Tutorial - 1

Ques.1. → **Asymptotic Notations** - It give us an idea about how good a given algorithm is, as compared to some other algorithm.

There are 3 types of widely used asymptotic notation -

i) Big O (O)
ii) Big omega (Ω)
iii) Big Theta (Θ)

i) **Big O Notation** → This notation defines an upper bound of an algorithm, it bounds a func^n only from above.

ii) **Omega Notation** → Just as Big O notation provides an asymptotic upper bound on a func^n, Ω notation provides an asymptotic lower bound.

iii) **Theta Notation** → This notation bounds a func^n from above & below, so it defines exact asymptotic behaviour.

eg. - $f(n) = \sum_{i=1}^{n} i \times 2^i$ → $T(n) = \Omega(2^n)$
→ $T(n) = O(n2^n)$
→ $T(n) = \Theta(n2^n)$

Ques.2. → Time complexity of - for (i=1 to n){
i = i*2; }

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

$t_k = ar^{k-1}$

$n = 2^{k-1}$

$\log_2 n = k-1$

$k = \log_2 n + 1$

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

$$\boxed{T(n) = O(\log_2 n)}$$

Ques3. $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$

$T(0) = 1$

$T(n) = 3T(n-1)$ —①

put $n = n-1$ in eq ①,

$T(n-1) = 3T(n-2)$ —②

put in eq ①,

$T(n-1) = 3^2 T(n-2)$ —③

put $n = n-2$ in eq ①,

$T(n-2) = 3T(n-3)$ —④

put in eq ③,

$T(n) = 3^3 T(n-3)$ —⑤

for some constant k,

$T(n) = 3^k T(n-k)$ —⑥

put $n-k=0$, $\Rightarrow k=n$

$T(n) = 3^n . T(0)$

$\rightarrow \boxed{T(n) = O(3^n)}$

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

$T(n) = 2T(n-1)-1$ —①

put $n = n-1$,

$T(n-1) = 2T(n-2)-1$ —②

put in eq ①,

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

$T(n) = 4T(n-2)-2-1$ —③

put $n=n-2$ in eq ①

$T(n-2) = 2T(n-3)-1$ —④

put in eq ③,

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

$T(n) = 8T(n-3)-4-2-1$ —⑤

for some constant k, $T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - \dots - 1$ —⑥

put $n-k=0 \Rightarrow n=k$

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

$a = 2^{n-1}$, $r = +1/2$, $S = 2^n \frac{[(1/2)^n - 1]}{+\frac{1}{2} - 1} = 2^n [2^{-n} - 1]$

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

$\rightarrow \boxed{T(n) = O(2^n)}$

Ques.5.

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

$i = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$

$s = 1+3+6+10+15+ \dots + T_n$ — (1)

$s = 1+3+6+10+ \dots + T_{n-1} + T_n$ — (2)

sub. eq (2) from eq (1)

$0 = 1+2+3+4+ \dots n - T_n$

$T_k = 1+2+3+4+ \dots k$

$T_k = \frac{k(k+1)}{2}$

for k iterations,

$1+2+3+ \dots + k <= n$

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

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

$O(k^2) <= n$

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

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

Ques.6. void function (int n) {

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

$\because i^2 \leq n \Rightarrow i \leq \sqrt{n}$

$i = 1, 2, 3, 4 \dots, \sqrt{n}$

$\sum_{i=1}^{n} 1+2+3+4 \dots + \sqrt{n}$

$\Rightarrow T(n) = \frac{\sqrt{n}(\sqrt{n}+1)}{2} = \frac{n\sqrt{n}}{2}$

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

Ques. 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 $k = k^2$

$k = 1, 2, 4, 8, \dots n$

$a=1,\ k=2 \Rightarrow k = \log_2 n$

i	j	k
n/2	logn	logn * logn
n/2+1	logn	
n	logn	logn * logn

$T(n) = O(\frac{n}{2} * \log n * \log n)$

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

→ Ques. 8 → function (int n){
if (n==1) return; // 0
for (i=1 to n){ // n
for (i=1 to n){ // n
printf("*");
}
}
function (n-3); } // T(n/3)

→ $T(n) = T(n/3) + n^2$

using master's Method,

$a=1,\ b=3,\ f(n) = n^2$

$c = \log_3 1 = 0$

$n^c = 1 > n^2$

→ $\boxed{T(n) = \theta(n^2)}$

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

for $i=1$, $j \Rightarrow 1, 2, 3, 4, \ldots, n = n$

for $i=2$, $j = 1, 3, 3, \ldots, n = n/2$

for $i=3$, $j = 1, 4, 7, \ldots n = n/3$

⋮

for $i=n$, $j=1 \qquad = 1$

$$\Rightarrow \sum_{j=n}^{1} n + \frac{n}{2} + \frac{n}{3} + \ldots + 1$$

$$= \sum_{j=n}^{1} n\left[1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right]$$

$$= \sum_{j=n}^{1} n(\log n)$$

$$\rightarrow \boxed{T(n) = O(n\log n)}$$

Ques. 10. for func. n^k and c^n, what is asymptotic relationship b/w these func.? Assume that $k>=1$ and $c>1$ are constants. Find the value of c and n_0 for which relation holds.

Relation b/w n^k and c^n is $\underline{n^k = O(c^n)}$

as $n^k \leq a c^n$

$\forall n \geq n_0$ and some constant $a > 0$

for $n_0 = 1$

$c = 2$

$\Rightarrow 1^k \leq a 2^1$

$$\boxed{n_0 = 1 \text{ and } c = 2}$$