Time and Space Complexity

Time Complexity of an algorithm is the representation of the amount of time required by the algorithm to execute to completion

★ kisi bhi algorithm ko pura-puri tarike se Execute hone me jo bhi time chaiye hoga! Uss time ko represent krey toh use TIME COMPLEXITIY Bolengy ALGORITHM ki!

★ Denotion $\boxed{t(n)}$ → Time taken Input of size (n).

- Agar Hume kisi ALGORITHM ki EFFICIENCY CHECK krna ho toh 2 Factors dekhengy:

  ↳ kitna ELIGIBLE ya kitna NIPUNT hai

① Time Factor

The time is calculated or measured by counting the no. of key operations.
- such as comparison in sorting algorithm.

② Space Factor

The space is calculated or measured by counting the maximum memory space required by algorithm.

Space Complexity of an algorithm represents the amt. of memory space the algorithm needed in its life cycle.

Time Complexity

→ kisi bhi PROBLEM ko solve krne ke Humare pass n SOLUTION Ho sakty hai!

→ kisi bhi PROGRAM ko solve krne ke n NO. OF SOLUTION Ho sakty hai!

QUES: To find the Square of a number?

Solution 1:

```
int n=5;
for(int i=1; i<n; i++)
{
    n = n + 5;
}
return n
```

→ Time yaha (n) pe depend kr raha hai! Jitni (n) ki value hogi utni baar time badega!

→ Yaha loop (n) no. of times chalega!

→ Jesey-Jesey (n) ki value badegi wesey-wesey time bhi badega

∴ Time complexity will be (n) atleast

★ TIME COMPLEXITY hum estimate krty hai by calculating the no. of elementary steps performed by an algorithm to finish execution.

SOLUTION 2 :

```
int n=5;
return n*n;
```

→ Yaha Time Complexity constant hogi kuki Yaha time (n) ki value per depend nahi krta!

TYPES OF NOTATIONS FOR TIME COMPLEXITY

- **Big Oh** : denotes "fewer than or same as" <expression> iterations
- **Big Omega** : denotes "more than or same as" <expression> iterations
- **Big Theta** : denotes "the same as" <expression> iterations
- **Little Oh** : denotes "fewer than" <expression> iterations
- **Little Omega** : denotes "more than" <expression> iterations

There are different types of TIME COMPLEXITIES used:

- Constant Time → $O(1)$
- Linear Time → $O(n)$
- Logarithmic Time → $O(\log n)$
- Quadratic Time → $O(n^2)$
- Cubic Time → $O(n^3)$

And many more complex notations like exponential time, ~~Quadratic~~ Quasilinear time, Factorial time, etc

☆ O(1) [CONSTANT TIME] → Time Taken will not depend on Input size

An Algorithm is said to have constant time of Order $\boxed{O(1)}$ when it is not dependent on the input size (n).

Irrespective of the input size (n), the runtime will always be same.

Time Taken For Small Input	=	Time Taken For Large Input

∴ Time (2 * 5) = Time (200 * 500)

↓

WHY ?

∴ 2, 5, 200, 500 → All will take 32 bits → Time Consumed is Same

↓

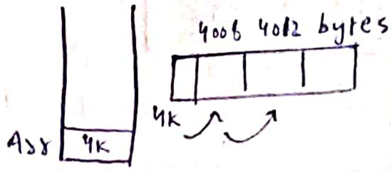
Becoz

↓

- CPU ko sirf 32 bits process krni hai.

- Essa iss liye becoz 2, 5, 200, 500 all are Integers
- Integer size in Java = 4 Bytes = 32 bits

Type : O(1)
Initial Eq : T(n) = k
Final Eq : T(n) = k
Exampl : + - * / % ^ / &



Array me kisi bhi Position par set / get krna hai toh time equal hi lagega har position ke liye!

☆ O(n) [LINEAR TIME]

Agar input ka size badega Time consumed bhi badega!

