TODO BEFORE

Preparation before next session

⚠ *You will need to present your research during the next session*

LEARN

1. Read this guide about algorithm complexity

RESEARCH

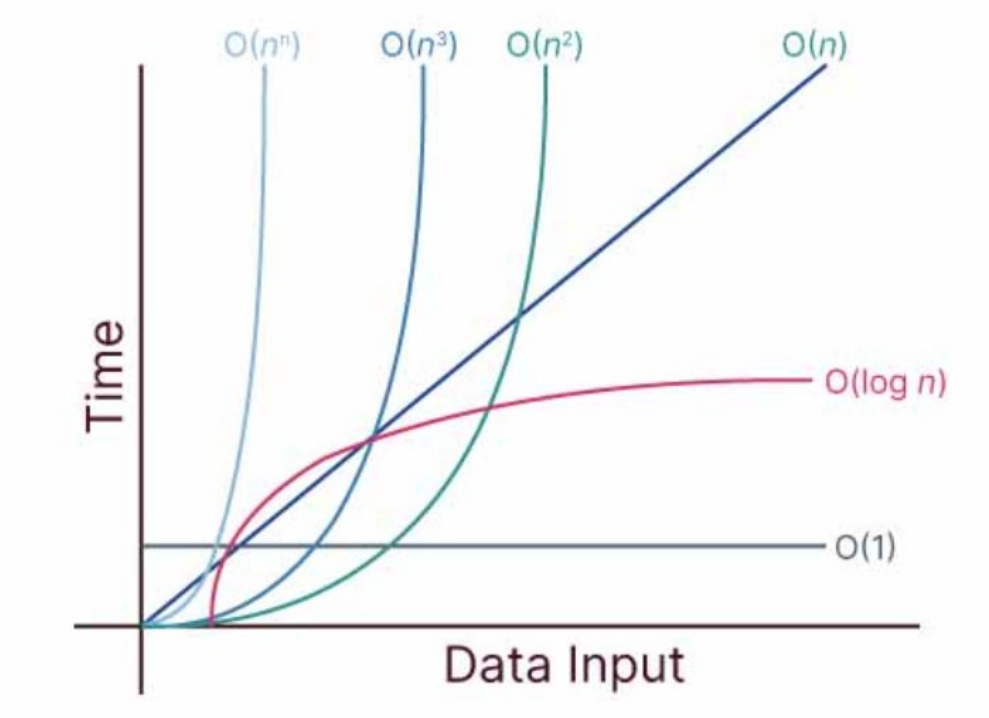
Make researches on the following points

Q1 – What does the **time complexity** means ?

Q2 – What are the **impacts** of an algorithm with a **Hight complexity** ?

ALGORITHM ADVANCED

W2-S3 – Complexity & Big O notation



CADT IDT

Session Objectives

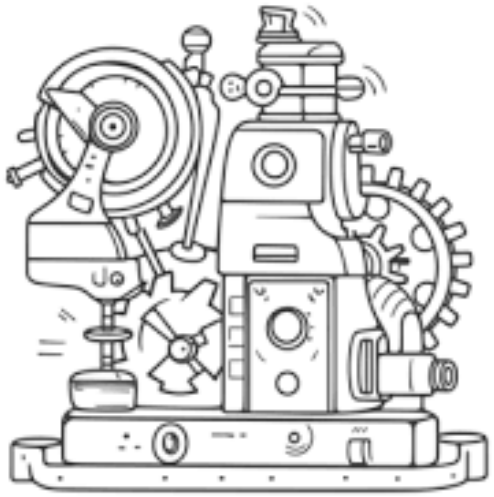
- Understand the concept of **time complexity**
- Understand **5 Big O Notations**
  - ✓ O (1) — Constant time complexity
  - ✓ O (log n) — Logarithmic time complexity
  - ✓ O(n) — Linear time complexity
  - ✓ O (n log n) — Log-linear time complexity
  - ✓ O(n2) — Quadratic time complexity
- **Calculate** the time complexity in different use cases

