COSC 2436 Linked List Review

Linked List: Question #1

Write a function that gets the size of a linked list <u>using recursion</u>. <u>Not allowed to use loops.</u>

```
struct node{
  int data;
  node *next;
};

int getSize(node *head){



}
```

Linked List: Question #1

```
 int getSize(node *head){
  if(head == nullptr)
    return 0
  return 1 + getSize(head->next);
}
```

Solution

Base Case: When head is null → return 0

head = 4 → 2 → 3 → 6 → null

getSize(head) head = 4

1 + getSize(head → next) head = 2

1 + getSize(head → next) head = 3

1 + getSize(head → next) head = 6

1 + getSize(head → next) head = null

return 0

Linked List: Question #2

Write a function that appends a linked list at the end of another linked list. Your function should return the head of the new linked list. Example:

list1 = 4 → 2 → 3 → 4 → 6

list2 = 5 → 0 → 4 → 9

newList = 4 → 2 → 3 → 4 → 6 → 5 → 0 → 4 → 9

```
struct node{
  int val;
  node *next;
};

node *append(node *list1, node *list2){



}
```

Solution

What is the simplest case?

When one of the lists are null

list1 = 4 → 2 → 3 → 4 → 6 →null
list2 → nullptr
return list1

What if both lists are not null?

Insert head of list2 after the last node in list1

list1 = 4 → 2 → 3 → 4 → 6 →null
list2 = 5 → 0 → 4 → 9 →null

Merge lists:

newList = 4 → 2 → 3 → 4 → 6 → list2
OR
newList = 4 → 2 → 3 → 4 → 6 → 5 → 0 → 4 → 9 →null

Potential Solution

How do you insert list2 head at the end of list1?

Since no tail pointer was given, loop to the end of list1. Set the last_node->next = list2

```
while(list1 != nullptr){
  list1 = list1->next;
}
list1->next = list2;

return newList;
}
```

What errors do you see in the loop above?

1. Since while loop terminates when list1 reaches nullptr, will attempt to dereference a nullptr (segmentation fault)
2. Did not save the head of list1!

Linked List: Question #2

```
node *append(node *list1, node *list2){
  if(list1 == nullptr)
    return list2;

  else if(list2 == nullptr)
    return list1;

  node *newList = list1;
  while(list1->next != nullptr){
    list1 = list1->next;
  }
  list1->next = list2;

  return newList;
}
```

Linked List: Question #3

Write a function that removes the n^{th} from the end node of a linked list. Your function should return the head of the altered linked list. You can assume n is always valid. Example:

node *removeNthFromEnd(1 → 2 → 3 → 4 → 5, 2) => 1 → 2 → 3 → 5

```
struct node{
  int val;
  node *next;
};

node *removeNthFromEnd(node *head, int n){



}
```

Steps to solution

1. Find the length of the list

```
int length = 0;
node *cu = head;
while(cu != nullptr){
   length++;
   cu = cu->next;
}
```

2. Two pointers (cu and prev)

head = 4 → 2 → 3 → 6 → null

delete index 2

cur = head

loop stops at cur

head = 4 → 2 → 3 → 6 → null

cur = 3

cur

Need to set **node 2's** next pointer = **node 6**

prev pointer behind cu to save address of node before deleted node

Steps to solution

3. For loop length - n times

```
length = length - n;
node *prev = nullptr;
cu = head;
for(int i = 0; i < length; i++){
  prev = cu;
  cu = cu->next;
}
```

4. Edge case (if deleted node is head)

```
if(cu == head){
  head = head->next;
  delete cu;
  return head;
}
```

Linked List #3

```
node *removeNthFromEnd(node *head, int n){
  int length = 0;
  node *cu = head;
  while(cu != nullptr){
    length++;
    cu = cu->next;
  }
  length = length - n;
  node *prev = nullptr;
  cu = head;
  for(int i = 0; i < length; i++){
    prev = cu;
    cu = cu->next;
  }
  if(cu == head){
    head = head->next;
    delete cu;
    return head;
  }
  prev->next = cu->next;
  delete cu;
  return head;
}
```

