Date. _____
Page No. _____

Assignment - 01

Q1. What do understand by Asymptotic notations. Define different Asymptotic notation with example.

Asymtotic notations are set of mathematical tools used to describe the behaviour of functions as their input size approach infinity. They are often used to analyze the time and space complexity of algorithms.

There are three main type of asymtotic notation:

① Big O notation (O): This notation provides an upper bound on the growth rate of a function. It represents the worst case running time of an algorithm, which is the maximum amount of time it could take to complete. For example we say that the algorithm has a time complexity of O(n), we mean that the algorithm's running time grows at most linearly with the size of input.

e.g. :-
```
int sum = 0;
for (int i = 1; i <= n; i++) {
sum += i;
}
```

② Omega Notation : This notation provides a lower bound on the growth rate of a function. It represents the best-case running time of an algorithm, which is the minimum amount of time it could take to complete. For example if we say that an algorithm has a time complexity of Ω(n)

Date.
Page No.

we mean that the algorithm's running time grows at least linearly with the size of its input.

(3) Theta notation (Θ): This notation provides both an upper and a lower bound on the growth rate of a function. It represent the average case running time of algorithm, which is the expected amount of time it would take to complete. For example, if we say that an algorithm has a time complexity of $\Theta(n)$ we mean that algorithm's running time grows timely with size of its input and there are no faster and slower growth rates.

Ex
```
def bubble sort (list):
    n = len (list)
    for i in range (n):
        for j in range (n-i+1):
            if list [i] > list [j+1],
                list [j], list [j+1] = list [j+1], list [j]
    return list
```

The avg. - case time complexity is $\Theta(n^2)$

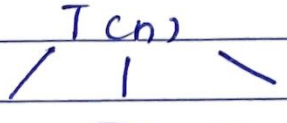
Date. _____
Page No. _____

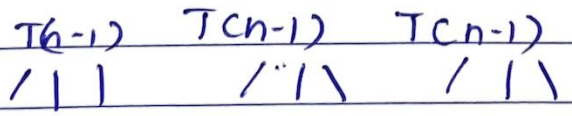
Q What should be the time complexity for
(i = 1 to n) { i = i * 2; }

Sol. The time complexity of the loop
for (i = 1 to n)
{ i = i * 2;
}
can be determined by counting the number of iteration that the loop will execute as a function of the input size 'n'
Here the value of i is being denoted in each iteration, loop terminate when i becomes > n.
$2^k = n$ ∴ $k = \log(n)$
Time complexity = $O(\log n)$

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

Ans. The time complexity of Recursive function can be determined by analyzing the number of function call it makes as a function of the input size 'n'. Each call to T(n) results 3 calls to T(n-1) until n reaches 0, at which point the function return 1. This can be represented using tree





T(n-2) T(n-2) T(n-2) T(n-2) T(n-2) T(n-2)....

The height of tree is n, at even level there are 3 nodes The total no. of calls is 3^n
Time complexity $O(3^n)$

Date. _____
Page No. _____

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

$T(0) = 1$

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

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

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

$T(4) = 2T(3) - 1 = 1$

and so on

So $T(n) = 1$ for any value of n

that is it is a constant

Time complexity $= O(1)$

Qs. What should be time complexity of

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

Sol. The time complexity of the given while is O(sqrt(n))

The loop iterates until the value of s become greater than n, At each iteration i is intremented by 1. and s is updated to s+i, the number of iterations is $i^2 + i - 2(n-s) > 0$, this can be solved by Quadratic formula.

Date. _____
Page No. _____

Q6. void function (int n) {
int i, count = 0;
for (i = 1; i * i <= n; i++)
count ++
}

Ans. The time complexity of the given function is O (sqrt (n)). Therefore loop iterates from i = 1 to i ; i * i <= n . The loop will execute for all values of i from 1 to the largest integer less than or equal to the square root of n.

Q7. 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. Time complexity is O(n² log (n)),
The function consist of 3 nested loop that iterate over variable i , j and k the first loop take n/2 iterati The second loop iterates over the variable j from 1 to n in power of 2 , which takes log 2 (n) iteration. The third loop iterates over the variable k from 1 to n in power of 2 , which also takes log 2 (n) iteration.

Date. _____
Page No. _____

Q8. Function (int n) {
if (n == 1)
return;
for (i = 1 to n)
{
for (j = 1 to n)
{ printf ("*"); }
function (n-3); }

The function is a recursive functions that is called with argument n-3, it contains two nested loop that iterate over the variable i and j. The outer loop iterate n times and inner loops also iterate n times.
∴ n * n times. At each recursive call the value of n is decreased by 3. The function will be called a total of n/3 times recursively until n=1.
Time complexity is $O(n^2 (n/3)^2)$.

Q9. void function (int n) {
for (i = 1 to n)
{ for (j = 1; j <= n; j = j+1)
printf ("*")
}}

This function consist of two nested loop that iterate over the variable i and j the counter loop iterate over n times and inner loop iterate n/i times $n + n/2 + n/3 + \ldots + 1$.
This is harmonic series
$\log(n) + 0.57 + 2 + O(1/n)$
Time complexity is $O(n \log(n))$.

Date.
Page No.

Q10. For the function, n^n and e^n. What is the asymphonic notation b/w these function
Assume that $k >= 1$ and $c > 1$ are constants. Find out the value of c and no. for which relation holds

Sol $n^k = O(c^n)$ as n approaches infinity
n^k is bounded above by c^n.