EXPLAIN

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

INPUTS

Number of input = N



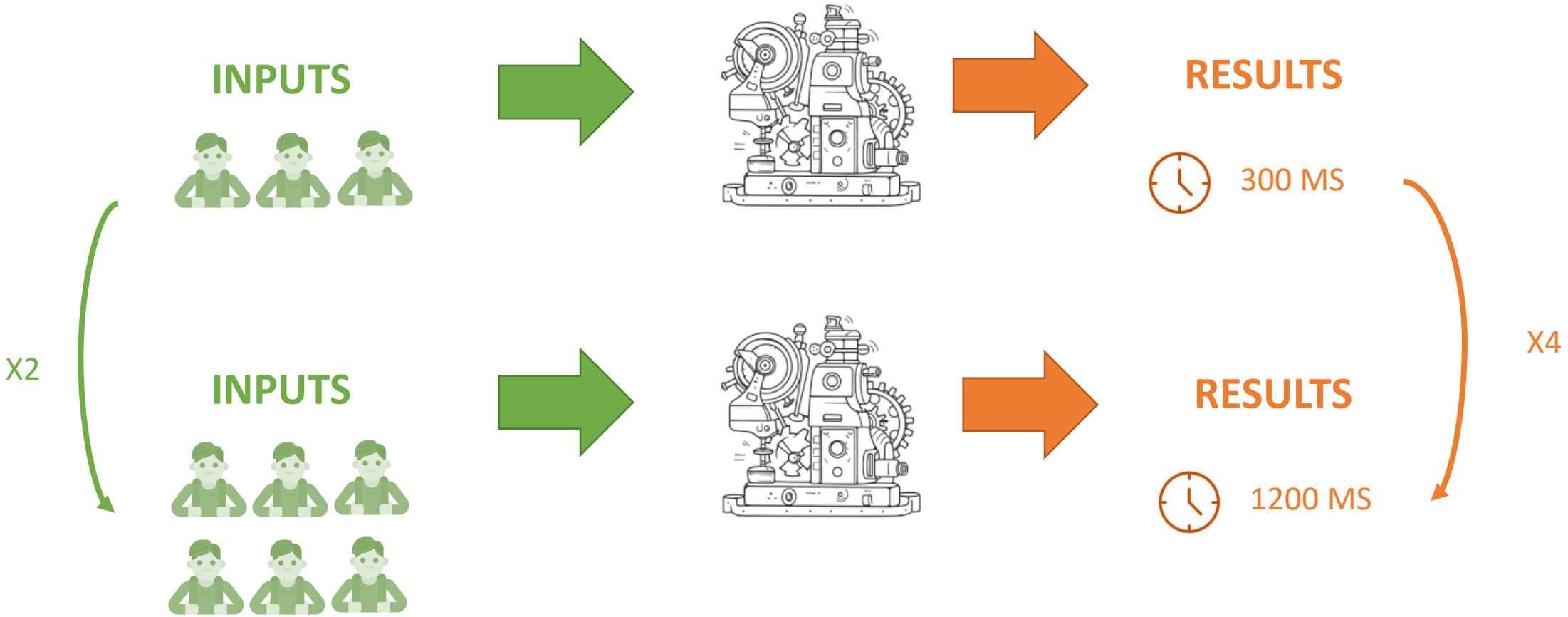
ALGORITHM

RESULTS

EXPLAIN

The **Time complexity**

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



EXPLAIN

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)

```
cout<<"Enter the value M:“;

cin>>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

EXPLAIN

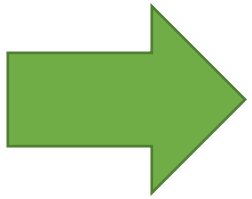
Big O - *An example*

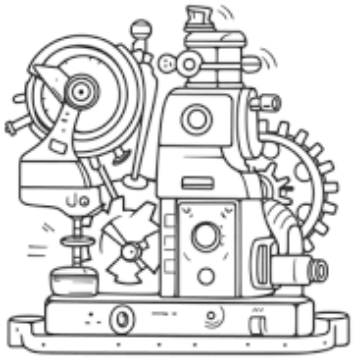
You lost you pen !

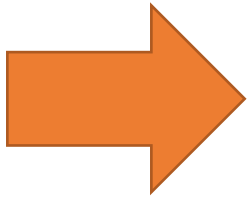
- ✓ You gave your pen to a student, in a classroom of 100 students
- ✓ You have to find that pen without knowing to whom you gave it.
- ✓ *You need an algorithm to find you pen !*

INPUTS

The 100 students (N=100)







RESULTS

The student name

EXPLAIN

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.

EXPLAIN

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.

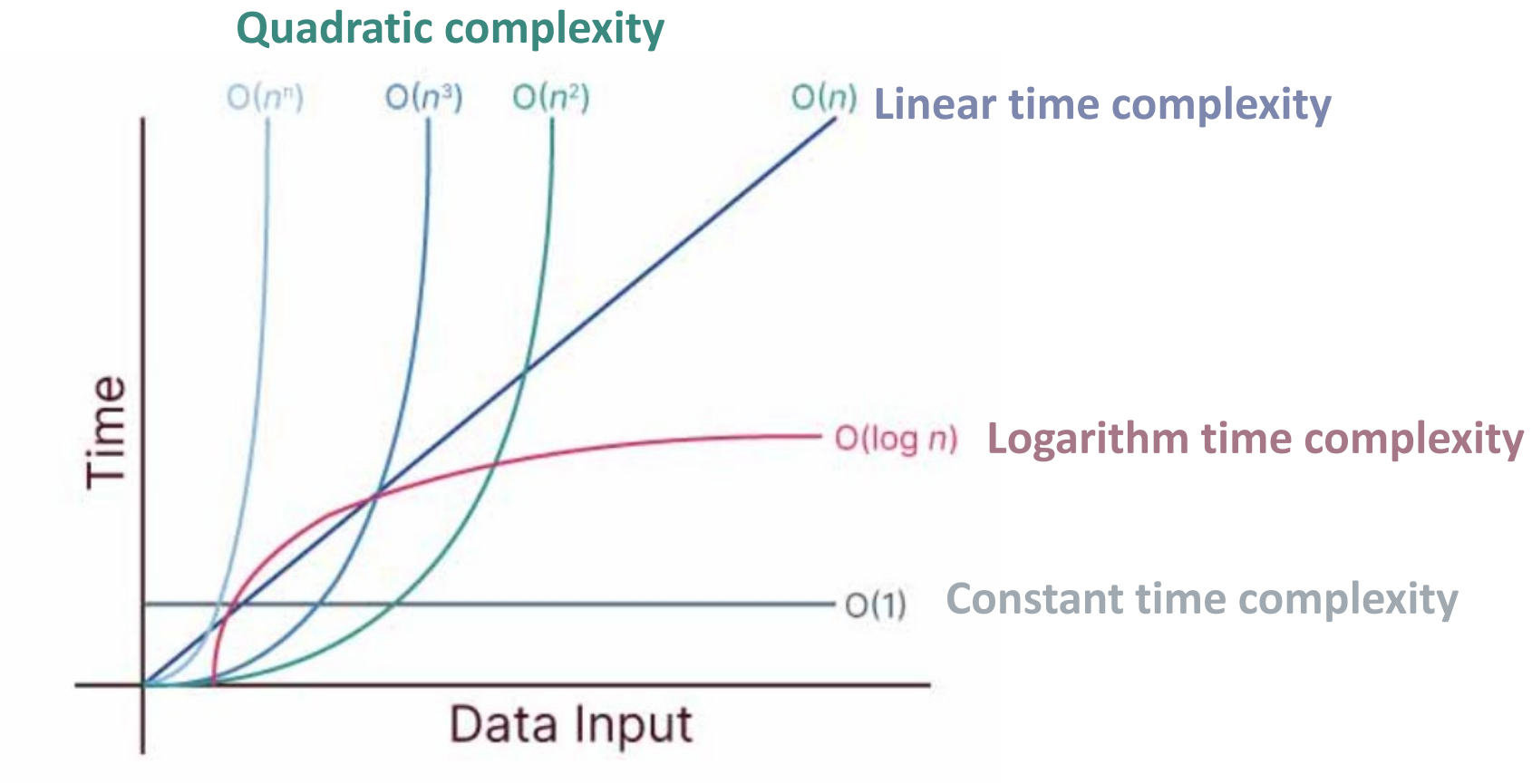
if one student had the pen and only they knew it

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

EXPLAIN

Big O notations

We will explore 5 types of complexity today



How to **compute** complexity ?

1. Identify the algorithm **main block**
2. Identify how **many time** each block need to **be executed**
3. Count **elementary operations** on **each block**

Block complexity

4. Infer the whole algorithm **complexity**

An example of workflow …

DEMO

1 Identify the algorithm **main block**

```
cout<<“Enter the value N:”;
cin>>n;

cout<<“Enter the value M:”;
cin>>n;

s = n
n = m
m = s


cout<<“ N value is: ”<<n<<endl;
cout<<“ M value is: ”<<m<<endl;
```

How many main block
This algorithm is composed of?

Can you identify what
Does this algorithm perform?

DEMO

1 Identify the **main blocks**

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

MAIN BLOCK

```
cout<<“Enter the value N:”;
cin>>n;

cout<<“Enter the value M:”;
cin>>n;

s = n
n = m
m = s


cout<<“ N value is: ”<<n<<endl;
cout<<“ M value is: ”<<m<<endl;
```

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

DEMO

2

Count each block **elementary operations**

MAIN BLOCK

2 EO

```
cout<<“Enter the value N:”;
cin>>n;
```

2 EO

```
cout<<“Enter the value M:”;
cin>>n;
```

3 EO

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

2 EO

