Assignment

Name : Akash Rawat
Section : CE
Roll no. : 10

ans1 Asymptotic notations are methods/languages using which we can define the running time of the algorithm based on in input size.
There are mainly 3 asymptotic notations.

1 **Big-O Notation** :— It represents the upper bound of the running time of an algorithm. Thus it gives the worst-case complexity of an algorithm.

eg: O(log n) = Binary Search
O(n) = Simple search.

(2) **Omega Notation** :— It represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

(3) **Theta Notation (Θ)** : Theta notation encloses the function from above & below. Since it represents the upper and lower bound of the running time of an algorithm it is used for analyzing the average-case complexity of an algorithm.

ans2 for (i=1 to n)
{ i = i*2 }

1, 2, 4, 8 ..... n

$2^{K-1} = n$

$(K-1)\log 2 = \log n$

$K = \log_2 n + 1$

Time Complexity $= O(\log_2 n)$

ans3 $T(n) = 3T(n-1) \quad n > 0$

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

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

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

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

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

$T(n) = 27T(n-3)$

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

$n-K=0$

$n=K$

$T(n) = 3^n * 1$

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

ans 4

$$T(n) = 2T(n-1)$$
$$T(n-1) = 2T(n-2)$$
$$T(n) = 4T(n-2)$$
$$T(n-2) = 2T(n-3)$$
$$T(n) = 8T(n-3)$$
$$T(n) = 2^K T(n-K)$$
$$n-K = 0$$
$$T(n) = 2^n = O(2^n)$$

ans 5

$i = 1$

$S = 1$

The value of i increases by 1 at each interval of $n = \frac{K(K+1)}{2} \Rightarrow K = \sqrt{n}$

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

ans 6

$i^2 <= n \Rightarrow i <= \sqrt{n}$

Time Complexity $= O(\sqrt{n})$ //

ans 7

for 1st loop $\rightarrow n/2$

for 2nd loop $\rightarrow \log n$

for 3rd loop $\rightarrow \log n$

Time Complexity $= O(n(\log n)^2)$ //

ans8

n^2 = Recurrence

$T(n) = T(n-3)$
$T(n-3) = T(n-6)$
$T(n) = T(n-6)$
$T(n) = T(n-9)$
$Tn = T(n-3^k)$

$n - 3^K = 0$
$n = 3^K$
$\log n = K \log 3$

$K = \log_n 3$

$n^2 > \log n^3$

Time Complexity $= O(n^3)$

ans9

$n + n/2 + n/3 + n/4 + \dots n/n$

$n\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}\right)$

Time Complexity = $n \log n$.

ans 11 The loop variable i is incremented by 1, 2, 3, 4 until it becomes greater than or equal to n. The values of i is $n(n+1)/2$ after n iterations. So if loops runs n times, then $n(n+1)/2 < n$. Therefore

Time Complexity can be written as $\sqrt{n}$ //

ans 12 fibonacii Series.

0 1 1 2 3 5 8 13 . . . . . .

$$fib(n) = \begin{cases} 1, 0 & (n = 1 \text{ or } n = 0 \text{ resp}) \\ fib(n-2) + fib(n-1), & n > 1 \end{cases}$$

The recursive equation for fibonacii Series is:

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

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

Solving it using backword Substitution.

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

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

$$T(n) = 2[2T(n-2) + 1] + 1$$

$$= 4T(n-2) + 3$$

Next we can substitute $T(n-2) = 2T(n-3) + 1$

$$T(n) = 2[2[2T(n-3) + 1] + 1] = 8T(n-3) + 7$$

⋮

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

Now we can find K and therefore solve by substituting in $T(0) = 1$

For $T(0)$, we can see that $n - K = 0$

Rearranging we get $K = n$. Now substituting in our values for $T(0)$ & K, we get.

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

Time Complexity $= O(2^n)$

Space Complexity would be : $O(1)$, Since the program does not use extra spaces.

ans14 $T(n) = T(n/4) + T(n/2) + n^2$

Assuming $T(n/2) > T(n/4)$

$$T(n) = T(n/2) + cn^2.$$

Applying master's theorem.

$a = 1 \quad b = 2$

$f(n) = \cancel{cn^2}\ cn^2 / n^{\log_2 1}$

As, $(n^2 > n^{\log 2}$

So, $T(n) = O(f(n))$

$T(n) = n^2$

$= O(n^2)$ //

ans15 $\quad n + n/2 + n/3 + n/4 + \cdots$

$n\left(1 + \frac{1}{2} + 1/3 + 1/4 + \cdots\right)$

$n(\log n) = n \log n$ //

ans16 $\quad n = i^K$

$\log n = K \log i$

$K = \log_i n$

$O(\log n)$ //

ans18 (a) $100 < \log\log n < \log n < \log(n!) <$ root $n < n < n \log n < n^2 <$
$2^n < 2^{2n} = 4^n < n!$

(b) $1 < \log\log n < \sqrt{\log(n)} < \log(n) < \log 2n < 2\log n < n < 2n < 4n < n\log n$
$< \log(n!) < n^2 < 2(2^n) < n!$

(c) $96 < \log_8 n < \log_2 n < \log(n!) < 5n < n\log n < n\log_2 n <$
$8n^2 < 7n^3 < 8^{2n-6} < n!$

ans10 Since polynomials grow slower than exponentialy n^k has an ~~as~~ asymptotic upper bound of $O(a^n)$