Time complexity

- It is the time taken by an algorithm to run.
- As a function of length of the input

Big o Notations(Max bound)   Theta(avg)   Omega(lower bound)

Constant time O(1) :   for(int i=0; i<10; i++) So here this loop runs only 10 times.

Linear time O(n) :   for(int i=0; i<n; i++) So here this loop runs only n times.

Quadratic time O(n^2):

```
for(int i=0; i<n; i++)
{
        for(int i=0; i<n; i++)
}
```

2 loops

Cun=bic time O(n^3):

```
for(int i=0; i<n; i++)
{
        for(int i=0; i<n; i++)
        {
                for(int i=0; i<n; i++)
        }
}
```

3 loops

O(n!)   - High time complexity
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1) - Low time complexity

Note:

Always ignore the constant and lower degree
For ex -

F(n) -> 2n^2 + 3n Answer - O(n^2)
F(n) -> n/4 Answer - O(n)

10^8 operation rule : Most of the modern machine can perform 10^8 operations per second

Constraints :

1 < n < 10^6

1 < n < 1000

- < [10..11] - O(n!), O(n^6)
- < [15..18] - O(2^n * n2)
- < 100 - O(n^4)
- < 400 - O(n^3)
- < 2000 - O(n^2 * log n)
- < 10^4 - O(n^2)
- < 10^6 - O(n log n)
- < 10^8 - O(n), O(log n)