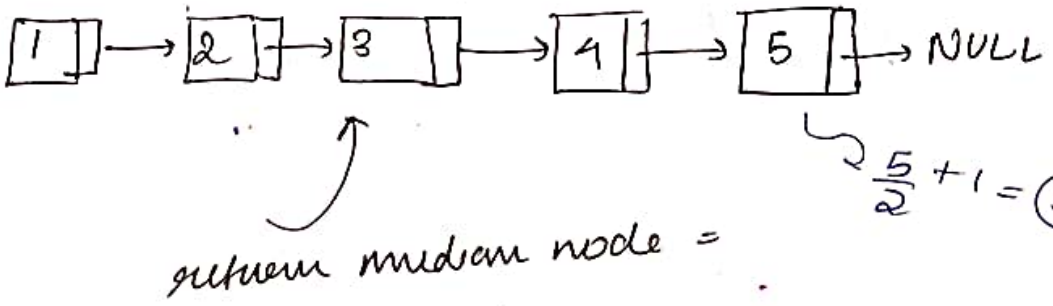
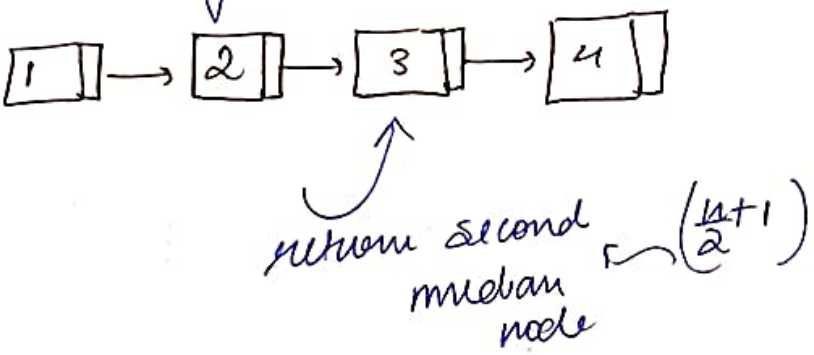
FIND MIDDLE ELEMENT IN A LINKED LIST

problem: Given the head of a linked list.



if even no of elements



BRUTE:

Traverse and count all the node in first pass

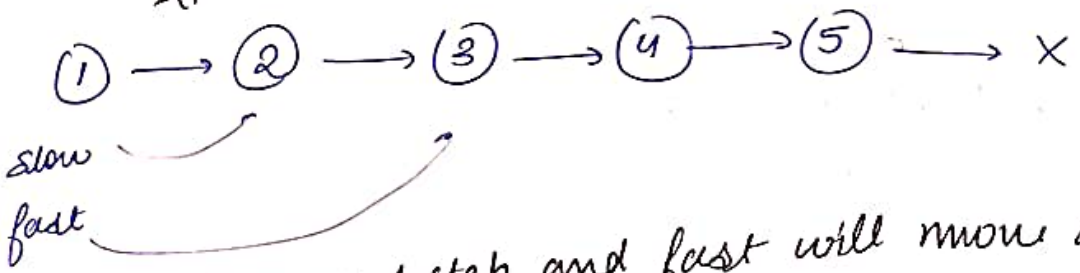
get median value by $\left(\frac{n}{2}+1\right)$

Now traverse again upto $\frac{n}{2}+1$ = become 0 and return that pointer

```
mid = (count/2 + 1)
temp = head
while (temp != null){
    mid--;
    if (mid == 0) return temp;
    temp = temp -> next;
}
```

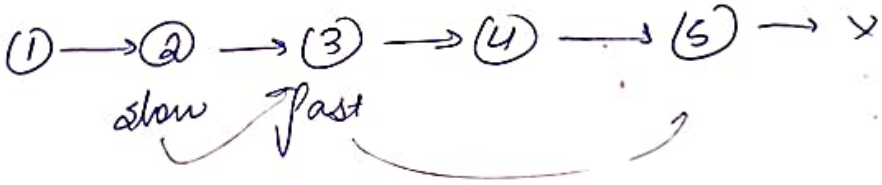
OPTIMAL:

