# Middle of the linked list

## 876. Middle of the Linked List

Easy 9807 289 Add to List Share

Given the `head` of a singly linked list, return *the middle node of the linked list.*

If there are two middle nodes, return **the second middle** node.

**Example 1:**


```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

**Example 2:**


```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.
```

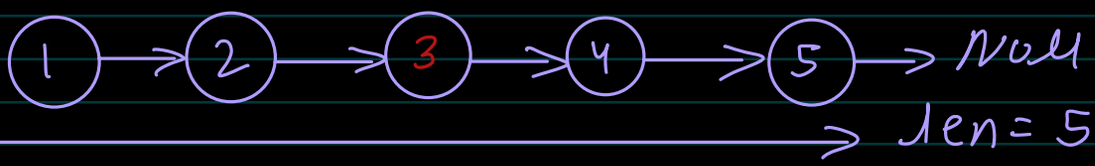
# Brute force :-

1st approch will be to Find the length of Linked List

so

→ Create a temp node and a variable to count the length of LL traverse to the end

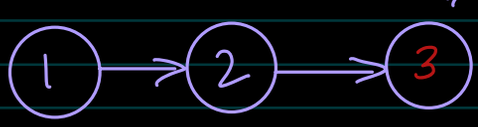
→ Now we got the length run another loop to traverse to the middle element.




Now len = len/2 ;

while (len --)

temp




return

```cpp
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* temp=head;
        int len=0;
        while(temp!=NULL){
            len++;
            temp=temp->next;
        }
        len=len/2;
        temp=head;
        while(len--){
            temp=temp->next;
        }
        return temp;

    }
};
```

T.C - O(n) + O(n/2)

S.C - O(1)

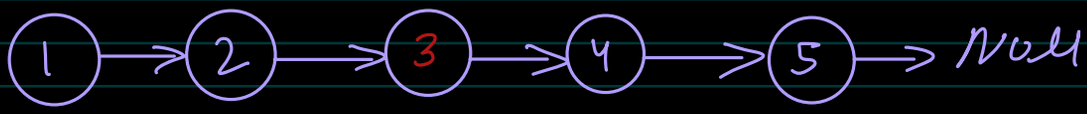
# Optimal approch:-

Fast and slow pointer approch

Slow → which moves ahead by 1 Node per iteration

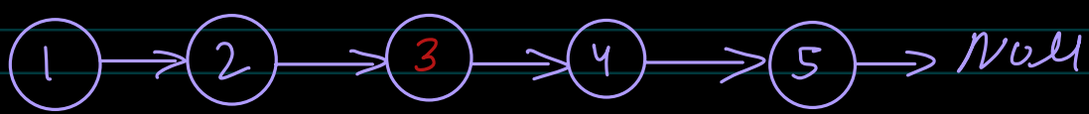
Fast → which moves ahead by 2 Node per iteration

1st iteration



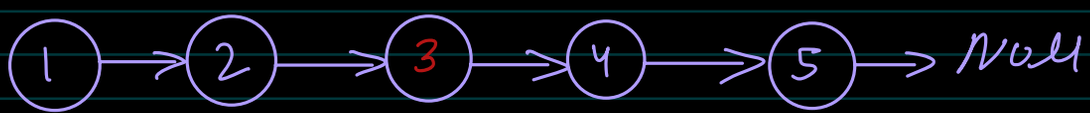

Slow Fast

2st iteration while (Fast != Null && Fast->next != Null)




Slow Fast

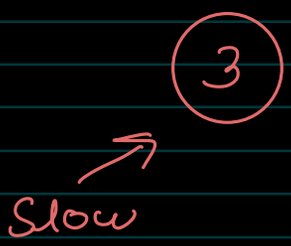
3nd




Slow Fast

while (Fast != Null && Fast->next != Null) Fals

Now Since fast Reaches to the end so slow must in middle




return Slow

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* Slow=head;
        ListNode* Fast=head;
        while(Fast!=NULL && Fast->next!=NULL){
            Slow=Slow->next;
            Fast=Fast->next->next;
        }
        return Slow;

    }
};
```

T.C - O(n)

S.C - O(1)