Data Structure & Algorithms

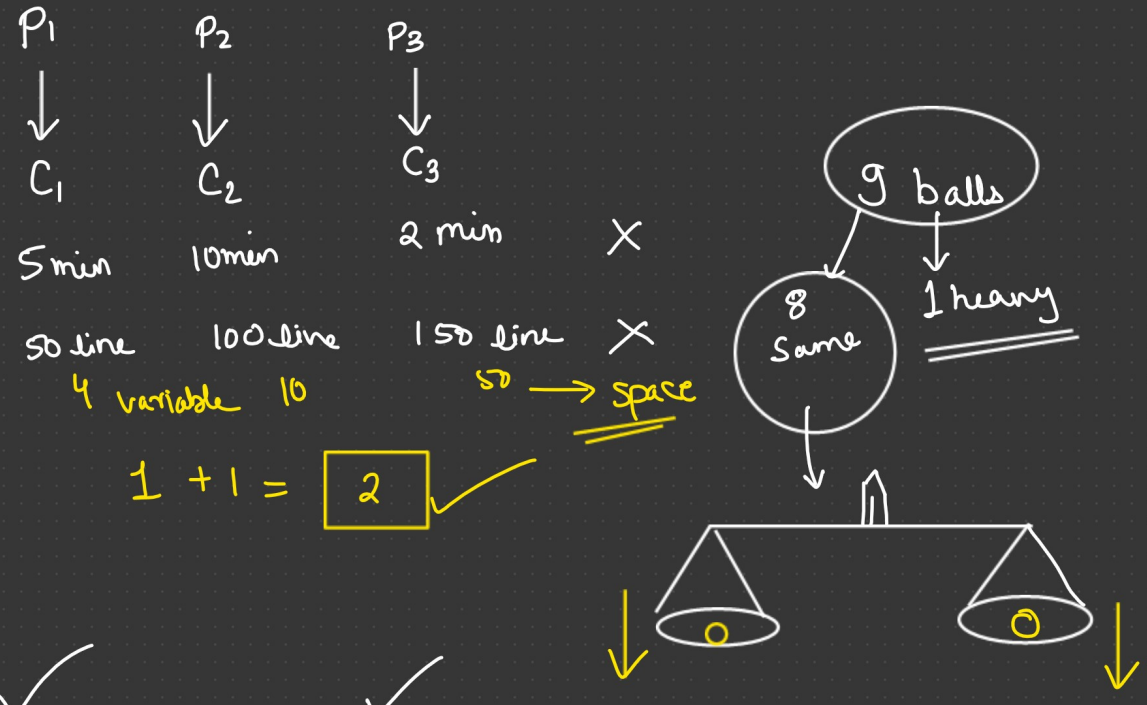
1) Time Complexity + Space complexity

Data Structure

① Linear DS ⟶ Array, Linked list, Stack, Queue

② Non- linear DS ⟶ Tree, Graph

Analysis of Algorithm



① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

G_1 G_2 G_3

1 2 3

```
int a[9] = { 1, 1, 1, 1, 2, 1, 1, 1, 1 };

for (i=1; i<=8; i++)        ⟶ 8 times
{
    if (a[0] < a[i])        ⟶ 8 time ⟶ 17 line
        return i;           ⟶ 1
}

if ( a[0] + a[1] + a[2] == a[3] + a[4] + a[5])  ⟶ 1
{
    if ( a[6] == a[7])      ⟶ 1
        return 8;
    else if( a[6] > a[7])   ⟶ 1
        return 6;
    else
        return 7;           ⟶ 1
}
else if(          >          )  ⟶ 1

else {            }   ⟶ 2
```