```
cout<<“ N value is: ”<<n<<endl;
cout<<“ M value is: ”<<m<<endl;
```

9 EO

DEMO

3

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

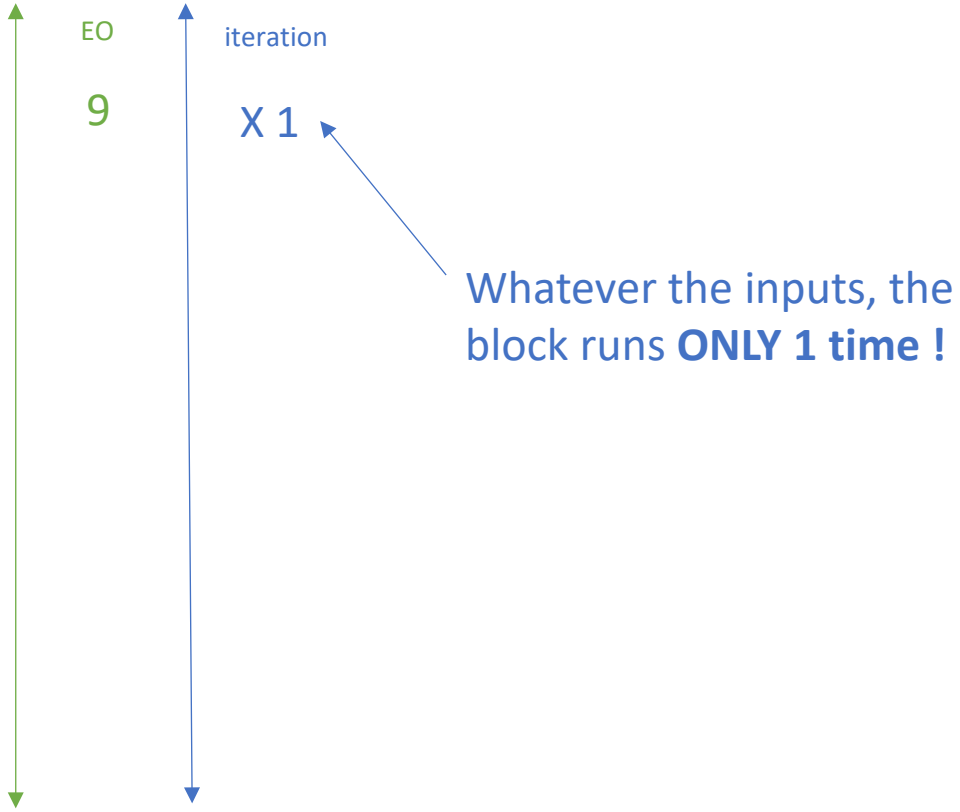
INIT BLOCK

```
cout<<“Enter the value N:”;
cin>>n;

cout<<“Enter the value M:”;
cin>>n;

s = n
n = m
m = s

cout<<“ N value is: ”<<n<<endl;
cout<<“ M value is: ”<<m<<endl;
```



DEMO

4

Infer the **whole algorithm complexity**

INIT BLOCK

```
cout<<“Enter the value N:”;
cin>>n;

cout<<“Enter the value M:”;
cin>>n;

s = n
n = m
m = s

cout<<“ N value is: ”<<n<<endl;
cout<<“ M value is: ”<<m<<endl;
```

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

```
cout<<“Enter the value N:”;
cin>>n;

if (n % 2 == 0)
        cout<<“ the number is even”<<endl;
else
        cout<<“ the number is odd”<<endl;
```

SOLUTION

Conditional branches

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

MAIN BLOCK

```
cout<<“Enter the value N:”;
cin>>n;

if (n % 2 == 0)
        cout<<“ the number is even”<<endl;
else
        cout<<“ the number is odd”<<endl;
```

2 EO

2 EO

1 EO

1 EO

Only 1 block , as no specific branches here

Max (1, 1) = 1

EO

ITERATIONS PER BLOCK

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

```
int sum(int t[]) {

    int n = 20;
    int s = 0;


    for (int i = 0; i < n; i++) {
        s += t[i];
    }


    return s;
}
```

SOLUTION

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

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

```
int sum(int t[]) {
    int n = 20;
    int s = 0;

    for (int i = 0; i < n; i++) {
        s += t[i];
    }

    return s;
}
```

INIT BLOCK — 2 EO

LOOP 1

RETURN BLOCK

EO

2

1

ITERATIONS PER BLOCK

X 1

1 + (n+1) + n + 2n
= 4n + 2=82

X 1

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

O (85)

O (1)

Constant complexity !

ACTIVITY

Activity 3

Analyze the complexity of this algorithm following the 4 steps

```
cout<<“Enter the value N:”;
cin>>n;


for (int i = 0; i < 10; i++)


