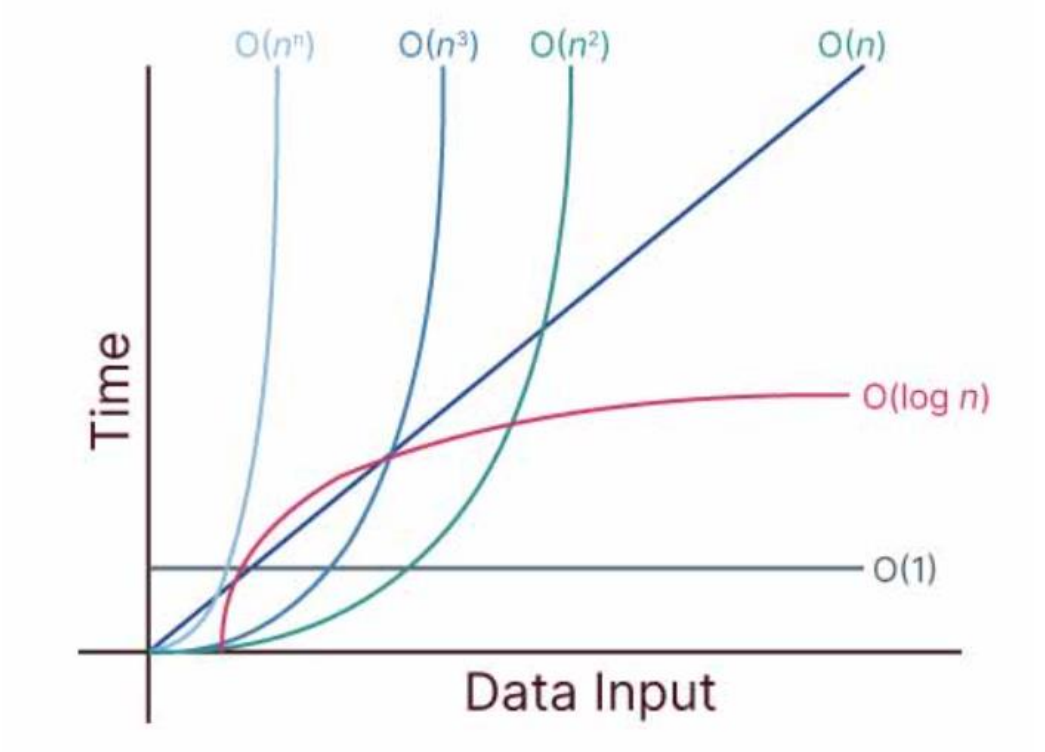
Algorithm Complexity & Big O notation



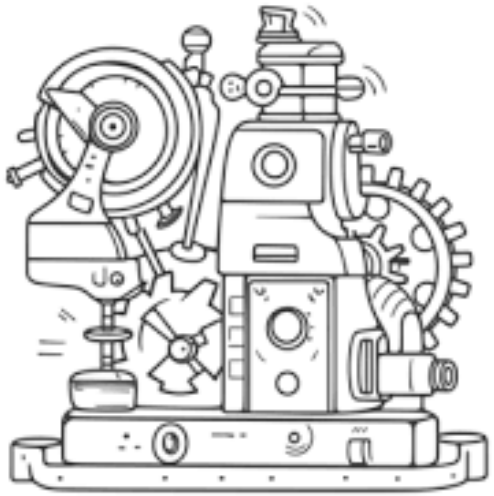
Objectives

- Understand the concept of **time complexity**
- Understand **5 Big O Notations**
  - ✓ O (1) Constant time complexity
  - ✓ O (log n) Logarithmic time complexity
  - ✓ O(n) Linear time complexity
  - ✓ O (n log n) Log-linear time complexity
  - ✓ $O(n^2)$ Quadratic time complexity
- **Calculate** the time complexity in different use cases