Linked List: Question #4

Write a function that removes the n^{th} from the end node of a linked list. Your function should return the head of the altered linked list. You can assume n is always valid. <u>You can only do it in one pass.</u> Example:

node *removeNthFromEnd(1 → 2 → 3 → 4 → 5, 2) => 1 → 2 → 3 → 5

```
struct node{
  int val;
  node *next;
};

node *removeNthFromEnd(node *head , int n){



}
```

Solution

Fast and Slow pointer:

Ex. Remove 2^{nd} node from the end

1. Fast pointer traverses 2 nodes (two iterations)

fast

6 → 3 → 2 → 4 → 1 →null

slow

fast

6 → 3 → 2 → 4 → 1 →null

slow

fast

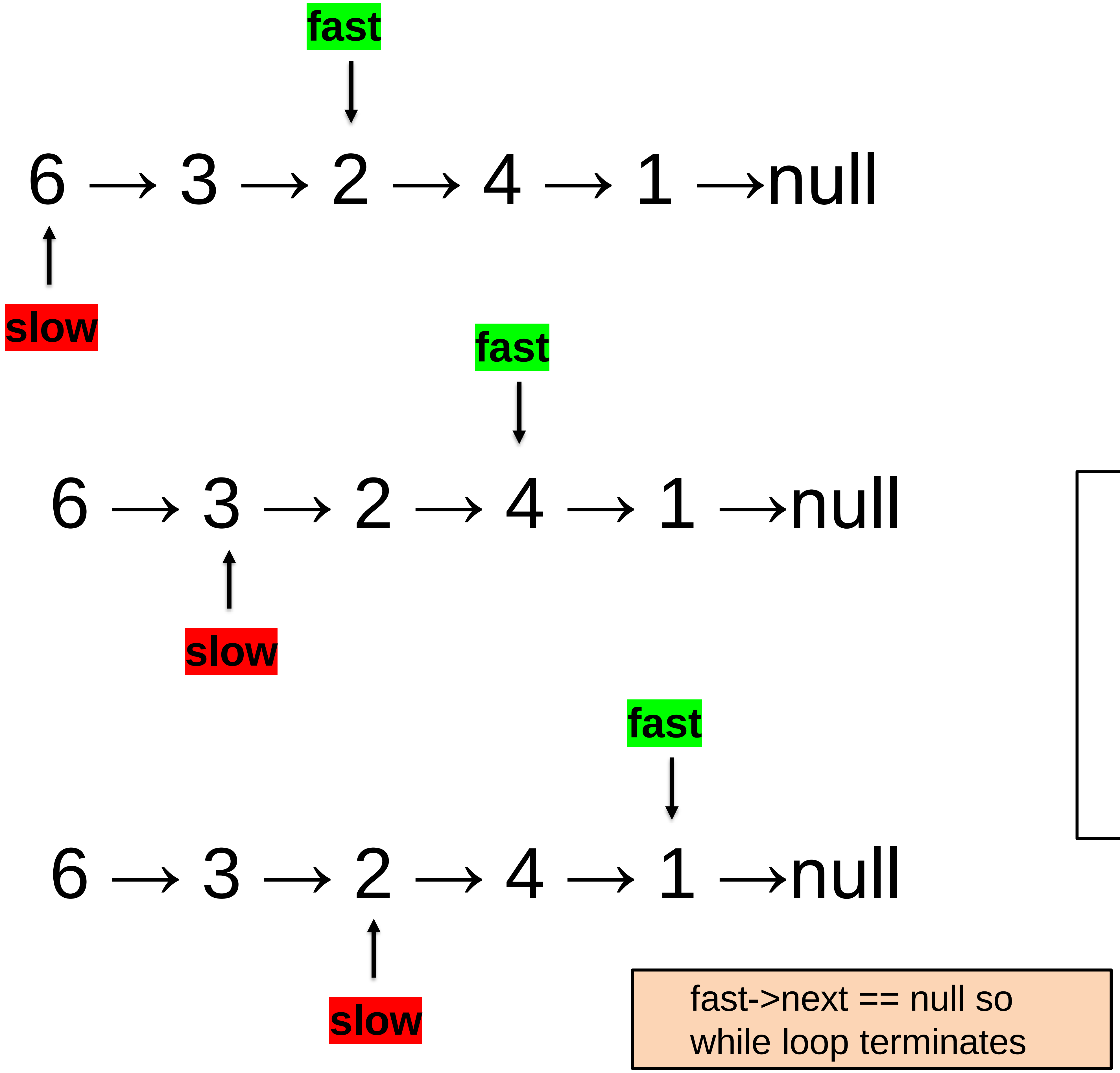
6 → 3 → 2 → 4 → 1 →null

slow

```
node *fast = head;
node *slow = head;
for(int I = 0; I < n; I++){
   fast = fast->next;
}
```

Ex. Remove 2nd node from the end

2. Using fast pointer's new starting position iterate BOTH the <u>fast</u> and <u>slow</u> pointer while fast->next != nullptr

