Tutorial Sheet - 1

Sol 1:- Asymptotic Notation :- These notations are used to tell the complexity of an algorithm when the i/p is very large.

→ It describes the algorithm efficiency and performance in a meaningful way. It describes the behaviour of time or space complexity for large instance characteristics.

- The asymptotic notion of an algorithm is classified into 5 types:-

(i) Big ~~O~~ notation (O) :- Asymptotic

Sol 2: for (i=1 to n)'

```
{
    i *= 2;
}
```

Time complexity for a loop means no. of times the loop has to run.

→ for loop above the loop will run for following values of i:

i	1	2	4	8	16 - - -	2^k
value	2^0	2^1	2^2	2^3	2^4 . .	2^n

$i = 1, 2, 4, 8, 16, \ldots 2^k$ this means k times

i.e $2^k = n$ [Taking log both sides]

$k \log_2 2 = \log_2 n$

$k = \log_2 n \quad [\because \log_a a = 1]$

$\therefore \boxed{TC = O(\log n)}$

Sol 3:- $T(n) = \left\{ \begin{array}{ll} 3T(n-1) & , n > 0 \\ 1 & \end{array} \right\}$

By forward substitution

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

$T(0) = 1$

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

$T(2) = 3T(2-1)$
$= 3T(1)$
$= 3 \times 3 = 3^2$

$T(3) = 3T(3-1)$
$= 3T(2)$
$= 3 * 3^2 = 3^3 \ldots\ldots T(n) = 3^n$

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

<u>Sol 4:</u> $T(n) = \begin{cases} 2T(n-1) - 1, & n > 0 \\ 1 \end{cases}$

By forward substitution

$T(0) = 1$

$T(1) = 2T(n-1) - 1$
$= 2T(1-1) - 1 = 2T(0) - 1$
$= 2 \times 1 - 1 = (2-1)$

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

$T(3) = 2T(3-1) - 1$
$= 2T(2) - 1 = 2 \times (2^2 - 2^1 - 1) - 1$
$= 2^3 - 2^2 - 2^1 - 1$
$\vdots$

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

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

Sol5: int i=1, s=1

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

$S_i = S_{i-1} + i$

The value of 'i' increases by one for each Value contained in 'S' at the i^{th} iteration is the sum of the first 'i' +ve integers. If K is the total no. of iterations taken by any program then while loop terminates if $1+2+3+\cdots+K$

$$= [K(K+1)/2] > n$$

So' $K = O(\sqrt{n})$

$\therefore \boxed{T.C = O(\sqrt{n})}$

Sol6:

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

$] \; O(n)$

$\boxed{T.C = O(n)}$

Sol:7

```
void func (int n)
{
  int i, j, k, count =0;
  for(i=n/2; i<=n; ++i)          ——> O(n)
    for(j=1; j<=n; j=j*2)        ——> O(log n)
      for(k=1; k<=n; k=k*2)      ——> O(log n)
         count ++;
}
```

$TC = n * \log n * \log n = O(n \log^2 n)$

Sol8: func (int n)

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

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

Sol9: Void function (int n)

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

$T.C. = O(n) * O(n) = O(n^2)$

Sol10 n^k is $O(c^n)$

$n^k = O(c^n)$.

$=$