An algorithm get **inputs** and produce **results**

INPUTS

Number of input = N

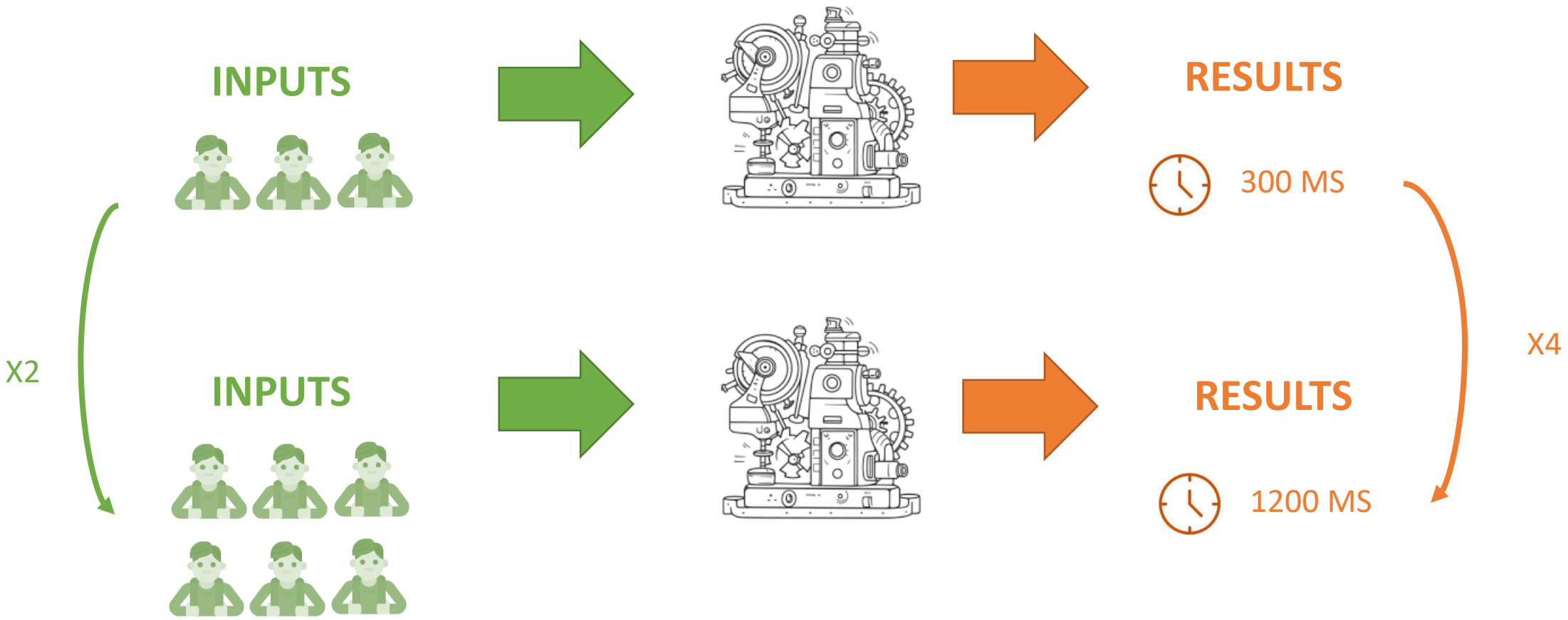


ALGORITHM

RESULTS

The **Time complexity**

*Represents **how runtime grows** regarding the input size*



Time is related to the number of **elementary operations** executed

- ✓ They refers to **atomic smallest** operations
- ✓ They always take the **same time to be executed** (for the same machine, same language)

```
print("Enter the value M:")

read(m)

s = 2 * 8

n = 2 + s + 3
```

1 write operation

1 assignment operation

1 computation + 1 assignment

2 computation + 1 assignment

7 EO for this algorithm

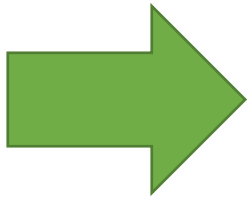
Big O - *An example*

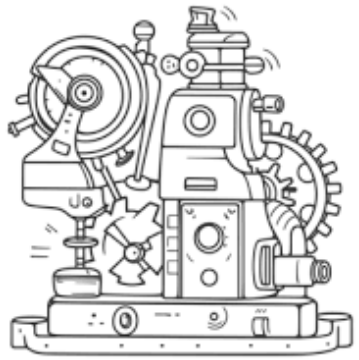
You lost you pen !

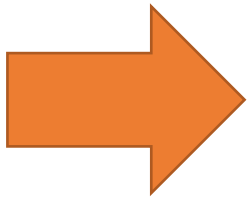
- ✓ You gave your pen to a student, in a classroom of 100 students
- ✓ You have to find that pen without knowing to whom you gave it.
- ✓ *You need an algorithm to find you pen !*

INPUTS

The 100 students (N=100)







RESULTS

The student name

Big O - *An example*

*Depending of **your strategy**, you might have different time complexities*

O order		**Example in our case**
O(n2)	**Quadratic time** complexity	✓ You go and ask the first person in the class if he has the pen. ✓ Also, you ask this person about the other 99 people in the classroom if they have that pen and so on
O(n)	**Linear time** complexity	✓ You go and ask each student individually
O(log n)	**Logarithmic time** complexity	✓ I divide the class into two groups, then ask: “Is it on the left side, or the right side of the classroom?” ✓ I take that group and divide it into two and ask again, and so on.