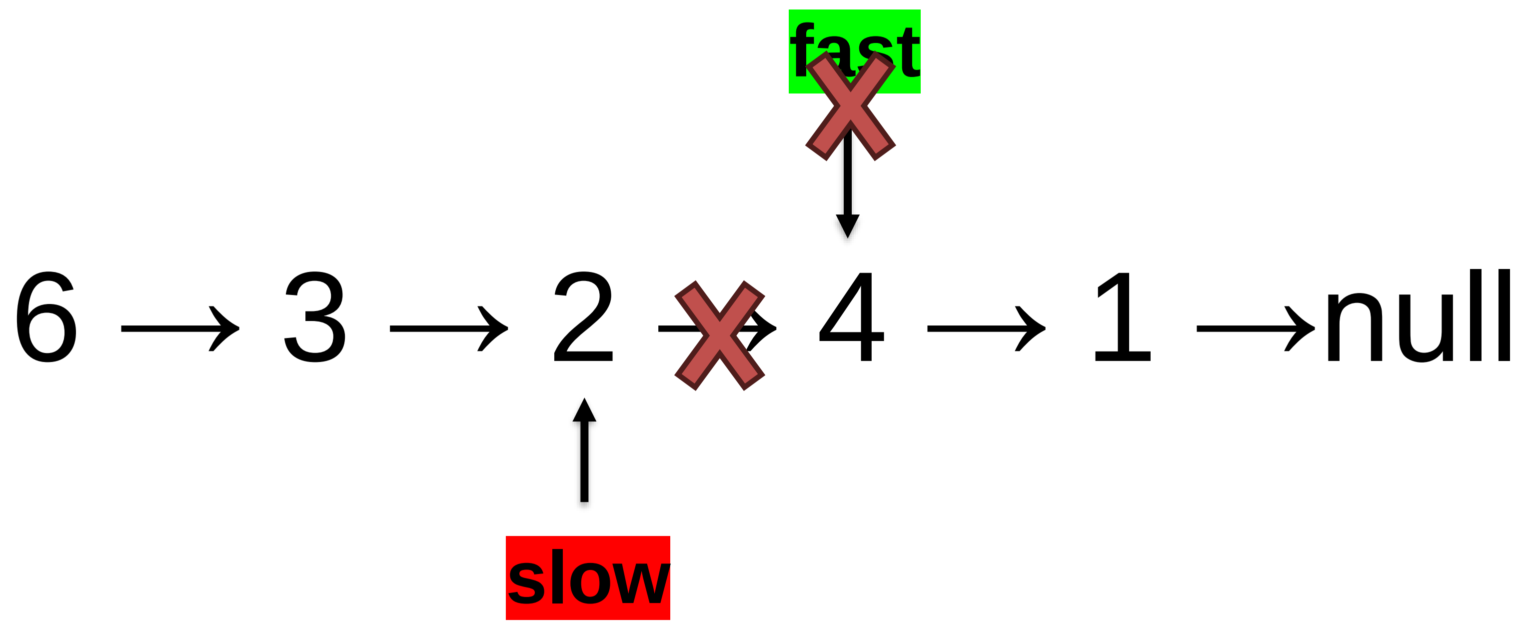


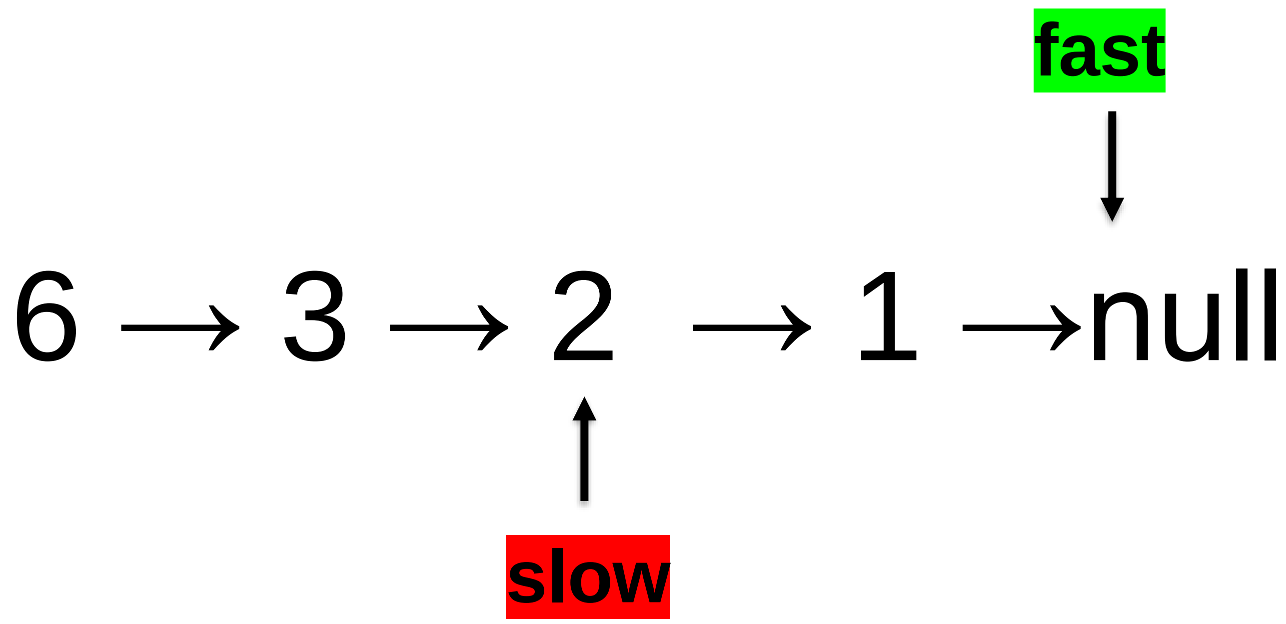

```
while(fast->next != nullptr){
   fast = fast->next;
   slow = slow->next;
}
```

Slow pointer is now at the correct position BEFORE the <u>2nd node from the end)</u>

