DAA

ASSIGNMENT-1

① **Asymptotic Notation** →

Asymptotic Notation defines the time taken by an algorithm to run for a given input

(i) Big O notation (O) → It represent the upper bound or the maximum time that an algorithm can take to execute.

Eg :- $O(n)$ → Accessing elements of 1-D array

$O(n^2)$ → Bubble Sort

$O(n \log n)$ → Quick Sort

(ii) Omega Notation (Ω) → It represent the lower bound or the ~~ma~~ minimum time ~~that~~ that an algorithm can take to execute

Eg :- ~~O(1) → Se~~

$\Omega(1)$ → Searching an element in 1-D array (If the element is at 1st position)

$\Omega(n)$ → Bubble Sort (We are given sorted Array)

iii) Theta Notation (θ) → It represent the lower as well as upper bound or the average time that an algorithm can take to execute

Eg :- $\theta(n)$ → Searching element

$\theta(n^2)$ → Bubble sort

(2) Time comlexity of

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

$i = 1, 2, 4, 8, \text{----} n$

~~$n k = a r^{(k-1)}$~~ $\qquad n = a r^{k-1}$

~~$k =$~~ $m = 1 \qquad n = 1 \cdot 2^{k-1}$

$n = \frac{2^k}{2}$

$2n = 2^k$

$\log_2 2n = \log_2 2^k$

$\log_2 2 + \log_2 n = k \log_2 2$

$k = 1 + \log_2 n$

Time Complexity $= O(\log_2 n)$

(3) $T(n) = 3(T(n-1))$.

$T(n) = \begin{cases} 3T(n-1) & , n > 0 \\ 1 & , \text{otherwise} \end{cases}$

$T(0) = 1$

$T(n) = 3T(n-1) \quad — (1)$

put $n = n-1$

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

$T(n-1) = 3T(n-2) \quad — (2)$

Put (2) in (1)

$T(n) = 3 \cdot 3T(n-2) \quad - (2)$

Put $n = n-2$

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

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

Put in (2)

$T(n) = 3 \cdot 3 \cdot 3\, T(n-3) \quad - (3)$

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

Put $n-k = 1$

$\boxed{k = n-1}$

$T(n) = 3^{n-1} T(n-n-1)$

$T(n) = \frac{3^n}{3} T(+1)$

$T(n) = \frac{1}{3} \cdot 3^n$

$= O(3^n)$

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

$T(0) = 1$

$T(n) = 2T(n-1) - 1 \rightarrow (1)$

Put $n = n-1$ in (1)

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

$T(n-1) = 2T(n-2) - 1 \quad - (2)$

Put (2) in (1)

$T(n) = 2 \cdot 2T(n-2) - 1 - 1 \quad - (3)$

Put $n = n-2$

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

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

Put in (3)

$T(n) = 2 \cdot 2 \cdot 2\, T(n-3) - 1 - 1 - 1$

$T(n) = 2^k T(n-k) - k$

Put $n-k = 1$

$k = n-1$

$T(n) = 2^{n-1} T(\cancel{n} - \cancel{n} + 1) - n \cancel{+1} - 1$

$T(n) = \frac{1}{2} \cdot 2^n\, T(1) - n \cancel{+} 1$

$= O(2^n)$

(5)

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

let $n = 5$

$i = 1$

while ($s <= 5$)

{

i++; $i = 2, 3, 4,$

$s = s+i$; $s = 1 + 2$

~~s=1, 3, 6,~~

~~s=1+2+3+~~

$i = 2, 3, 4, 5, 6, \ldots\ldots k$

$s = 1 + 2 + 3 + 4 + \cdots\cdots n \Rightarrow 1, 3, 6, 10,$

$T = a + (n-1)d$

$T = 1 + (n-1) \Rightarrow O(n)$

(7) 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 ++;
}

$k = 1, 2, 4, 8, \ldots n$

$n = a r^{k-1}$

$n = 1 \cdot 2^{k-1}$

$n = \frac{2^k}{2}$

$2n = 2^k$

$\log 2n = k \log 2$

$\boxed{k = \log_2 n}$

same for j loop

$= \log_2 n$

The first loop will execute $\frac{n}{2}$ times

$T = \frac{n}{2} \times \log_2 n \times \log_2 n$

$T = \frac{n}{2} \log_2^2 n$

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

(7)

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

if $n = 5$ if $n = 10$ if $n = ~~15~~\ 20$

$i = 1, 2$ $i = 1, 2, 3$ $i = 1, 2, 3, 4$

So the loop runs for $\sqrt{n}$ times for all cases

$i = 1, 2, 3, 4, - - - \sqrt{n}$

$O(\sqrt{n})$

(8)

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

(9)

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

i	j	
1	1, 2, 3, 4, ---- n	n times
2	1, 3, 5, 7, ---- n	n/2 times
3	1, 4, 7, ---- n	n/3 times
⋮		
n	1, 1+n	n/n time
n times	$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots \frac{n}{n}$	

$$= n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$$

$\rightarrow O(\log n)$

T.C = $O(n \log n)$