⇒ Time & Space Complexity

↳ Relation between Input size and Running Time

```
main() {
    Syso("Hello");   1
                     ↳ constant time
    Syso("Hello");   2
}
```

1 sec = 10^9 operations

```
main() {                          n=1, 2, 3, ..., 10^6
    for(int i=0; i<n; i++) { }    n operation
        Syso("Hi");
    }
}
```

Time:- how many operations we are performing in a program.

↳ Types of operations

	input	output operation
↳ linear operation ⇒	n	$\longrightarrow n$
↳ quadratic operation ⇒	n	$\longrightarrow n^2$
↳ cubic operations ⇒	n	$\longrightarrow n^3$
↳ logarithmic operation ⇒	n	$\longrightarrow \log(n)$
↳ constant operations ⇒	n	$\longrightarrow 1$

Ex 1

```
Public static void main () {
    Scanner scn = new Scanner ( );
    int n = scn.nextInt();
    for ( int i=0; i<n; i++) {
        Syso ("hello");
    }
}
```

n times

	input	operation
ipput	n ⇒ 1 ⟶	1
	n ⇒ 10 ⟶	10
	n ⇒ 90 ⟶	90
	n ⇒ 100000 ⟶	100000

relation

input ∝ n

↳ O(n)

→ Complexity (Time)

1) Best Case :- when least no. of operations has to be done

2) Average Case :- when average no. of operations has to be done

3) Worst Case :- when most no. of operations has to be done

Ques You have an array and you have to find "1" in it

2	2	5	6	8	3	4	1
0	1	2	3	4	5	6	7

worst

n operations

```
for ( int i=0; i<n; i++){
    if ( arr[i] == 1 ) {
        Syso (i);
    } break;
}
```

1 sec = 10^9

1 → 7

n = 7 70000000

Ex 2

```
Public static void main () {
    Scanner scn = new Scanner ( );
    int n = scn.nextInt();
    for ( int i=0; i<n; i++) {
        for ( int j=0; j<m; j++) {
            Syso("Hello");
        }
    }
}
```

n = 1 → i=0, j=0 } 1 time

n = 2

i=0, j=0, 1 }
i=1, j=0, 1 } 4 times

n = 3

{ i=0, j=0,1,2
  i=1, j=0,1,2
  i=2, j=0,1,2 } 9 times

n = 4 → 16 times

input	input	operation
	n = 1	1
	n = 10	100
	n = 90	8100
	n = 100000 (10^5)	10^{10}

relation

input $\propto n^2$ → $O(n^2)$

input $\propto n \times m$ → $O(n \times m)$

Ex 3

```
Public static void main () {
    Scanner scn = new Scanner ( );
    int n = scn.nextInt();

    for ( int i=0; i<n; i++) {
        Syso ("Hello");
    }

    for ( int j=0; j<m; j++) {
        Syso("Hello");
    }
}
```

n operations

m operations

Input $\propto$ n+m

e.g., n = 2
m = 3
(5)

e.g., n = 5 ✓✓
m = 500000 ✓✓

O(m)

Capital O notation

↳ O(n+m)

Note:-

when we add complexities (Time) then smaller value will get neglected (bez larger value have more significance)

Compare

	$O(n)$	$O(n^2)$	$O(n^3)$
$n = 1$	1	1	1
$n = 2$	2	4	8
$n = 3$	3	9	27
⋮	⋮	⋮	⋮
$n = 10^5$	10^5	10^{10}	10^{15}
	1 sec		10 min

1. Constant Time Complexity - O(1) fastest

2. Logarithmic Time Complexity - O(log n) fast

3. Linear Time Complexity - O(n) average

4. O(n log n) Time Complexity bad

5. Quadratic Time Complexity - O(n2) very bad

6) $O(n^3)$ ⟶ very very bad

```
Public static void main (){
    Scanner scn = new Scanner ();
    int n = scn.nextInt(); //1, 5, 50000
    Syso ("Hello");
}
```

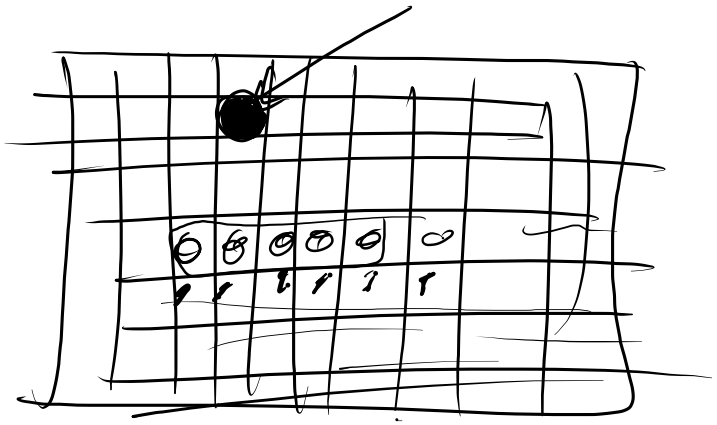
$$O(1) > O(\log n) > O(n) > O(n \log n) > O(n^2) > O(n^3)$$

⇒ Space Complexity // no. of variables uses

```
Public static void main () {
    Scanner scn = new Scanner ( );
    int n = scn.nextInt();
    for ( int i=0; i<n; i++) {
        Syso ("hello");
    }
}
```

← 1, 2, 1000, 10^5

O(1) //constant

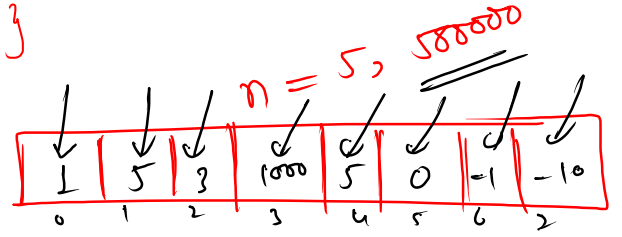


```
Public static void main () {
    Scanner scn = new Scanner ( );
    int n = scn.nextInt();
    int [] arr = new int [n];

    for ( int i=0; i<n; i++) {
        Syso ( arr[i]);
    }
}
```

← 5, 5000

O(n)



Space operation ∝ n (input)