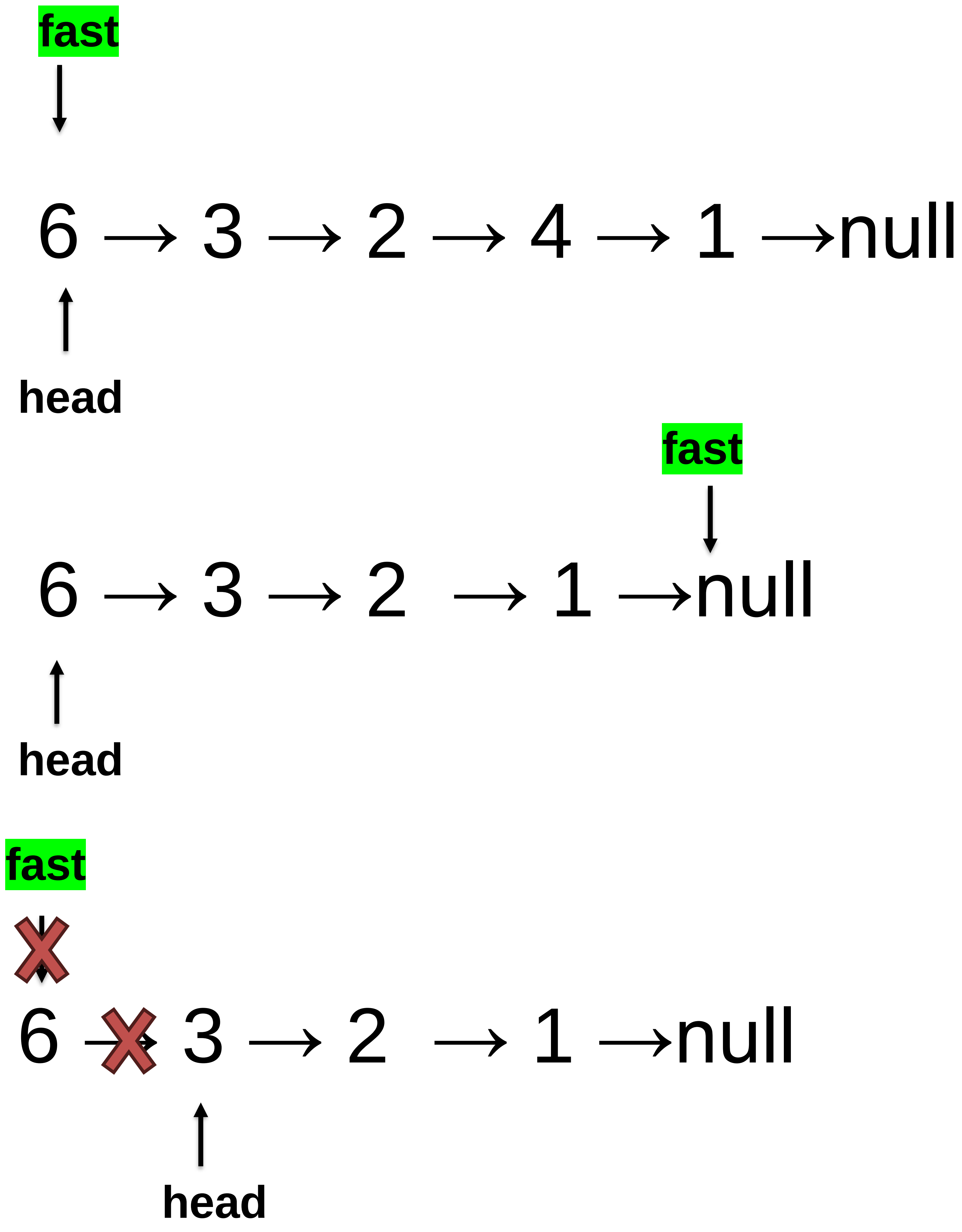
Ex. Remove 2^{nd} node from the end

3. Change the pointer of slow->next and fast
Delete the 2^{nd} to last node (deallocate memory)






```
fast = slow->next;
slow->next = slow->next->next;
delete fast;
return head;
```



Edge Case: If deleted node is head

n = size of list

Ex. Remove 5th node from the end

1. Fast traverses entire list and becomes nullptr
2. Conditional that updates the head and deletes previous head

```
if(fast == nullptr){
  fast = head;
  head = head->next;
  delete fast;
  return head;
}
```