- An algorithm is said to have LINEAR time complexity when the running time increases linearly with the size of input.

$$n_1 = 2n_2$$
$$T(n_1) = 2T(n_2)$$

- when the function involves checking the values in input data, such function has the Time Complexity of order O(n).
  - ↳ Agar data ki har values par agar koi function chalta hai toh uss function ki Time Complexity ho jati hai of Order O(n).

Example : LINEAR SEARCH

Ques : Find element in an Array.

Solution

```
public static ~~int~~ int linearSearch (int[] arr, int x)
{   for (int i=0; i<arr.length; i++)
    {
        if (arr[i] = x) {
            return i;
        }
    }
    return -1;
}
```

Time input ki Length pe depend krega!

Yeh comparison check (i) pe depend krta hai. Jitni Bār yeh check chalega utni baar time will increase!

worst Case — no. not in array

∴ For 1 comparison = k time

				n

k → k → k → k → k, nk times

$$\therefore = k + k + \dots + nk$$
$$= (k)(n)$$
$$= nk$$

$$\boxed{T(n) \propto n}$$

Type : O(n)
Initial Eq : T(n) = kn
Final Eq : T(n) ∝ n
Example : Linear Search

0	1	2	3
10	20	30	40

index

∴ x = 50

∴ comparison check : arr[index] = 50

$$\therefore T(n) = k + T(\cancel{n}-1)$$
$$T(\cancel{n-1}) = k + T(\cancel{n-2})$$
$$T(\cancel{n-2}) = k + T(\cancel{n-3})$$
$$\vdots$$
$$T(\cancel{1}) = k$$

After x equations

☆ ∴ T(1) ≃ T(n-3)

$$\therefore T(n) = xk$$

$$\therefore T(n) = nk$$

$$\therefore \boxed{T(n) \propto n}$$

∴ T(n) = 4K

n (n-1) (n-2) (n-3) ---- 1

Yeh AP hai!

∴ 5 4 3 2 1, ∴ x = n

5 equations → Har baar (n) ke saath (1) subtract hoga!

☆ O(log n) [LOGARITHMIC TIME]

- An algorithm is said to be logarithmic time complexity when it reduces the size of the input data in each stip. This means → That the number of operations is not same as the input size.
- The no. of operations get reduced as the input size increases.
- The Algorithms with logarithmic Time Complexity are found in Binary Tree or Binary Search Function
  ↓
  This involves the search of a given value in an array by splitting the array into two and starting the search in one split → This ensures that the operation is not done on every element of the data

Example : BINARY SEARCH

$x = 70$

low4 → ↙ high4 mid2 ↓

10	20	30	40	50	60	70	80	90	100
0	1	2	3	4	5	6	7	8	9

low1 → 0; mid1 → 4; low2, low3 → 5; high3 → 6; high1, high2 → 9

① $mid = \frac{0+9}{2} = 4.5 = 4$ (base)

$50 < 70$
arr[mid] < 70
↳ then,
low = mid + 1

] Hum (n) element search krey hai
1 comparison = k time
Aab bachey huay elements ke liye $T(\frac{n}{2})$ lagega!

$$T(n) = K + T(\cancel{\tfrac{n}{2}})$$

② $mid = \frac{5+9}{2} = 7$

arr[mid] = 80
$70 < 80$
$70 <$ arr[mid]
↳ then,
high = mid - 1

] Hum (n/2) elements search kr rahey hai
1 comparison = k time
Bachey huay elements ke liye $T(\frac{n}{4})$

$$T(\cancel{\tfrac{n}{2}}) = K + T(\cancel{\tfrac{n}{4}})$$

③ $mid = \frac{5+6}{2} = 5$

arr[mid] = 60
$60 < 70$ ↳ ⓧ
∴ low = mid + 1

] Hum (n/4) elements search kr rahy hai
1 comparison = K time
Bachey huay elements ke liye $T(\frac{n}{8})$

$$T(\cancel{\tfrac{n}{4}}) = K + T(\cancel{\tfrac{n}{8}})$$

④ $mid = \frac{6+6}{2} = 6$

arr[mid] = 70 = x

] $\frac{n}{8} = \frac{10}{8} \approx 1 = 1$
1 element ko compare krne keliye k time

$$T(\cancel{1}) = K$$

$$T(n) = 4K$$

How many equations will be formed ?

