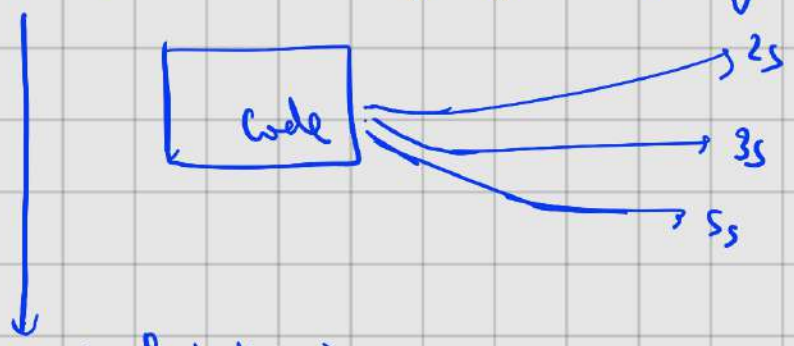
What is Time Complexity ▶

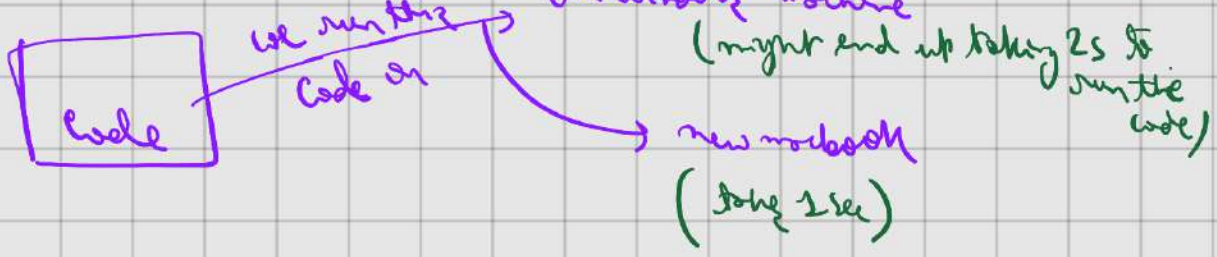
Whenever we write a code, might end up taking 2s, 3s or 5s on a machine



→ But this is not time complexity

∴ Time Complexity != Time Taken (TC)

Eg:

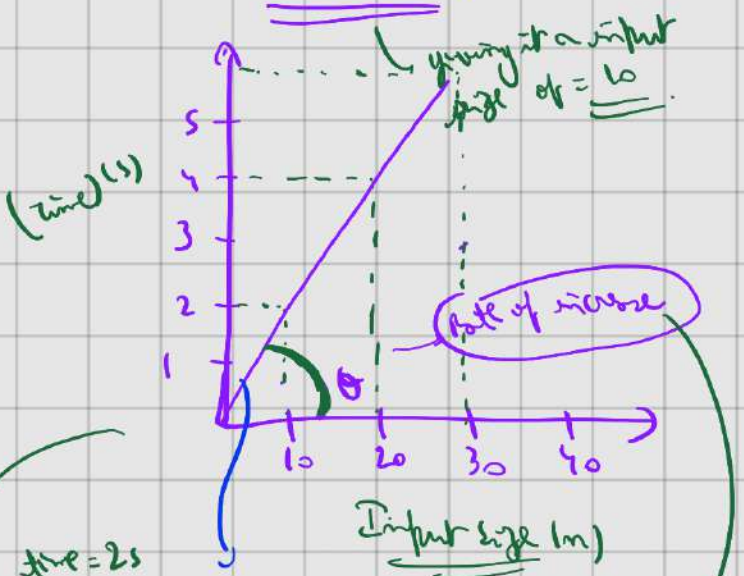


→ Depending on the system, code ends up taking different time.

∴ Why Time Complexity cannot be said as time taken
(Since, it is dependent on system or configuration)

Time Complexity : The rate at which the time taken increases with respect to the input size.

