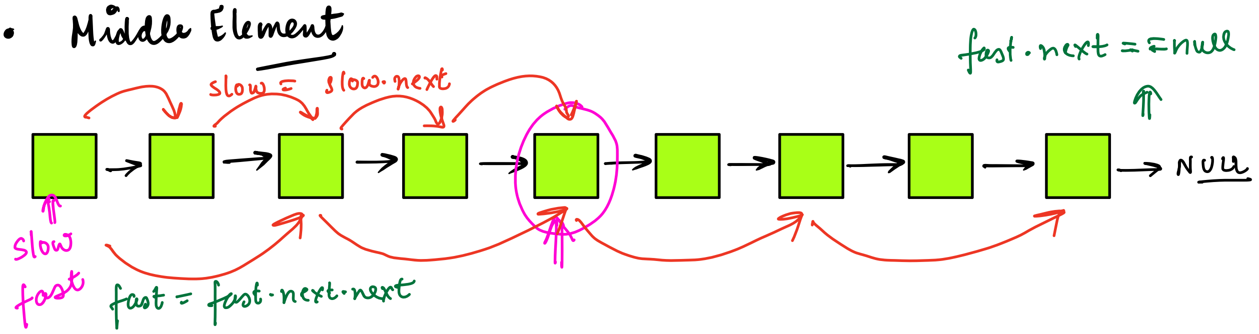
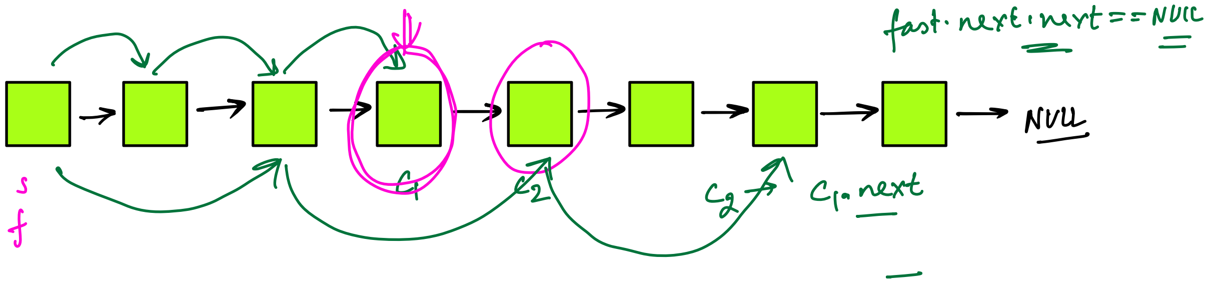
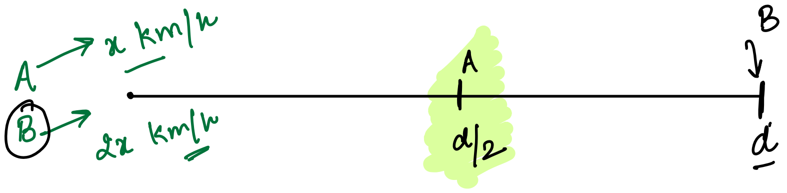
- ## Middle Element





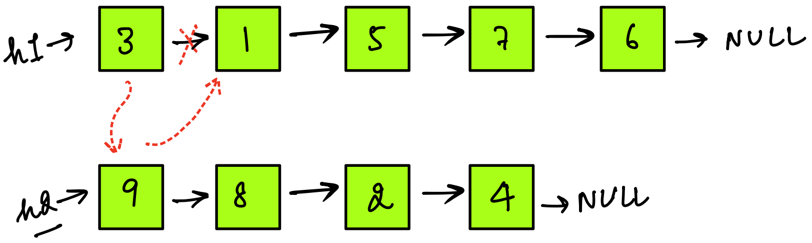
1) Find the length of LL → simple traversal ≡ N

2) $\frac{N}{2}$th element of LL



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

Q Merge 2 LL.



len(h1) >= len(h2)

3 → 9 → 1 → 8 → 5 → 2 → 7 → 4 → 6

```
curr = h1

while ( h2 != null)                      curr != null &
{   temp = h2;                                 h2 != null
    h2 = h2.next;
     temp.next = curr.next;
      curr.next = temp;
      curr = curr.next.next          // curr = temp.next
}
return h1;
```

- Reorder the LL.