4 line

Rate of Growth

Types of Equation:—

1) Linear eqn ⟶ $y = mx + c \Rightarrow ax + by + c = 0$

2) quadratic eqn ⟶ $ax^2 + bx + c = 0$

3) Cubic eqn ⟶ $ax^3 + bx^2 + cx + d = 0$

4) bi-quadratic eqn → $ax^4 + bx^3 + cx^2 + dx + e = 0$

5) log. eqn → $a\log x + b$

6) expo. eqn. → e^{2x}, 3^x, 4^{2x+1}

7) Polynomial eqn ⟶ $ax^n + bx^{n-1} + \ldots + c = 0$

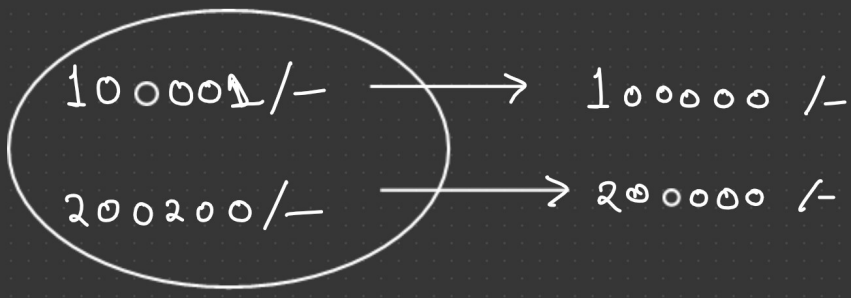
degree $= n$

$\log x < ax+b < ax^2 < ax^3 < ax^4 \ldots < e^x$

$f(x) = 2x^2 + 3x - 1$, find $f(2)$, $x = 2$

$f(x, y) = 3x^2 - 2y + 3$ find $f(3, 4)$ $x = 3, y = 4$

$A(x, y) = x \times y$ $\quad A(2, 3)$



Linear eqn :- $f(n) = 3n + 5$

$n = 1$, 8
$n = 2$, 11
$n = 3$, 14
$n = 10$, 35
$n = 100$, 305
$n = 1000$, 3005 $\rightarrow$ 3000

neglible

$f(n) = 3n$

$f(n) = n$

2) Quadratic eqn :- $f(n) = an^2 + bn + c$

$f(n) = 2n^2 + 5n - 3 \approx n^2$

neglible

3) Cubic eqn :- $f(n) = 4n^3 - 2n^2 + 3n - 5 \approx n^3$

4) log. eqn :- $f(n) = 2\log n + 3 \approx \log n$

5) expo. eqn :- $f(n) = 3^n + 2 \approx 3^n$

Rate of Growth

1) Best Case :- Lower bound

2) Average case :- Tight bound

3) Worst Case :- Upper bound

Lower bound <= Tight bound <= Upper bound

Time Complexity & its Name

1) Constant $\longrightarrow$ 1
2) Logarithmic $\longrightarrow$ $\log n$
3) Linear $\longrightarrow$ n
4) Linear logarithmic $\longrightarrow$ $n \log n$
5) Quadratic $\longrightarrow$ n^2
6) Cubic $\longrightarrow$ n^3
7) exponential $\longrightarrow$ c^x; 2^n, 3^n

