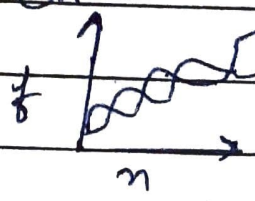
# Assignment :- 01
## Design AND Analysis of Algorithm

**Ans 1** Asymptotic notations to analyse an algo. running time identifying its behaviour as the input size for the algorithm increases. These notations are used to tell the complexity of an algo. when input is large.

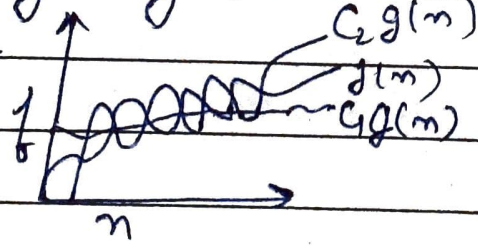
Type of Asymptotic notations

1) Big- Oh



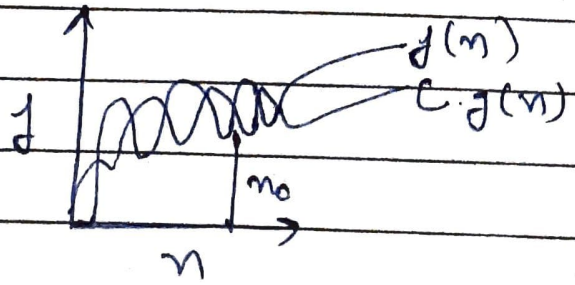

$f(n) = o(g(n))$ if & only if $f(n) \leq C.g(n)$ $\forall\ n \geq n_0$

2) ~~Big omega (Ω)~~ Theta




$f(n) = \theta(g(n))$ if & only if $C_1 g(n) \leq f(n) \leq C_2 g(n)$ $\forall\ n \geq \max(n_1, n_2)$

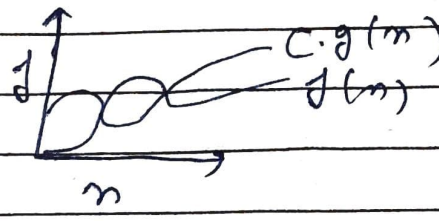
3) Big omega (Ω)




$f(n) = \Omega g(n)$ if & only if $f(n) \geq C g(n)$ $\forall\ n \geq n_0$

DATE : / 20
PAGE No

4) Small-Oh (o)




$f(n) = o(g(n))$

$f(n) < C \cdot g(n) \; \forall \; n > n_0$

Ans 2 $i = 1, 2, 4, 8 \ldots n$

$2^0, 2^1, 2^2, 2^3 \ldots 2^k \rightarrow G.P$

$a = 1, \; r = 2$

$t_k = ar^{k-1}$

$= 1 \times 2^{k-1}$

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

$2^k = 2n \qquad K = \log_2(2n)$

$K = \log_2(n) + \log_2(2)$

$= \log_2 n + 1$

$\therefore TC = O(\log n + 1)$

$= O(\log n)$

Ans 3 $T(n) = 3T(n-1) \quad -(1) \qquad n > 0$

$T(1) = 1$

but $n = n-1$ in Eq$^n$ (1)

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

put $T(n-1)$ in Eq$^n$ (1)

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

$T(n) = 9T(n-2) \quad -(3)$

put $n = n-2$ in Eq$^n$ (1)

$T(n-2) = 3T(n-3) \quad -(4)$

put $T(n-2)$ in Eq$^n$ (3)

$T(n) = 9(3T(n-3))$

$T(n) = 27T(n-3)$

DATE : / 20
PAGE No

$T(n) = 3^k T(n-k)$ — (5)

$\because T(1) = 1$

$n - k = 1$

$k = n - 1$ — (6)

from (5) & (6)

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

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

$\Rightarrow \boxed{T.C = O(3^n)}$

Ans 4 $T(n) = 2T(n-1) - 1$ — (1)

$T(1) = 1$

put $n = n-1$ in Eqn (1)

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

put $T(n-1)$ in Eqn (1)

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

$T(n) = 4T(n-2) - 2 - 1$ — (2)

put $n = n-2$ in Eqn (1)

