Time Complexity

A user has written a code and B user has also written a code for the same problem; it can be any problem. We need to finish the code or we need it to run faster.

Ex: If we want to download a file from internet, we will use an application that will help us to download fast. We can do this using TC

Time Complexity, is a rough idea of the time a code will take to execute.

TC is an area in computer science, we use to predict time within which the program will give us an output or we will try to understand the time the program takes to run.

Ex:

INPUT:	OUTPUT:

```
i = 0
while i < 3:
    print (i)
    i += 1
```
	0
	1
	2

Until ‘i’ value is equal to three, the code will run. So, if while block is taking 1 sec for one line, then we are repeating it 3 times. So the TC will 3 sec which can be said as order of 3;

O (1) + O (1) + O (1) = O (3)

Let’s say,

```
i = 0
print (i)
print ("Hello")
if x  > 5:
    print ("hello")
```

since there is no loop in this code, the time complexity will be order of 1.

If a statement is getting repeated many times, then the order of 1 will also increase in the same proportion.

```
i = 0
while i < n:
   print(i)
```

If n = 10^5, the TC will be 10^5. So, we can say that this code has a time complexity in the order of 1.

We are interested in rate of growth of time with respect to the inputs taken during the program execution.

LOOP INCREASES YOUR TIME COMPLEXITY.

```
for i in range (100):

        print (i)
```

The time complexity for this will be 100, (0 – 100). If we put n in place of 100, it will be time complexity in the order of n. O(n)

`for i in range (0, n):`
`   print (i)`
	O (n)

`for i in range (0, 2*n):`
`   print (i)`
	O (2*n)

O (n) + O (2*n) = O (n + 2*n)

In a polynomial, $ax^n + bx^{n-1} + cx^{n-2}$ …. The power will be ‘n’. in the same way, for this program, O (n) in which the power of n is 1. So the time complexity for this program will be 1.

$2n^3 + 4n^2 + 5 \rightarrow \quad O(n^3)$

$4x^3 + 5x \rightarrow \quad O(n^3)$

$x\ (4x + 3) \rightarrow O(x^2)$

The highest power of n will be the Time Complexity of this equation.

```python
for i in range (0, n):
    print (i)

for j in range (0, n**3):
    print (j)
```

the time complexity for O (n^3) + O (n) will be O (n^3)

```python
for i in range (2):
    for j in range (2):
        print (i,j)
```

The above code will be running for 4 times. when I = 0, j will have 2 values and whn I = 1, then j will have 2 values. So, 2 X 2 = 4. So the time complexity of this program will be O (4)

```python
for i in range  (n):
    for j in range (n):
        print (i*j)
```

TC will be O (n**2)

I will have values from 0 to n-1. For each I, you have 0 – n-1 values of j. So, this code will run for O (n**2)

INPUT:

```python
i = 100
while  (i > 0):
    print (i)
    i = i // 5
```

OUTPUT:

```
100
20
4
```

Time complexity = O (3)

```
i = n
while i > 0:
    print (i)
    i = i // 5
```

In this, i = n for the first time **N/5** and it will become **$N/5^2$** for the second time and the third time it will become **$N/(5^3)$** until it come to **$N/5^N$.**

So, whenever we are diving something by K repetitively, and you are getting the above series, such series is called Logarithmic Series. So, the time complexity for this will be **$\log_5 N$ or can be written as O ($\log_5 N$)**

```
i = n
while i > 0:
    print (i)
    i = i // 2
```

$\mathbf{\frac{N}{2} + \frac{N}{2^2} + \frac{N}{2^3} + \ldots\ldots\ldots\ldots}$ the time complexity will be O ($\log_2 N$)

```
i = 0
while i < n:
    print (i)
    i = i * 2
```

the output for this program will be 2 4 6 8 16 32

So, the time complexity for this will be O ($\log_2 N$)

MCQ 1

```
for i in range (n):
    i = 0
    while i < n:
        print (i)
        i = i*2
```

ANS: O (n*log$_2$ n)

MCQ 2

```
for i in range (n**2):
    for j in range (n):
        if j == 0
        break
    print (i)
```

ANS: O (n^2)

MCQ 3:

```
i = 0
while i < n
    while i < n/2
        i += 1
```

ANS: O (n)