Asymptotic Notation

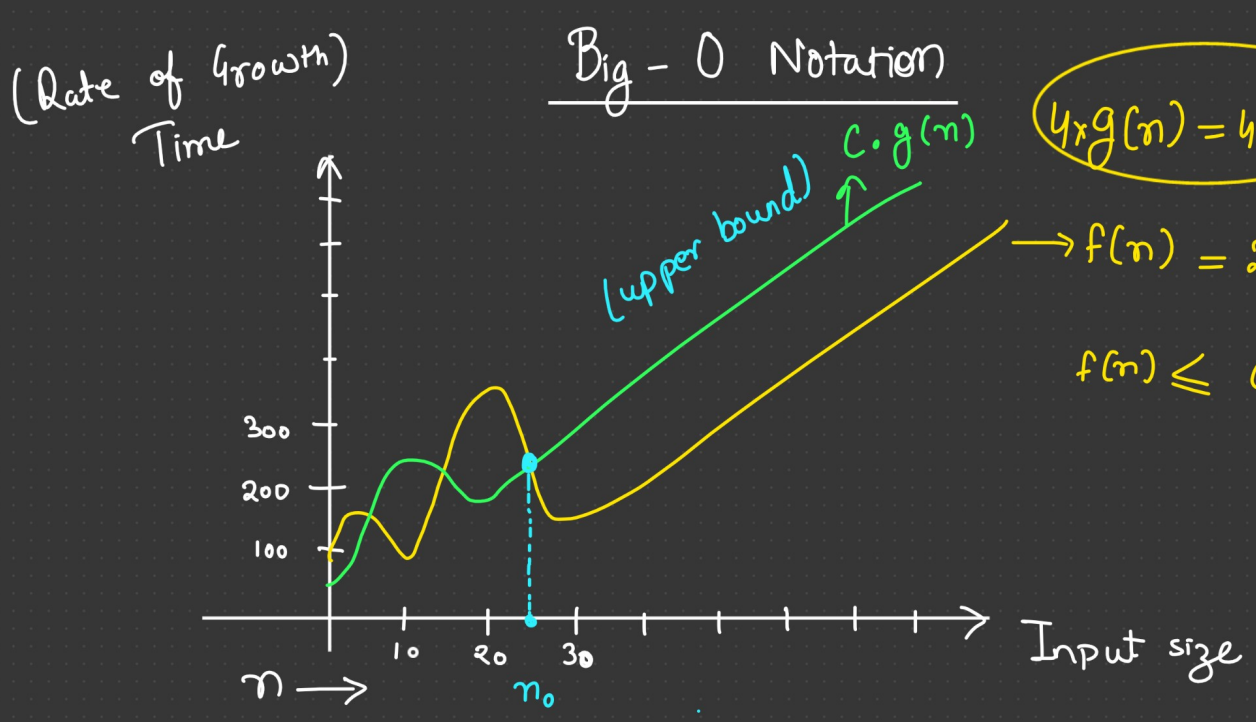
1) Best Case ⟶ Omega Notation = Ω

2) Average Case ⟶ Theta Notation = θ

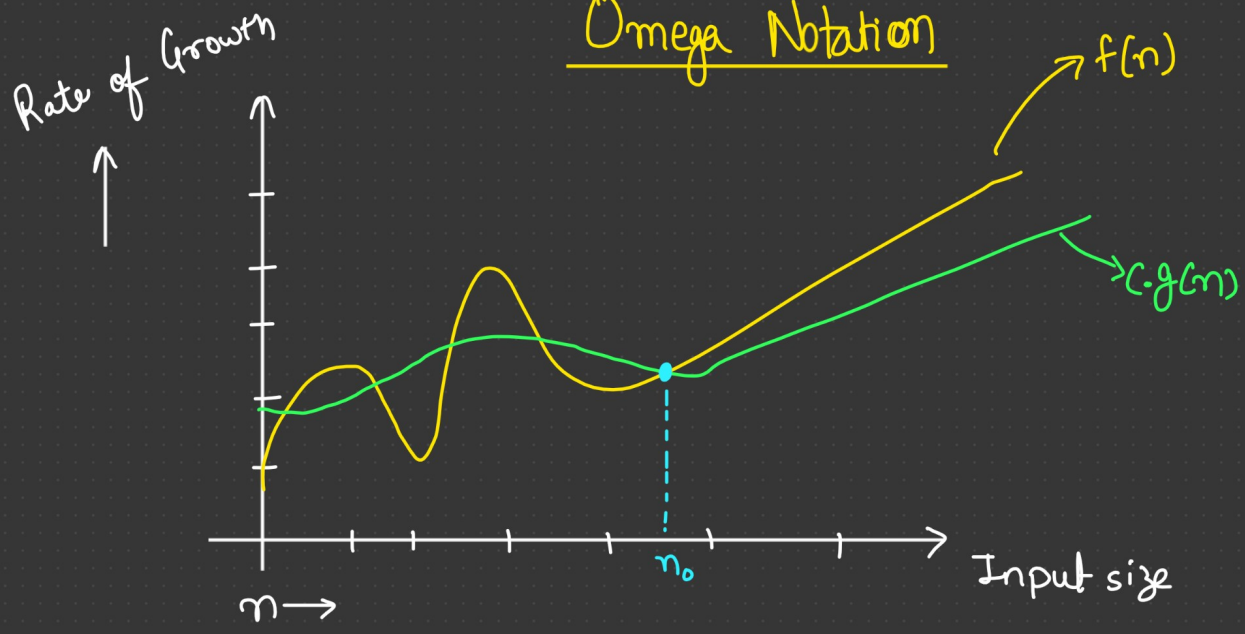
3) Worst Case ⟶ Big-O Notation = O

Ex:- $\Omega(1)$, $\theta(n^2)$, $O(n^3)$

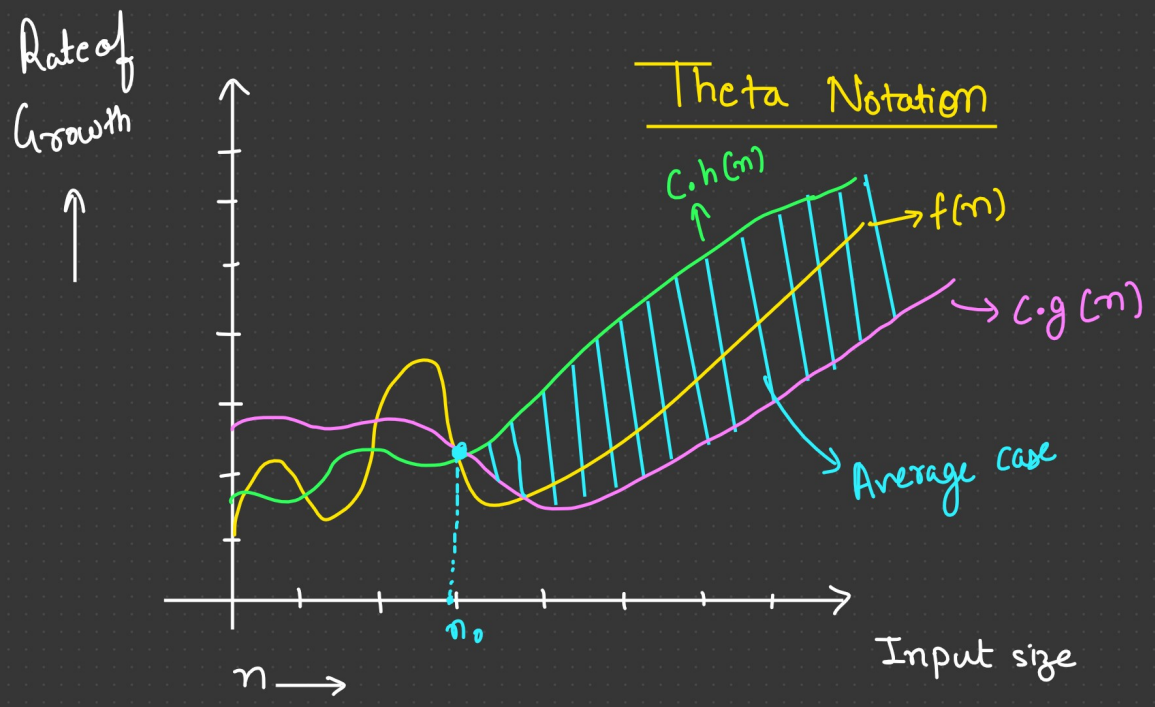
Big - O Notation



Omega Notation



Theta Notation



1) Constant Time :-

```
int main()
{
   int x, y, z;      ⟶ 3
   z = x+y;          ⟶ 2
   cout<< z;         ⟶ 1
   return 0;         ⟶ 1
}                      7
```

7, 100, 1000

O(1)

```
int main()
{
  int a, b, c, d, e, f;
  a = b+c+d;
  f = 2*a-3*b+c/f;
  cout << a<<b;
  return 0;
}
```

O(1)

2) Linear Time :-

```
int main()
{
   int a, b, c, d;        ⟶ 4
   a = b + c;             ⟶ 2
   Cin >> n;              ⟶ 1
   for(int i=1; i< n; i++) ⟶ 2 × (n)
   {
      a++;                ⟶ 1 × (n-1)
   }
   return 0;              ⟶ 1
}
```

Constant = O(1)

+

Linear = O(n)

+

constant = O(1)

O(1) + O(n) = O(n)

$$f(n) = 4+2+1+2n+(n-1)+1 = 3n+7$$

$f(n) = 3n + 7 \longrightarrow$ Linear eqn.

