Assignment 1

DAA

Date.
Page No.

Q 1.

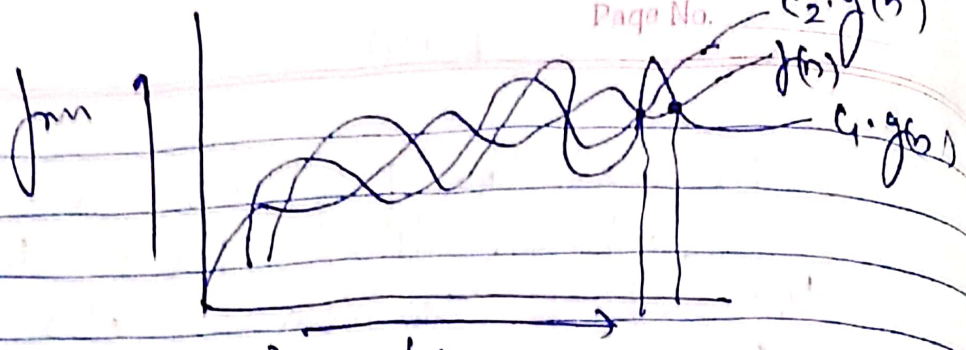
Asymptotic Notation are mathematical tools to represent the time Complexity of algorithms for asymptotic analysis

the main idea of asymptotic analysis is to have a measure of the efficiency of algorithm that don't depend on machine's specific constant and doesn't require algorithm to be implemented and time taken by the program to be compared.

* following are the asymptotic notation that are mostly used

1) Θ Notation :- the theta Notation bonds a function from above and below, so it defines exact asymptotic

behaviours



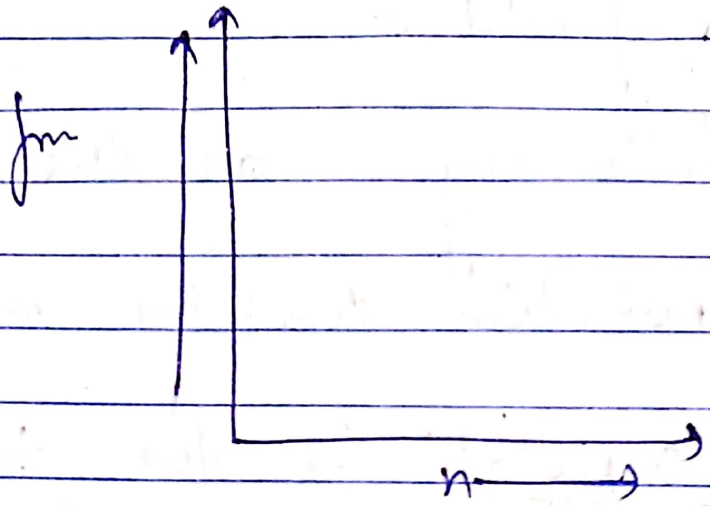
$\Theta(g(n)) = f(n)$

if $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$c_1, c_2 > 0$ and

$\max(n_1, n_2) \geq n$

2) Big oh Notation :- it defines an upper bound of an algorithm. it bounds a function only from above