NOTE: Condition should be checked <u>first</u> before iterating slow pointer

- If true, return updated head

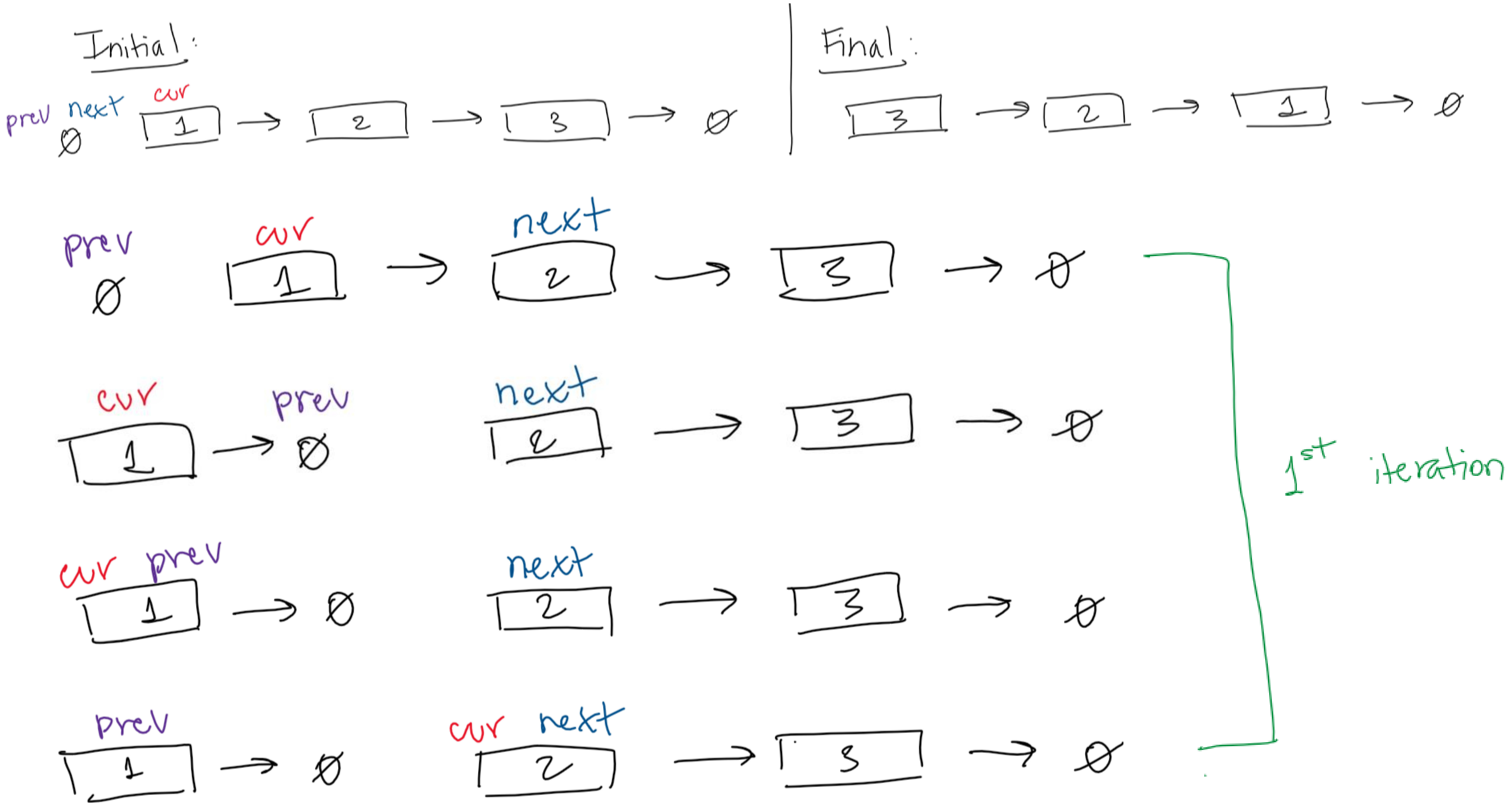
Linked List #4

```
node *removeNthFromEnd(node *head , int n){
  node *fast = head;
  node *slow = head;
  for(int I = 0; I < n; I++){
    fast = fast->next;
  }
  if(fast == nullptr){
    fast = head;
    head = head->next;
    delete fast;
    return head;
  }
  while(fast->next != nullptr){
    fast = fast->next;
    slow = slow->next;
  }
  fast = slow->next;
  slow->next = slow->next->next;
  delete fast;
  return head;
}
```