GP : $n \quad \frac{n}{2} \quad \frac{n}{4} \quad \dots \quad 1$ ↳ x^{th} term

∴ $a = n$ → first element
$r = \frac{1}{2}$ → gap / move

∴ find a_x ?
↳ last term (x^{th} term) of GP

∴ How to calculate a_x (x^{th} term) in GP.

∴ $a = n$
$r = \frac{1}{2}$
$a_x = 1$

$\boxed{a_x = a r^{x-1}} \Rightarrow \boxed{1 = n\left(\frac{1}{2}\right)^{x-1}}$

∴ $\boxed{1 = \frac{n\ (1)^{x-1}}{(2)^{x-1}}}$

∴ $\boxed{1 = \frac{n}{2^{x-1}}} \Rightarrow \boxed{(2)^{x-1} = n}$

⇓ Taking $\log_2$ both side

$\boxed{\log_2 \left((2)^{x-1}\right) = \log_2 (n)}$

$\boxed{\log_2 (2)^{x-1} = \log_2 (n)}$

⇓

$\boxed{(x-1) \log_2 (2)^1 = \log_2 (n)} \Rightarrow \boxed{(x-1) \underbrace{\log_2 2}_{1} = \log_2 n} \Rightarrow \boxed{(x-1)1 = \log_2 n}$

$\boxed{(x-1) = \log_2 n}$

This is our equation ← $\boxed{x = \log_2 n + 1}$

∴ Binary Search Time Complexity ⇒ $\boxed{T(n) = x * K}$

∴ Here $x = \log_2 n + 1$

⇒ $\boxed{T(n) = (\log_2 n + 1) \cdot K} \Rightarrow \boxed{T(n) = K \log_2 n + K}$

∴ ☆ $\boxed{T(n) \propto \log n}$

Type : $O(\log n)$
Initial Eq : $T(n) = K + T(\frac{n}{2})$
Final Eq : $T(n) \propto \log n$
Example : Binary Search
Binary Tree

Eg 1 : $n_1 = 4n_2$
∴ $T(n_1) \propto \log_2 (4n_2)$ {∴ $T(n) \propto \log n$}
$T(n_1) \propto \log_2 (2^2 n_2)$
$\boxed{T(n_1) = 2\,T(n_2)}$

Eg 2 : $n_1 = 1024\, n_2 = 2^{10} n_2$
∴ $T(n_1) \propto \log_2 (2^{10} n_2)$
$\boxed{T(n_1) = 10\, T(n_2)}$

Eg 3: $n_1 = 10^9 (n_2)$ → Isko 2 ki power me express kro!

$\therefore T(n_1) = T(10^9 n_2)$

$\{\therefore 10^9 = 2^{30}\}$

$T(n_1) = T(2^{30} n_2)$

$\therefore T(n_1) \propto \log_2(2^{30} n_2)$

$$\boxed{T(n_1) = 30\,T(n_2)}$$

Input : 10^9 times

Time : 30 time bada!

* NOTE TO REMEMBER *

- Jis bhi ALGORITHM ki Time complexity $O(\log n)$ hai toh yeh accha Algorithm hai → WHY?
- Becoz input Bahut Jyada Badne pe bhi Time Bahut Jyada nahi badta!

