Totally fair — these terms sound abstract until you see **what they're actually measuring**. Let's ground this in your exact code and think like a machine, not a textbook.

1️⃣ What Time Complexity really means

Time complexity = how many steps the code runs as input grows

Not actual seconds. Just **how work increases**.

Your code

```
for (int i = 1; i < nums.length; i++) {
    sum = sum + nums[i];
    nums[i] = sum;
}
```

If array size = `n`

- Loop runs **once per element**
- For `n` elements → about `n` operations

So we say:

🩷 Time Complexity = O(n)

Real meaning:

- If array size doubles → work doubles
- If array size = 1,000 → ~1,000 steps
- If array size = 1,000,000 → ~1,000,000 steps

That's it. No magic.

2️⃣ What Space Complexity really means

Space complexity = extra memory used by your algorithm

⚠️ Important: We **do NOT count the input array** because it's already given.

What extra memory do you use?

```
int sum = nums[0];
```

That's it.

- `sum` → one integer
- `i` → loop variable

These do **not grow** with input size.

Whether array has:

- 5 elements
- 5 million elements

You still use:

- 1 `int`
- 1 loop variable

So we say:

🆗 Space Complexity = O(1) (constant space)

Real meaning:

- Extra memory stays **constant**
- Does not depend on `n`

3️⃣ Why we say "O(1) extra space"

Because:

- We reused the input array (`nums`)
- We did NOT create a new array like:

```
int[] result = new int[n]; // ❌ extra space O(n)
```

If you had done that, space would be **O(n)**.

4️⃣ Tiny comparison (this will click ⬜)

❌ Extra space O(n)

```
int[] result = new int[n];
```

Memory grows with input → O(n)

✅ Extra space O(1)

```
int sum = 0;
```

Memory stays same → O(1)

One-line intuition (remember this forever)

- **O(n) time** → loop depends on input size
- **O(1) space** → only fixed number of variables

Space complexity is about growth, not absolute memory. A small fixed variable counts as **O(1) constant space** even though it technically uses memory.

So we consider space complexity constant even if we take extra variable or array **till it's not growing with input**.

- **Extra variable(s)** (like `sum`, `i`, `temp`) → **O(1)** because they exist only once, no matter how big the input is.
- **Extra array / collection** whose size depends on input `n` → **O(n)** because memory grows with input.

So the rule is:

> **Space complexity = how memory usage grows with input size.** If it doesn't grow, it's constant → O(1).

Even if you declare 2–3 extra variables, it's still O(1) — only **memory proportional to `n` or more** counts as O(n), $O(n^2)$, etc.

You got it perfectly.