Big O - *An example*

*Choosing your algorithm also depend on the **context..***

O order		**Example in our case**
O(n2)	**Quadratic time** complexity	✓ You go and ask the first person in the class if he has the pen. ✓ Also, you ask this person about the other 99 people in the classroom if they have that pen and so on
O(n)	**Linear time** complexity	✓ You go and ask each student individually
O(log n)	**Logarithmic time** complexity	✓ I divide the class into two groups, then ask: "Is it on the left side, or the right side of the classroom?" ✓ I take that group and divide it into two and ask again, and so on.

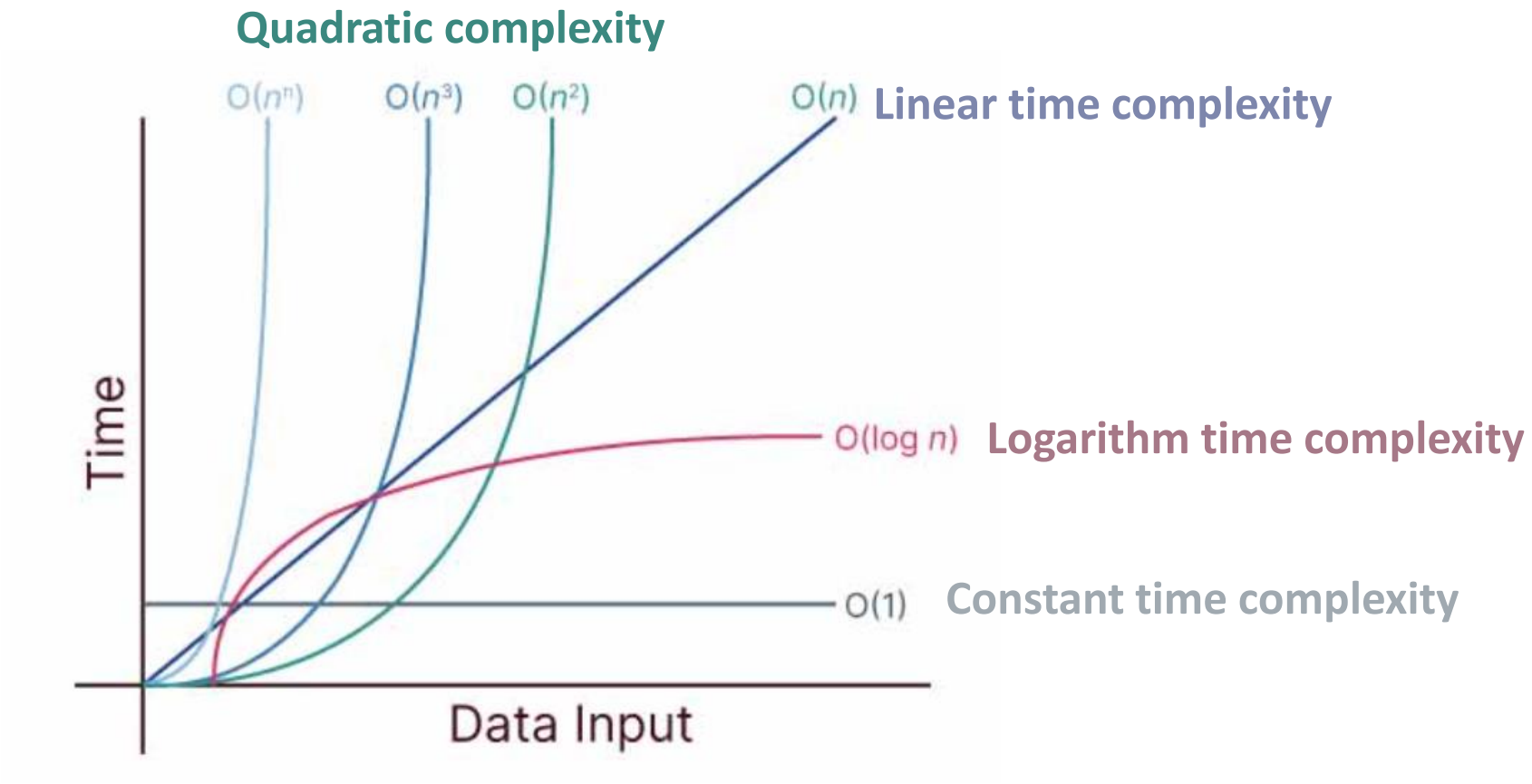
if only one student knows on which student the pen is hidden.

if one student had the pen and only they knew it

If all the students knew, but would only tell me if I guessed the right side.

Big O notations

We will explore 5 types of complexity today



How to **compute** complexity ?