$\therefore T(n_1) \propto \log(n_1)$ $\qquad$ $\therefore T(n_2) \propto \log(n_2)$

$T(n_1) = K \log(n_1) - (1)$ $\qquad$ $T(n_2) = K \log(n_2) - (2)$

Divide $\frac{(1)}{(2)}$:

$$\frac{T(n_1) = \cancel{K} \log(n_1)}{T(n_2) = \cancel{K} \log(n_2)} \Rightarrow \frac{T(n_1)}{T(n_2)} = \frac{\log(n_1)}{\log(n_2)}$$

$\therefore \{n_1 = 1024\, n_2\} \therefore \{1024 = 2^{10}\}$

$$\frac{T(n_1)}{T(n_2)} = \frac{\log(1024\, n_2)}{\log(n_2)} \Rightarrow \frac{T(n_1)}{T(n_2)} = \frac{\log(2^{10} n_2)}{\log(n_2)}$$

$$\therefore \frac{T(n_1)}{T(n_2)} = 10\, \frac{\cancel{\log(n_2)}}{\cancel{\log(n_2)}}$$

$$\therefore \boxed{T(n_1) = 10\, T(n_2)}$$

IMP

FOR LOOP

→ $O(\log_2(n))$ →

```
for (int i=n; i>=1; i=i/2){
   //xyz
}
```

★ TRICK LOOP ★

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

$\boxed{\therefore T(n) \propto \log(n)}$ → $\boxed{O(\log n)}$

→ $O(n)$ →

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

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

$\boxed{\therefore T(n) \propto n}$

```
for(int i=1; i>=n; i--){
}
```

// $T(n) = Kn$

$\boxed{\therefore T(n) \propto n}$

→ $\boxed{O(n)}$

Ques: Find the Time Complexity of written programs of Power of no.

Solution: $x^n = ?$

Program 1

```
p s v power(int x, int n)
{ int (n==0){ //Base case
    return 1;
  }      → x ki power (n-1)
  int xpnm1 = power(x,n-1); //Recursive Call (Faith) }  t(n-1)
  int xpn = xpnm1 * x; }  K constant time  //Self work
  return xpn;
         ↳ x ki power n
}
```

$t(n) = K + t(n-1)$

$\Downarrow$

$\therefore \boxed{t(n) \propto n}$

Program 2

```
p s v power(int x, int n){
  { if(n==0){ //Base Case
      return 1;
    }
    int xpnb2 = power(x, n/2); //Recursive (Faith) Call   t(n/2)
    int xpn = xpnb2 * xpnb2;
      if(n%2==1) //Self work          K constant time
        { xpn = xpn*x;
        }
    return xpn;
  }
```

$t(n) = K + t\left(\frac{n}{2}\right)$

$\Downarrow$

$\therefore \boxed{t(n) \propto \log n}$

Program 3

```
p s v power(int x, int n){
  { if(n==0){
      return 1;
    }
    if (n%2==0){
    return power(x,n/2)*power(x,n/2);
                t(n/2)           t(n/2)
    } else {
    return power(x,n/2)*power(x,n/2)*x;
                t(n/2)           t(n/2)   K
    }
}
```

$\therefore t(n) = t\left(\frac{n}{2}\right) + t\left(\frac{n}{2}\right) + K$

$\therefore \boxed{t(n) = K + 2t\left(\frac{n}{2}\right)}$

$\times 2^0 \rightarrow 2^0 t(n) = 2^0 K + 2^1 \cancel{t(n/2)}$

$\times 2^1 \rightarrow 2^1 \cancel{t\left(\frac{n}{2}\right)} = 2^1 K + 2^2 \cancel{t(n/4)}$

$\times 2^2 \rightarrow \cancel{2^2 t\left(\frac{n}{4}\right)} = 2^2 K + 2^3 \cancel{t(n/8)}$

$\vdots$

$\times 2^{x-1} \rightarrow \cancel{2^{x-1} t(1)} = 2^{x-1} K$

$t(n) = 2^0 K + 2^1 K + 2^2 K + 2^{x-1} K$

$\therefore GP = \frac{a(r^n - 1)}{r-1} \Rightarrow t(n) = \frac{K(2^x - 1)}{2-1}$

$t(n) = \frac{K(2^x - 1)}{1} = K(2^x - 1) = 2^x K - K \Rightarrow 2^{\log_2 n + 1} K - K = K(2^{\log_2 n}) \cdot (2)^1 - K$

$= 2Kn - K$

$= K(2n - 1)$

$\therefore \boxed{x = \log_2 n + 1}$

$\therefore \boxed{t(n) \propto n}$