Date.
Page No.

DAA ASSIGNMENT

Ans 1- Asymptotic notation describes the growth rate of algorithms as their input size approaches infinity.
Commonly used notations are.

(i) Big O (O) :- Represents the upper bound on the growth rate.
Ex -> If an algorithm has a time complexity of O(n), it means the worst case running time grows linearly with the input size (n).

(ii) Omega (Ω) :- represents a lower bounds on the growth rate. if an algorithm has a time complexity of $\Omega(n^2)$, it means the worst case time running grows at least quadrically with the input size n.

(iii) Theta (Θ) :- represents both upper and lower bounds. indicating a tight bound on the growth rate. If an algorithm has a time complexity of $\Theta(n)$, it means the worst case running time grows linearly, and there is well defined constant factor.

Ans 2 -> for (int i = 0; i < n; i = i*2)
{
=
}

Taking log both sides
$\log_2 n = (K-1)\log_2 2 \Rightarrow \log_2 n = K-1$
$K = \log_2 n + 1$
$TC \Rightarrow O(\log_2 n)$

Date. ________
Page No. ________

Ans 3-> $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$
$T(n) = 3T(n-1)$ put $n = n-1$
$T(n-1) = 3T(n-2)$ $n = n-2$
$T(n-2) = 3T(n-3)$

And so on
$T(K) = 3^K T(n-K)$
put $n-K = 0$, $\Rightarrow n = K$
$T(n) = 3^n T(0)$
$T(n) = 3^n$

Ans 4-> $T(n) = \{2T(n-1) - 1$ if $n>0$, otherwise $1\}$
$T(n) = 2T(n-1) - 1$ — (i)
put $n = n-1$ in (i) $T(n-1) = 2T(n-2) - 1$
put $n = n-2$ in (i) $T(n-2) = 2T(n-3) - 1$

If we expand this, we get
$T(n) = 2^K T(n-K) - K$

We will continue this until $n-K = 0$, $K = n$
$T(n) = 2^n T(0) - n$
$T(n) = 2^n - n$

Time complexity $\Rightarrow 2^n$

(5)- int i = 0, S = 1;
while (S <= n)
{
i++;
S = S + i;
printf("#");
}

i =	1	2	3	4	5
S =	1	3	6	10	15

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

Date.
Page No.

(6) void function (int n)

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

i =	1	2	3	4
i^2 =	1	4	9	16

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

(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++
}
```

$\frac{n}{2} \times \log_2(n) \times \log_2(n)$

complexity = $O(n \log^2(n))$

(8) function (int n)

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

$T(n) = O(n^2) + T(n-3)$

$TC = O(n^3)$

Date.
Page No.

09- Void function (int n)

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

i = 1, 2, 3, 4 - - -

j = 1, 3, 6, 10 - - -

i = 1 , n times

i = 2 , n/2 times

i = 3 , n/3 times

$1 + n/2 + n/3 + \dots + 1$

complexity = $O(n \log n)$

10- n^k $(k \geq 1)$ c^n $(c > 1)$

c^n grows faster than n^k.