1. Identify the algorithm **main blocs**
2. Identify how **many time** each bloc need to **be executed**
3. Count **elementary operations** on **each bloc**
4. Infer the whole algorithm **complexity**

An example of workflow …

DEMO

1 Identify the algorithm **main blocs**

```
print("Enter the value N:")
read(n)

print("Enter the value M:")
read(m)

s = n
n = m
m =s

print("N value is: ", N)
print("M value is: ", M)
```

How many main blocs
This algorithm is composed of?

Can you identify what
Does this algorithm perform?

DEMO

1 Identify the **main blocs**

Only 1 bloc, as this algorithm as not specific branching here

MAIN BLOC

```
print("Enter the value N:")
read(n)

print("Enter the value M:")
read(m)

s = n
n = m
m = s


print("N value is: ", N)
print("M value is: ", M)
```

This algorithm just **swap** the 2 values, n and m

DEMO

2

Count each bloc **elementary operations (EO)**

MAIN BLOC

2 EO

```
print("Enter the value N:")
read(n)
```

2 EO

```
print("Enter the value M:")
read(m)
```

3 EO

```
s = n
n = m
m =s
```

2 EO

```
print("N value is: ", N)
print("M value is: ", M)
```

9 EO

DEMO

3

Identify how **many times** the bloc need to **be executed**
– depending on the input (here n, m)

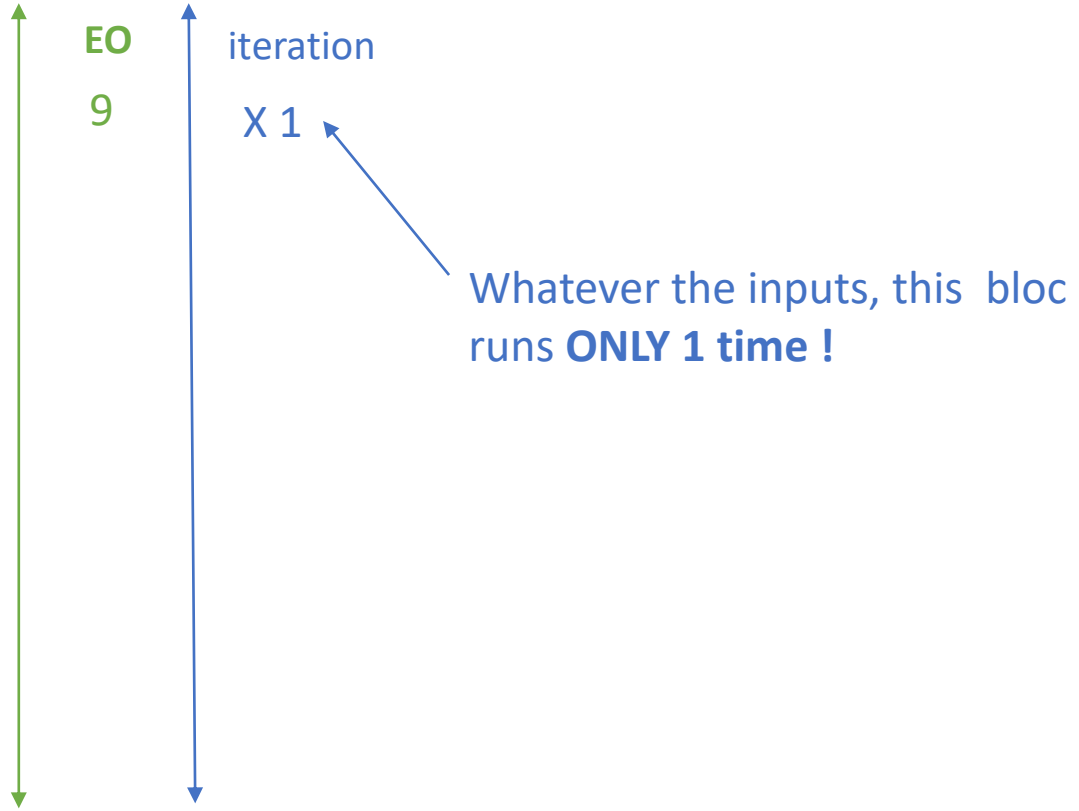
INIT BLOC

```
print("Enter the value N:")
read(n)

print("Enter the value M:")
read(m)

s = n
n = m
m =s

print("N value is: ", N)
print("M value is: ", M)
```



DEMO

4 Infer the **whole algorithm complexity**

INIT BLOC

```
print("Enter the value N:")
read(n)

print("Enter the value M:")
read(m)

s = n
n = m
m =s

print("N value is: ", N)
print("M value is: ", M)
```

EO

9

iteration

X 1

9 x (1)

O (**9**)

Constants can be **ignored**
In the complexity computation

O (1)

Constant complexity

NOW YOU KNOW

O (1)