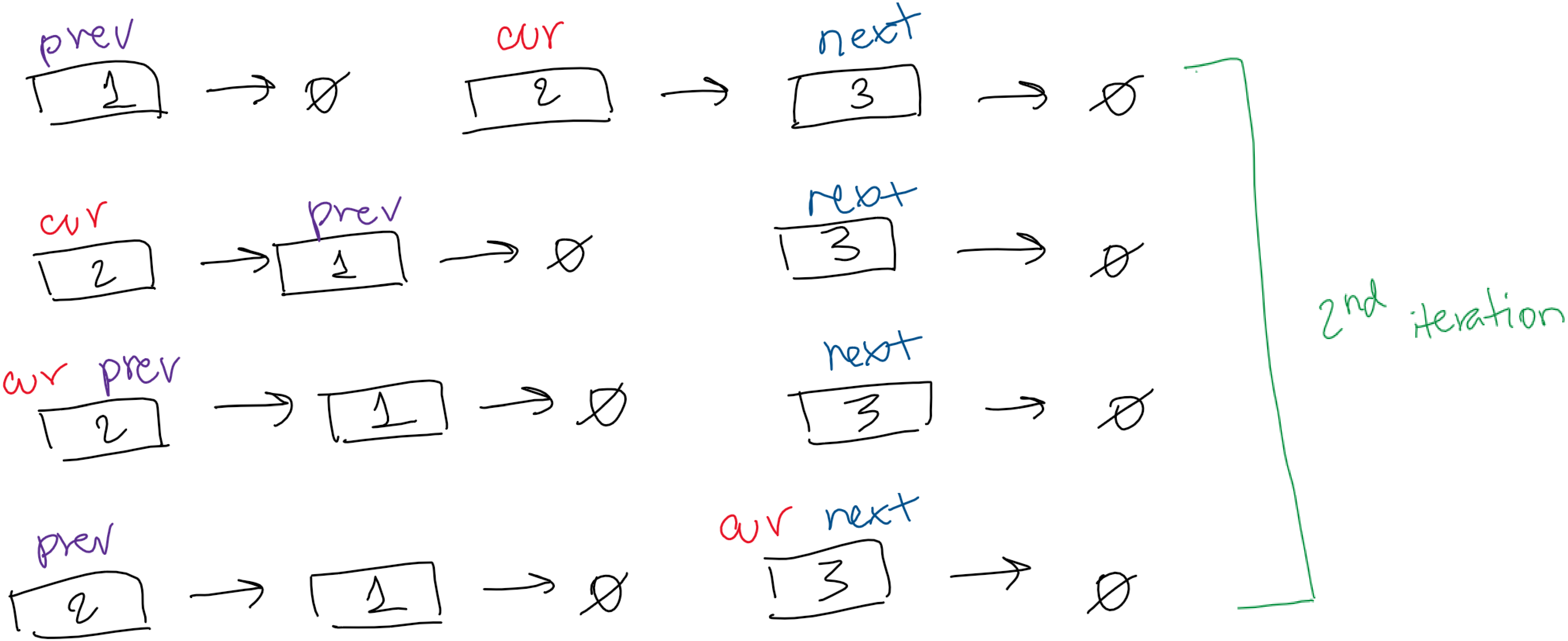
Linked List: Question #5

Write a function to reverse a linked list. Your function should return the head of the reversed linked list. You cannot use any additional data structures in your function.