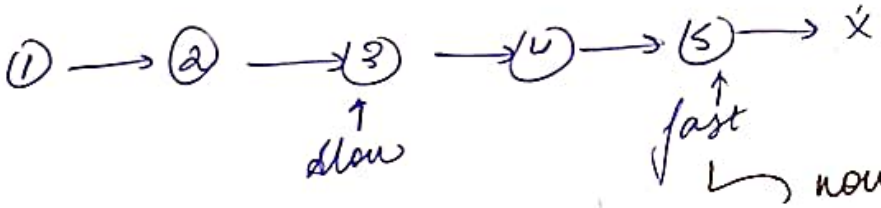
when odd length



slow will move 1 step and fast will move 2 step ahead at once

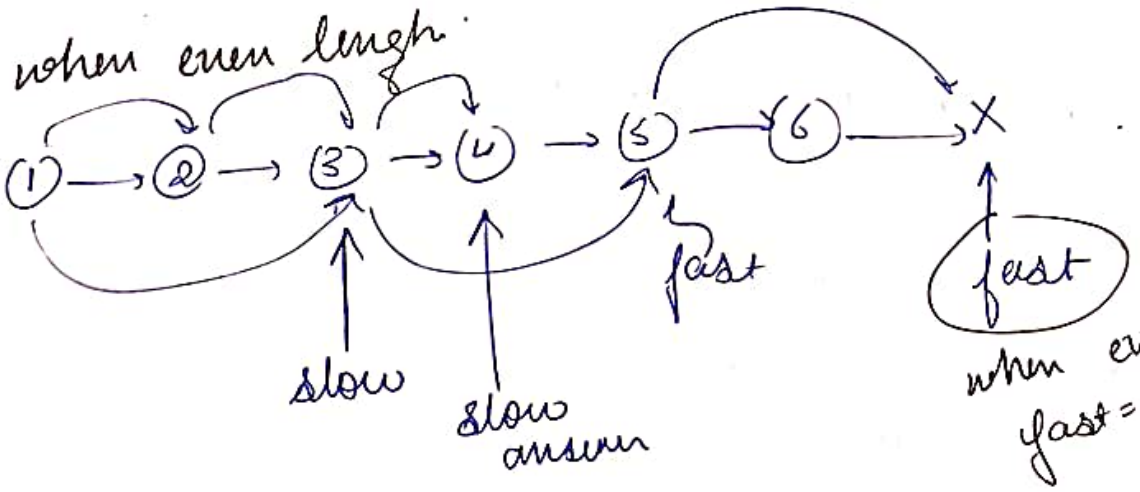
∴





↳ now fast is at end then slow will point to midan

when even length

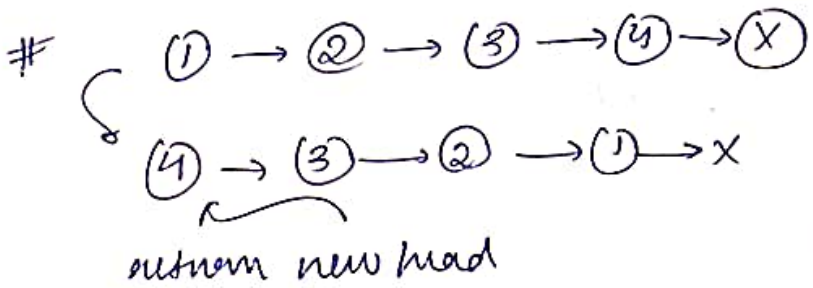


intution → p → travelling at n speed ∴ n distance

p2 → travelling at $\frac{n}{2}$ speed ∴ $\frac{n}{2}$ distance ↳ midpoint

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

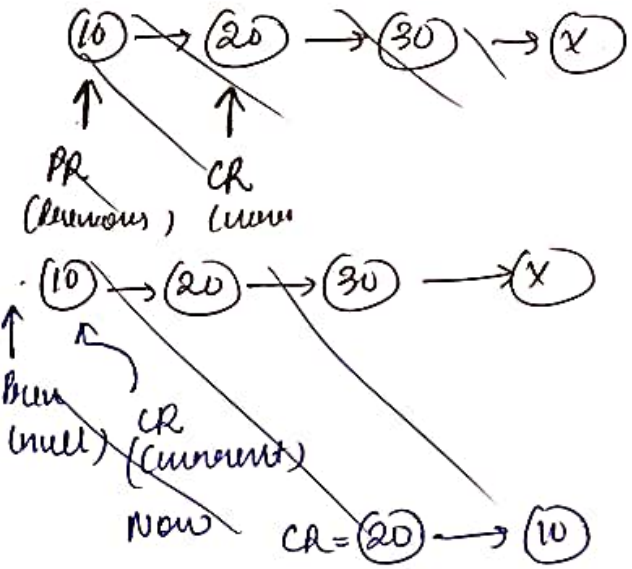
REVERSE A LINKED LIST

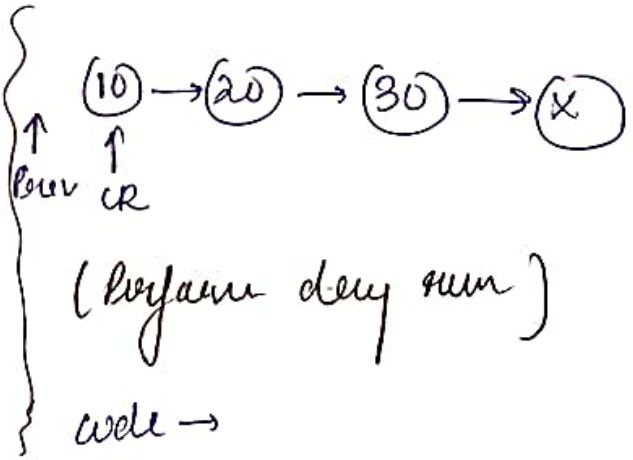


BRUTE

traverse list and add data to stack
traverse again and overwrite data by popping element.

OPTIMAL : (Iterative)

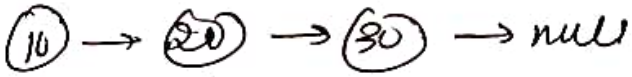





```
Node current = head;
Node previous = null;
while (current != null){
Node nxt = current.next;
current.next = previous;
previous = current;
current = nxt;
