🧠Two Pointers — One-Page Recognition Cheat Sheet

> **Goal:** Instantly recognize *when* to use two pointers, *which type*, and *which loop* — without thinking about specific problems.

👇The Golden Rule

> **If pointer movement depends on logic →** `while`**.**
> **If one pointer simply scans once →** `for`**.**

🤩Converging Pointers (Opposite Ends)

How it looks
- One pointer starts at the **beginning**
- One pointer starts at the **end**
- They move **toward each other**

Recognition clues - “Find a pair” / “check two values together”
- Array or string is **sorted** (or order matters)
- You can **discard one side** after a comparison

Core intuition

> Start wide, then shrink the search space intelligently.

Pointer movement logic - Need a bigger result → move left pointer right
- Need a smaller result → move right pointer left

Code pattern

```
left = 0
right = n - 1
while left < right:
    if condition_met:
        return result
    elif need_bigger:
        left += 1
```

```
    else:
        right -= 1
```

Loop: `while left < right`

😜Same-Direction Pointers (Slow–Fast, Same Speed)

How it looks
- Both pointers move **left → right**
- One pointer *reads*, the other *writes*

Recognition clues - “Modify array in place”
- “Remove / keep / compress elements”
- Final result is a **prefix** of the array

Core intuition

> One pointer explores, the other builds the answer.

Pointer roles - `fast` → scans every element
- `slow` → tracks where the next valid value goes

Code pattern

```
slow = 0
for fast in range(n):
    if keep(nums[fast]):
        nums[slow] = nums[fast]
        slow += 1
```

Loop: `for` (fast pointer)

🤔Same-Direction Pointers (Slow–Fast, Same Speed)

How it looks
- Both pointers move **left → right**
- One pointer *reads*, the other *writes*

Recognition clues - “Modify array in place”
- “Remove / keep / compress elements”
- Final result is a **prefix** of the array

Core intuition

> One pointer explores, the other builds the answer.

Pointer roles - `fast` → scans every element
- `slow` → tracks where the next valid value goes

Loop: `for` (fast pointer)

🤔Same-Direction Pointers (Slow–Fast, Different Speed)

How it looks
- Both pointers start at the same place
- One moves **faster** than the other

Recognition clues - Mentions of **cycles**, **loops**, or **meeting points**
- Works on **linked lists** or number sequences
- You are not comparing values — only movement

Core intuition

> Speed difference reveals structure (cycle or midpoint).

Pointer movement - Slow: 1 step
- Fast: 2 steps

Loop: `while fast and fast.next`

😈Sliding Window (Expand + Shrink)

How it looks
- Left and right pointers form a **window**
- Window grows, then shrinks

Recognition clues - “Subarray” / “substring”
- “Longest / shortest / maximum / minimum”
- Condition becomes valid or invalid dynamically

Core intuition

> Grow until broken, shrink until fixed.

Pointer roles - Right pointer → expands window
- Left pointer → shrinks window

Loop: `for` (expand) + `while` (shrink)

🤭Partitioning Pointers

How it looks
- Two pointers rearrange elements **in place**
- Often involves swapping

Recognition clues - “Group elements by condition”
- “Reorder without extra space”
- Not fully sorting — just separating

Core intuition

> Push bad elements to one side, good to the other.

Pointer behavior - One pointer scans for misplaced elements
- Other pointer marks boundary

Loop: `while left <= right`

🤮Fixed + Two Moving Pointers (Nested)

How it looks
- One index is **fixed**
- Two pointers search around it

Recognition clues - “Find combinations of 3 or more”
- One value chosen, others adjusted

Core intuition

> Reduce a multi-value problem into a two-value problem.

Loop - Outer: `for` (fix one value)
- Inner: `while` (two pointers)

👎🏿 Ultra-Fast Recognition Table

Question pattern	Pointer type	Loop
Pair from sorted data	Converging	`while`
In-place removal	Same-dir (same speed)	`for`
Cycle / midpoint	Slow–fast (diff speed)	`while`
Subarray / substring	Sliding window	`for` + `while`
Grouping / rearranging	Partitioning	`while`

 Final Memory Hook

Two pointers = coordinated movement
Loop choice = who controls movement
Patterns > syntax

Print this. Read it before solving. Recognition will become automatic.