5) Logarithmic Time:-

$i = n = 2^K$

```
int main()
{
    int i, n, a, b, c;          ---> constant
                                     +
    a = b + c;                  ---> constant
                                     +
    for (i = 1; i <= n; i = i*2)
    {
        b++;   ---> 1 + 1 + 1 + 1 + 1   } log2 n
    }
    return 0;
}
```

$= C_1 + C_2 + \log_2 n = O(\log n)$

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

$\boxed{2^0, 2^1, 2^2, 2^3}, \ldots, 2^K > n$

$2^K \simeq n$

$\log(2^K) \simeq \log n$

$K \log 2 = \log n$

$\boxed{K = \dfrac{\log n}{\log 2} \simeq \log_2 n}$

```
int main()
{
    int i, n, a, b, c;
    a+b;
    for(i=1; i<n; i=i*3)
    {
        b++;
    }
    return 0;
}
```

$\longrightarrow$ constant

$\longrightarrow \log_3 n$

$i = 1, 3^1, 3^2, 3^3, 3^4, \ldots, 3^K$

$3^K = n$

$\log(3^K) = \log n$

$K \log(3) = \log n$

$K = \frac{\log n}{\log 3}$

$\boxed{K = \log_3 n}$

$T = \log_3 n + C = O(\log n)$

```
int main()
{ int i, n, a,b,c;
   c = a/b;
   for (i= n ; i>=1 ; i=i/2)
   {
       c++;
   }
   return 0;
}
```

→ constant

log n

$$\boxed{O(\log n)}$$

$$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, \frac{n}{2^k} \rightarrow (k+1) \text{ times}$$

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$\boxed{k = (\log_2 n)}$$

$\boxed{n = 16}$ $\quad i = 16, 8, 4, 2, 1$

$1 + 1$

```
int main()
{
    int i, n, j, a, b, c;     } → constant
      a = b + c;              }

    for ( i = 0 ; i < n ; i++ ) → n
                                   x
    {
        for ( j = 1 ; j < n ; j = j * 2 ) → log n
        {
            c--;
        }
    }
    return 0;
}
```

$$T = c + n \times \log n$$

$$= \boxed{O(n \log n)}$$

Recurrence Relation

$n = 3$

fact(3)

3×f(2)

f(2)

2×f(1)

Recursion :– function calling itself is called Recursion.

$$\lfloor\underline{5} = 5 \times \lfloor\underline{4}$$
$$= 5 \times 4 \times \lfloor\underline{3}$$
$$= 5 \times 4 \times 3 \times \lfloor\underline{2}$$
$$= 5 \times 4 \times 3 \times 2 \times \lfloor\underline{1}$$
$$= 5 \times 4 \times 3 \times 2 \times 1$$

```
fact(n)
{
        int f = 1;
  for(i=1 ; i<=n; i++)
      f = f*i;
}
```

```
for(i=n; ; i--)
```

```
fact(n)
{
    if(n==1) return 1;
   else
return n*fact(n-1);
}
```

$fact(n) \longrightarrow T(n)$

{

if (n <= 1)
  return 1;   ] constant +

else
  return n * fact(n-1); $\longrightarrow$ n times

constant + T(n-1)

}

$\Rightarrow O(n)$

$f(n) \rightarrow f(n-1) \rightarrow f(n-2) \; --- \; -- \rightarrow f(1) \longrightarrow O(n)$

$$T(n) = 1 + T(n-1), \quad n > 1$$
$$T(1) = 1, \quad \boxed{n = 1}$$

Recurrence Relation

- Back Substitution
- Tree Method
- Masters Theorem

Back Substitution Method

$$\left[\begin{array}{lll} T(n) = T(n-1) + 1 & , & n > 1 \\ T(1) = 1 & , & n = 1 \end{array}\right]$$

