrow \quad 1 + 2 + \cdots + k \leq n$$

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

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

- ignoring constants & lower order term

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

$$k = \sqrt{n}$$

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

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 outermost loop $\rightarrow$ $O(n)$

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

6.

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

Let Loop will iterate for k times

$\Rightarrow k^2 <= n$

$\rightarrow k = \sqrt{n}$

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

9. void function (int n) {

```
for ( i = 1 to n) {
        for (j=1 ; j<=n ; j=j+i)
            print ("*")
    }
}
```

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

Time complexity $= O(\log n)$

for outer loop,

Time complexity $= O(n)$

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

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

$$\boxed{n^k = O(c^n)}$$

i.e. $n^k \leq c_1 \cdot (c^n)$

$\Rightarrow n^k = c_1 \cdot c^n$

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

$(2)^2 = c_1 \cdot (2)^2$

$4 = c_1 \cdot 4$

$\boxed{c_1 = 1}$

$\therefore$ for $c_1 = 1$, the relation holds