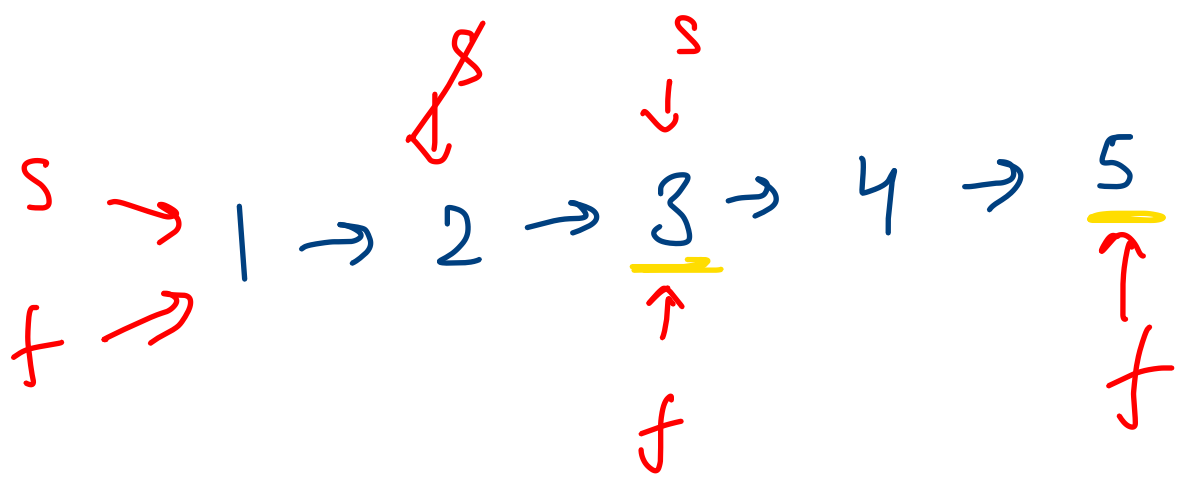
# Mid-point of LL

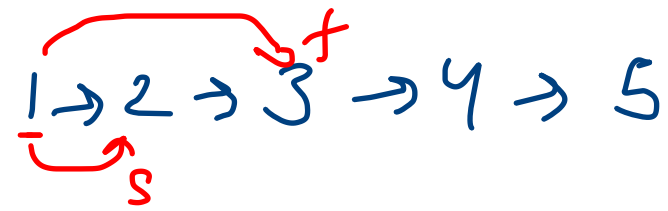
$1 \to 2 \to \boxed{3} \to 4 \to 5 \to NULL$ — odd length (5)

$1 \to 2 \to \boxed{3} \to 4 \to 5 \to 6$ — even length (6)

slow / fast → pointers




→ fast is moving 2x speed of slow, so when fast will at last position, slow will be halfway that is mid point.




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

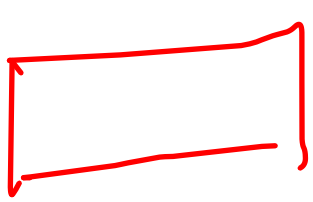
```java
public static Node<Integer> midPoint(Node<Integer> head){
    // base case
    if(head == null || head.next == null){
        return head;
    }

    Node<Integer> slow = head;
    Node<Integer> fast = head;
    while(fast.next != null && fast.next.next != null){
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;

}
```

TC- O(n)

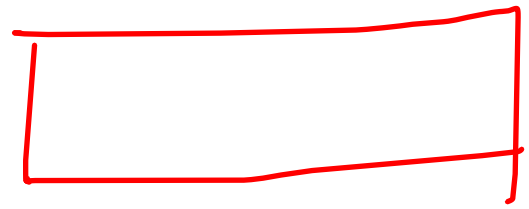
SC - O(1)

```java
public static Node<Integer> takeInput(){
    Node<Integer> head = null , tail = null;
    Scanner s = new Scanner(System.in);
    int data = s.nextInt();

    while(data != -1){
        Node<Integer> newNode = new Node<Integer> (data);
        if(head == null){
            head = newNode;
            tail = newNode;
        }
        else{
            tail.next = newNode;
            tail = newNode;
        }
        data = s.nextInt();
    }
    return head;
}
```