Constant time complexity

*Algorithm execution time **does not change** when inputs grow !*

ACTIVITY

Activity 1

Analyze the complexity of this algorithm following the workflow

```
print("Enter the value N:")
read(n)

if n mod 2==0 then
     print("the number is even")
else
     print("the number is odd")
```

A. How many elementary operations (EO) are there?
B. What is the time complexity of this algorithm?

SOLUTION

Conditional branches

*For conditional branch, **we compute the max** btw the different branch complexities*

MAIN BLOC

```
print("Enter the value N:")
read(n)

if n mod 2==0 then
        print("the number is even")
else
        print("the number is odd")
```

2 EO

2 EO

1 EO

1 EO

Only 1 bloc , as no specific branches here

Max (1, 1) = 1

EO

ITERATIONS PER BLOC

5

X 1

5 x 1

Constants can be **ignored** In the complexity computation

O (5)

O (1)

Constant complexity !

ACTIVITY

Activity 2

Analyze the complexity of this algorithm following the 4 steps

```
function sum(t: int[]) : int

      n = 20
      s = 0


      for i from 0 to n, do
            s = s + t[i]


      return s
```

A. How many elementary operations (EO) are there?
B. What is the time complexity of this algorithm?

SOLUTION

A loop, but with a **constant number of iterations**!

You might have put O(n) for the loop bloc, but n here ***is NOT an input !***

```
function sum(t: int[]) : int
```

EO

ITERATIONS PER BLOC

INIT BLOC — 2 EO

```
n = 20
s = 0
```

2

X 1

LOOP 1

```
for i from 0 to n, do
    s = s + t[i]
```

3 EO

3

X 21

n is a constant
Not an input

RETURN BLOC

```
return s
```

1

X 1

Constants can be **ignored**
In the complexity computation

O (66)

O (1)

Constant complexity !

ACTIVITY

Activity 3

Analyze the complexity of this algorithm following the 4 steps

```
print("Enter the value N:")
read(n)


for i from 1 to 10, do


    for j from 1 to n, do
        print(“ITC”)
```

A. How many elementary operations (EO) are there?
B. What is the time complexity of this algorithm?

NOW YOU KNOW

O (n)

Linear time complexity

*Algorithm execution time **change linearly** when inputs grow !*

ACTIVITY

Activity 4

Analyze the complexity of this algorithm following the 4 steps

```
print("Enter the value N:")
read(n)


for i from 1 to n, do


    for j from 1 to i, do
        print(“ITC”)
```

SOLUTION

First let s compute the number of iterations of the 2 loops

INIT BLOC

```
print("Enter the value N:")
read(n)
```

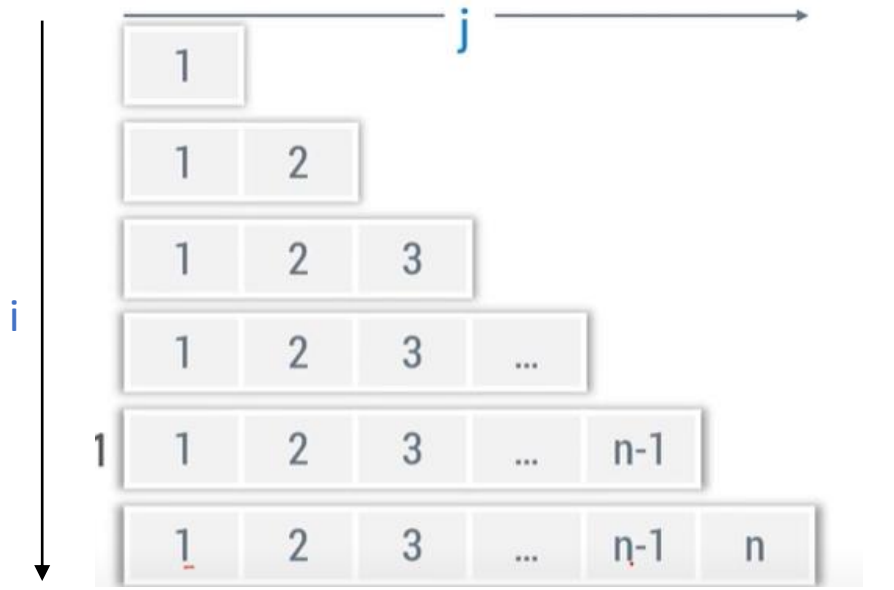
OUTER LOOP

```
for i from 1 to n, do
```