(3 and 7 are negligible)

$$f(n) = O(n) = \Omega(n) = \Theta(n)$$

```
int main()
{
    int a, b, c;          ⟶ 3
    a = b + c;            ⟶ 2
```

} Constant = O(1)

```
    for ( int i=0 ; i < n/2 ; i++)   ⟶ 3 × (n/2)
    {
        a++;                         ⟶ 1 × (n/2)
    }
```

} Linear = O(n)

```
    for (int j = 0 ; j < n ; j = j+2)   ⟶ 3 × (n/2)
    {
        b = a*2 - c ;                   ⟶ 3 × (n/2)
    }
```

} Linear = O(n)

```
    return 0;             ⟶ 1
```

} constant = O(1)

```
}
```

$$\max\left(O(1) + O(n) + O(n) + O(1)\right) = O(n)$$

$$f(n) = 3 + 2 + \frac{3n}{2} + \frac{n}{2} + \frac{3n}{2} + \frac{3n}{2} + 1$$

$$f(n) = 10\frac{n}{2} + 6 = 5n + 6 \approx O(n)$$

$$\frac{9n}{2} \approx n$$

3) Quadratic Time :-

```
int main()
{
  int a,b,c;                                  ---> 3
  for(int i=0; i<n; i++)                      ---> 2xn
  {                                                      x  ---> 4n^2
    for(int j=1; j<n; j++)                    ---> 2xn
    {
        a = b+c;                              ---> 2n^2
    }
  }                                                              n^2
  cout<<a;                                    ---> 1             +
  return 0;                                   ---> 1             Constant
}
```

$$O(n^2)$$

$$f(n) = 3 + 4n^2 + 2n^2 + 2 = 6n^2 + 5$$

$$f(n) = O(n^2)$$

```
int main()
{
    int a, b, c, n;
    a = b + c/2 + n;
    for( int i = 1; i < n/3 ; i++)
    {
        for(int j = 1 ; j < n ; j = j+2)
        {
            a++;
        }
    return 0;
    }
}
```

} → constant → O(1)

→ n/3

→ n/2

O(n)

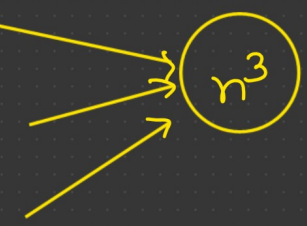
4) Cubic Time Complexity :-

```
int main()
{
    int x, y, z, n;

    x = y + z;

    for (int i=0; i<n ; i++)
    {
        for (int j=0; j<n ; j++)
        {
            for (int k=0; k<n; k++)
            {
                x++;
            }
        }
    }

    return 0;
}
```



```
int main()
{
    int a, b, c, n;
    b = n;
    if (a > 0)
    {
        for( int i = 0 ; i<n ; i++)
            b++;
    }
    else {
        while (b != 0)
        {
            b--;
            for( int j = 0; j<n ; j++)
                c++;
        }
    }
    return 0;
}
```

→ constant → O(1)

for(int i = 0 ; i<n ; i++) → O(n)

while (b != 0) → n

for(int j = 0; j<n ; j++) → n

O(n) × O(n) → $O(n^2)$

Worst case - $O(n^2)$

Best case - $\Omega(n)$

5) Logarithmic Time complexity :-

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

1, 2, 4, 8, 16, 32, ...