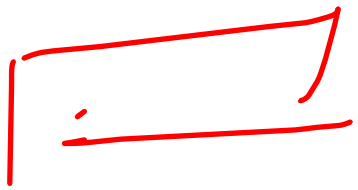
```java
import java.util.Scanner;
import java.util.LinkedList;

class Node<T>{
    T data;
    Node<T> next;

    Node(T data){
        this.data = data;
        next = null;
    }
}

class Main {
    public static void main(String[] args) {
        Node<Integer> head = takeInput();
        Node<Integer> data = midPoint(head);
        System.out.println(data.data);
    }
```



— arr 1



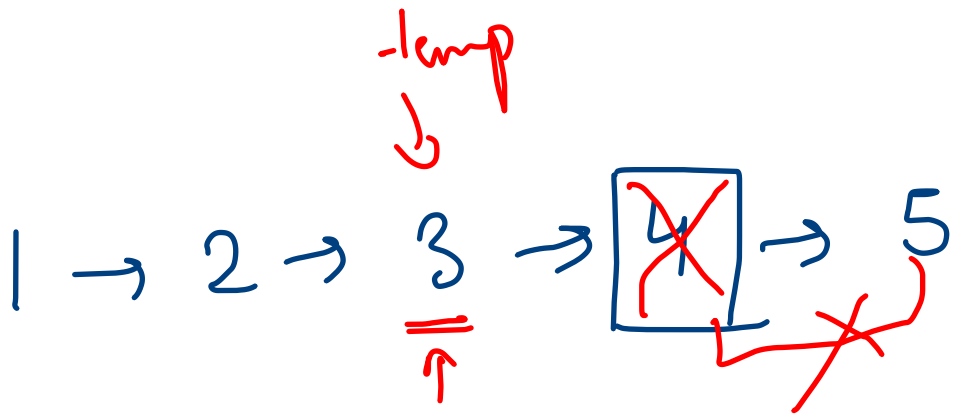
— $(n+m)$ $O(n)$



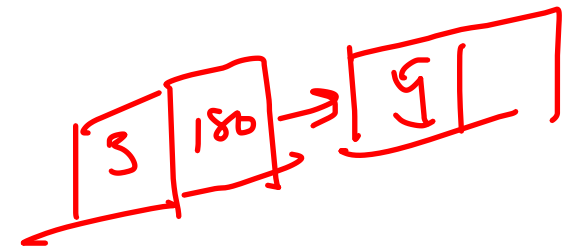
— arr 2

SC → extra or auxiliary space required apart from given

# Remove node from LL




temp.next = temp.next.next




first make

then break

# Remove nth node from last

1. Initialize two pointers

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

2. Move fast pointer n steps ahead

```
while (n--> 0)
    fast = fast.next;

if (fast == null)
    return head.next
```

3. Move both pointers until fast reaches end.

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

4. Delete the node

```
slow.next = slow.next.next;
```

# Dry Run

$1 \to 2 \to 3 \to 4 \to 5$

(s, f)

n = 2, 1, 0

slow = 1

fast = ~~1~~ ~~2~~ 3

2 > 0 T

1 > 0 T

0 > 0 F

## After 1st move

$1 \to 2 \to 3 \to 4 \to 5$

f

## After 2nd move

$1 \to 2 \to 3 \to 4 \to 5$

s

f

$1 \to 2 \to 3 \to 4 \to 5$

$\cancel{s}$ $\quad s \quad \cancel{f} \quad f$

Move 1 $\to$ $s \to 2$

$f \to 4$

Move 2 $\to$ $s \to 3$

$f \to 5$

$1 \to 2 \to 3 \to \cancel{4} \to 5$

$s \quad f$

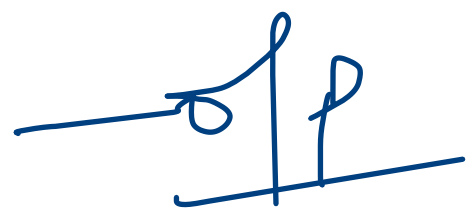
$3 \to 4$

$3 \to 5$

$1 \to 2 \to 3 \to 5$

$s$

Head $\to$ $1 \to 2 \to 3 \to 5$



```java
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode slow = head;
        ListNode fast = head;

        while(n-- > 0){
            fast= fast.next;
        }
        if(fast == null){
            return head.next;
        }

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

TC → O(n)

SC → O(1)