Shifa
DS1
58

Assignment-1

Ques 1:- What do you understand by Asymptotic notations. Define different Asymptotic notations with examples.

Ans:- Asymptotic notations are mathematical tools used to describe the behaviour of functions as their inputs approach certain values, usually infinity.

(1) Big O notation (O): Represents the upper bound of the function's growth rate.

ex:- (f(n) = O(g(n))) means that (g(n)) is an upper bound on (f(n))

(2) Omega notation (Ω): Represents the lower bound of a function's growth rate.

ex:- (f(n) = Ω(g(n)) means that (g(n)) is a lower bound on (f(n))

(3) Theta notation (Θ): Represents both upper and lower bounds of a function's growth rate.

ex:- (f(n)) = Θ(g(n)) means that (g(n)) is both an upper and lower bound on (f(n)).

Ques 2:- What should be the time complexity of -

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

→ The time complexity of the given loop can be determined by analyzing how many iterations it performs as a function of the input size (n).

In this loop, the value of (i) is being doubled in each iteration until it exceeds (n).

Initially, (i=1) and in each iteration, (i) is multiplied by 2. So, the values of (i) in the iterations would be: 1, 2, 4, 8, 16 ... (2^k), where (2^k) is the largest power of 2 less than or equal to (n).

Hence, the time complexity of the given loop is

$O(\log(n))$

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

→ If $(n>0)$, then $(T(n) = 3T(n-1))$

if $(n=0)$, then $(T(0) = 1)$

By recurrence relation.

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

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

We continue the process until $(n-K = 0)$ i.e., $(K=n)$. At this point, $(T(0))$ is reached which is defined as 1.

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

Hence, Time complexity is $(O(3^n))$

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

⇒ if $(n>0)$, then $(T(n) = 2T(n-1)-1)$
if $(n=0)$, then $(T(0) = 1)$.

By recurrence relation

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

So,
$[T(n) = 2^kT(n-k) - (2^k-1)]$

We continue until $(n-k=0)$ i.e., $(k=n)$.
Here, it is reached till $\{T(0)\}$ i.e., 1.

So, $(T(n) = 2^nT(0) - (2^n-1) = 2^n - (2^n-1) = 1)$

Hence, time complexity is $(O(1))$

Ques 5. What should be time complexity of -

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

⇒ After 1st iteration =
$s = 1+2 = 3$
2nd iteration
$s = 3+3 = 6$.
3rd iteration
$s = 6+4 = 10$.

$1+2+3+4+5+\dots+k$.

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

s = finding value of K

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

$$\frac{K^2+K}{2} > n.$$

$$K^2+K > 2n.$$

$$K > \sqrt{n}.$$

So Time complexity is $O(K)$.

$$= \boxed{O\sqrt{n}}$$

Ques 6. TC of -

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

→ Loop have condition

$i*i \to i*i \leq n$ so,

Loop will continue until if become greater than root n.

count → take constant time.

$$\boxed{O\sqrt{n}}$$

__/__/__

Ques 7. outer loop → n/2 times.

middle j loop → $O(\log n)$ times because j double in iteration

inner loop → $O(\log_2 n)$ times because k is double in each iteration.

Total no. of iterations $= O(n/2 \cdot \log n \cdot \log_2 n)$
$= O(n \log_2^2 n)$

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

Ques 8. Base case is when n=1, then function without further recursion.

→ outer loop runs from i=1 to n.
→ inner loop runs from j = 1 to n.

~~$O(n^2)$ since, loop inside loop~~

The recursive call is made with the parameter 'n-3'. Assuming the initial value of 'n' is divisible by 3, The recursive call approximately n/3 times before reaching the base case where 'n' becomes 1

$T(n) = T(n-3) + O(n^2)$ → over all time complexity

Ques 9. In inner loop, variable 'j' is incremented by 'i' in each iteration. The no. of iterations of the inner loop depends on value of 'i'. Let's denote the no. of iterations of inner loop for specific 'i' as $\frac{n}{i}$

Calculate total no. of iterations :-

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$$

Sum of h.p $O(\log n)$.

Time complexity $\rightarrow O(n \log n)$