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9

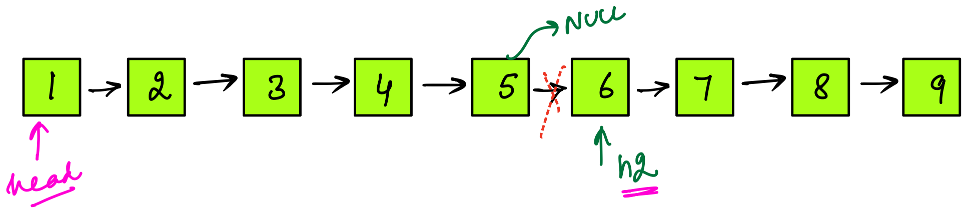
⇓ Reorder

1 → 9 → 2 → 8 → 3 → 7 → 4 → 6 → 5

$$a_1 \to a_2 \to a_3 \to a_4 \to a_5 \; . \; . \; . \; . \quad a_{n-3} \to a_{n-2} \to a_{n-1} \to a_n$$

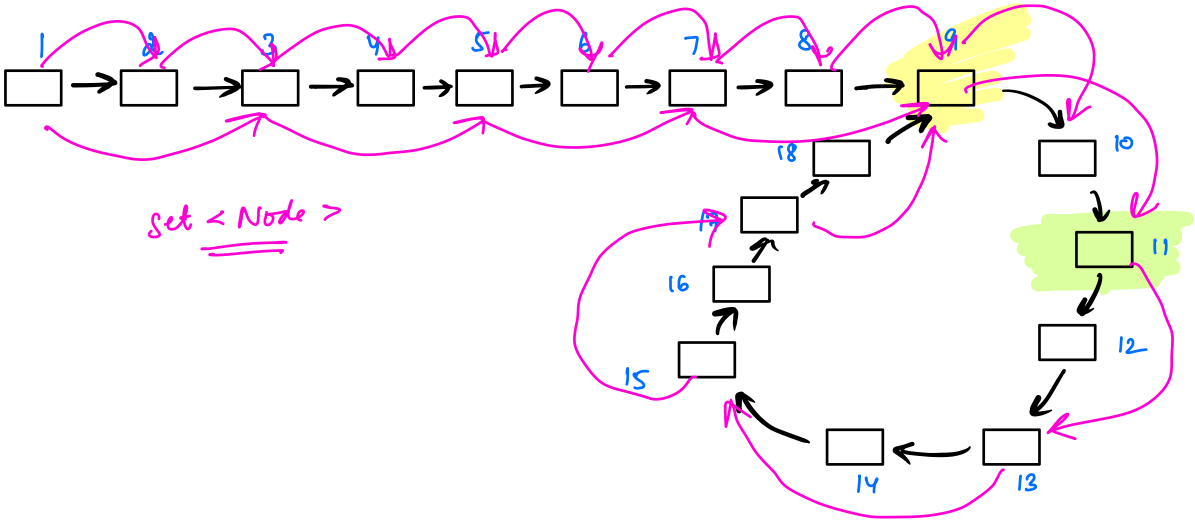
$$\Downarrow$$

$$a_1 \to a_n \to a_2 \to a_{n-1} \to a_3 \to a_{n-2} \; \cdots$$

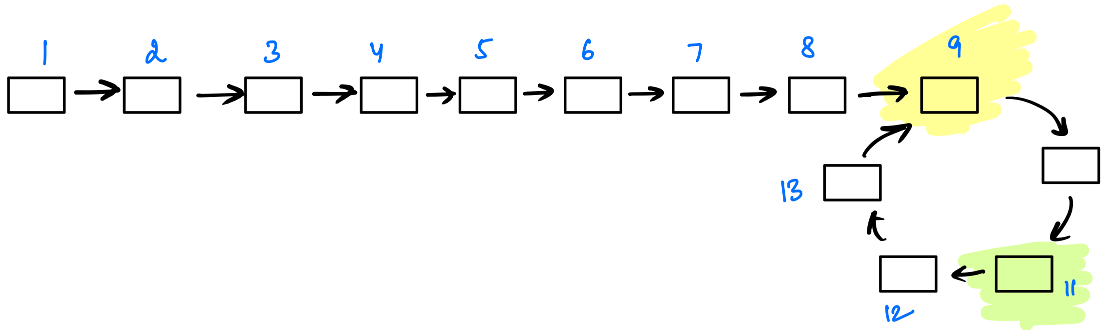


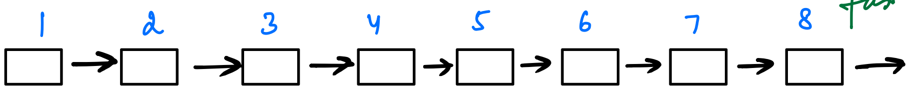
```
Node m = middle(head);
Node h2 = m.next;
     m.next = null;
        h2 = reverse(h2);
     return      merge(h1, h2);
```

Q LL , detect if there is a loop inside the LL.

