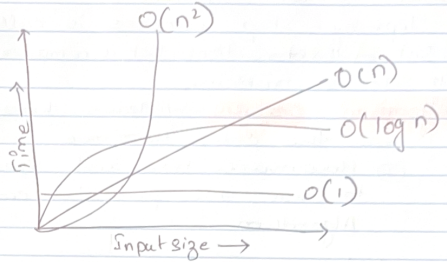
Big O Notation.



Time Complexities:-

1. O(1) or Constant time.
2. O(log(n)) or Logarithmic time.
3. O(n) or Linear time.
4. O(n²) or Exponential time
5. O(n!) or Factorial time.

① O(1) → Constant time. The Algorithm takes the same amount of time regardless of the input size.

② O(log n) → Logarithmic time. The algorithm time grows logarithmically as the input size increases, often due to dividing the problem in half.

③ O(n) → Linear time. The time grows linearly with the input size.

④ O(n log n) → Log-Linear time. The time grows in proportion to 'n' times 'Log n', often seen in efficient sorting

algorithms.

5) $O(n^2)$ → Quadratic time. The time grows quadratically with the input size, common in algorithms with nested loops.

A Comparison of Some Common Functions:-

	Function	Name		Function	Name
①	Any constant	Constant	⑦	n^2	Quadratic
②	$\log n$	Logarithmic	⑧	n^3	Cubic
③	$\log^2 n$	Log-square	⑨	n^4	Quartic
④	$\sqrt{n}$	Root-n	⑩	2^n	Exponential
⑤	~~Linear~~ n	Linear	⑪	e^n	Exponential
⑥	$n \log n$	Linearithmic	⑫	$n!$	n-Factorial.

Formulae

List of handy formulas which can be helpful when calculating the time complexity of an algorithm.

	Summation	Equation.
①	$\left(\sum_{i=1}^{n} c\right) = c+c+c+\ldots+c$	cn
②	$\left(\sum_{i=1}^{n} i\right) = 1+2+3+\ldots+n$	$\frac{n(n+1)}{2}$
③	$\left(\sum_{i=1}^{n} i^2\right) = 1+4+9+\ldots+n^2$	$\frac{n(n+1)(2n+1)}{6}$
④	$\left(\sum_{i=0}^{n} r^i\right) = r^0+r^1+r^2+\ldots+r^n$	$\frac{(r^{n+1}-1)}{r-1}$

General Tips.

1. Every time a list or Array gets iterated over C# length times, it is most likely in O(n) time.
2. When you see a problem where the number of elements in the problem space gets halved each time, it will most probably be in O(log n) runtime.
3. Whenever you have a single nested loop, the problem is most likely in quadratic (n^2) time.

* Example. → Running Time Complexity. (for-loop).

```
class Sum
{
    int n=10;          -0
    int sum=0;         -0
    for (int i=0; i<n; i++)      1 | 3(n+1) | 3n
    {
        sum += 1;      -> 3n
    }
    System.out.println(sum);     1
}
```

$= 8 + 9n$ → Run-Time complexity.

Output:- 10. ↓

Detailed Step-by-Step process for Run-time Complexity:-

`int n=10;` Running time Complexity = 1
`int Sum=0;` " " = 1
`int i=0;` " " = 1
`i<n` → The test operation i<n comprises of three steps
① Access the value of i. 1
② Access the value of n. 1
③ Compare the values of i and n. 1 } 3

→ on careful analysis you will notice that the ~~for~~ statement is executed for each value of i.
* Therefore, the statement is executed n+1 times.
* Therefore the total statement occurs for 3(n+1) times

`Sum += i;` — 3 (Accessing, adding, updating).
↓ This statement is executed n times because of for loop.
Therefore, in total this statement accounts for 3n primitive operations
`i++;` - 3 (Accessing, adding, updating) occurs n time → 3n
`S.O.P(Sum);` — 2 (Accessing, Displaying)

Common Complexity Scenarios:-

1. Simple for-loop with an increment of size 1.

```
for(int x=0; x<n; x++){
    // Statement(s) take constant time
}
```

Explanation:- ① The initilization x=0; runs once - 1

② The comparision Statement x<n ; runs n+1 times.

