

What is the Fast and Slow Pointers Pattern?

The **Fast and Slow Pointers pattern** (also known as **Floyd's Cycle Detection Algorithm** or the **Tortoise and Hare technique**) is a two-pointer strategy used to efficiently traverse data structures where:

- Elements are **connected in a sequence** (e.g., linked list, number transformations, circular arrays).
- You can "move" step by step through the structure.

You use:

- Slow Pointer: moves **1 step at a time**.
- Fast Pointer: moves **2 steps at a time**.

Where is this pattern useful?

This pattern is typically used in problems involving:

1. **Cycle detection** (in linked lists, number sequences, etc.)
2. **Finding the middle of a list**
3. **Detecting palindromes in linked lists**
4. **Finding the start of a loop**
5. **Math-based problems that create a repetitive cycle** (e.g., Happy Number)

How to Recognize When to Apply Fast and Slow Pointers

Look for these signs:

Signal	Description
Repetition	You're repeatedly transforming or traversing values (e.g., linked list traversal, square-sum of digits).
Cycle possibility	The structure or transformation could lead to a loop (like a cycle in a linked list or in a number chain).
Need for midpoint	You need to find the middle element in one pass.
O(1) space required	You're asked to solve in constant space — fast & slow pointers do not use extra memory.

How to Apply the Pattern – Step-by-Step

Step 1: Initialize two pointers

```
slow = head;
fast = head;
```

Step 2: Traverse the structure

- Move slow one step.
- Move fast two steps.

```
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
    ...
}
```

Step 3: Watch for a meeting point

- If `slow == fast` → there is a cycle (for detection problems).
- If fast hits null → no cycle (or list has ended).

Step 4: Additional logic

Depending on the goal:







- If looking for **cycle start**, reset slow and move both one step until they meet.
- If looking for **middle**, return slow when fast hits end.
- If looking for **cycle** in transformations (e.g., Happy Number), treat transformation as “next” movement.







Benefits of Fast and Slow Pointers

Benefit	Explanation
O(n) Time	Linear traversal — no nested loops needed
O(1) Space	Only two pointers used — no hashmap or extra storage
No modification	Works on original structure without altering it
Midpoint access	Efficient way to find middle of list
Cycle detection	Elegant way to detect loop existence or entry point

Example Use Case Patterns

Use Case	Question Type	What to Observe
Detect cycle	Linked List	Traverse nodes, check if slow == fast
Find loop start	Linked List	After meeting, reset slow to head
Find middle	Linked List	Stop when fast == null
Happy Number	Integer Problem	Apply next-transform function repeatedly
Circular array loop	Array + Modulo	Move with $(i + \text{nums}[i]) \% n$, apply cycle logic

#	Problem Name	Topic	Platform	Link
1	Linked List Cycle Detection	Detect cycle	LeetCode	Link 
2	Linked List Cycle II	Find start of cycle	LeetCode	Link 
3	Middle of the Linked List	Find middle node	LeetCode	Link 
4	Happy Number	Detect cycle in number transformation	LeetCode 	
5	Palindrome Linked List	Check if palindrome	LeetCode	Link 
6	Remove Nth Node From	Use two-pointer gap	LeetCode	Link 

	End			
7	Reorder List	Rearranging with middle	LeetCode	Link 
8	Find the Duplicate Number	Cycle detection in array	LeetCode	Link 
9	Circular Array Loop	Detect loop with direction	LeetCode	Link 
10	Detect Cycle in Circular Linked List	Custom implementation	GFG 	
11	Intersection of Two Linked Lists	Cycle trick (alternative method)	LeetCode	Link 
12	Merge in Between Linked Lists	Mid finding and pointer manipulation	LeetCode	Link 
13	Find Middle of Circular Linked List	Variation of middle node	GFG	Link 
14	Sort List (Merge Sort)	Midpoint splitting	LeetCode	Link 
15	Detect Loop Length in Linked List	Count cycle length	GFG	Link 