}
return previous;
```

O(n)

OPTIMAL: (Recursive)



code

```
usingRecursion (Node current) {

    if(current == null || current.next == null) return current;
```

if only one element

return last element which will become new head

```
    Node newHead = usingRecursion(current.next);
```

to we will help on returning this new head

```
    Node front = current.next;
    front.next = current;
    current.next = null;
```

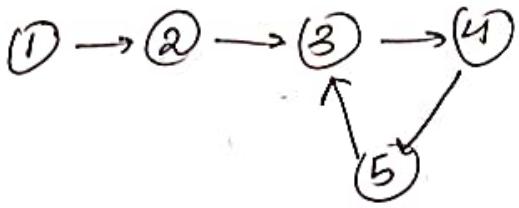
performing link change of two nodes in context

```
    return newHead;
}
```

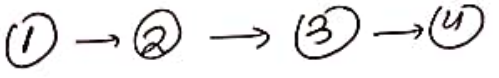
help on returning last (new head)

O(1)

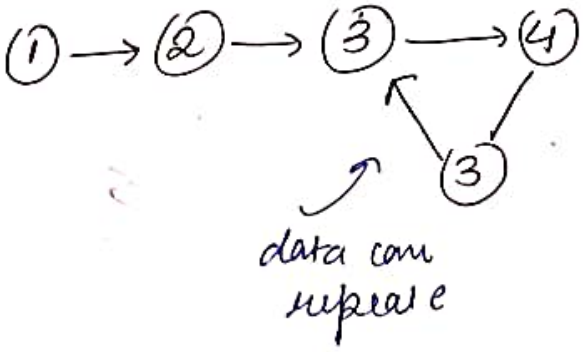
DETECT A LOOP OR CYCLE



→ true, loop exists



→ x false as no loop



→ true

BRUTE:

traverse and store node in set data structure

if node already present ∴ there is a loop return true;

if reached end return false;