        for (int j = 0; j < n; j++)
                cout<<“CADT”;
```

SOLUTION

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

In this case, the first loop is a constant loop, which will not impact the final complexity

INIT BLOCK

2 EO

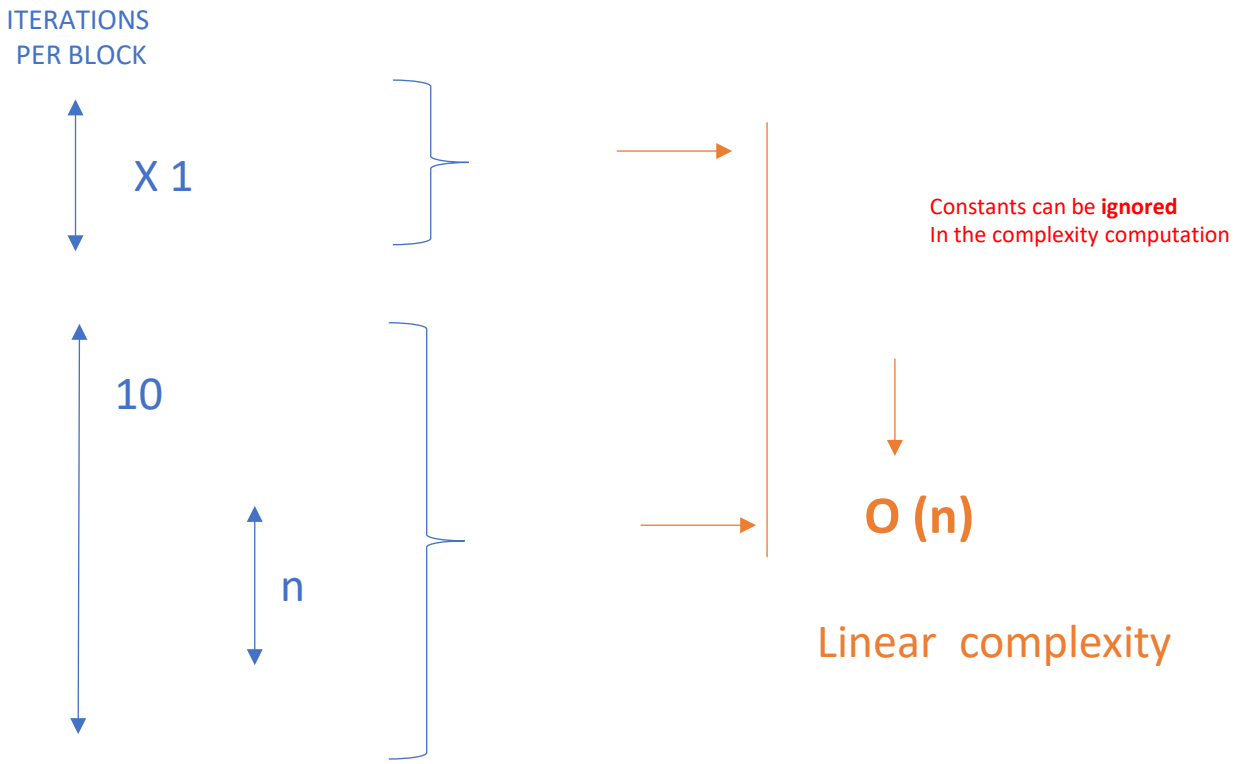
```
cout<<“Enter the value N:”;
cin>>n;
```

OUTER LOOP

```
for (int i = 0; i < 10; i++)
```

INNER LOOP

```
for (int j = 0; j < n; i++)
        cout<<“CADT”;
```



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

```
cout<<“Enter the value N:”;
cin>>n;


for (int i = 0; i < n; i++)


        for (int j = 0; j < i; i++)
                cout<<“CADT”;
```

SOLUTION

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

INIT BLOCK

```
cout<<“Enter the value N:”;
cin>>n;
```

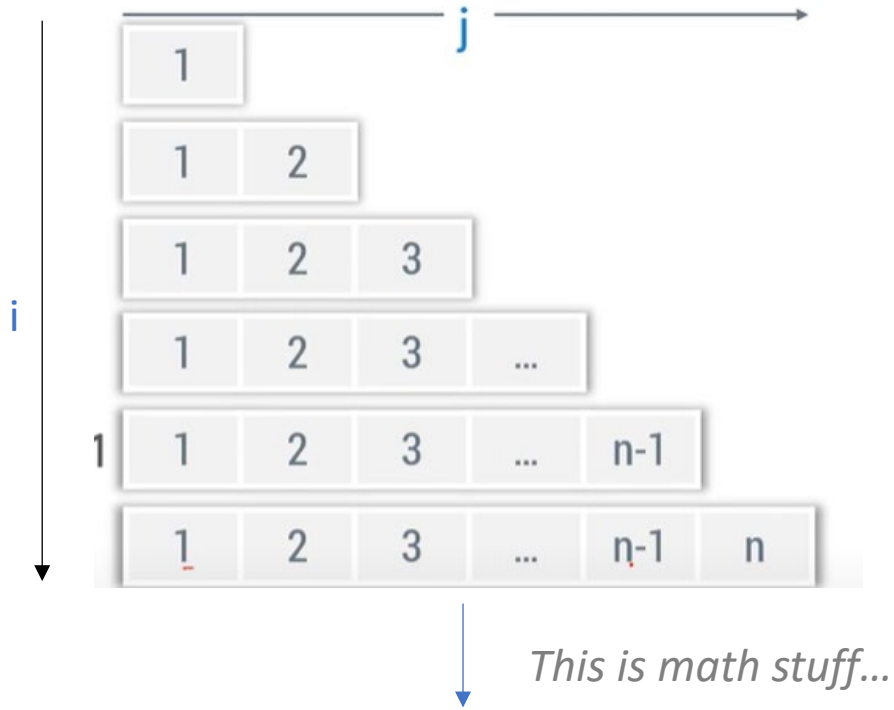
OUTER LOOP

```
for (int i = 0; i < n; i++)
```

INNER LOOP

```
for (int j = 0; j < i; i++)
        cout<<“CADT”;
```

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…

INIT BLOCK

2 EO