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

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

$$T(n) = [T(n-1)] + 1$$

$$= [T(n-2) + 1] + 1 = T(n-2) + 2$$

$$= [T(n-3) + 1] + 2 = T(n-3) + 3$$

$$\vdots$$

$$= T(\underbrace{n-k}) + k$$

$$\boxed{n - k = 1}$$

$$\boxed{k = n - 1}$$

$$= T(1) + k = 1 + k = \cancel{1} + n - \cancel{1} = n = \boxed{O(n)}$$

Tree Method :-

$$\begin{cases} T(n) = 2T(n/2) + 1, & n > 1 \\ T(1) = 1, & n = 1 \end{cases}$$

$$T(n/2) = 2T\left(\frac{n}{4}\right) + 1$$

$$T(n) = 1 + 2 + 4 + 8 + \dots + 2^K$$

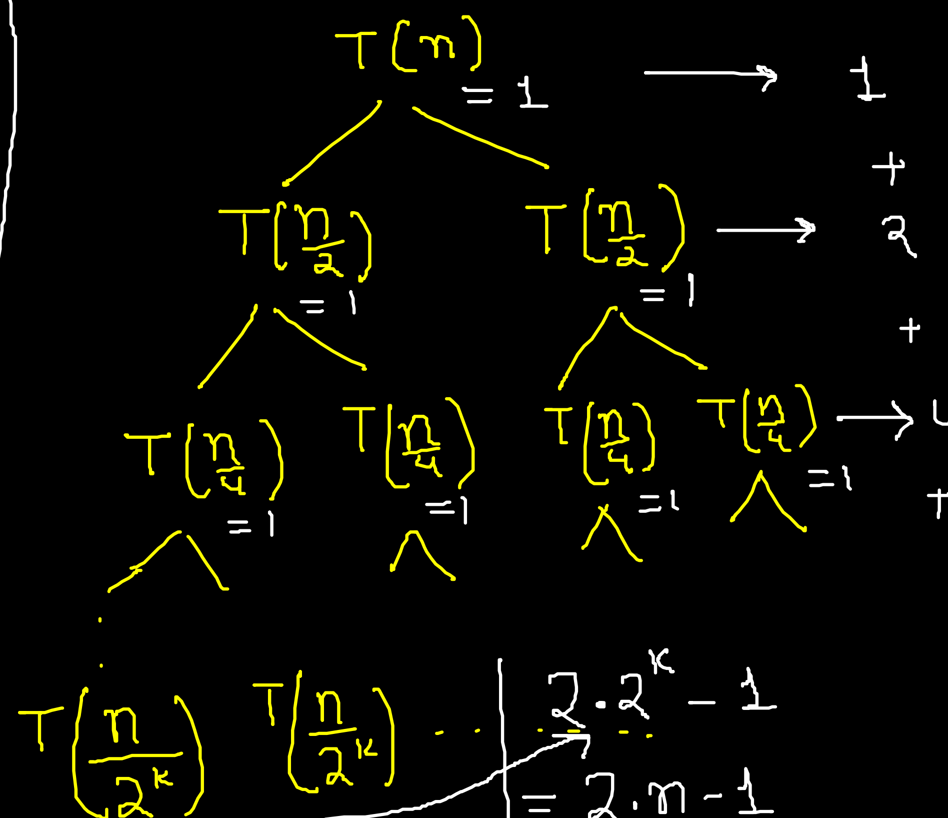
$$S_n = \frac{a(r^n - 1)}{r - 1}$$

r = common ratio

a = first term, n = no. of terms

$$S_n = \frac{1(2^{K+1} - 1)}{2 - 1}$$

$$= (2^{K+1} - 1)$$



$2^K = n \Rightarrow \boxed{\frac{n}{2^K} = 1}$

$$2 \cdot 2^K - 1$$

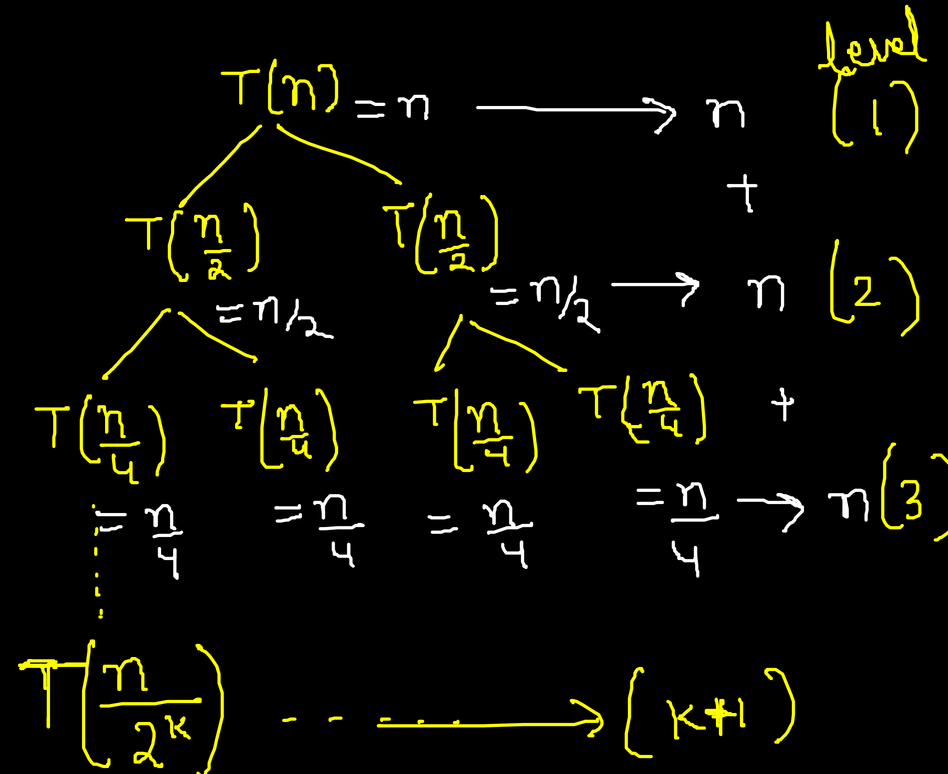
$$= 2 \cdot n - 1$$

$$= O(n)$$

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n, & n > 1 \\ T(1) = 1, & n = 1 \end{cases}$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$$



$$\text{Total} = n + n + \ldots + \text{no. of level}$$

