TIME COMPLEXITY

- The rate at which the time required to run a code, changes with respect to the input size, is considered the time complexity. **Basically, the time complexity of a particular code depends on the given input size, not on the machine used to run the code.**

- Time complexity does not refer to the time taken by the machine to execute a particular code. We should not compare the two different codes on the basis of the time taken by a machine as the time is dependent on it.

How we will represent the time complexity of any code:

To represent the time complexity, we generally use the Big O notation. The Big O notation looks like the following: **O( )** , Where time taken by the code is specified inside the parenthesis.

Here come the three rules, that we are going to follow while calculating the time complexity:

1. We will always ***calculate the time complexity for the worst-case scenario***.
2. We will ***avoid including the constant terms***.
3. We will also ***avoid the lower values.***

Let's discuss all of them in detail.

calculate the time complexity for the worst-case scenario.

Best Case: This term refers to the case where the code takes the least amount of time to get executed. For example, if the mark is 10(i.e. < 25), only the first line will be executed and the rest of the lines will be skipped. So, the least amount of steps i.e. only 2 steps are required in this case. This is an example of the best case.

Worst Case: This term refers to the case where the code takes the maximum amount of time to get executed. For example, if the mark is 70(i.e. > 65), the last line will be executed after checking all the above conditions. So, the maximum amount of steps i.e. 4 steps are required in this case. This is an example of the worst case.

Average Case: This term is pretty self-explanatory. This is basically the case between the best and the worst.

Now, as we always want our system to serve the maximum number of clients, we need to calculate the time complexity for the worst-case scenario. With this, we can actually judge the robustness of any code or any system.

Avoid including the constant terms

Let's understand this rule considering the time complexity: $O(4N^3 + 3N^2 + 8)$. Now, if we consider the value of N as 10^5 the time complexity will be like this: $O(4*10^{15} + 3*10^{10} + 8)$. In this case, the constant term 8 is very less significant in terms of changing the time complexity with different values of N. That is why we should avoid the constant terms while calculating the time complexity.

Avoid the lower values

Now, in the previous example, the given time complexity is $O(4N^3 + 3N^2 + 8)$ and we have reduced it to $O(4N^3 + 3N^2)$. Here, we can clearly observe that if the value of N is a large number, the second term i.e. $3N^2$ will also be a less significant term. For example, if the value of N is 10^5 then the term $3*10^{10}$ becomes less significant with respect to $4*10^{15}$. So, we can also avoid the lower values and the final time complexity will be $O(4N^3)$

NOTE *- A point to remember is that we can actually ignore the constant coefficients as well. For example, considering the time complexity $O(4N^3)$ as $O(N^3)$ is also correct.*

The two most common are the Theta notation(θ) and the Omega notation(Ω). The differences are shown in the below table:

Big O notation	Theta notation(θ)	Omega notation(Ω)
Represents the worst-case time complexity i.e. the upper bound.	Represents the average-case time complexity.	Represents the best-case time complexity i.e. the lower bound.

SPACE COMPLEXITY

- The term space complexity generally refers to the memory space that a code uses while being executed. Again space complexity is also dependent on the machine and so we are going to **represent the space complexity using the Big O notation** instead of using the standard units of memory like MB, GB, etc.
- Space complexity generally represents the summation of auxiliary space and the input space. Auxiliary space refers to the space that we use additionally to solve a problem. And input space refers to the space that we use to store the inputs.

```
Input(a)//1 Input space
Input(b)//1 Input space
c = a+b
//c-> 1 auxiliary space
```

The variables a and b are used to store the inputs but c refers to the space we are using to solve the problem and c is the auxiliary space. Here the space complexity will be O(3).