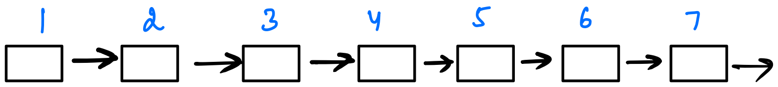


Set < Node >

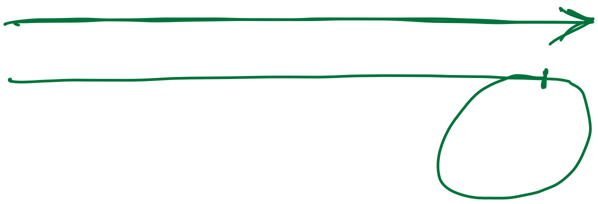




fast.next.next == null



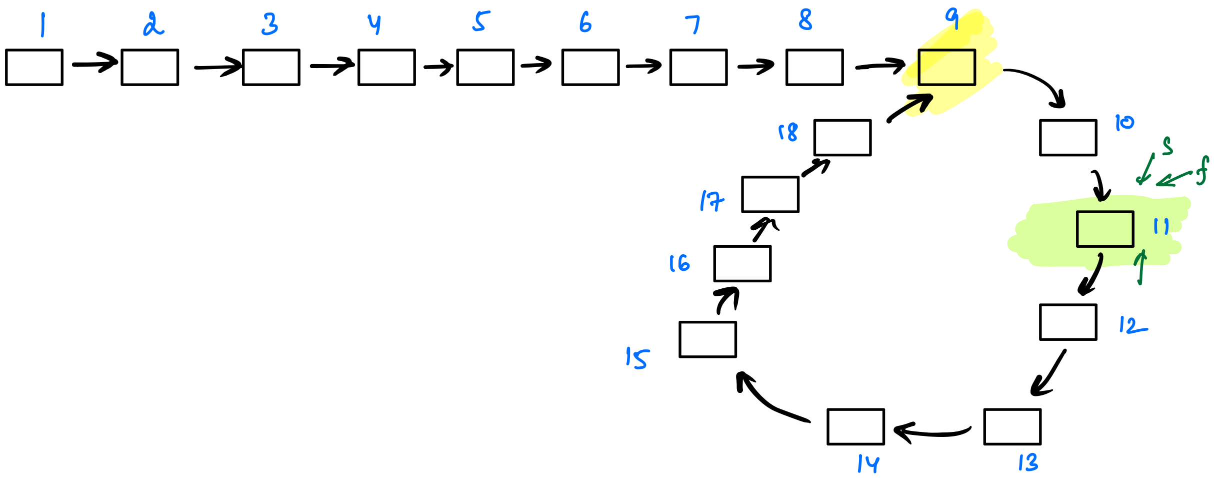
A – cycle
B – Bike

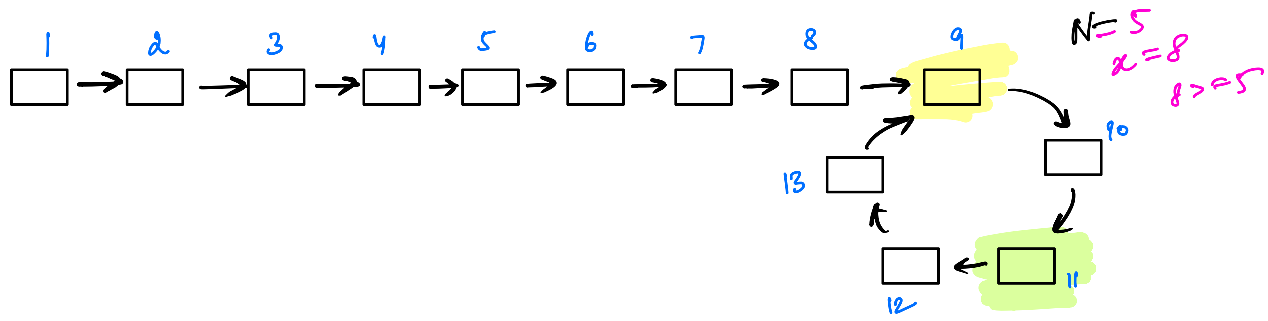


you take slow / fast pointers, if there exists a loop, these two will definitely meet

```
if( head == null || head->nxt == null )return false;

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

Start point of the loop!