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

put $T(n-2)$ in Eqn (2)

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

$T(n) = 8T(n-3) - 4 - 2 - 1$ — (3)

⋮

$T(n) = 2^k [T(n-k)] - 2^{k-1} - 2^{k-2} \cdots 2^1 - 2^0$ — (4)

$T(1) = (1)$

$n - k = 1$

$k = n - 1$ — (5)

from (4) & (5)

$T(n) = 2^{n-1} [T(n-(n-1))] - 2^{n-2} - 2^{n-3} \cdots 2^0$

$= 2^{n-2} - 2^{n-1} - 2^{n-3} \cdots 1$

$= \frac{1}{2}[2^n - (2^n - 1)]$

DATE : / 20
PAGE No

$= \frac{1}{2} \times 1 = \frac{1}{2}$ $\boxed{T.C = O(1)}$

Ans 5 $n = 1, 3, 6 \cdots K \longrightarrow AP$

$T.C = \frac{K(K+1)}{2}$

$\frac{O(K^2+K)}{2}$

$O(K^2) \Rightarrow \boxed{T.C = O(n^2)}$

Ans 6

```
Void function (int n) {
    int i, count = 0;          → 1
    for (i = 1; i*i <= n; i++) {
        count++; }             (n+1)^2
    }                          n
```

$\Rightarrow 1 + 1 + (n+1)^2 + n + n$

$\Rightarrow 2 + 2n + n^2 + 1 + 2n$

$\Rightarrow O(n^2 + 4n + 3)$

$O(n^2)$ //

Ans 7

```
Void function (int n) {
    int i, j, k, count = 0;
    for (i = n/2; i <= n; i++)          — O(n)
        for (j = 1; j <= n; j = j*2)    — log(n)
            for (k = 1; k <= n; k = k*2)
                count++;                — log(n)
```

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

$= \log^2(n)$

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

DATE : / 20
PAGE No

Ans 8 function (int n) {
if (n == 1) return; → 1
for (i = 1 to n)
for (j = 1 to n) } → n * n
printf ("*") → 1
}
}
function (n-3) → n * n²
}

$1 + n^2 + 1 + n^3$

$n^3 + n^2 + 2 \Rightarrow O(n^3)$

Ans 9

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

| i | j | times |
|---|---|---|
| 1 | 1 → n | (n+1)/2 |
| 2 | 1 → n | (n+1)/2 |
| ⋮ | | |
| n | 1 → n | (n+1)/2 |

$T.C = \log n \left(\frac{n+1}{2}\right)$

$= O\left(\frac{n+1}{2} \log n\right)$

$= O(n \log n)$

Ans 10 $n^k \leq C.a^n$

$a^n + n^k \leq C.a^n \rightarrow a^n$

$a^n + n^k \leq a^n (C-1)$

$\frac{a^n + n^k}{a^n} \leq (C-1)$

$C \geq 1 + \frac{n^k}{a^{n_0}} + 1$

$C \geq 2 + \frac{n_0^k}{a^n}$

$C \geq 2 + \frac{n_0^1}{1.5^n}$

$n_0 = 1$

$C \geq 2 + \frac{1}{1.5}$

$C \geq 3.0 + 1$

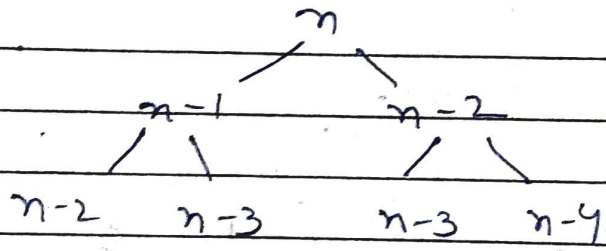
$C \geq 4$

DATE: / 20
PAGE No

Ans 11 Time complexity = O(n).
The execution of different code lines are
1) while ⇒ (n-1)
2) i=i+j ⇒ (n)
3) j++ ⇒ (n)

T.C = n+n+n-1
= 3n-1
T.C = O(3n-1)
T.C = O(n)

