DAA

Tutorial 1

Ans1:- Asymptotic notations:-

→ Asymptotic notations are used to tell the complexity of an algorithm when the I/P is very large.

→ Asymptotic Notations are languages to express the required time & space by an algorithm to solve a given problem. or It is a function to describe the performance of an algorithm.

Eg

Time complexity	Space complexity
(n^2)	(n) extra space that an
approx. no. of instructions	algorithm takes except input.

Ans2:-

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

1, 2, 4, ----- n

$a=1,\ r=2$

$t_K = a r^{K-1} = 1 \cdot 2^{k-1} \longrightarrow n = \frac{2^K}{2}$

$2^K = 2n \longrightarrow K = \log_2(2n)$

$K = \log_2(n) + 1$

$T.C = \log_2(n)$

Ans3:- $T(n) = \{ 3T(n-1) \text{ if } n>0, \text{ otherwise } 1 \}$

using substitution method :-

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

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

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

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

...

...

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

$= 3^n T(0)$

$= 3^n$

$T.C = O(3^n)$

Ans4:- $T(n) = \{ 2T(n-1) - 1 \text{ if } n>0, \text{ otherwise } 1 \}$

Using substitution method:-

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

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

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

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

$= 2^3 T(n-3) - 2^2 - 2^1 - 2^0$

...

...

$= 2^n T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots 2^0$

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

$= 1$

$T.C = O(1)$

Ans5:-

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

We can define the term 's' a/c to relation $S_i = S_{i-1} + 1$. The value of 'i' increases by one for each iteration. The value contained in 's' at the i^{th} position iteration is the sum of the first 'i' the integers.

If K is total no. of iterations taken by the program, then while loop terminates if:-

$$1+2+3+\cdots K = \frac{K(K+1)}{2} > n$$

$$\Rightarrow K = O(\sqrt{n})$$

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

Ans6:-

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

loop ends if $i^2 > n$

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

Ans7:-

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

```
for (k=1; k<=n; k=k*2)  ] execute log n times
    count++;
```

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

Ans 8:-

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

Recurrence rel^n : $T(n) = T(n-3) + cn^2$

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

Ans 9:-

```
void function (int n) {
  for (i=1 to n) {          --> This loop execute n times
    for (j=1 ; j<=n; j=j+i) --> This executes j times with
      printf ("*");             j increase by the rate
    }                           of i.
}
```

$\Rightarrow$ Inner loop executes n/i times for each value of i.
Its running times is $n \times \left(\sum_{i=1}^{n} n/i \right)$

$= O(n \log n)$

Ans 10:- The asymptotic relationship b/w the functions n^k and a^n is

$$n^k = O(a^n) \qquad k >= 1,\ a > 1$$

$$n^k \leq c \cdot a^n \quad \forall \quad n \geq n_0$$

$$\rightarrow \frac{n^k}{a^n} \leq c$$