2^0 2^1 2^2 2^3 2^4 . . . . . $2^K > n$

K+1 times

$2^K = n$

$\log(2^K) = \log(n)$ $\quad [\because \log a^b = b \log a]$

$K \log(2) = \log(n)$

$K = \frac{\log n}{\log 2} = \log_2 n \approx O(\log n)$

```
int main()
{
  int a, b, c, n;

   c++;

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

1, 3, 9, 27, ...

3^0 3^1 3^2 3^3 .... $3^K > n$

K times

$3^K = n$

$\log 3^K = \log n$

$K \log 3 = \log n$

$K = \frac{\log n}{\log 3} = \log_3 n$

$\approx O(\log n)$

```
int main()
{
  int a, b, c, n;

    a = b+c;

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

K times

$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots$

$\frac{n}{2^0}, \frac{n}{2^1}, \frac{n}{2^2}, \frac{n}{2^3}, \ldots \frac{n}{2^K} < 1$

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

$$\Rightarrow n = 2^k$$

$$\Rightarrow \log n = \log 2^k$$

$$\log n = k \times \log 2$$

$$k = \frac{\log n}{\log 2} = \log_2 n$$

$$\approx O(\log n)$$

```
int main()
{
  int a, b, c, n;

  a = b+c;

  for (int i=0; i<n; i = i+2)
  {
    for (int j=1; j<n; j = j*2)
    {
      c++;
    }
  }

  return 0;
}
```

$\frac{n}{2}$ → $\frac{n}{2} \times \log n$

$\log n$ → $\frac{n}{2} \times \log n$

$$\approx O(n \log n)$$

Recurrence Relation

Recursion :- function calling itself is called recursion.

$$\lfloor 5 = 5 \times \lfloor 4$$
$$= 5 \times 4 \times \lfloor 3$$
$$= 5 \times 4 \times 3 \times \lfloor 2$$
$$= 5 \times 4 \times 3 \times 2 \times \lfloor 1$$
$$= 5 \times 4 \times 3 \times 2 \times 1 = 120$$

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

$\downarrow$

$O(n)$

```
int fact(n) → T(n)
{
    if (n==1)
        return 1;        } O(1)
    else
        return n * fact(n-1);
}
```