Ans 12 The main working of fibonacci series is
f(n) = f(n-1) + f(n-2)




T(n) = 1+2+4+ -- $2^n$
a = 1, r = 2, $\frac{a(r-1)}{r-1} = \frac{1(2^{n+1}-1)}{2-1}$
$= 2^{n+1}$
$T(n) = O(2^{n+1}) = O(2^n \cdot 2^1) = O(2^n)$ //

Ans 13 O(n(log n))

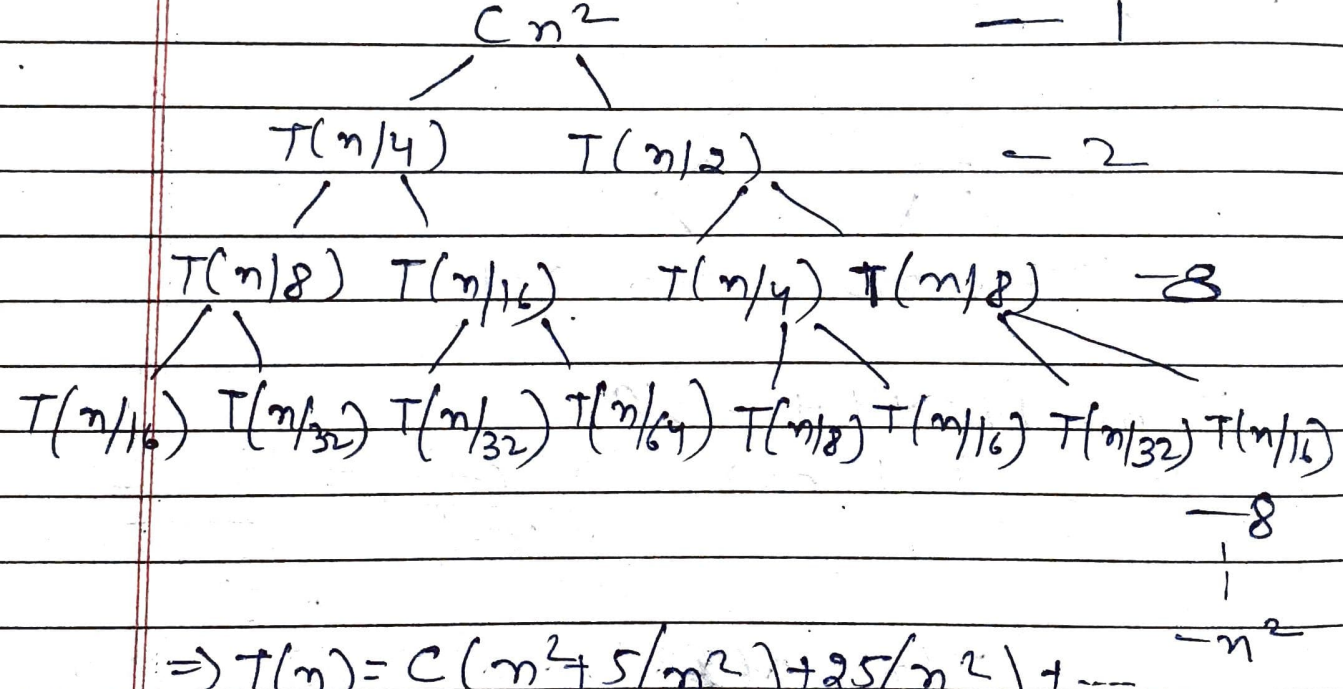
```
int n;
for(int i=0; i<n; i++){
    for(int j=n; j>0; j/=2){
        printF("*");
    }
}
```

DATE : / 20
PAGE No

Ans13 $O(n^3)$

```
int i,j,k;
for (i=1; i<=n; i++){
    for(j=1;j<=n;j++)
    {
    for(k=1; k<=n; k++){
        printf("*");
    }
}
}
```

Ans14 $T(n) = T(n/4) + T(n/2) + Cn^2$




$\Rightarrow T(n) = C\left(n^2 + 5\left(\frac{n^2}{16}\right) + 25\left(\frac{n^2}{256}\right) + \dots\right)$

ratio = 5/16.