Old Laptop



giving it a input size of = 10

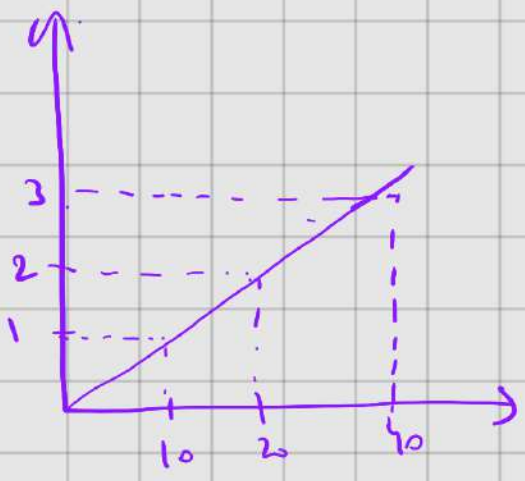
For n=10, time = 2s (input size)
for n=20, time: 4s
n=30, t = 6s

time increasing angle of θ

it the rate at which the time increases, is called as time complexity.

(depending upon input, if more input, may take more time)

New Macbook



for input size 10 → taking 1 sec
for input size 20 → 2s
40 → 3s

---

* We time complexity is not computed in seconds/minutes. (T.C)

* Every piece of code, takes time in terms of (Big-Oh Notation)

O( )

Capital O — inside it write how much time is taken

#

```
for(i=1; i<=5; i++)
{
    cout << "\n" << i;
}
```

Step 1 → i = 1 ; Assigning
Step 2 → i <= 5 , Comparison
Step 3 → cout << "\n" << i ; Printing
Step 4 → i++ - Increment
Step 5 → i <= 5
Step 6 → cout << "\n" << i ;

↳ whatever is the total no. of steps, it what we write inside O( ). i.e. name Big-Oh Notation.

↳ Big-Oh of this will be no. of steps, this code will take.

↳ We just can't count the steps manually everytime, i.e. step 1 step 2, etc.

↳ So, 3 rule comes

1) Always compute time complexity in terms of worst case scenario

i.e TC → worst case scenario.

2) Avoid Constants

3) Avoid lower Values.

#

```
for(i=1; i<=5; i++)
{
    cout << "\n" << i;
}
```

→ On a higher level, this for loop runs for 5 times

↳ At every time,
- Increment
- Check
- Print

→ 3 steps

→ For every time, it is operating 3 times,

ie $5 \times 3 = 15$ times

∴ $O(15)$ in terms of number.

↳ Time Complexity

→ Generally we don't represent the code in numbers.

So, #

```
for(i=1; i<=N; i++)
{
    cout << "\n" << i;
}
```

↳ ∴ Here, T.C = $O(N \times 3)$

$\leq O(3N)$

[∵ at every iteration, 3 things will happen, and it will run for N iterations]

Best Case Average Case Worst Case

#

```
if (marks < 25) cout << "Grade D"
else if (marks < 45) cout << "Grade C"
else if (marks < 65) cout << "Grade B"
else cout << "Grade A"
```

Best Case Scenario:

↳ when the program take the least amount of time

↳ if someone gives an input of marks = 10, first line gets executed and, Grade D, gets printed, and none of the other lines will be executed.

↳ O(2) → (Check, Print)
[for marks = 10]

→ But if entered, marks = 70, → it gives O(4) (really ? otherwise)

So, Always compute the time complexity in Worst Case Scenario.

↳ Here Best Case → O(2) → for marks = 10
Worst Case → O(4) → for marks = 70

∴ Average case will be = $\frac{\text{Best} + \text{Worst}}{2}$

→ Always think of the worse that can happen.

- Build a system for 1 person
- Build for 1 million (have to think about scaling up)

$O(4N^3 + 3N^2 + 8)$ → the no. of operations, that our code is taking.

↳ If input size, $= N = 10^5$.

∴ $O(4 \times \underline{\underline{10^{15}}} + 3 \times \underline{\underline{10^{10}}} + 8)$

This 8 doesn't have any significance before huge numbers i.e $\underline{\underline{10^{15}}}$, $\underline{\underline{10^{10}}}$

↳ i.e. why we don't consider constants.

`int n = 2;` → this is one operation,

```
for(i=1; i<=N; i++)
{
    cout << "Satbat";
}
```

→ Time complexity

$= O(N \times 3 + 1)$

↓ we won't consider unit operations

$\simeq O(N \times 3)$

∴ Whenever, input size is very large, those constants have very less significance.

#

$O(4N^3 + 3N^2 + 8)$

↳ If input size, $= N = 10^5$.