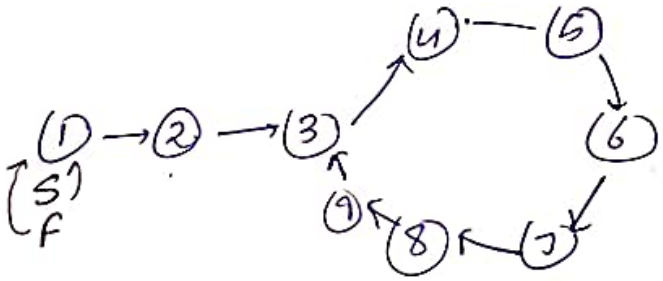
```
Node temp = head;
HashSet<Node> set = new HashSet<>();

while(temp != NULL){
    if(set.contains(temp)) return true;
    else set.add(temp);
    temp = temp.next;
}
return false;
```

O(1)

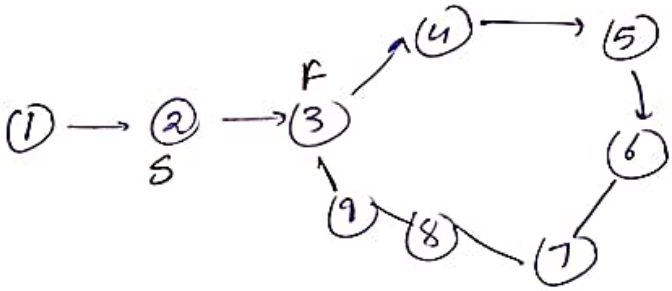
O(n) space

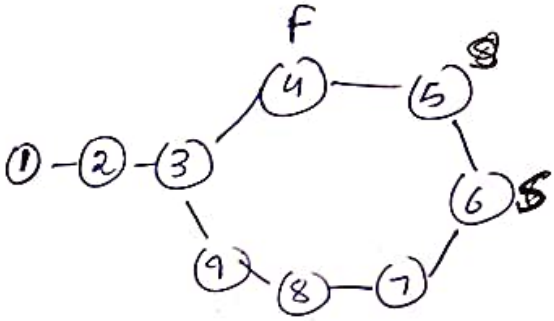
OPTIMAL:

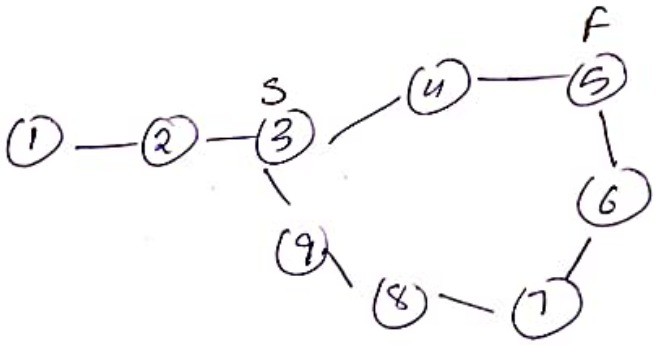
detect a loop by TORTOISE & HARE algorithm