$= \frac{n^2}{1 - 5/16} \Rightarrow O(n^2)$ //

DATE: / 20
PAGE No

Ans 15.
```
int fun(int n){
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j+=i){
            O(1);}
    }
```

| i | j | times |
|---|---|---|
| 1 | 1→n | n |
| 2 | 1→n | n/2 |
| ! | | |
| n | 1→n | n/n |

$T(n) = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{n}$

$= n(1 + 1/2 + 1/3 + \cdots + 1/n)$

$= T(n) = n \log(n)$

Ans 16. $i = 2, 2^c, 2^{c^2}, 2^{c^3} \ldots 2^{c \log_c(\log n)}$

The last term has to be $<= n$.

$2^{c \log_c(\log n)} = 2^{\log n} = n$

There are in total $\log_c(\log(n))$ iterations

Each takes a constant time to run

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

Ans 18 a) $100 < \log\log n < \log n < \sqrt{n} < n < n \log n = \log(n!)$
$< n^2 < 2^n < 2^2 < 4^n < n!$

b) $1 < \log\log(n) < \sqrt{\log(n)} < \log n < 2n < 4n$
$< 2(2n) < \log(2n!) < 2\log(n) < n < n\log n$
$= \log(n!) < n < n^2$

c) $96 < \log_2(n) = \log_8(n) < n\log_8(n) =$
$n\log_2(n) = \log(n!) < 5n < 8n^2 < 7n^3$
$< 8^{2n}$

Ans 19
```
for (int i=0 to n-1){
    if (Array[i]=Key){
        return i; }
    }
    return -1;
}
```

Ans 20 a) Iterative Insertion sort

```
Void Insertion sort (int arr[], int n)
{
int i, temp, j;
for (int i=1; i<=n-1; i++)
    temp = arr[i];
    j = i-1;
while (j>=0 && arr[j]>temp){
    arr[j+1] = arr[j];
    j = j-1; }
    arr[j+1] = temp;
```

b) Recursive Insertion Sort

```
Void Insertion sort (int arr[], int n){
    if (n<2)
        return;
    Insertion Sort (arr, n-1);
    last = arr[n-1]; j = n-2;
while (j>=0 && arr[j] > temp){
    arr[j+1] = arr[j];
    j = j-1; }
    arr[j+1] = last;
}
```

DATE : / 20
PAGE No

Ans21

| Algorithm | Best Case | Avg. Case | Worst Case |
|---|---|---|---|
| Bubble | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Selection | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Insertion | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| Merge | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |
| Quick | $O(n \log n)$ | $O(n \log n)$ | $O(n^2)$ |
| Heap | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |

Ans22

| Algorithm | In place | Stable | Online |
|---|---|---|---|
| Bubble | ✓ | ✓ | ✗ |
| Selection | ✓ | ✗ | ✗ |
| Insertion | ✓ | ✓ | ✓ |
| Merge | ✗ | ✓ | ✗ |
| Quick | ✗ | ✗ | ✗ |
| Heap | ✓ | ✗ | ✗ |

Ans23. Iterative Binary Search

```
Int Binary search (int arr[], int l, int r, int x)
    while (l <= r){
        int m = (l + r)/2;
        if (arr[m] = x;
            return m;
        else if (arr[m] < x)
            l = m + 1;
        else r = m - 1; }
        return -1;
    }
```

DATE : / 20
PAGE No

## Recursive Binary Search

```
int Binary search (int arr[], int l, int r,
                                  int x)
    if (l > r)
        return -1;
    int m = (l+r)/2;
    if (arr[m] == x)
        return m;
    else if (arr[m] < x)
    return Binary search (arr, m+1, r, x);
    else
    return Binary search (arr, l, m-1, x);
    }.
```

## Time Complexity,

Linear (Recursive) → O(n)
Binary ( " ) → O(n)
Linear (Iterative) → O(1)
Binary ( " ) → O(1)

## Space Complexity

Linear (Recursive) → O(1)
Binary ( " ) → O(logn)
Linear (Iterative) → O(1)
Binary ( " ). → O(1).

Ans 24. Recurrence Relation for binary search
$T(n) = T(n/2) + 1.$