🔎 Searching an Element in a Linked List

🧠 First, Intuition (Think Like This)

Imagine 5 people standing in a line.
Each person holds the shoulder of the next person.

If you want to find **Rahul**, what will you do?

You:

1. Start from the first person.
2. Ask: “Are you Rahul?”
3. If no → move to next.
4. Repeat until:
    - You find Rahul ✅
    - Or line ends ❌

That’s exactly how searching works in a **Linked List**.

📌 What is Happening Internally?

In a linked list:

- Each node has:
    - data
    - next (pointer to next node)

To search:

- Start from head
- Traverse node by node
- Compare current.data == key
- If found → return position / true
- If reach NULL → not found

⏱️ Time & Space Complexity

- **Time Complexity:** O(n)
  (Worst case → traverse entire list)
- **Space Complexity:** O(1)
  (No extra memory used)

Unlike arrays ❗
Binary search is NOT possible because:

- No random access
- Must move sequentially

🔍 **Algorithm (Logic Steps)**

1. If head == NULL → return false
2. Set current = head
3. While current != NULL:
    - If current.data == key → return true
    - Move current = current.next
4. Return false

🧪 Dry Run Example

Linked List:

10 → 20 → 30 → 40 → NULL

Search Key = 30

Step	Current Node	Comparison	Result
1	10	10==30 ❌	Move
2	20	20==30 ❌	Move
3	30	30==30 ✅	Found

Return → TRUE

If Search Key = 50

Step	Current Node	Comparison	Result
1	10	❌	Move
2	20	❌	Move
3	30	❌	Move
4	40	❌	Move

Step Current Node Comparison Result

5 NULL Stop Not Found

Return → FALSE

💻 **Code Implementations**

✅ **C++ Code**

```
#include <iostream>
using namespace std;

class Node {
public:
   int data;
   Node* next;

   Node(int val) {
      data = val;
      next = NULL;
   }
};

bool search(Node* head, int key) {
   Node* current = head;

   while (current != NULL) {
      if (current->data == key) {
         return true;
      }
      current = current->next;
   }
```

```
    return false;
}

int main() {
    Node* head = new Node(10);
    head->next = new Node(20);
    head->next->next = new Node(30);
    head->next->next->next = new Node(40);

    int key = 30;

    if (search(head, key))
        cout << "Element Found";
    else
        cout << "Element Not Found";

    return 0;
}
```

✅ **Java Code**

```
class Node {
    int data;
    Node next;

    Node(int val) {
        data = val;
        next = null;
    }
}
```

```
public class LinkedListSearch {

    static boolean search(Node head, int key) {
        Node current = head;

        while (current != null) {
            if (current.data == key) {
                return true;
            }
            current = current.next;
        }

        return false;
    }

    public static void main(String[] args) {
        Node head = new Node(10);
        head.next = new Node(20);
        head.next.next = new Node(30);
        head.next.next.next = new Node(40);

        int key = 30;

        if (search(head, key))
            System.out.println("Element Found");
        else
            System.out.println("Element Not Found");
    }
}
```

✅ **Python Code**

```
class Node:
    def __init__(self, val):
        self.data = val
        self.next = None

def search(head, key):
    current = head

    while current:
        if current.data == key:
            return True
        current = current.next

    return False

# Creating Linked List
head = Node(10)
head.next = Node(20)
head.next.next = Node(30)
head.next.next.next = Node(40)

key = 30

if search(head, key):
    print("Element Found")
else:
    print("Element Not Found")
```

🎯 **Why Searching is Slower in Linked List Than Array?**

Array	**Linked List**

Array	**Linked List**
Random Access	Sequential Access
Can do Binary Search	Cannot
O(log n) possible	Always O(n)