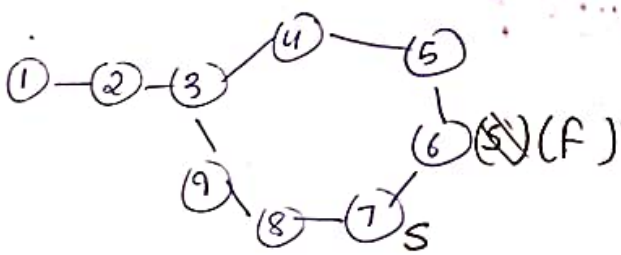


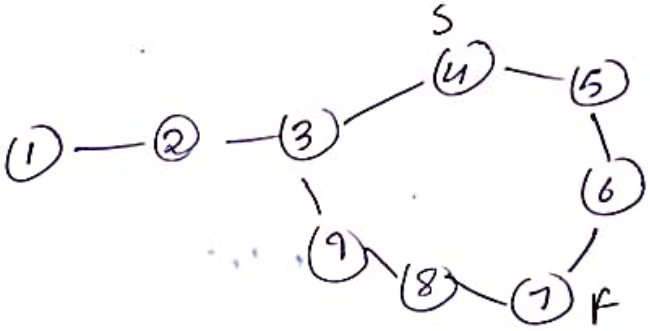
slow pointer 1 step
fast pointer 2 step

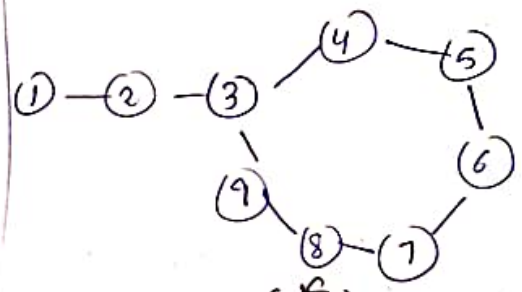






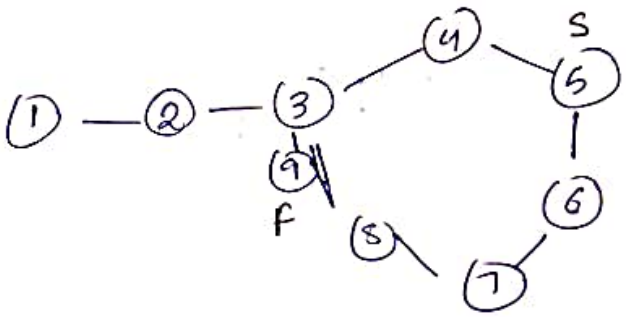






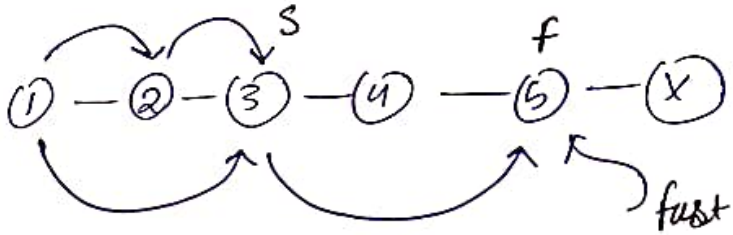
↳ eventually meets

∴ loop



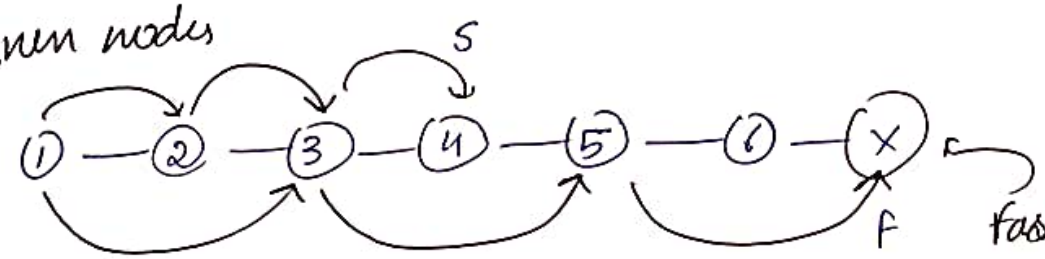
distance reduction between slow and fast is 1 ∴ they will always collide;

if odd no of nodes



fast pointer will be on last node in case of odd nodes

even nodes



fast will be on null pointer.

code →

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

O(n) → time complexity

O(1) → space