Date.
Page No.

Assignment 1

Q) What do you understand by Asymptotic notations. Define different Asymptotic notation with example.

Sol^n Asymptotic notation are a set of mathematical tools used to describe the behavior of function as their input sizes approach infinity. They are often used to analyze the time and space complexity of algorithm.

There are 3 main types of asymptotic notation:

1) Big Oh 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.

eg:
```
int add = 0;
for (int i = 1; i <= n; i++) {
    add = add + i;
}
```
The time complexity is O(n).

2) omega notation (Ω):- This notation provides a lower bound on the growth rate of a function. It represents the best-case scenario running time of an algorithm, which is the minimum amount of time it could take to complete. If we say that an algorithm has time complexity of Ω(n), we mean that the algorithm's running time grows at least linearly with the size of its input.

Date.
Page No.

2) 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 an algorithm, which is the expected amount of time it would take to complete.

```
eg:- def bubbleSort (lst):
        n = len(lst)
        for i in range(n):
            for j in range(n-i-1):
                if lst[j] > lst[j+1]:
                    lst[j], lst[j+1] =
                        lst[j+1], lst[j]
        return lst
```

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

or what should be time complexity of for(i=1; i<n; i=i*2);

Sol:- The time complexity of the loop

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

can be determine 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 $i \geq n$

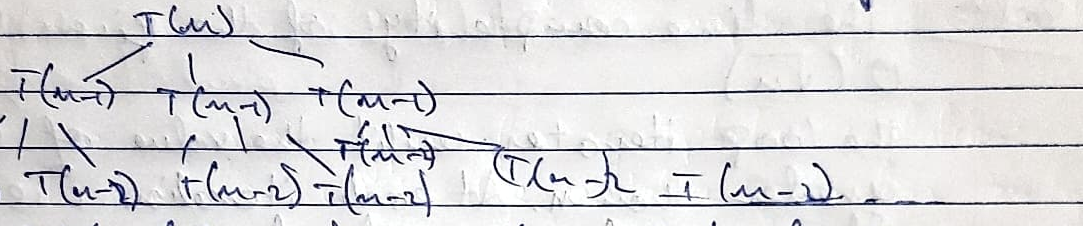
$2^k = n \Rightarrow k = \log(n)$

Time complexity = $O(\log n)$.

Date.
Page No.

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

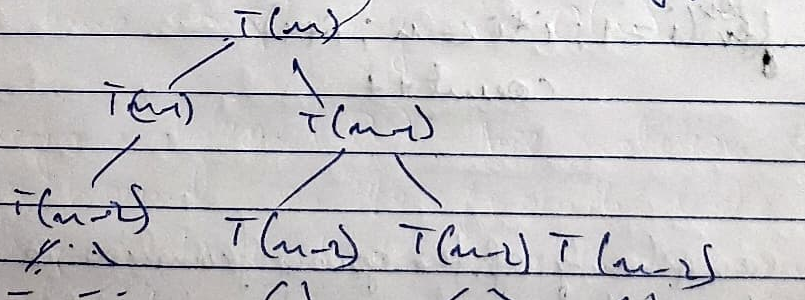
Sol: The time complexity of Recursive function can be determined by analyzing the number of function call it makes as a function of the input size 'n' each call to T(n) result in 3 calls to T(n-1) until n Reaches 0, at which point the function return 1. This can be represent using tree.



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

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

Sol: Input size n, Each T(n) result is 2 calls to T(n-1) until n reaches 0, at point of time return 1



is n at each level 2 nodes, Total function calls is 2^n, Time complexity $O(2^n) * O(1)$
$= O(2^n)$

Date.
Page No.

Q5) what should be time complexity of:

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

Sol^n: The time complexity of the given loop is $O(\sqrt{n})$

The loop iterates until the value of s becomes greater than n. At each iteration i is incremented by 1. and s is updated to s+i, the number of iterations required to reach $s > n$, $s.t\ (i+1) > n$

so, $i^2 + i - 2(n-s) > 0$. this is solve by quadratic formula.

Q6) Time complexity of:

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

Sol^n: The time complexity of the given function is $O(\sqrt{n})$, The for loop iterates from $i=1$ to $i*i<=n$, The loop will execute for all values of i from 1 to the longest integer less than or equal to the square root of n.

Date.
Page No.

Q7) void function(int n) {
int i, j, k count = 0;
for (i = n/2; i <= n; i = i + 2)
for (i = 1; i <= n; i = i * 2)
for (k = 1; k <= n; k = k * 2)
count++

Sol^n Time complexity is O(n² log(n))
The function consist of three nested loop that iterate over variable i, j and k the first loop take n/2 iteration, the second loop iterates over the variable i 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.

Q8) Time complexity of:
function(int n) {
if (n == 1) return;
for (i = 1 to n) {
for (i = 1 to n) {
Printf ("*");
}
}
function(n-3);
}

Sol^n the function is a Recursive function that is called with argument n-3, it contain two nested loop that iterate over the variable i and j. The outer loop iterate n times

Date.
Page No.

and the inner loop also iterates n times. i.e. 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 $O(n^2(n/3)^2)$.

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

Sol: The function is a Recursive function that is called with argument n-3. it contain two nested loop that iterate over the variable i and j, the outer loop iterate n times and the inner loop also iterate n times i.e. n*n times.

Sol: Time complexity is $O(n \log(n))$.
The function consist of two function nested loop that iterate over the variable i and j the outer loop iterate over n time and inner loop iterate n/i times,
So, $n + n/2 + n/3 + \dots + 1$, this is Harmonic series. $\log(n) + 0.577 + O(1/n)$.

Q(h) For the function n^k and c^n, what is the asymptotic relationship b/w these func? Assume that $k \geq 1$ and $c > 1$ are constants. Find value of c and no for which relation holds?

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