n : $O(1)$; $fact(n-1)$: $T(n-1)$

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

$\downarrow$

$O(n)$

$f(n) + f(n-1) + f(n-2) + \_ \_ \_ \_ + f(1)$

$\downarrow \quad \downarrow \quad \downarrow \quad \_ \_ \_ \_ \downarrow$

$1 + 1 + 1 + \_ \_ \_ \_ + 1 = n \approx O(n)$

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

- 1) Back substitution
- 2) Tree Method
- 3) Masters Theorem ⭐

Back Substitution

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

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

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

$n \rightarrow n-1 \rightarrow n-2 \rightarrow \vdots \rightarrow 1$

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

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

$= T(n-3) + 3$

$\vdots$

$= T(n-K) + K$

$= T(1) + n - 1$

$= 1 + n - 1 = n$

$= \underline{\underline{O(n)}}$

$n - K = 1$

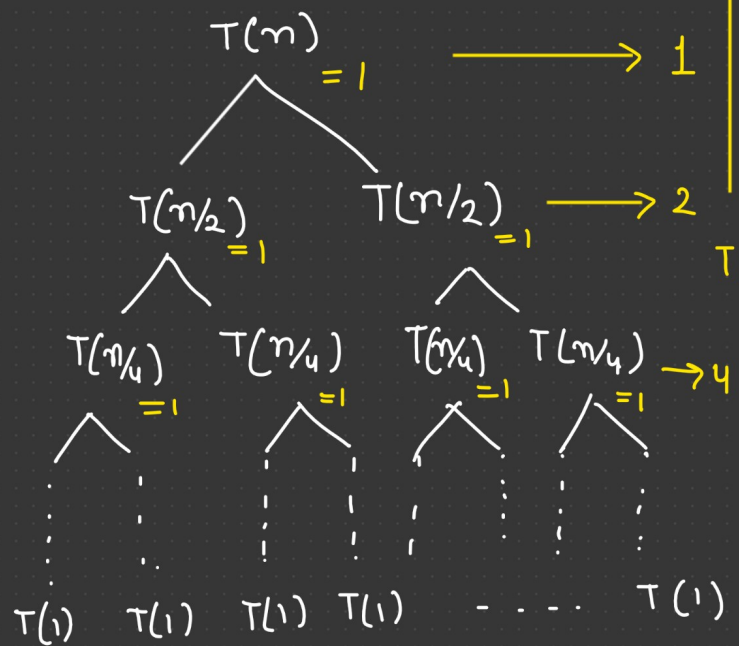
$K = n - 1$

Tree Method

$T(n) = 2T(n/2) + 1, \quad n > 1$

$T(1) = 1 \quad , n = 1$

```
int fun(n)
{
    if (n == 1)
        return 1;
    else
        return fun(n/2) + fun(n/2);
}
```



$T(n/2) = 2T(n/4) + 1$

$n \to n/2 \to n/4 \to n/8 \to \vdots \to 1$

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

$$1 + 2 + 4 + \ldots\ldots\ldots + 2^K$$

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

$$\Rightarrow n = 2^K$$

Sum of Geometric Progression :-

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

a = first term

n = no. of terms

r = common ratio

$$r = \frac{2}{1} = \frac{4}{2} = 2$$

$$n = K + 1$$

$$a = 1$$

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

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

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

$$= 2n - 1$$

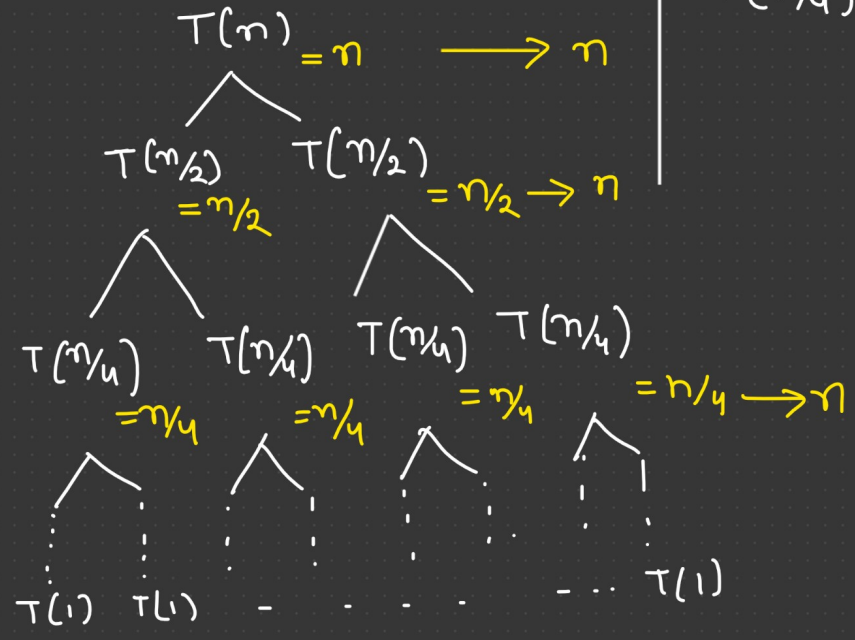
$$= O(n)$$

$$T(n) = 2T(n/2) + n \quad , n > 1$$

$$T(1) = 1 \quad , n = 1$$

$$T(n/2) = 2T(n/4) + n/2$$

$$T(n/4) = 2T(n/8) + n/4$$



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

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

$= n \times \text{no. of levels}.$

$= n \times (\log_2 n + 1)$

$= n \log n + n = O(n \log n)$

no. of level $= K + 1$

$\because n = 2^K$

$\log n = \log 2^K$

$\log n = K \log 2$

$K = \frac{\log n}{\log 2}$

$K = \log_2 n$

Masters Theorem

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

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta\left(n^k \log^p n\right)$$