$$= n \times \text{no. of level} = n\log n + n$$

$$= n \times (k+1) \qquad = \boxed{O(n\log n)}$$

$$= n \times (\log n + 1)$$

$$\frac{n}{2^k} = 1 \rightarrow n = 2^k$$

$$\boxed{k = \log_2 n}$$

Master Theorem

the form $T(n) = aT(\frac{n}{b}) + \boxed{\Theta(n^k log^p n)}$, where $\boxed{a \geq 1}$, $b > 1$, $\boxed{k \geq 0}$ and p is a real number, then:

1) If $a > b^k$, then $T(n) = \Theta(n^{log_b^a})$

2) If $a = b^k$

a. If $p > -1$, then $T(n) = \Theta(n^{log_b^a} log^{p+1} n)$

b. If $p = -1$, then $T(n) = \Theta(n^{log_b^a} loglogn)$

c. If $p < -1$, then $T(n) = \Theta(n^{log_b^a})$

3) If $\boxed{a < b^k}$

a. If $p \geq 0$, then $T(n) = \Theta(n^k log^p n) = \Theta(n)$

b. If $p < 0$, then $T(n) = O(n^k)$

$T(n) = 3T(\frac{n}{4}) + \boxed{n}$

$a = 3$, $b = 4$, $K = 1$ $p = 0$

$b^K = 4$

$\boxed{a < b^K}$

$3 < 4$

Master Theorem

$T(n) = 3T(n/2) + n^2$ $= O(n^2)$

$a = 3,\ b = 2,\ K = 2,\ P = 0,\ b^K = 4$

$a < b^K$ $\quad \Theta(n^K \log^P n) = O(n^2)$

$T(n) = 4T(n/2) + n^2$ $= O(n^2 \log n)$

$$\left[\log_2 4 = \frac{\log 4}{\log 2} = \frac{\log 2^2}{\log 2} = 2\frac{\log 2}{\log 2} = 2\right]$$