After slow & fast meet together, Take one pointer at head and one pointer at meeting point, move these with slow speed, They will again meet at starting pt of loop.

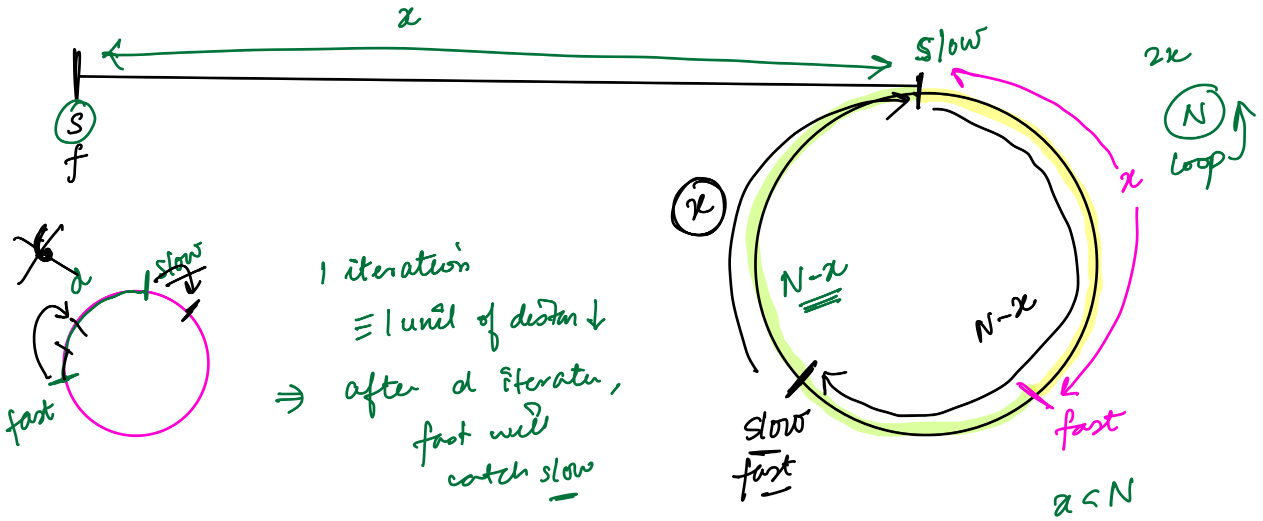
+

```
// get slow & fast at your meeting pt.

    slow= head;

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

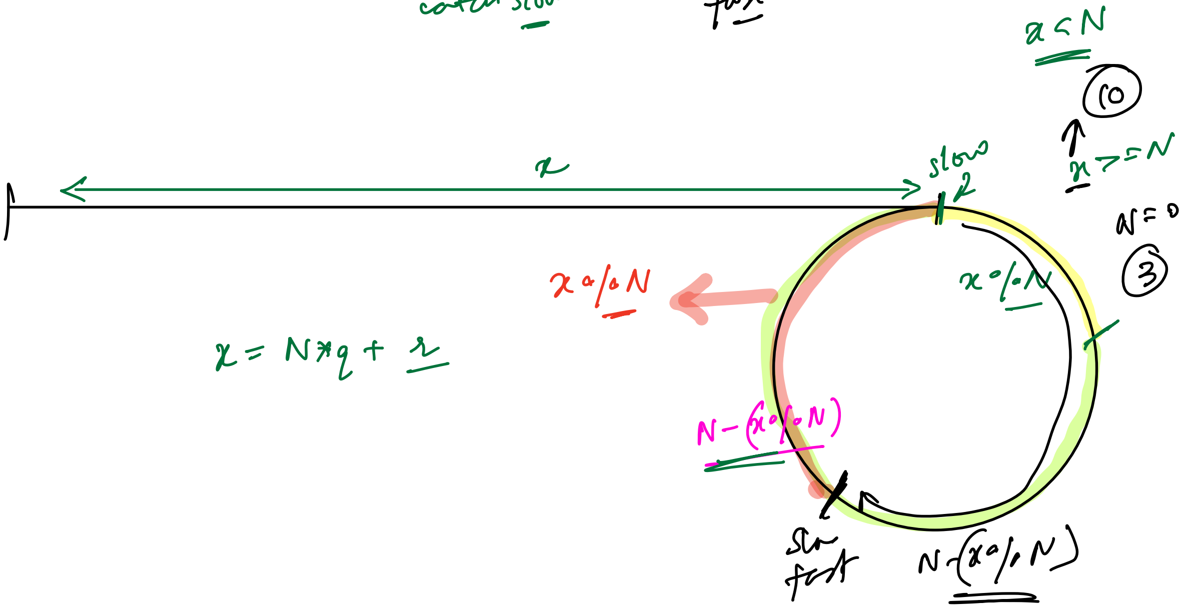
Remove the Loop

```
Node * temp = start;
while( temp->next != start)
{   temp = temp->next; }
  temp->next = NULL;
```

start of the loop found in prev code



1 iteration
≡ 1 unit of distan ↓
⇒ after d iteratn, fast will catch slow



$x = N * q + r$