ASSIGNMENT - 1

1) What do you understand by Asymptotic notations. Define different ~~typ~~ Asymptotic Notation with examples.

Asymptotic Notation :-

Whenever we want to perform analysis of an algorithm, we need to calculate the complexity of that algorithm. But when we calculate the complexity of an algorithm it does not provide the exact amount of resource required. So instead of taking the exact amount of resource, we represent that complexity in a general form which produces the basic nature of that algorithm. We use that general form for analysis process

- Asymptotic notation ~~is an~~ of an algorithm is a mathematical representation of its complexity.

(i) Big oh (O):-

Consider function f(n) as time complexity of an algorithm and g(n) is the most significant term.

If $f(n) <= c\,g(n)$ for all $n >= n_0$

$c > 0$ and $n_0 >= 1$

Then we can represent f(n) as O(g(n))

$$f(n) = O(g(n))$$

Eg: $f(n) = 3n+2$ $g(n) = n$

If we want to represent f(n) as O(g(n)) then it must satisfy $f(n) <= c\,g(n)$ for all values of $c > 0$ and $n_0 >= 1$

$f(n) <= c\, g(n)$

$3n + 2 <= c\, n$

here $c = 2$ and $n >= 2$

hence ~~it~~ the expression is TRUE

So $3n + 2 = O(n)$

(ii) Big Omega (Ω):-

If $f(n) >= c\, g(n) \ \forall \ n >= n_0$

$c > 0$ and $n_0 >= 1$

Then $f(n) = \Omega\, g(n)$

Eg:

$f(n) = 3n + 2$

$g(n) = n$

$f(n) >= c\, g(n)$

$3n + 2 >= c\, n$

Here, $c = 1$ and $n >= 1$

This expression is true.

So $3n + 2 = \Omega(n)$

(iii) Big Theta (θ):-

If $c_1 g(n) <= f(n) <= c_2 g(n) \ \forall \ n > n_0$

$c_1 > 0$, $c_2 > 0$ and $n_0 >= 1$

Then $f(n) = \theta(g(n))$

Eg: $f(n) = 3n + 2$

$g(n) = n$

$c_1 g(n) <= f(n) <= c_2 g(n)$

$c_1 n <= 3n + 2 <= c_2 n$

this condition is true for all values of ~~$c_1 > 0$, $c_2 > 0$~~

~~and~~ $c_1 = 1$, $c_2 = 4$ $n >= 2$

So $3n + 2 = \theta(n)$

2) What should be time complexity of:

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

$i = 1, 2, 4, 8, \ldots\ldots, n$ - k terms

Here, it is a GP

$a_n = a r^{n-1}$

$n = 1 \cdot 2^{k-1}$

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

$2n = 2^k$

Taking log both side:

$\cancel{2}\log(2n) = \cancel{\log}(2^k)$

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

$k = \log(2n)$

$k = \log 2 + \log n$

$k = 1 + \log n$

$O(\log n)$

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

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

Put $n = n-1$

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

Put (2) in (1)

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

Let $n = n-2$

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

Put (4) in (3)

$T(n) = 9(3T(n-3))$

$T(n) = 27T(n-3)$ - ⑤

Let $n = n-3$

$T(n-3) = \cancel{27}\{3T(n-4)\}$

Put in ⑤

$T(n) = 27(3T(n-4))$

<u>Generalised form:</u>

$T(n) = 3^k[kT(n-(k+1))]$

$T(0) = 1$

$n-k-1 = 0$

$\boxed{k = n-1}$

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

$3^{n-1}(n-1)T(\cancel{0})]$

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

$= \frac{n3^n}{\cancel{3}} - \frac{3^n}{3}$

<u>Ans</u> $\boxed{O(3^n)}$

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

$T(n) = 2T(n-1)-1$ - ① $T(0) = 1$

Let $n = n-1$

$T(n-1) = 2T(n-2)-1$ - ②

Put in eq ①

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

$T(n) = 4T(n-2)-2-1$ - ③

Let $n = n-2$

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

Put in eq ③

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

$= 8T(n-3)-4-2-1$

$T(n) = 8T(n-3) - 4 - 2 - 1$

~~Generalized form~~

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

~~$n - k = 0$~~

~~$k = n$~~

$T(n) = 8T(n-3) - 7$

Generalized form

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

$n-(k+1) = 0$

$k = n-1$

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

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

$= 1$

Ans = $\boxed{O(1)}$

5) What should be time complexity of

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

i	s
2	3
3	6
4	10
5	15
⋮	⋮

$s = \frac{i(i+1)}{2}$

$= \frac{i^2 + i}{2} >= n$

$i^2 + i = 2n$

$i = \sqrt{2n-1}$

$O(\sqrt{n})$

6) Time complexity of -

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

Let assume input size is n when

i	after loop i become
1	1
2	4
3	9
4	16
5	25

$i^2 <= n$

$i^2 = n$

$i = \sqrt{n}$

$O(\sqrt{n})$

7) Time complexity of

```
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 ++;
            }
        }
    }
}
```

i j k

$\frac{n}{2}$ logn logn

in loop $i \rightarrow$ $\frac{n}{2}$

in loop $h \rightarrow$ 1 2 4 8 ...

$k = \log n$

in loop ~~f~~ $j \rightarrow$ 1 2 4 8 ....

$\frac{a_2}{a_1} = \frac{2}{1} = 2$

$n = 2^{k-1}$

$k = \log n$

As it ~~complexity~~ $O(n \log^2 n)$

8) Time complexity of -

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