INNER LOOP

```
    for j from 1 to i, do
        print(“ITC”)
```

ITERATIONS
PER BLOC



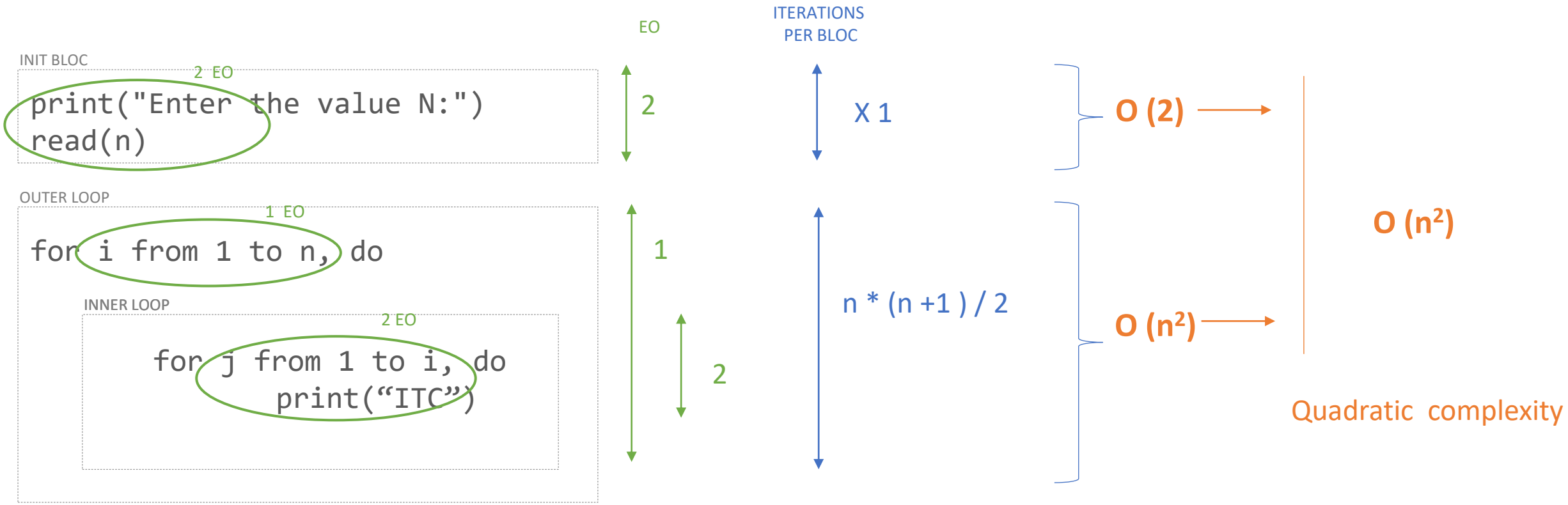
This is math stuff…

$n * (n + 1) / 2$

SOLUTION

With **nested loops**, complexity are **multiplied**

In this case, each nested loop depends on n… We got a N^2 - quadratic complexity…



NOW YOU KNOW

$O(n^2)$

Quadratic time complexity

*the execution time increases proportionally to the **square** of the size of the input data*

ACTIVITY

Activity 5

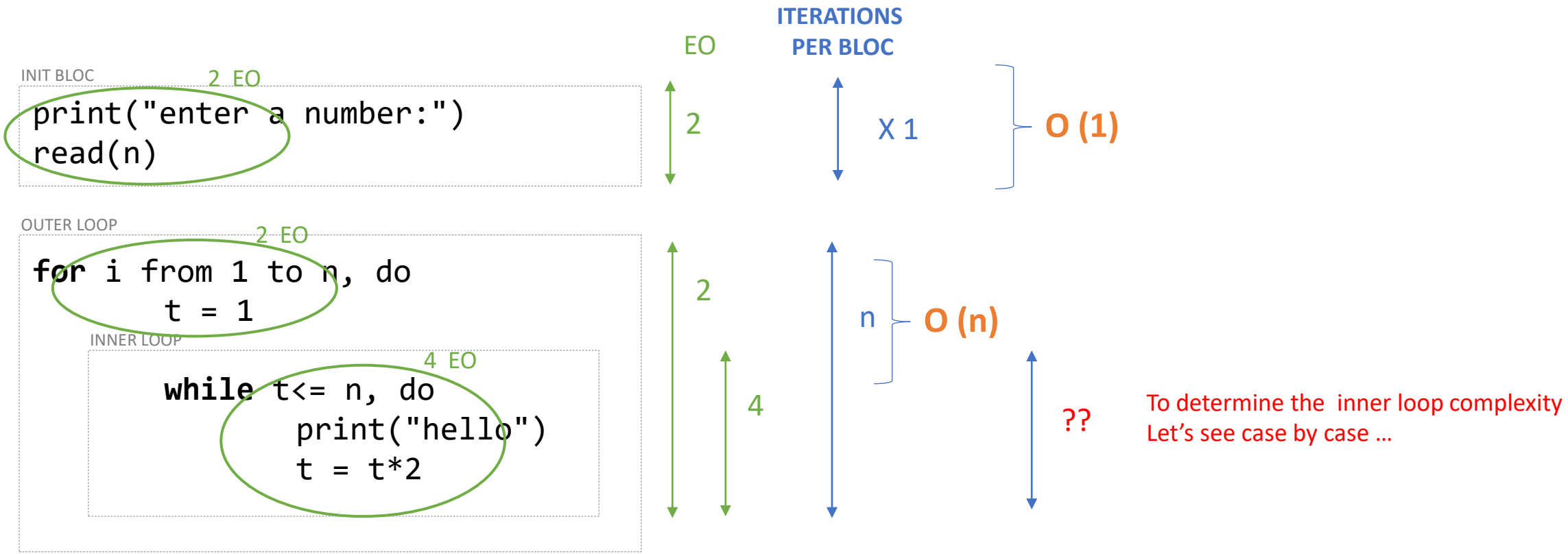
Analyze the complexity of this algorithm

```
print("enter a number:")
read(n)


for i from 1 to n, do
      t = 1
      INNER LOOP
      while t <= n, do
            print("hello")
            t = t*2
```