```
cout<<“Enter the value N:”;
cin>>n;
```

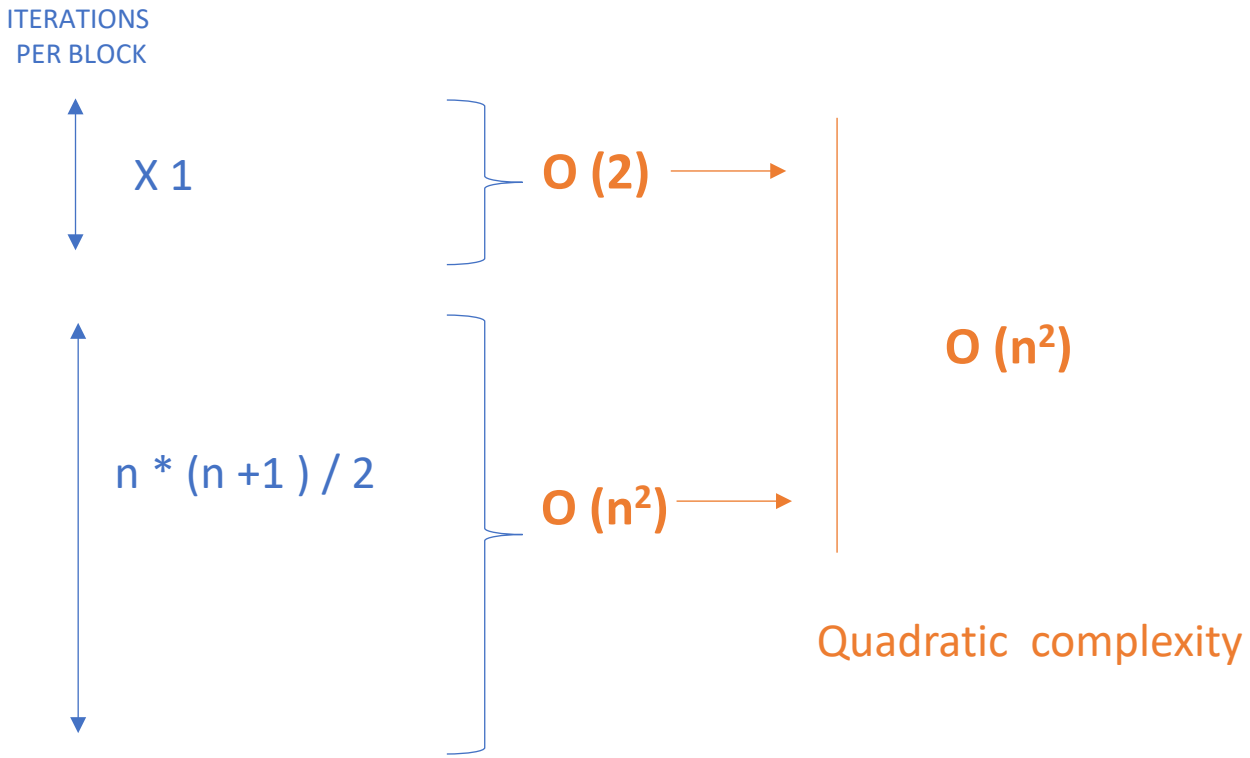
OUTER LOOP

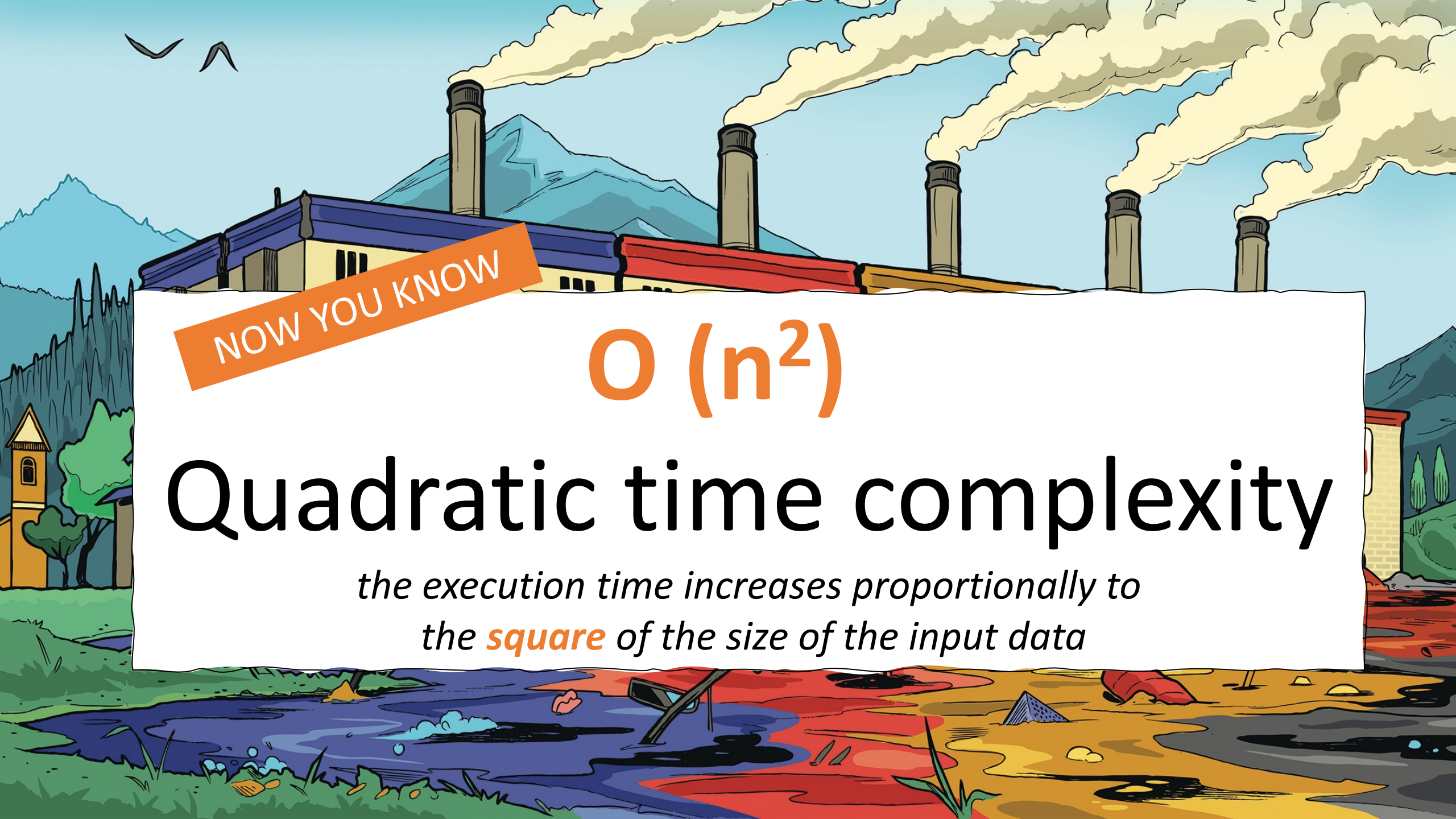
1 EO

