Assignment - 1

Date / /

1) Asymptotic Notations: Asymptotic notations are used to write fastest and slowest possible running time for an algorithm. These are also referred to as 'best case' and 'worst case' scenarios respectively.

"In asymptotic notations, we derive the complexity concerning the size of the input. (Example in terms of n)".

"These notations are important because without expanding the cost of running the algorithm, we can't estimate the complexity of the algorithm.

Different notations:

1) Big - oh notation: Big-oh is the formal method of expressing the upper bound of an algorithms running time. It is the measure of the longest amount of time. The function $f(n) = O(g(n))$.

Eg: $3n + 2 = O(n)$ for all $n \geq 1$.

Date / /

2) Omega ( ) Notation: The function $f(n) = \Omega(g(n))$ if and only if there exists positive constant c and n_0 such that:

$f(n) \geq k * g(n)$ for all n, $n \geq n_0$.

3) Theta (θ): The function $f(n) = \theta(g(n))$ if and only if there exists positive constant k_1, k_2 and k_0 such that

$k_1 * g(n) \leq f(n) \leq k_2 g(n)$ for all n, $n \geq n_0$.

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

No. of Loops:	i:
1	2
2	2^2
3	2^3
:	:
:	:
:	:
K	2^K

So, $2^K = n$

$K = \log_2 n$

Hence, Time complexity: $O(\log_2(n))$ Ans

Date / /

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

$T(n) = 3T(n-1)$ — ①

Put $n = n-1$ in ①

$T(n-1) = 3T(n-2)$ — ②

Substitute ② in ①:

$T(n) = 3\{3T(n-2)\}$ — ③

Put $n = n-2$ in ①

$T(n-2) = 3T(n-3)$ — ④

Substitute ④ in ③:

$T(n) = 3^2\{3T(n-3)\}$ — ⑤

$T(n) = 3^3\{T(n-3)\}$ — ⑥

$T(n) = 3^K\{T(n-K)\}$ — ⑦

When $K = n$:

$n-K = 0$

$n = K$

$T(n) = 3^n\{T(0)\}$

So,

$T(n) = 3^n$

$T(n) = O(3^n)$ Ans

Date / /

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

$T(n) = 2T(n-1) - 1$ — ①

Put $n = n-1$ in ①:

$T(n-1) = 2T(n-2) - 1$ — ②

Sub. ② in ①:

$T(n) = 2\{2T(n-2) - 1\} - 1$ — ③

Put $n = n-2$ in ①:

$T(n-2) = 2T(n-3) - 1$ — ④

Sub. ④ in ③:

$T(n) = 2\{4T(n-3) - 2\} - 1$ — ⑤

$T(n) = 8T(n-3) - 4 - 1$ — ⑥

$T(n) = 4T(n-2) - 2 - 1$ — ③

Put $n = n-2$ in ①:

$T(n-2) = 2T(n-3) - 1$ — ④

Sub ④ in ③:

$T(n) = 4\{2T(n-3) - 1\} - 2 - 1$ — ⑤

$T(n) = 8T(n-3) - 4 - 2 - 1$ — ⑥

$T(n) = 2^3 T(n-3) - 2^2 - 2^1 - 2^0$ — ⑦

$T(n) = 2^K (T(n-K)) - 2^{K-1} - 2^{K-2} - 2^{K-3}$

$n - K = 0$

$n = K$

Date / /

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

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

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

$T(n) = 1$

So,

$T(n) = O(1)$ Ans

5)

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

6)

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

$\sum_{i=1}^{n} O(1)$ $\quad i*i<=n$

$i^2 <= n$

$i <= \sqrt{n}$

$O(\sqrt{n})$ Ans

Date / /

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++;

1st Loop: 1 : $i = \frac{n}{2} + 0$

2 : $i = \frac{n}{2} + 1$

3 : $i = \frac{n}{2} + 2$

k : $i = \frac{n}{2} + (k-1)$

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

$k - 1 = n - \frac{n}{2}$

$k - 1 = \frac{n}{2}$

$k = \frac{n}{2} - 1$

$k = n$

1st Loop : $O(n)$

Date / /

2nd Loop: 1: $i = 1 = 2^0$
2: $i = 2 = 2^1$
3: $i = 4 = 2^2$
4: $i = 8 = 2^3$
K: $i = 2^{K-1}$

$2^{K-1} = n$
$K-1 = \log n$
$K = \log_2 n - 1$
$K = O(\log n)$
2nd Loop: $O(\log n)$

3rd Loop: $O(\log n)$

Time complexity: $n * \log n * \log n$
$\Rightarrow O(n(\log n)^2)$ Ans

8)

9)

```
Void function (int n){
    for(i=1 to n){
        for(j=1; j<=n; j=j+i)
            Printf("*");
    }
}
```

1st Loop: i: 1 to n
$O(n)$

Date / /

2nd Loop:
when $i=1$ (n times)
when $i=2$ ($\frac{n}{2}$ times)
when $i=3$ ($\frac{n}{3}$ times)
when $i=n$ ($\frac{n}{n}$ times)

So, $n + \frac{n}{2} + \frac{n}{3} + - - - - 1$

$n\left(1 + \frac{1}{2} + \frac{1}{3} + - - - - + \frac{1}{n}\right)$

$n \log n$

2nd: $O(n \log n)$

Time complexity: $n^2 \log n$

$\Rightarrow O(n^2)$ Ans

5)

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

i	s
1	2
2	2+2
3	2+2+3
4	2+2+3+4
n	2+2+3+4- - - n

Date / /

$T(n) = 1 + 2 + 3 + 4 + \dots + n$

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

$T(n) = n(n+1)$

$T(n) = O(n^2)$ Ans

8)

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

T.C : $\frac{n}{3} * n * n = \frac{n^3}{3}$

So, T.C : $O(n^3)$ Ans