$f(n) = O(g(n))$

iff

$f(n) <= c \cdot g(n)$ A

$n >= n_0$ and $c > 0$

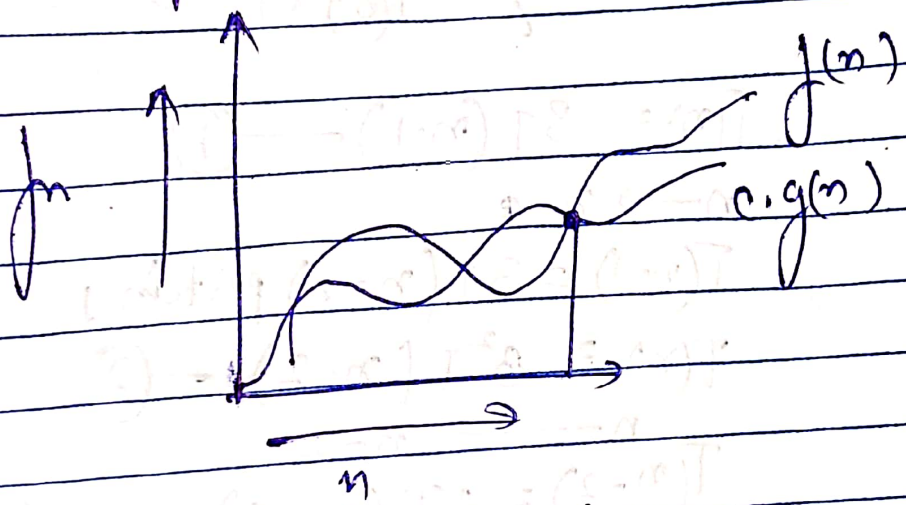
Date. _____

Page No. _____

3) Omega Notation :- Omega Notation provides an asymptotic lower bound

$$f(n) = \Omega(g(n))$$

$g(n)$ is "tight" lower bound of $f(n)$



$$f(n) = \Omega(g(n))$$

$f(n) > c \cdot g(n)$ $\forall$ $n > n_0$ and $c > 0$

Q 2 for (i = 1 to n)
i = i * 2

iteration 1 $\quad 1 \quad 2^0 \quad i=1$

iteration 2 $\quad 2 \quad 2^1 \quad i=2$ Date. Page No.

" $\quad 3 \quad 2^2 \quad i=4$

|

iter $K \quad 2^{K-1}$

$2^{K-1} \geq n \Rightarrow K-1 \log_2 = \log n$

$K = (\log n + 1)$

Complexity $= O(\log n)$

Q

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

$T(n) = 3T(n-1)$ —— (1)

$n \rightarrow n-1$

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

$T(n) = 3^2 T(n-2)$ —— (2)

$n \rightarrow n-2$

$T(n-2) = 3T(n-3)$ — put in (2)

$T(n) = 3^3 T(n-3)$ —— (3)

from (1) (2) & (3)

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

$n-K = 0 \quad n = K$

$T(n) = 3^K T(0)$

$T(n) = O(3^n) \quad \because n = K$

Q

$T(n) = 2T(n-1)$ $n > 0$ — (1)

$T(0) = 1$

Date.
Page No.

- $n \longrightarrow n-1$

$T(n-1) = 2T(n-2)$ ⟶ put in (1)

$T(n) = 2^2 T(n-2)$ — (2)

$n \longrightarrow n-2$

$T(n-2) = 2T(n-3)$ — put in (2)

$T(n) = 2^3 T(n-3)$ — (3)

from (1) (2) (3)

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

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

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

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

$n-k = 0 \quad n = k$

$T(n) = 2^n T(0)$

$\boxed{T(n) = \Theta(2^n)}$

```
int i=1
s=1
while (s<=n)
    i++;
    s+=i;
    print("*");
```

		S
i=1	s=s+i = 1+1	
i=2	s=s+i = 1+1	
i=3	s=s+i = 1+2	
i=4	s=s+i = 1+2+3	
i=5	s=s+i = 1+2+3+4	
⋮	⋮	
n	s=s+i	1+2+3+4...n

$1+2+3+4+\dots+n$

$$= \frac{n(n+1)}{2} \Rightarrow \frac{n^2+n}{2}$$

$\Rightarrow O(n^2)$ $\because$ $n^2 >> n$, $n \geq 0$

Q6 Time Complexity of

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

at every iteration i increase by 1
& goto n times

Date.
Page No.

i=1 i++;
i=2 i++
i*2 (k-1)
k*k<= n
k^2<2n
k<2√n
O(√n)

Q 7. void function (int n)
int i, count=0;
for(i=n/2; i<=n; i++) — (1)
for(j=1; j<n; j=j*2) — (2)
for(k=1; k<=n; k=k*2) — (3)
count++;

loops
first loop → loop 2

iteration(i) n/2 (n) ∴ O(n)

iter 1 j=2^0 → k=2^0
iter 2 j=2^1 → k=2^1
iter 3 j=2^2 → k=2^2
iter K j=2^k → k=2^l
∴ K=log(n) l=log(n)

final Complexity O(n log^2 n)