```
for (int i = 0; i < n; i++)
```

INNER LOOP

```
for (int j = 0; j < i; i++)
    cout<<“CADT”;
```




NOW YOU KNOW
O (n²)
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 following the 4 steps

```
cout<<“Enter the value N:”;
cin>>n;


for (int i = 0; i < n*n; i++)


        for (int j = 0; j < i; i++)
                cout<<“CADT”;
```

SOLUTION

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

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

INIT BLOCK

```
cout<<“Enter the value N:”;
cin>>n;
```

OUTER LOOP

```
for (int i = 0; i < n*n; i++)
```

INNER LOOP

```
for (int j = 0; j < i; j++)
    cout<<“CADT”;
```

ITERATIONS PER BLOCK

X 1 → O (2)

$\frac{(n^2-1)n^2}{2}$ → O (n^4)

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

O (n^4)

O (n^4)

Quadratic complexity

ACTIVITY

Activity 6

Analyze the complexity of this algorithm following the 4 steps

```
cout<<“Enter the value N:”;
cin>>n;


for (int i = 0; i < n; i++)
        t = 1
```

INNER LOOP

```
        while (t<= n)
                cout<<“Hello”;
                t = t*2
```

SOLUTION

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

INIT BLOCK

```
cout<<"Enter the value N:";
cin>>n;
```

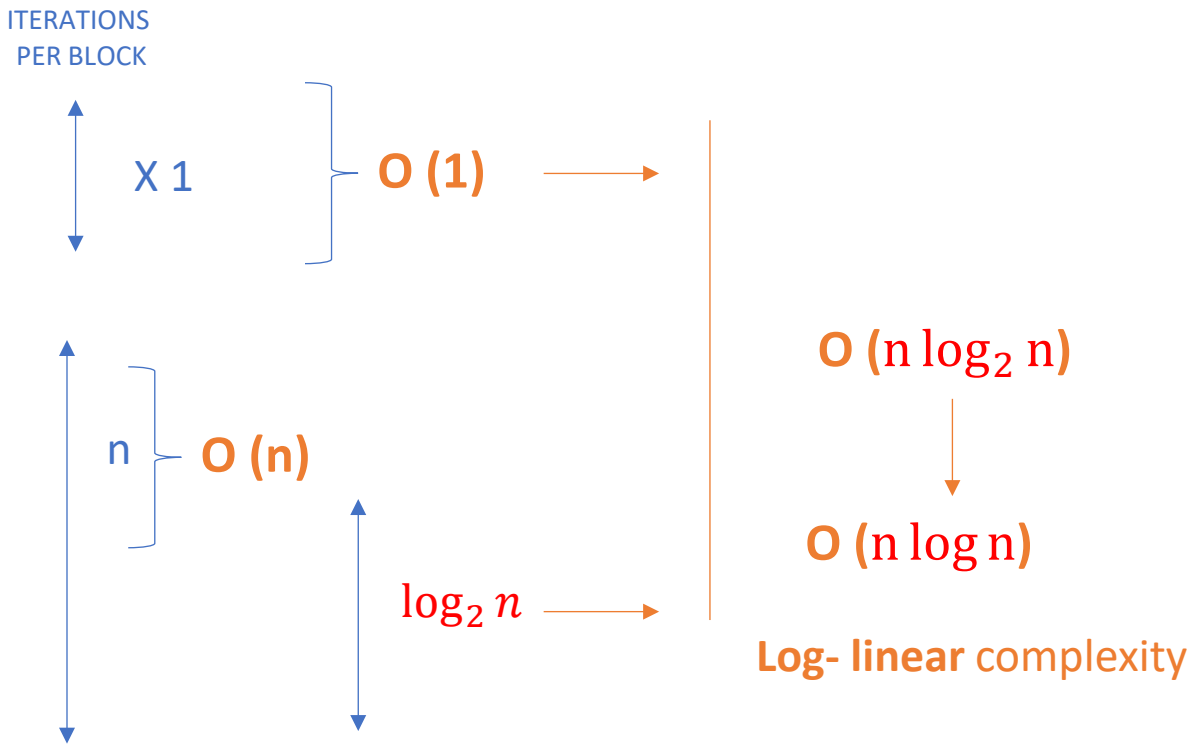
OUTER LOOP