③ The constant time statement inthe loop itself runs n times. cn.

④ The loop increment statement runs x++, n times. n times.

∴ 1+(n+1) + cn + n ⇒ 2n + cn + 2

∴ n(2+c) + 2

∴ O(n) (by dropping lower order terms)

2. For-Loop with increments of k.

```
for(int x=0; x<n; x+=k){
    // Statement(s) take constant time
}
```

Running Time Complexity

$2 + n\left(\frac{2+c}{k}\right) = O(n)$

Explanation:- ① The initialization x=0; runs once.

② Then x is incremented by k until it reaches n.

[0, k, 2k, 3k, ... (nk) < n] → $\frac{n}{k} + 1$

③ Hence, the incrementation part x+=k for the loop takes $\frac{n}{k}$ times → $\frac{n}{k}$ times.

④ Constant c × $\frac{n}{k}$ times → $c \cdot \frac{n}{k}$ times

$= 1 + \frac{n}{K} + 1 + \frac{n}{K} + c\frac{n}{K} \Rightarrow 2\frac{n}{K} + \frac{cn}{K} + 2$

$\therefore \frac{n}{K}(2+c) + 2$ [∵ Here K is constant].

$\therefore O(n)$.

3. Simple Nested For Loop.

```
for(int i=0; i<n; i++){
    for(int j=0; j<m; j++){
        // Statement(s) that take(s) constant time
    }
}
```

Running Time Complexity.

$nm(2+c) + 2 + 4n = O(nm)$

Explanation:- The inner loop is a simple for loop that takes $(2m+2) + cm$ time and the outer loop runs in n times so it takes $n((2m+2) + cm)$ time. Additionally, the initialization, increment and test for the outer loop take $2n+2$ time so in total, the running time is.

$n((2m+2) + cm) + 2n + 2 \Rightarrow 2nm + 4n + cnm + 2$

$\Rightarrow nm(2+c) + 4n + 2$, which is $O(nm)$.

4. Nested For-Loop with dependent variables.

```
for(int i=0; i<n; i++){
    for(int j=0; j<i*i; j++){
        // Statement(s) that take(s) constant time
    }
}
```

Running Time Complexity:- $O(n^2)$

5. Nested for-Loop with Index Modification.

```
for(int i=0; i<n; i++){
    i*=2;
    for( int j=0; j<i; j++){
    // Statements that take constant time.
    }
}
```

Running Time Complexity :- $O(n)$

6. Loop with log n time complexity.

```
i = // constant
n = // constant
k = // constant
while ( i<n){
    i* = k;
    // Statement(s) that take(s) constant time
}
```

Running Time Complexity :- $\log_k(n) = O(\log_k(n))$

8. Big(O) of Nested loop with Addition.

```
class Nestedloop{
public static void main (String[]args){
    int n=10;                          1
    int sum = 0;                       1
    double pie = 3.14;                 1

for(int var=0; var<n; var = var+3){    n/3 + 1
    System.out.println ("pie" + pie);  n/3
    for (int j=0; j<n; j = j+2){       n/3 (n/2 + 1)
```

Statement	no. of Executions
int n=10;	1
int sum=0;	1
double pie=3.14;	1
for(int var=0; var<n; var = var+3){	n/3 + 1
System.out.println ("pie" + pie);	n/3
for (int j=0; j<n; j = j+2){	n/3 (n/2 + 1)
// Statements that take constant time.	n/3 * n/2
}	n/3 * n/2
}	n/3
}	1

```
        Sum++; n/3 * n/2
    System.out.println("Sum: " + Sum);
    n/3 * n/2
}
}
}
```

Running Time Complexity :-

$$1+1+1+1+\frac{n}{3}+1+\frac{n}{3}+\frac{n}{3}+\frac{n}{3}\times\left(\frac{n}{2}+1\right)+\left(\frac{n}{3}\times\frac{n}{2}\right)+\left(\frac{n}{3}\times\frac{n}{2}\right)+\left(\frac{n}{3}\times\frac{n}{2}\right)$$

$$= \frac{15+5n+2n^2}{3}$$

Big (O) Time complexity : $O(n^2)$