SOLUTION

We need to **identity** the **inner loop** iterations



SOLUTION

We need to **identity** the **inner loop** iterations

```
print("enter a number:")
read(n)

for i from 1 to n, do
        t = 1
```

INNER LOOP

```
  while t<= n, do
          print("hello")
          t = t*2
```

N	*1*	*10*	*300*
INNER LOOP ITERATIONS	1	4	9

If n = 10
- Iteration 1 : t = 1
- Iteration 2 : t = 2
- Iteration 3 : t = 4
- Iteration 4 : t = 8

t=16 => we stop

If n = 300
- Iteration 1 : t = 1
- Iteration 2 : t = 2
- …
- Iteration 9 : t = 256

t=512 => we stop

NOW YOU KNOW

Logarithm complexity : O(log N)

The **number of steps** required to complete the bloc grows **logarithmically**

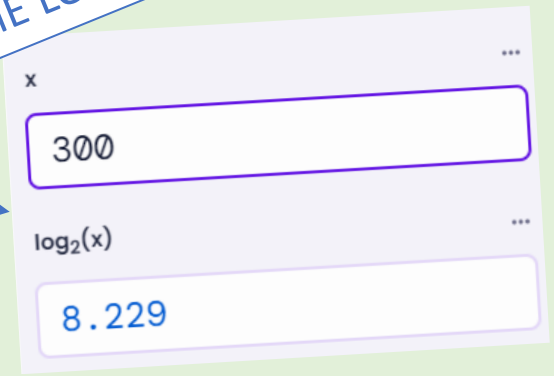
INNER LOOP

```
while t<= n, do
        print("hello")
        t = t*2
```

Log**2** …. as we multiply by 2

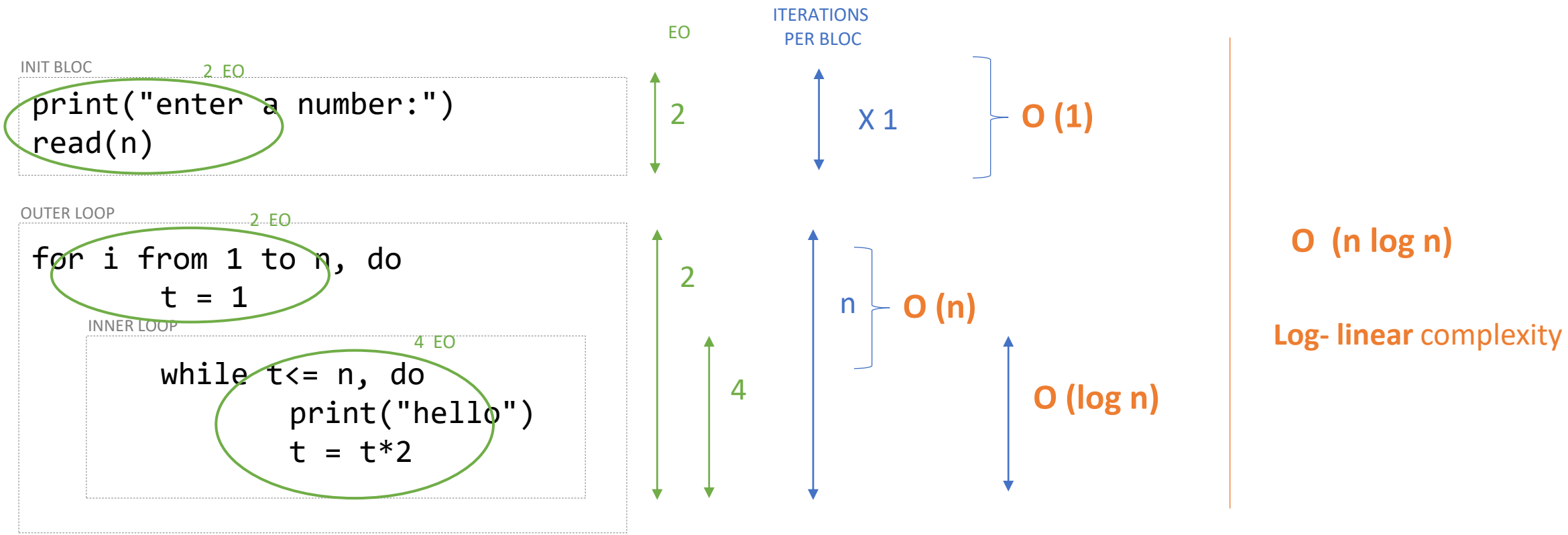
Bloc	*N=1*	*N=10*	*N=300*
INNER LOOP	1	4	9
1 + $Log_2(n)$	1	4.3	9.2