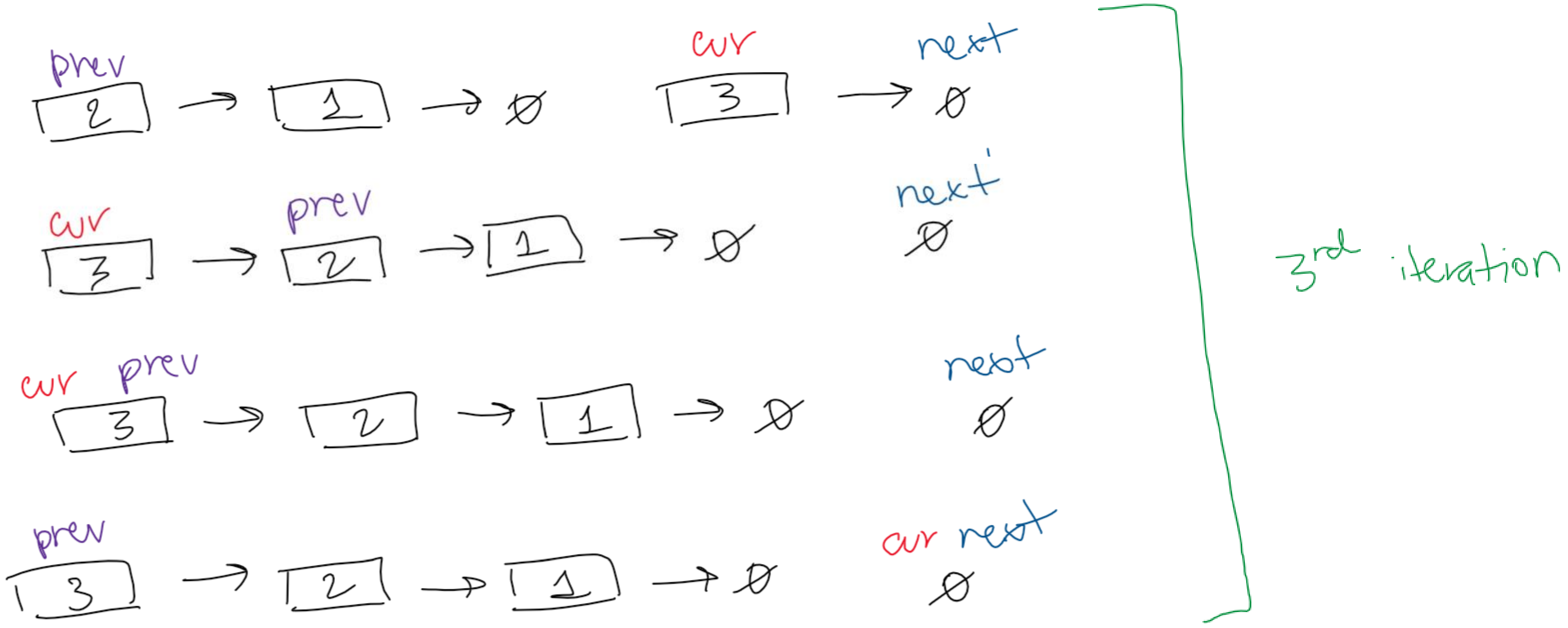
```
struct node{
   int val;
   node *next;
};

node *reverse(node *head){


}
```



Initial:
prev next cur
1 → 2 → 3 → ∅
Final:
3 → 2 → 1 → ∅
prev cur next
∅ 1 → 2 → 3 → ∅
cur prev next
1 → ∅ 2 → 3 → ∅
cur prev next
1 → ∅ 2 → 3 → ∅
prev cur next
1 → ∅ 2 → 3 → ∅
1st iteration



prev
cur
next
1
Ø
2
3
Ø
cur
prev
next
2
1
Ø
3
Ø
cur prev
next
2
1
Ø
3
Ø
prev
cur next
2
1
Ø
3
Ø
2nd iteration



cur == nullptr

return prev

Linked List: Question #5

```
node *reverse(node* head){
  node* cur = head;
  node* prev = nullptr;
  node* next = nullptr;
  while(cur != nullptr){
       next = cur->next;
       cur->next = prev;
       prev = cur;
       cur = next;
  }
  return prev;
}
```

Linked List: Question #6

Write a function to reverse a linked list <u>using recursion</u>. Your function should return the head of the reversed linked list. You cannot use any additional data structures in your function.

```
struct node{
  int val;
  node *next;
};

node *reverse(node *head){


}
```

Linked List: Question #6

```
node *reverse(node* head){
  if(head == nullptr || head->next == nullptr)
       return head;

  node* rest = reverse(head->next);
  head->next->next = head;
  head->next = nullptr;
  return rest;
}
```