1. Asymptotic Notation :- It defines the time taken by an algorithm to run for a given input

i) Big O notation (O) → It represent the upper bound or the maximum time that an algorithm can take to execute.

Eg → $O(n)$ → Accessing elements of 1-D array

$O(n^2)$ → Bubble sort

$O(n\log n)$ → quick sort

ii) Omega Notation (Ω) → It represent the lower bound or the minimum time that an algorithm can take to execute.

eg → $\Omega(1)$ → Searching an element in 1-D array (If the element is at 1st position)

$\Omega(n)$ → Bubble sort (the case given sorted arrays)

iii) Theta Notation (Θ) → It represent the lower as well as upper bound or the average time that an algorithm can take to execute.

eg → $\Theta(n)$ → Searching element

$\Theta(n^2)$ → Bubble Sort

2. Time complexity of

for(i=1 to n) $\rightarrow n$

{

i=i*2;

}

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

$n = ar^{k-1}$

$n = 1.2^{k-1}$

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

$2n = 2^k$

$\log_2 2n = \log_2 2^k$

$\log_2 2 + \log_2 n = k \log_2 2$

$k = 1 + \log_2 n$

Time complexity $= O(\log_2 n)$

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

$T(n) = 3T(n-1)$, $n>0$

$= 1$, otherwise

$T(0) = 1$

$T(n) = 3T(n-1)$ — (i)

put $n = n-1$

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

$T(n-1) = 3T(n-2)$ — (ii)

put (ii) in (i)

$T(n) = 3.3T(n-2)$ — (iii)

put $n = n-2$

$T(n-2) = 3T(n-2-1)$ —

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

put in (iii)

$T(n) = 3.3.3T(n-3)$

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

put $n-k = 1$

$k = n-1$

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

$= \frac{3^n}{3}.T(1)$

$= \frac{1}{3}.3^n$

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

4. $T(n) = \begin{cases} 2T(n-1)-1 \; ; & n>0 \\ 1 \; , & \text{otherwise} \end{cases}$

$T(0) = 1$

$T(n) = 2T(n-1) - 1 \quad -(i)$

put $n = n-1$ in (i)

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

$T(n-1) = 2T(n-2) - 1 \quad -(ii)$

put in eq $\rightarrow$ (i)

$T(n) = 2 \cdot 2T(n-2) - 1 - 1 \quad -(iii)$

put $n = n-2$

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

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

put in eq (iii)

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

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

put $n-k = 1$

$k = n-1$

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

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

$= O(2^n)$

5. int i=1, s=1;

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

$i = 2, 3, 4, 5, 6, \dots k$

$S = 1+2+3+4+ \dots n$

$T = a+(n-1)d$

$= 1+(n-1)$

$T \rightarrow O(n)$

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 ++;
            }
        }
    }
```

$k = 1, 2, 4, 8, \dots n$

$n = ar^{k-1}$

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

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

$2n = 2^k$

$\log 2n = k \log 2$

$\boxed{k = \log_2 n}$

Same for j loop

$= \log_2 n$

The first loop will execute $\frac{n}{2}$ times

$T = \frac{n}{2} \times \log_2 n \times \log_2 n$

$= \frac{n}{2} \log_2^2 n$

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

6. void function(int n){

int i, count=0

for(i=1; i*i<=n; i++)

{

count++;

}

if n=5 | if n=10 | if n=20

i=1,2 | i=1,2,3 | i=1,2,3,4

so loop runs $\sqrt{n}$ for all cases

$i = 1, 2, 3, 4, \dots, \sqrt{n}$

$O(\sqrt{n})$

8.
```
function (int n) {
      if (n==1):
          return
       for (i=1 to n)
            for (j= 1 to n)
                  printf("*");

      function(n-3);
}
```

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

i	j	
1	1, 2, 3, 4, .. n	n times
2	1, 3, 5, 7, .. n	n/2 times
3	1, 4, 7, ... n	n/3 times
⋮		
n	1, 1+n ...	n/n times
n times	$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots \frac{n}{n}$	

T. C = O(n log n)

$$= n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) = O(\log n)$$