Apriori Analysis

It is Standard Approch To Use , we called it as a Absolute Analysis ;

Determination of Order of Magnitude of a Statement is known as Apriori Analysis …

What is Order Of Magnitude ?

It means how many time our Programm is Execute ;while Running a Programm how many time statement is executing

Ex :

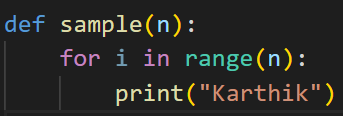
In Programm we use Loops why we use ?

If we want to Print Our Name 10 times we do two Approach :

1. Using Print Statement 10 times ;
2. Using Loop to Execute ;

Compare these two we prefer 2nd approach becozz our Code Quantity is less Compare to Previous One .

Example : 1



This is the Code if we want To Find the Time Complexity

It is Order of N 🡪 O(n)

Becozz here user define the number it might be anything , so how many time it will execute based 0 to n-1 like n = 10 means 0 to 9

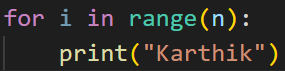
So the Order is O(n)

So this O(n) is Known As Big O Notation

This Big O Notation take care of the Worst Case Time Complexity ;

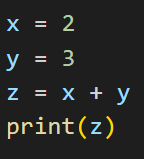
How will we Denote O(n) :

If we see our code



How many Time our print statement execute , that is what our time complexity , here it execute n time so O(n) .

Example 2 :

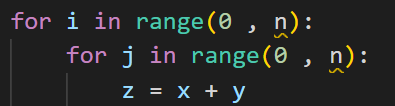


If we see now in this code what Is our time Complexity?

The Time Complexity is O(1) It Is Constant time

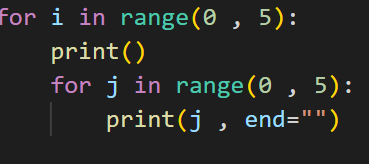
Becozz every set of code execute only one time there is no looping so this is Constant time complexity , So code time few millisecond to run this code .this is our purpose if our code run constant time we prefer that

Example 3 :



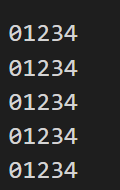
Here there are two Loops Nested Loop

Like our n is 5



Our Code Look Like this

Our Output is



So here it goes for 1st loop i = 0 and go to the j loop complete 0 to 4 becozz(5) again go to 1st loop now i = 1 and go to the j loop complete 0 to 4 becozz(5) .. likewise it will print until I = 4

So for this Our Time Complexity is n2+n+1

As we discuss this is Aprior Gives the Approx result

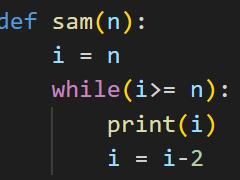
So in the Time Complexity n2 + n+1  
  
The Highest Polynomial is n2 ; so our time complexity is O(n2)

So this is the approx. result of the Above Code.

So in general our time complexity works based on the loop it is either n , n2 , log(n)

If we don’t have loop majorly it ends o(1)🡪Constant

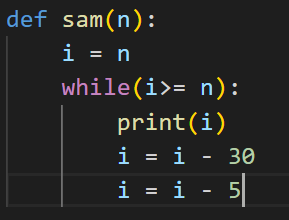
Now we are seeing While Loop :



Now we want to find the time complexity

Suppose n = 20 , every time I decrement by 2 so 20 , 18 , 16 , 14 ,……..2 , So it will Go Untill I = 2 ; so our code execute 10 times while the n = 20 , if n = 40 it will execute 20 times

So the Order is O(n/2) , becozz it is decrement by 2 , suppose



Now our Order is O(n/35) becozz decrement by 35

So The main thing is N only so

🡪 It will end up o(n)

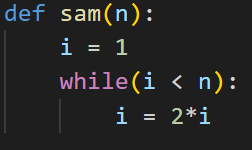
The Time Complexity Of the Problem is O(n).

Important Conclusion About Time Complexity :

* So Time Complexity Is Only About Loop ..

We fill find how many time our code runs whether it is Order O(n) , O(log n) , O(n2)

* If we have Nested Loop We fill Focus On the Larger Loop It is what we discuss previous degree of Polynomial (Highest Value)
* If there is No Loop At all we call it as Constant O(1).



Now we see n = 64 , start with I = 1 it will go and increment I = 2 then I = 4 , I = 8 , I = 16 , I = 32 , if I = 64 our condition is false

How many time our code runs , 6 time

If N is :

64 🡪 6 time

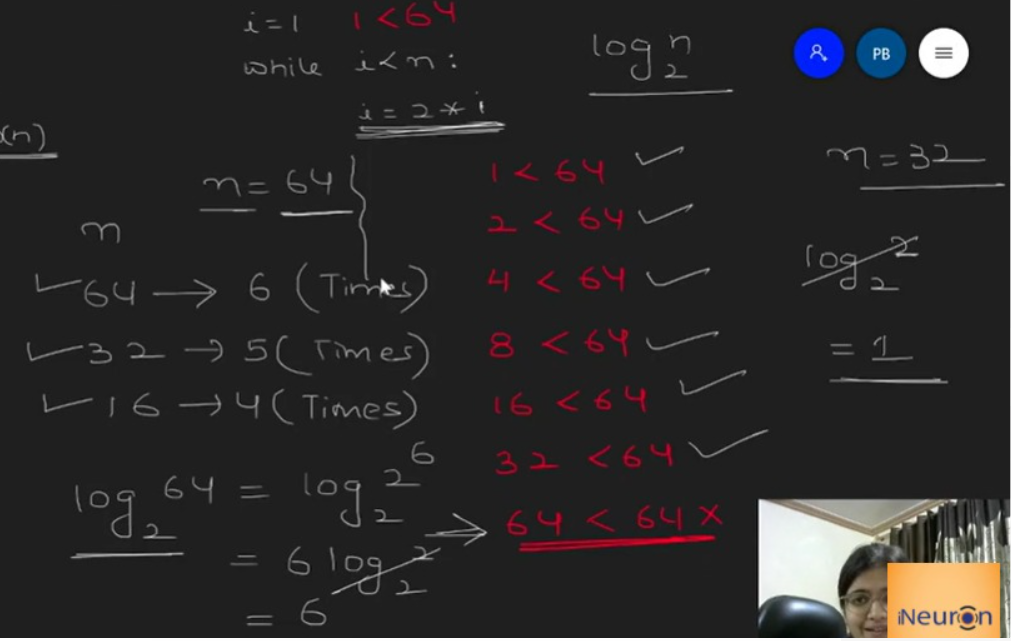
32 🡪 5 time

16 🡪 4 time

If we see log of 64 when base is 2 == 6;

Log64(base2) == 6;

Log32(base) == 5 ;



Now the Time Complexity is O(log2n).

If I = I \* 3 , our time complexity is o(log3n)