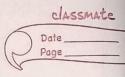
## Assignment

1. What do you understand by Agymptomatic notations? Ares It is a mathematical notation that describes the behaviour of a function as its input size approach infinity. It is used to analyze the time and space es complexity of algorithms.

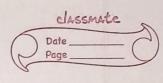
Different Types of Asymptomatic Notations are: i) Big Oh (0) - it is used to describe the apper bound it is the worst case scenario of an algorithm. f(n) = O(g(n)) Iff fln) & cg(n) of n >, no, some constant (c>0) space complexity of an algorithm. it is the best case scenario of an algorithm. f(n) = -2 (g(n)) if f(n) > c g(n)

of n? no, some constant (c>0) will Theta (0) - it is used to describe the light bound of the running time or space complexity of an algorithm. it is the average case scenario of an algorithm. f(n) = 0 (g(n)) iff C, g(n) & f(n) & c2g(n) 4 n > max(n, n2) some constant (G>066 C2>0)



200 - 100 m (v) Small Oh (o) - it is used to describe the -1-2-1strict upper bound of running time or space complexity of an algorithm. it is a more strict version of Big oh notation. f(n) = O(g(n))iff f(n) < c q(n) A n>00 , 4 c>0 v) Small Omega (w) - it is used to describe the strict lower bound of vurning time or space complexity of an algorithm. It is more strict version of Big-Omega notation. fln) = w(gln)) iff flad> c [g (a)] # n>no, # C>0 what should be time complexity of for (i=1 to n) { i=i\* 2; } i= 1, 2, 4, B, --- n an=1, M=2, n=k god (30) = 1 god (3)  $k = \log(2) + \log(n)$   $k = 1 + \log(n)$ in O ( dog (n))

3- T(n) = 2 3T(n-1) if n > 0, otherwise 13 Sol<sup>2</sup> By forward Cal 112 By forward Solution T(n) = 3T(n-1)T(0) = 1 T(1) = 3T(1-1) = 3T(0) = 3 T(2) = 3T(2-1) = 3T(1) = 3.3T(3) = 3T(3-1) = 3T(2) = 3.3.30(3n) = 0(n)4- T(n) = 2T(n-1)-1 if (n > 0), otherwise 1 Here,  $\alpha = 2$ ,  $\delta = 1$ ,  $c = \log_2(2) = 1$  $n^{c} = f(n)$ : T(n)= O(n. logn) 5- What should be time complexity of int i=1, s=1; while (s(=n) { i++; s+=i; 7 printf("#"); The input size is n Sol";>



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 ( b=1; k <= n; k= k\*2) count++; Sol?> Time complexity of inner most loop b = 1 to n; b = b\*2 => 1,2,4,8,---, kth term kth term = 2 12-1  $2n = 2^k$ log(2n) = 12 log(2) k = dog (2n) + dog (2) b = 1+ dog (20) it means for each value of jo this loop runs + logan) times Time Complexity of middle loop j=1 ton; j=j\*2 Means four each value of i, this loop runs (1+ log n) times Complexity of outer-most loop i = n/2 to n; i = i+1 = > n/2, n/2 + 1, n/2 + 2, n/2 + 3, ...,  $k^{dh}$ hth term = n+k

this loop will sun m/2 times.

Total Complexity = n + (1+ log2n) + (1+ log2n) =  $\frac{n + n \log_2 n + n (\log_2 n)^2}{2}$ 

n Jogan

 $\Rightarrow O(n(\log_2 n)^2)$ 

Time Complexity of void function (int n) { for (i=1 to n)

9-

SOUT

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

print( " \* ");

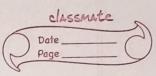
that iterates over the variable i and jo The outer loop iterates over in times and

inner loop iterates n/i times.

this is Helmonie Series

log(n) + 0,572 + 0(1/n)

in Time Complexity is O(n deg(n))



g-	Time Complexity of -
	Time Complexity of - function (int n)
	$\begin{cases} if (n==1) & return; \end{cases}$
	for (i=1 to n) s
	for (j=1 to n)
	printf(" * ");
	1
	function (n-3);
	3
Sol7>	A recurrière fination is given with two nested
	doop that iterates over i and j. The outer doop
	iterates n times and inner loop iterates n times.
	· n* is though
	at such weeks sine call value of a seriesed
	I continue the college total al m/3
	in not notiones.  At each recursive call, value of no secreased  Ly 3. function will be realled total of 19/3
-	times.  in Time Complexity = O(n2 (n/3)2)
	as Time complexity