```
for (int i = 0; i < n; i++)
        t = 1
```

INNER LOOP

4 EO

```
while (t<= n)
        cout<<"Hello";
        t = t*2
```



SOLUTION

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

```
cout<<“Enter the value N:”;
cin>>n;

for (int i = 0; i < n; i++)
        t = 1
```

INNER LOOP

```
   while (t<= n)
            cout<<“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 block grows **logarithmically**

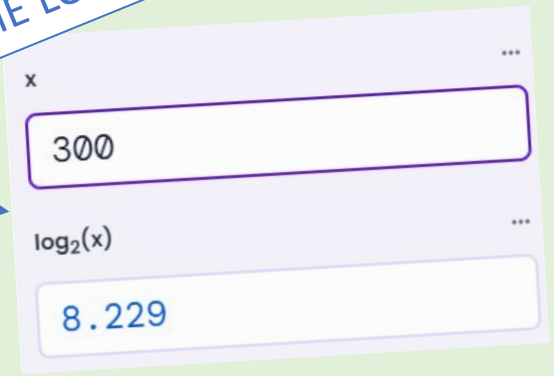
INNER LOOP

```
while (t<= n)
        cout<<“Hello”;
        t = t*2
```

Log**2** …. as we multiply by 2

Block	*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

INIT BLOCK — 2 EO

```
cout<<"Enter the value N:";
cin>>n;
```

OUTER LOOP — 2 EO

```
for (int i = 0; i < n; i++)
        t = 1
```

INNER LOOP — 4 EO

```
while (t<= n)
        cout<<"Hello";
        t = t*2
```

EO

2

4

ITERATIONS PER BLOCK

X 1

n

O (1)

O (n)

O (log n)

O (n log n)

Log- linear complexity

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

EXERCICES

EXERCICE 1

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;

isPrime = true

for (int i = 2; i < n-1; i++)
    if (n % i == 0)
        isPrime = false
        break

if (isPrime)
    cout<<“Prime number”;
else
    cout<<“Not a prime number”;
```

EXERCICE 2

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;

sum = 0

for (int i = 0; i < n; i++)
    sum = sum + i

cout<<“Sum is: ”<<sum<<endl;
```

EXERCICE 3

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;
cout<<“Enter the value X:”;
cin>>x;

left = 1
right = n

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

    if (array[mid] == x)
        cout<<“Element found at: ”<<mid<<end;
        break
    else if (array[mid] < x)
        left = mid + 1
    else
        right = mid - 1
```

EXERCICE 4

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;

for (int i = 0; i < n; i++)
    for (int j = 0; j < n-i-1; j++)
        if array[j] > array[j + 1] then
            temp = array[j]
            array[j] = array[j + 1]
            array[j + 1] = temp
```

EXERCICE 5

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;
cout<<“Enter the value X:”;
cin>>x;

count = 0

for (int i = 0; i < n; i++)
    if array[i] == x then
        count = count + 1

cout<<“Element found at: ”<<count<<end;
```

EXERCICE 6

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        result[i][j] = 0
        for (int k = 0; k < n; k++)
              result[i][j] = result[i][j] + matrix1[i][k] * matrix2[k][j]
```

EXERCICE 7

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;
cout<<“Enter the value X:”;
cin>>x;

found = false

for (int i = 0; i < n; i++)
    if (array[i] == x)
        found = true
        cout<<“Element found at: ”<<i<<end;
        break

if (!found)
       cout<<“Element not found”<<endl;
```

EXERCICE 8

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value of Radius:”;
cin>>radius;

area = 3.14f * radius * radius

cout<<“Area of the circle is: ”<area<<end;
```

EXERCICE 9

Evaluate the complexity based on the five types you have learned

```
cout<<“Enter the value N:”;
cin>>n;

maxValue = array[1]

for (int i = 1; i < n; i++)
    if (array[i] > maxValue)
       maxValue = array[i]

cout<<“Maximum value is: ”<<maxValue<<end
```

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

FOR NEXT TIME !

1 – At home

- Review the theory
- https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/
- Finalize the exercises

2 – Practice

- Sorting Algorithm

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**.