Ex:- $T(n) = 2T(\frac{n}{2}) + n$

$a = 2, \; b = 2, \; K = 1, \; p = 0$

the form $T(n) = aT(\frac{n}{b}) + \Theta(n^k log^p n)$, where $a \geq 1, b > 1, k \geq 0$ 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 $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

$n^{\log_{4/3}^2}$

$a = 2$

$b = 2$

$k = 1$

$P = 0$

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

Case 2 a:-

$= \theta(n \log n)$

$\log_2 2 = \frac{\log 2}{\log 2} = 1$

Ex:- $T(n) = 2T(n/2) + 1$

$a = 2, \quad b = 2, \quad K = 0, P = 0$

$2 > 2^0 \Rightarrow a > b^k$

Case 1 :-

$\theta = (n^{\log_b^a}) = n^{\log_2^2} = \theta(n)$

1) $T(n) = 3T(n/2) + n^2$ $\quad a=3,\ b=2,\ k=2,\ p=0$

$3 < 2^2 \Rightarrow n^k \log^p n = \underline{\theta(n^2)}$

---

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

$4 = 2^2 \Rightarrow \theta(n^{\log_2 4} \log n)$

$\Rightarrow \log_2 4 = \frac{\log 4}{\log 2} = \frac{\log 2^2}{\log 2} = 2\frac{\log 2}{\log 2} = 2 \Rightarrow \underline{\theta(n^2 \log n)}$

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

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

$1 < 2^2 \Rightarrow \theta(n^k \log^p n) = \underline{\underline{\theta(n^2)}}$

4) $T(n) = 2^n T(n/2) + n^n$ ✗ Not possible

5) $T(n) = 16T(n/2) + n = \theta(n^4)$

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

$a=\sqrt{2},\ b=2,\ k=0,\ p=1$

$\sqrt{2} > 2^0$

$n^{\log_2 \sqrt{2}} = n^{1/2}$

$\frac{\log \sqrt{2}}{\log 2} = \frac{1}{2}\frac{\log 2}{\log 2}$

7) $T(n) = 2T(n/2) + n/\log n = \theta(n \log\log n)$

8) $T(n) = \sqrt{2}T(n/2) + \log n = \theta(\sqrt{n})$

Master Theorem for Subtract & Conquer recurrence:-

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

$T(1) = 1 \quad , n = 1$

$a = 1, b = 1, f(n) = 1, K = 0$

$O(n^{K+1}) = O(n)$

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

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

1) $T(n) = 3T(n-1)$, if $n > 0$

$T(1) = 1$, if , $n \le 0$

$= O(3^n)$

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

$O(n^K a^{n/b}) = O(n^0 \cdot 3^n)$

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

$a = 2, b = 1, K = 2$

$O(n^K \cdot a^{n/b}) = O(n^2 \cdot 2^n)$

Find Running Time

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

$S = 1, 3, 6, 10, \ldots\ldots\ldots, n$

$1+2 \quad 1+2+3 \quad 1+2+3+4 \quad 1+2+3+\ldots+k$

$i = 1, 2, 3, 4$

$1 + 2 + 3 + \ldots + k = n$

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

$k^2 + k = 2n \Rightarrow k^2 = n$

$k = \sqrt{n}$

$O(\sqrt{n})$

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

}
```

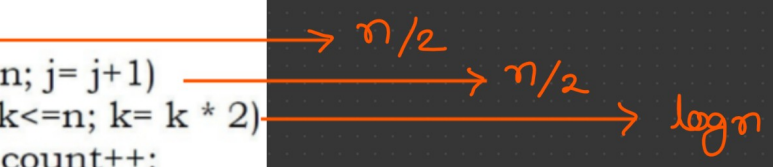
$i^2 = n$

$1, 2, 3, \ldots\ldots, k$

$k^2 = n$

$k = \sqrt{n} \Rightarrow O(\sqrt{n})$

```
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++;
}
```



$$\Rightarrow \frac{n}{2} \times \frac{n}{2} \times \log n \Rightarrow \underline{O(n^2 \log n)}$$