🔥 Framework for Explaining Code While Solving Problems

Use this **4-step approach** to structure your explanation:

1️⃣ Understand the Problem (Clarify & Break Down)

- **Repeat the problem in your own words.**
- Identify **inputs, outputs, and constraints**.
- Ask clarifying questions if needed.

✅ **Example for Two Sum:**
"We need to find two numbers in an array that sum to a target value. The function should return their indices. Given an array `[2, 7, 11, 15]` and target `9`, we return `[0, 1]` since `2 + 7 = 9`."

2️⃣ Discuss the Naïve Approach (Brute Force First)

- Walk through a basic solution.
- Highlight inefficiencies.

✅ **Example:**
*"A brute-force approach would be checking all pairs using two loops, but this has **O(n²)** time complexity, which is too slow for large inputs."*

python
CopierModifier

```
def two_sum(nums, target):
    for i in range(len(nums)):  # First number
        for j in range(i + 1, len(nums)):  # Second number
            if nums[i] + nums[j] == target:
                return [i, j]
```

🔴 **Issue: Too slow!**
✅ **We need a better approach. Let's optimize!**

3️⃣ Introduce an Optimized Approach (Think Data Structures & Patterns)

- Pick the best **data structure** for speed.
- Mention why it's better.

✅ **Example: Use HashMap for O(1) Lookup**
*"Instead of checking every pair, we store numbers in a **hashmap** so we can check in constant time whether the complement exists."*

python
CopierModifier

```
def two_sum(nums, target):
    hashmap = {}  # Dictionary to store seen numbers
    for i, num in enumerate(nums):  # Go through the list
        diff = target - num  # Find the complement
        if diff in hashmap:  # If complement exists, return indices
            return [hashmap[diff], i]
        hashmap[num] = i  # Store the number with its index
```

✅ **Key Takeaway:** "By using a hashmap, we reduce the time complexity from **O(n²) to O(n)**."

4️⃣ Walk Through an Example (Dry Run) & Edge Cases

- Pick an example and simulate the code execution.
- Mention edge cases:
  - Empty array?
  - Negative numbers?
  - Multiple solutions?

✅ **Example Dry Run:**
"Let's run it on `[2, 7, 11, 15]` with `target = 9`."

Step	num	diff (target - num)	hashmap	Action
1	2	7	`{}`	Store `{2: 0}`
2	7	2	`{2: 0}`	Match found! Return `[0, 1]`

✅ **Edge Cases Considered:**

- **Duplicates**: Does the solution handle `[3, 3]` for target `6`?
- **No Solution**: What if no two numbers sum to `target`?

🛠️ Cheat Sheet: How to Explain Different Algorithm Types

Algorithm Type	How to Explain During an Interview
Brute Force Approach	"First, I'll try a simple solution using loops. This takes O(n²) time because I check all pairs."
Optimized Approach	"Instead of brute force, I'll use a hashmap to reduce lookup time to O(1), making the solution O(n)."
Sorting-Based	"Sorting helps because once sorted, I can use two pointers instead of nested loops."
Sliding Window	"Instead of checking all substrings, I'll maintain a moving window to keep track of valid ones."
Dynamic Programming	"I'll store results of subproblems to avoid redundant calculations, improving efficiency."
Heap / Priority Queue	"A heap helps keep track of the top k elements efficiently in O(log n) instead of sorting O(n log n)."
Binary Search	"Since the array is sorted, I'll use binary search instead of linear search for O(log n) time."

🔥 Example: How to Explain Sliding Window Algorithm (Longest Substring Without Repeating Characters)

✅ Step 1: Explain Problem
"Find the longest substring without repeating characters. Given `'abcabcbb'`, the longest is `'abc'` with length `3`."

✅ Step 2: Brute Force & Why It's Slow
"I can check all substrings, but this is O(n³), which is too slow."

✅ Step 3: Optimized Solution - Sliding Window (O(n))
*"Instead of checking all substrings, I'll use a **moving window** and a hashmap to track characters."*

python
CopierModifier

```
def length_of_longest_substring(s):
    char_map = {}
    left = 0  # Left pointer
    max_length = 0
```

```
    for right, char in enumerate(s):  # Expand right
        if char in char_map and char_map[char] >= left:
            left = char_map[char] + 1  # Move left pointer
        char_map[char] = right
        max_length = max(max_length, right - left + 1)

    return max_length
```

✅ Step 4: Dry Run

Step	Window (`s[left:right]`)	Unique?	Action
1	`'a'`	✅	Expand right
2	`'ab'`	✅	Expand right
3	`'abc'`	✅	Expand right
4	`'abca'`	❌ (repeat)	Move left

✅ Step 5: Edge Cases

- **Empty string `""`?** ✅ Returns `0`
- **All unique characters?** ✅ Returns `len(s)`

🔥 Final Tips for Explaining Code in Interviews

1️⃣ **Talk out loud** while coding – don't just write silently.
2️⃣ **Follow the structured approach** (Understand → Brute Force → Optimize → Walkthrough).
3️⃣ **Use examples** to show correctness & edge cases.
4️⃣ **Keep explanations simple** – avoid unnecessary details.
5️⃣ **Practice with LeetCode Easy/Medium problems** while saying your thought process aloud.