PAGE _____
DATE _____

Assignment 1

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

=> Asymptotic ~~relations~~ notations are a set of mathematical tools used to describe the behaviour of function as their input sizes approaches infinity. They are often used to analyze the time & space complexity of algorithms.

There are 3 main types of asymptotic notations.

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

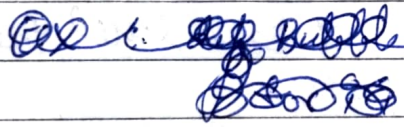
```
Ex:- int sum=0
     for(int i=1; i<=n; i++)
       { sum+ = i }
     // the TC is O(n).
```

② Omega notation (Ω): This notation provides a lower bound on the growth rate of a fn. It represents the best-case amt of time it could take to complete

PAGE _____
DATE _____

for ex. if we say that an algorithm has a TC of $\Omega(n)$ we mean that the algorithm's running time grows at least linearly with the size of the input

(3) Theta notation (Θ): This notation provides both upper & lower bound on the growth rate of a fn. It represents the avg-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 $O(n)$ we mean that the algo's running time grows linearly with the size of its inputs, & they are not faster or slower growth rates.



Q what should be the TC of:

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

The TC of the loop is can be determined by counting the no of iterations that it will execute as a fn of the input size of 'n'.

Here the value of i is being incremented in each iteration, loop terminates when i becomes > n

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

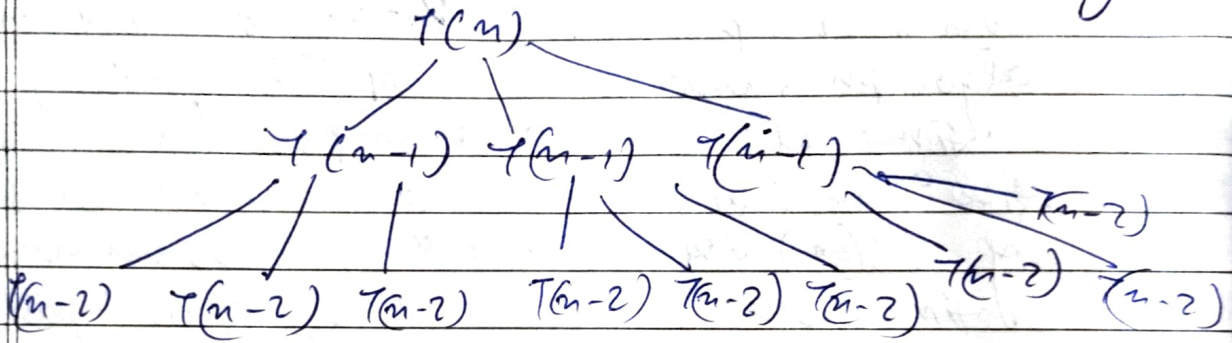
$k \log_2 2 = \log(n)$

$\therefore k = \log(n)$ $\quad \boxed{\because \log_a a = 1}$

$TC = O(\log n)$

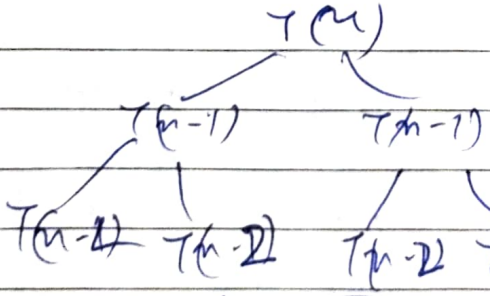
PAGE
DATE

Q $T(n) = \{ 3T(n-1)$ if $n>0$, otherwise $1\}$
The TC of Recursive fn can be determined by analyzing the no of fn calls it makes as a fn of the Input size 'n'.
Each call to $T(n)$ results in 3 calls to $T(n-1)$ until 'n' reaches 0, at which point the fn returns 1. This fn can be represented using tree



The height of the tree is n, at each level there are 3 nodes
The total no of calls is 3^n.
$\therefore T(n) = O(3^n)$.

Q $T(n) = \{ 2T(n-1)-1$ if $n>0$, otherwise $1\}$
Input size n, each $T(n)$ results in 2 calls to $T(n-1)$ until n reaches 0, at point it results 1.



at input 'n' at each level 2 nodes are created, Total fn calls is 2^n, $TC = O(2^n)$ & $O(1) = O(2^n)$.

PAGE _____
DATE _____

Q What should be the TC of

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

The TC 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 & s is updated to s+i, the no of iterations required to reach $s>n$, $s+(i+1)>n$

$i^2 + i - 2(n-s) > 0$,

this can be solved by using quadratic formula.

Q

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

The loop iterates from i=1 to $i*i<=n$. The loop will execute for all values of i from 1 to largest i s.t. $<=$ to $\sqrt{n}$.

$\therefore TC = O(\sqrt{n})$

Q

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

PAGE
DATE

TC is $O(n^2 \log(n))$

The fn consists of 3 nested loops that iterate over the variables i, j & k. The 1st loop takes multiple $n/2$ iterations, the 2nd one iterates from 1 to n in powers of 2, which takes $\log_2(n)$ iterations and same goes for the III loop. i.e. $\log_2(n)$.

$\therefore TC = O(n^2 \log(n))$

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

The fn is a recursive fn that is called with arg n-3 it contains 2 nested loops that iterates over i & j. The outer loop iterates n times & the inner one also. $\therefore n*n$ times for both loops. At each recursive call, the value of n is decremented by 3. The fn will be called a total of $n/3$ times recursively until $n=1$.

$\therefore$ TC is $O(n^2 (n/3)^2)$.

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

PAGE _____
DATE _____

The fn consists of two nested loops that iterates over i & j, the outer loop iterates over n times & inner one iterates over n/i times

∴ $n + n/2 + n/3 + \dots 1$, this is harmonic series

$\log(n) + 0.5 + O(1/n)$

$TC = O(n \log n)$.

Q for the fn n^k & c^n, what is the asymptotic relationship b/w these fn?
Assume that $k \geq 1$ & $c > 1$ are constants. Find out the value of c & n for relation holds.

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