∴ $O(1\times10^{15} + 3\times10^{10} + 8)$ → No significance in adding 10^{10} to 10^{15}

10^{10} has no significance in front of 10^{15}.

it's like adding 1 to 1000000

{ No of operations, will be just slightly more for 10^{10}, in front of 10^{15}, not much.

→ ∴ we avoid lower values, which does not change the bigger number's significance by much.

(Highest)
Big-oh (O) → Worst case [upper-bound]

(Other notations)
Theta (θ) → [Average Complexity]

Omega (Ω) → Best case [lower-Bound] or lowest

Q. Tell the time complexity of the code?

```
for(int i=0; i<N; i++)
{
    for(int j=0; j<N; j++)
    {
        // Block of code
        // taking constant time
    }
}
```

Ans:

```
for(int i=0; i<N; i++)        → outer loop starts from i=0 → i=N
                                  i.e. N times
{
    for(int j=0; j<N; j++)    → Inner loop starts from j=0
                                  i.e N times → j=N
    {
        // Block of code
        // taking constant time
    }
}
```

(j goes from 0 → N)

at i=0 → [j = 0, 1, 2, 3 ... N-1] → N times

when i=1 → [j = 0, 1, 2, 3, ... N] → N times

i = 2 → [j = 0, 1, 2, 3, ... N] → N times

⋮

i = N-1 → [j = 0, 1, 2, 3, ... , N] → N times

N times

$N + N + N + N + \cdots\cdots + N$

$\leq N \times N$

$O(N^2)$

Q.2

```
for(int i=0; i<N; i++)
{
    for(int j=0; j<=i; j++)
    {
        ==   → Block of code [taking constant time]
    }
}
```

Ex

For $i = 0$, → $[j=0]$ → 1 time

$i = 1$, → $[j = 0, 1]$ 2

$i = 2$, → $[j = 0, 1, 2]$ 3

⋮

$i = n-1$ → $[j = 0, 1, 2, \cdots, n-1]$ n

∴ No. of iterations $= 1 + 2 + 3 + 4 + \cdots + n$

$= \frac{N(N+1)}{2}$

$= \frac{N^2}{2} + \frac{N}{2}$ ≃ Best time complexity

$= O\left(\frac{N^2}{2}\right)$

$\simeq O(N^2)$

SPACE COMPLEXITY

↳ the memory space that our program takes.

→ Again, memory space, it will vary from machine to machine,
↳ we cannot be dependent on machine.

∴ why for space complexity calculations, we'll use the Big O Notation.

Space complexity → Exact definition:

Auxiliary Space → Space that we take to solve the Problem

\+

Input Space → the space that we take to store the input (problem)

```
# int a, int b;
cin>>a>>b; (given input to a & b)
C = a+b;
```

↳ Auxiliary Space that we are using in the extra 'c' variable

Coz if we are using an extra variable i.e. C to solve the problem

i.e. O( ) (in terms of Big O)

not in some terms of Bytes, MB, KB, GB.

ii. Auxilliary space.

Both of these can be referred as input space

↳ Conversely, O(3) = the space complexity
∴ 3 different variables

Ex #

int a[N]; → O(N)

↳ If you are defining an array of size n, it means we are consuming O(N) size (or O(N) space complexity)

---

int a, int b;

cin >> a >> b. (input in a & b)

b = (a+b);

(Task is to add both numbers)

Summation of a & b stored in b.

↳ It took lesser space, but not for interviews or big companies. (this is not done)

↳ Here you manipulated the data, you changed 'b'.

↳ In interview, never do anything to the input, unless the interviewer says to

↳ Always, take extra variable, extra array, extra nothing.

↳ Data should not be touched.

↳ You can do anything, taking the data, but don't do anything on the data.

↳ If you are using 2 arrays, TC will be O(2N), instead of (1 array, TC = O(N)), there is not much difference

---

In CP, you write code → you don't write your code on one of windows

you send it to the server

LC, GFG, CF
= = =

↳ Most servers, take

1 second ≃ 10^8 operations on the servers
~ for (roughly)

2 seconds ≃ many
2×10^8 (not 10^6)
operations

5 seconds ≃ many
5×10^8
operations

(TL)
If Time limit = 1 second

∴ Time Complexity (TC) of the snippet of your entire code,

should be ≃ $O(10^8)$ operations