CHECK THE LOG2 CALCULATOR



SOLUTION

We need to **identity** the **inner loop** iterations



NOW YOU KNOW

O (n log N)

Log Linear time complexity

the execution time increases proportionally
with the input size multiplied by the logarithm of n

Practices

EXERCICE 1

Evaluate the complexity based on the five types you have learned

```
print("Enter the value N:")
read(n)

sum = 0

for i from 1 to n, do
    sum = sum + i

print("Sum is: ", sum)
```

EXERCICE 2

Evaluate the complexity based on the five types you have learned

```
print("Enter the value N:")
read(n)

isPrime = true

for i from 2 to n-1, do
    if n mod i == 0 then
        isPrime = false
        break

if isPrime then
    print("Prime number")
else
    print("Not a prime number")
```

EXERCICE 3

Evaluate the complexity based on the five types you have learned

```
print("Enter the value of the radius:")
read(radius)

area = 3.14 * radius * radius

print("Area of the circle is: ", area)
```

EXERCICE 4

Evaluate the complexity based on the five types you have learned

```
print("Enter the value N:")
read(n)

maxValue = array[1]

for i from 2 to n, do
    if array[i] > maxValue then
        maxValue = array[i]

print("Maximum value is: ", maxValue)
```

EXERCICE 5

Evaluate the complexity based on the five types you have learned

```
print("Enter the value N:")
read(n)

for i from 1 to n, do
    for j from 1 to n - i, do
        if array[j] > array[j + 1] then
            temp = array[j]
            array[j] = array[j + 1]
            array[j + 1] = temp
```

EXERCICE 6

Evaluate the complexity based on the five types you have learned

```
print("Enter the value N:")
read(n)
print("Enter the value x:")
read(x)

count = 0

for i from 1 to n, do
    if array[i] == x then
        count = count + 1

print("Occurrence of x: ", count)
```

EXERCICE 7

Evaluate the complexity based on the five types you have learned

```
print("Enter the size of the matrix N:")
read(n)

for i from 1 to n, do
    for j from 1 to n, do
        result[i, j] = 0
        for k from 1 to n, do
            result[i, j] = result[i, j] + matrix1[i, k] * matrix2[k, j]
```

EXERCICE 8

Evaluate the complexity based on the five types you have learned

```
print("Enter the value N:")
read(n)
print("Enter the value x:")
read(x)

found = false

for i from 1 to n, do
    if array[i] == x then
        found = true
        print("Element found at: ", i)
        break

if not found then
    print("Element not found")
```

EXERCICE 9

Evaluate the complexity based on the five types you have learned

```
print("Enter the value N:")
read(n)
print("Enter the value x:")
read(x)

left = 1
right = n

while left <= right, do
    mid = (left + right) // 2

    if array[mid] == x then
        print("Element found at: ", mid)
        break
    elif array[mid] < x then
        left = mid + 1
    else
        right = mid - 1
```

You should know…

- Understand the concept of **time complexity**
- Understand **5 Big O Notations**

✓ O (1)	Constant time complexity
✓ O (log n)	Logarithmic time complexity
✓ O(n)	Linear time complexity
✓ O (n log n)	Log-linear time complexity
✓ O(n2)	Quadratic time complexity

- **Calculate** the time complexity in different use cases

3-2-1 Challenge

- ✓ List three things you **learned** today.
- ✓ List two **questions** you still have.
- ✓ List one aspect of the lesson or topic you **enjoyed**.