

FAST & SLOW POINTER PATTERN – DETAILED NOTES (COPY READY)

1 Fast & Slow Pointer kya hota hai?

Fast & Slow Pointer ek **DSA traversal technique** hai jisme do pointers same starting node se move karte hain, lekin **different speed** se.

- **Slow Pointer** → 1 step move karta hai
- **Fast Pointer** → 2 steps move karta hai

Ye technique mainly **linear data structures** ke liye use hoti hai:

- Linked List
- Array (index-based movement)
- String

👉 Ye pattern **comparison** ke liye nahi, balki **structure detect** karne ke liye hota hai.

2 Intuition (Samajhne ka tareeka)

Sochiye:

- Slow = 10 km/hr
- Fast = 20 km/hr

Case 1: Road straight hai

Fast kabhi slow ko nahi pakdega → **NO cycle**

Case 2: Road circular hai

Fast ghoom kar piche se slow ko pakad lega → **Cycle exists**

👉 Isi logic pe **cycle detection** kaam karta hai.

3 Fast = 2 steps hi kyun?

- Agar fast = 1 step → slow ke saath hi chalega (kabhi meet nahi)
- Agar fast ≥ 3 steps → slow ko skip kar sakta hai ✗
- **Fast = 2 steps** → guaranteed meeting in cycle ✓

Istiyeh **Floyd's Algorithm** me fast hamesha 2 step leta hai.

4 Brute Force Approach (Cycle Detection in Linked List)

Idea

- Har visited node ka **address** store karo
- Agar koi node dobara mile \rightarrow cycle exist karti hai

Algorithm

1. Empty set banao
 2. Head se traversal start karo
 3. Har node ke liye:
 - Agar node set me hai \rightarrow cycle
 - Nahi hai \rightarrow set me daal do
 4. Agar NULL aa gaya \rightarrow no cycle
-

⌚ Complexity

- **Time:** $O(n)$
 - **Space:** $O(n)$  (extra memory)
-

✓ Brute Force C++ Code

```

#include <unordered_set>
using namespace std;

class ListNode {
public:
    int val;
    ListNode* next;
    ListNode(int x) {
        val = x;
        next = NULL;
    }
};

bool hasCycle(ListNode* head) {
    unordered_set<ListNode*> visited;

    while (head != NULL) {
        if (visited.count(head)) {
            return true; // cycle detected
        }
        visited.insert(head);
        head = head->next;
    }
    return false; // no cycle
}

```

5 Optimised Approach (Fast & Slow Pointer)

Idea

- Slow → 1 step
- Fast → 2 steps
- Agar dono kabhi same node pe aa gaye → cycle

Algorithm (Step-by-step)

1. slow = head, fast = head

2. Jab tak:

- fast != NULL
- fast->next != NULL

3. Loop me:

- slow = slow->next

- fast = fast->next->next
4. Agar slow == fast → cycle detected
5. Loop exit ho jaye → no cycle
-

⌚ Complexity

- **Time:** O(n)
 - **Space:** O(1) 
-

✓ Optimised C++ Code

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if (head == NULL) return false;

        ListNode* slow = head;
        ListNode* fast = head;

        while (fast != NULL && fast->next != NULL) {
            slow = slow->next;           // 1 step
            fast = fast->next->next;    // 2 steps

            if (slow == fast) {
                return true;             // cycle detected
            }
        }
        return false; // no cycle
    }
};
```

6 Common Mistake (INTERVIEW FAVORITE)

✗ Galat condition:

```
cpp

while (fast != NULL || fast->next != NULL)
```

❗ Problem:
Agar fast == NULL hua aur fast->next access kiya → **runtime error**

 Correct:



```
while (fast != NULL && fast->next != NULL)
```

7 Fast & Slow Pointer Template (YAAD KAR LO)

```
slow = head;
fast = head;

while (fast != NULL && fast->next != NULL) {
    slow = slow->next;
    fast = fast->next->next;

    if (slow == fast) {
        // cycle / condition found
    }
}
```

8 Fast & Slow vs Two Pointer

Two Pointer

Different start points

Condition based movement

Comparison problems

Palindrome, pair sum

Fast & Slow Pointer

Same start

Fixed speed

Structural problems

Cycle, middle

9 Interview Keywords (Signal Words)

Agar question me ye words aaye:

- cycle / loop
- circular linked list
- middle element
- repeated node

 Turant **Fast & Slow Pointer** socho.

10 Final Summary (1–2 Lines)

- Brute force = easy but extra space
- Fast & Slow = **best**, $O(1)$ space
- Floyd's Algorithm = must-know for interviews