Assignment - 1:

(1) What do you understand by Asymptotic notations. Define different Asymptotic notation with examples.

→ Asymptotic notations is used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n. These notations are mathematically tools to represent the complexities.

There are three notations commonly used

↳ Big Oh Notations

↳ gives an upper bound for a function f(n) to within a constant factor.

↳ Big omega Notation

↳ (Ω) Notation gives a lower bound for a function f(n) to within a constant factor.

↳ Big Theta Notation

↳ (Θ) Notation gives bound for a function f(n) to within a constant factor.

(2) What should be time complexity of

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

↳ for(i=1; i<=n; i=i*2)

Iter 1 $i = 1 = 2^0$

" 2 $i = 2 = 2^1$

" 3 $i = 4 = 2^2$

" 4 $i = 8 = 2^3$

:

:

Iter p $i = 2^{p-1} = n$

$2^{p-1} = n$

$p-1 = \log_2 n$

$p = \log_2 n + 1$ ⇒ $O(\log n)$ Ans

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

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

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

$T(1) = 3T(0)$

$T(1) = 3\times 1$

$T(1) = 3$

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

$= 3T(2)$

$= 3.9$

$= 27$

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

$= 3T(1)$

$= 3.3$

$= 9$

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

$= 3T(3)$

$= 3.27$

$= 81$

$T(n) = 3+9+27+81+\dots+n$

$T(n) = 3^n$

Hence, Time Complexity $= O(n)$ Ans

④ $T(n) = 2T(n-1)-1$ if $n>0$ otherwise 1

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

$T(1) = 2.T(0)-1$

$= 2.1-1$

$= 1$

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

$= 2T(1)-1$

$= 2\times 1-1$

$T(2) = 1$

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

$= 2T(2)-1$

$= 2.1-1$

$\vdots$

$T(n) = 1$

$O(1)$ Ans

⑤ Time complexity

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

iter 1 $s = 1+2$

" 2 $s = 1+2+3$

" 3 $s = 1+2+3+4$

⋮

iter k $\sum_{i=1}^{k} = n$

$$\sum_{i=1}^{k} = \frac{k(k+1)}{2} = \Theta(k^2)$$

$$\Theta(k^2) = n$$

$$K = \Theta(\sqrt{2})$$

⑥ Time complexity

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

$i*i<=n$

$i<=\frac{n}{i}$

iter $1 : i = 1$

" $2 : i = 2$

⋮

iter $K : K = \frac{n}{i}$

$$K = \frac{n}{i} = K = \frac{n}{K}$$

$$K^2 = n$$

$$K = \sqrt{n}$$

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

(7) Time complexity of

void function (int n)

{

int i, j, k, count = 0; $\longrightarrow O(1)$

for (i = n/2; i <= n; i++)

{

for (j = 1; j <= n; j = j * 2)

{

for (K = 1; k <= n; k = k * 2)

count ++;

}

}

$\hookrightarrow$ iter 1 $i = n/2 + 0$

" 2 $i = n/2 + 1$

" 3 $i = n/2 + 2$

⋮

iter k $i = n/2 + k - 1 = n$

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

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

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

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

$\hookrightarrow O(n)$

iter 1 $j = 1 = 2^0$

iter 2 $j = 2 = 2^1$

iter 3 $j = 4 = 2^2$

iter 4 $j = 8 = 2^3$

⋮

iter k $j = 2^{k-1} = n$

$k - 1 = \log n$

$k = \log n + 1$ $\hookrightarrow O(\log n)$

iter 1 $k=1$
iter 2 $k=2$
iter 3 $k=4$
iter 4 $k=8$

$1 \times n \times \log n \times \log n$
$\hookrightarrow O(n(\log n)^2)$ w

⋮

iter $k = 2^{P-1} = n$

$2^{P-1} = n$

$P-1 = \log n$

$P = \log n + 1$

$\hookrightarrow O(\log n)$

(8) Time Complexity of

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

$\hookrightarrow$ can't be found because to find the time complexity it must be algorithm as there is no terminating point so it is not a algorithm.

(9) Time complexity of

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

}
```

↳ iter 1 i=1 → O(1)

" 1 j=1

" 2 j=2 //

same for the previous loop as well