$a = 4,\ b = 2,\ K = 2,\ P = 0,\ b^K = 4$

$a = b^K$ $\quad \Theta(n^{\log_b a} \log^{P+1} n) = \Theta(n^2 \log n)$

$T(n) = T(n/2) + n^2$ $= O(n^2)$

$a = 1,\ b = 2,\ K = 2,\ P = 0 = b^K = 4$

$a < b^K$

Master Theorem

$T(n) = 2^n T(n/2) + n^n$ ✗ we cannot apply master theorem

$\because a \neq$ constant

$T(n) = 16T(n/4) + n$ $= O(n^2)$ $a = 16,\ b = 4,\ k = 1,\ p = 0,\ b^k = 4$

$a > b^k$ $O(n^{\log_4 16}) = O(n^2)$

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

$$\left[\frac{\log 16}{\log 4} = \frac{\log 4^2}{\log 4} = 2 \frac{\log 4}{\log 4} = 2 \right]$$

Master Theorem

$a=2,\ b=2,\ K=1,\ P=-1$

T(n) = 2T (n/2) + n / logn $= O(n \log \log n)$

$a=\sqrt{2}=1.414,\ b=2,\ K=0,\ P=1 \rightarrow a > b^k$

T(n) = √2 T (n/2) + log n $= O(\sqrt{n})$

$a=3,\ b=2,\ K=1,\ P=0 \rightarrow a > b^k$

T(n) = 3T (n/2) + n $= O(n^{\log 3})$

$$\left[\frac{1}{\log n} = (\log n)^{-1} \Rightarrow P = -1 \right]$$

1) $\log_b a = \frac{\log a}{\log b}$

2) $\log a^b = b \log a$

3) $\log(m \times n) = \log m + \log n$

4) $\log(m/n) = \log m - \log n$

Youtube channel :-

Prateek Jain Academy
9555031137

Master Theorem

$$\begin{bmatrix} T(n) = T(n-1) + 1 & , n > 1 \\ T(1) = 1 & , n = 1 \end{bmatrix}$$

$T(n) \longrightarrow T(n/a) \; \ldots\ldots$ ✓

$\searrow T(n-a) \; \ldots\ldots$ ✓

Master Theorem

Master Theorem for subtract and conquer recurrence

Let $T(n)$ be a function defined on positive n, and having the property

$$T(n) = \begin{cases} c, & \text{if } n \le 1 \\ aT(n-b) + f(n), & \text{if } n > 1 \end{cases}$$

for some constants $c, a > 0, b \ge 0, k \ge 0$, and function $f(n)$. If $f(n)$ is in $O(n^k)$, then

$$T(n) = \begin{cases} O(n^k), & \text{if } a < 1 \\ O(n^{k+1}), & \text{if } a = 1 \\ O\left(n^k a^{\frac{n}{b}}\right), & \text{if } a > 1 \end{cases}$$

Master Theorem for subtract and conquer recurrence

Find the complexity of the below recurrence:

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

9555031137

$T(n) = aT(n-b) + n^k$

$= c$

$a = 3,\ b = 1,\ c = 1,\ k = 0$

$a > 1$

$\theta(n^k . a^{n/b}) = \theta(n^0 . 3^n) = \theta(3^n)$

Sat = 9 AM – 1 PM

Sun = 9 AM – 1 PM

Tues = 8 PM – 10 PM

Thus = 8 PM – 10 PM

(doubt)

Master Theorem for subtract and conquer recurrence

Find the complexity of the below recurrence:

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

$a = 2,\ b = 1,\ c = 1,\ k = 0$

$f(n) = 1$

$n^k = 1$

$n^0 = 1$

$\Theta(n^k \cdot a^{n/b})$

$\Theta(n^0 \cdot 2^n)$

$= \Theta(2^n)$

Find Running Time

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

$s > n$

i	s
1	1
2	2
3	4
4	
5	
6	
⋮	

K times

$O(\sqrt{n})$

$S = (1 + 2 + 3 + 4 + \_ \_ \_ \_ + K) > n$

$\frac{K(K+1)}{2} > n$

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

$K^2 + K = 2n$

$K^2 = n \Rightarrow K = \sqrt{n}$

Find Running Time

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


  }
```

Find Running Time

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

Find Running Time

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

Find Running Time

```
function( int n ) {
        if(n == 1) return;
        for(int i = 1 ; i <= n ; i + + ) {
                for(int j= 1 ; j <= n ; j + + ) {
                        printf("*" );
                        break;
                }
        }
}
```

Find Running Time

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