Assignment :- 01

Q1 What do you mean by Asymptotic Notation? Define different Asymptotic Notation with example?

Ans: Asymptotic Notation is a mathematical notation used to describe the behaviour of functions as their input approaches infinity. It provides a convenient way to analyze the time & space Complexity. There are five asymptotic notations:

1) Big O notation (O):
→ It describes the tight upper bound of an algorithm's time & space complexity

2) Omega Notation (Ω):
→ It describes the tight lower bound of an algorithm's time & space complexity.

3) Theta (Θ) Notation:
→ It describes lower & upper bound

4) Little O notation (o):
→ Describes an upper bound on function's growth

5) Little Omega Notation (ω):
→ Describes a lower bound on function's growth.

Example:

```
bool linear_search(int arr[], int size, int t){
    for(int i=0; i<size; i++){
        if(arr[i] == target){
            return true;
        }
    }
    return false
}
```

Best Case : $\Omega(1)$
Worst Case : $O(n)$
Average Case : $\theta(n)$

Q2 What should be the time complexity of

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

for each iteration i multiplied by 2 until i < n
this shows a logarithm pattern
Let no of iteration = k
so $2^k \leq n$

Taking log both side

$k \log_2 2 \leq \log_2(n)$

$k \leq \log_2(n)$

So the T.C is $O(\log n)$

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

we have $T(0) = 1$,

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

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

put value of $T(n-1)$ in eq (i)

$T(n) = 3 * 3T(n-2)$ ... (i)

put $n = n-2$ in eq (i)

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

put value of $T(n-2)$ in eq (i)

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

$T(n) \Rightarrow 3^K T(n-K)$

we need to find value of k such that

$n-k = 0$ so $n = K$

Substitute $k = n$

$T(n) = 3^n T(0)$

Since $T(0)$ is 1

so

$T(n) = 3^n$

Therefore Time complexity of recurrence relation $O(3^n)$

Q4 $T(n) = \{2T(n-1) - 1$ if $n > 0$. otherwise $1\}$

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

$T(0) = 1$

$T(1) = 2T(0) - 1$

put value of $T(0)$ which is 1

$= 2 \times 1 - 1 = 1$

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

$= 1$

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

$= 2 \times 1 - 1 = 1$

for $T(n)$ its is always 1 for all $n \geq 0$

$T(n) = 1 + 1 + 1 \ldots \ldots T(n) = 1$

so the time complexity will $O(1)$

Q5 What should be time complexity of

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

for each iteration i increases by 1 & s updated to s+i.

for i = 1, s = 1;
for i = 2, s = 3
for i = 3, s = 6,
for i = 4, s = 10;
for i = 5, s = 15;

It forms an arithmetic sequence for the sum of first k term

$$\frac{k(k+1)}{2}$$

it will $\frac{k(k+1)}{2} \leq n$

$$k^2 + k \leq 2n$$

Solving this will give us $O(\sqrt{n})$

Q6 Time Complexity of :-

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

The loop will iterate till $i^2 \leq n$

for $i = 1$ $i^2 \leq n$

for $i = 2$ $i^2 \leq n$

and so on

for some value k such that $k^2 > n$, & then $(k-1)^2 \leq n$

so loop will run $k-1$

Therefore Time Complexity is $O(\sqrt{n})$

Q7 Time Complexity of

```
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++
}
```

Ans first loop will run $n/2$ times

second loop will run $(\log n)$ because variable j gets doubled after each iteration

(iii) and last loop will also run $\log(n)$

So the total iterations

$\Rightarrow O(n/2 * \log n * \log n)$

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

8 Time Complexity of -

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

for the value of n the nested loop will run TC: $O(n^2)$

for next call : $(n-3)^2$

for another next : $(n-6)^2$

Total Time Complexity = $O(n^2) + O((n-3)^2) + O((n-6)^2) + \ldots$

Therefore, the overall time complexity is approximately $O((n/3)^2)$, simplifies to $O(n^2/9)$

Thus total time Complexity $O(n^2)$

9 Time Complexity of -

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

The outer loop runs n times
The inner loop has a variable increment
$j = j + i$. The no. of iterations n/i.
Then therefore total iterations
$= 1 + n/2 + n/3 + \ldots + n/n$.

The Time Complexity will $O(n \log n)$
according to harmonic series.

Q10 for the functions, n^k, & c^n, what is asymptotic relationship b/w these functions.?
Assume that $k \geq 1$ & $c > 1$ are constants.
find out values of c & n_0 for which relation holds.

Ans When n is large enough, the exponential function c^n will eventually grow much faster than any polynomial function n^k where k is constant
c^n grows faster than n^k

Let choose some arbitary value for c & n_0
We need to find value such that c^n eventually become greater than n^k
We can adjust c & n_0 until we find the smallest value that satisfy this condition.
for any chosen value of c, there will exist some n_0 beyond which c^n exceed n^k. The specific value of c & n_0 will depend on choice of k & c.
In general as n increases, c^n will surpass n^k for a fix k, To find we need more details about analysis