PAGE No.
DATE

Tutorial 1

A.1) Asymptotic Notations :- It is used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n.
There are 3 different notations big O, big theta (Θ) & big Omega (Ω)

i) Big O :- It defines an upper bound of an algorithm. It counts how many iterations an algorithm will take in the worst-case scenario with an input n.

ii) Big Ω - Notation :- Big (Ω Omega) notation describes best running time of algorithm.
for eg :- Bubble sort algorithm.

iii) Θ Notation :- Theta notation encloses f() from above and below. It represents upper & lower bound of algorithm. Used for calculating average-time complexity.

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

loop will run upto n times
i.e 1 ..... n.
Inside loop: i = i * 2 i.e i = 1 * 2 (for first value)
i = 2
i = 4 (2nd value)
i = 8 (3rd val).

$\Rightarrow i = 1, 2, 4, 8, 16 \ldots n$

$= 1 \cdot 2^1, 2^2, 2^3, 2^4 \ldots 2^n$

$\Rightarrow 2^k = n \Rightarrow k = \log n$

$\therefore T(n) = O(n)$ i.e $O(\log n)$

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

$n = n-1$

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

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

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

$T(n) = 9T(n-2)$

for $n = n-1$

$\Rightarrow T(n) = 9T(n-1-2)$

$T(n-1) = 9T(n-3)$

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

$= 27T(n-3)$

$\Rightarrow 3T(n-1), 9T(n-2), 27T(n-3) \ldots$

$\ldots 3^n T(n-n)$

$= 3^n T(0)$

$= 3^n$

$\Rightarrow T(n) = O(3^n) = O(3^n)$

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

$n = n-1$

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

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

$\Rightarrow T(n) = 2(2T(n-2)+1) - 1$

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

for $n = n-1$

$\Rightarrow T(n-1) = 4T(n-1-2) - 1$

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

$\Rightarrow T(n) = 2^2(2T(n-3)+1) - 2 - 1$

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

$2T(n-1) - 1,\ 4T(n-2) - 2 - 1,\ 8T(n-3) - 4 - 2 - 1$

$\ldots\ 2^n T(n-n) - 1$

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

$T(n) = 1.$

$\Rightarrow T(1) = O(1)$

Q.5.)

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

$1+2+3\ldots+k = \frac{k(k+1)}{2} > n$

$\Rightarrow k^2 = n$

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

A.6)

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

$i=1 \Rightarrow 1*1 <= n \Rightarrow 1 <= n$, i++

for $i=2$ $1*2 \leq n \Rightarrow 2 <= n$.

$1*n \leq n \Rightarrow n \leq n$

$1, 2, 3, \ldots\ldots, n \rightarrow$ A.P.

$\text{Sum} \rightarrow \frac{n(n+1)}{2} = O(n^2)$

A.7)

```
void function (int n)
{
int i, j, k, count=0;
for(i=n/2 ; i<=n; i++)        → n/2
 for(j=1; j<=n; j=j*2)        → log2n
 for(k=1; k<=n; k=k*2)        → log2n
   count++;
}
```

$\Rightarrow T(n) = n \log_2^2 n$

A.8)

```
function(int n)
{
  if(n==1)
    return;
```

```
for(i=1 to n)          → O(n)
{
  for(j=1 to n)        → O(n)
  {
    printf("*");
  }
}
function(n-3);
}
```

$\Rightarrow T() = O(n^2)$.

A.9)
```
Void function(int n)
{
  for(i=1 to n)
  {
    for(j=1; j<=n; j=j+i)
      printf("*");
  }
}
```

i	1	2	3	.	.	.	n
1	1	2	3	.	.	.	(n-1)
2	2	3	4	.	.	.	n

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

A.10) function:- n^k, c^n.

$\Rightarrow$ Relation b/w the two is:- $n^k = O(c^n)$, Ans