Q(i) time complexity

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

$O(n^3)$

operation performs in the function is $O(n^2)$ & function called by recursion its also n times

so $n \times n^2 \Rightarrow O(n^3)$

Q(9) Time Complexity of

Date.
Page No.

```
void function(int n)
{
    for(i=1 to n){                    (1)
        for(j=1 ; j<n ; j=j+1) {      (2)
            print("*");
        }
    }
}
```

for loop 1 it run (n) times
for loop(2) it also run n times
so its Complexity became

loop1 Complexity * loop2 Complexity

$O(n) * O(n)$

$T(n) \quad O(n^2)$

Q(10) $n^k \quad a^n$

$k >= 1 \quad a > 1$

taking $k = a = 2$

$n^2 \quad 2^n$

we can say $n^2 = O(2^k)$

$n^k = O(a^n)$

Q 11 void fn(int n)
{
int i=0, j=1;
while(i<n){
i=i+j;
j++
}
}

Date.
Page No.

for(n) {	iteration	j	i	operation
	iter 1	j=1	i=0	i=0+1
	iter 2	j=2	i=1	i=0+1+2
	iter 3	j=3	i=3	i=0+1+2+3
	K	j=K	i=	i=0+1+2+...K

$1+2+3+4+\dots+K$

$\frac{K^2+K}{2} < n$

$K^2 >> K$

$K^2 \approx K$

$K^2 - 1 \approx n$ — (1)

$K^2 < n$

$O(K^2-1)$ from (1)

$O(n^2)$

Q (12)

```
int fib(int n)
if(n<=1)
    return n
return fib(n-1) + fib(n-2)
```

Q.

$$T(n) = \begin{cases} T(n-1) + T(n-2) \\ T(0) = T(1) = n \end{cases}$$

Q. $T(n) = T(n-1) + T(n-2)$

Let us Consider

$T(n-1) \approx T(n-2)$

because $n >>$ than 1 and 2

$T(n) = 2T(n-2) + C$

↓ No. of Operation which will be Constant every time

$T(n) = 2T(n-2) + C$ ——— (1)

$n \longrightarrow n-2$

$T(n-2) = 2T(n-4) + C$ — put in (1)

$T(n) = 2(2T(n-4) + C) + C$

$T(n) = 2^2 T(n-4) + 2C + C$

$T(n) = 2^2 T(n-4) + 3C$ ——— (2)

$n \longrightarrow n-4$

$T(n-4) = 2T(n-6) + C$ — put in (2)

$T(n) = 2^2 (2T(n-6) + C) + 3C$

(13) Write programs which have complexity

Date.
Page No.

* $n \log n$

```
for(i=0; i<=n; i++)
    for(j=1; j<n; j=j*2)
        Cout++
```

* $O(n^3)$

```
for(i=0; i<=n; i++)
    for(j=1; j<=n; j++)
        for(k=1; k<=n; k++)
            Cout++
```

* $\log(\log n)$

```
int do_something(int n)
{
    if (n<=2)
        return 1;
    else
        return(Do something(floor
        (sqrt(n))+n));
```

$T(n) = 2^3 T(n-6) + 4c + 3c$

Date:
Page No.

$T(n) = 2^3 T(n-6) + 7c$

$\vdots$

$T(n) = 2^k (n-2k) + (2^k - 1)c$

$n - 2k = 0 \quad k = n/2$

$T(n) = 2^{n/2} T(0) + (2^{n/2} - 1)c$

$T(n) = (1+c)\, 2^{n/2} - c$

$T(n) \propto 2^{n/2}$ (for lower bound)

for upper bound

$T(n-2) \geq T(n-1)$

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

$\vdots$

$4T(n-2) + 3$

$\vdots$

$8T(n-3) + 7c$

$\vdots$

$2^k T(n-k) + (2^k + 1) + c$

$T(n) = 2^n T(0) + 2^n - 1c$

$T(n) = 2^n (1+c)$

$T(n) = 2^n$ for upper bond.