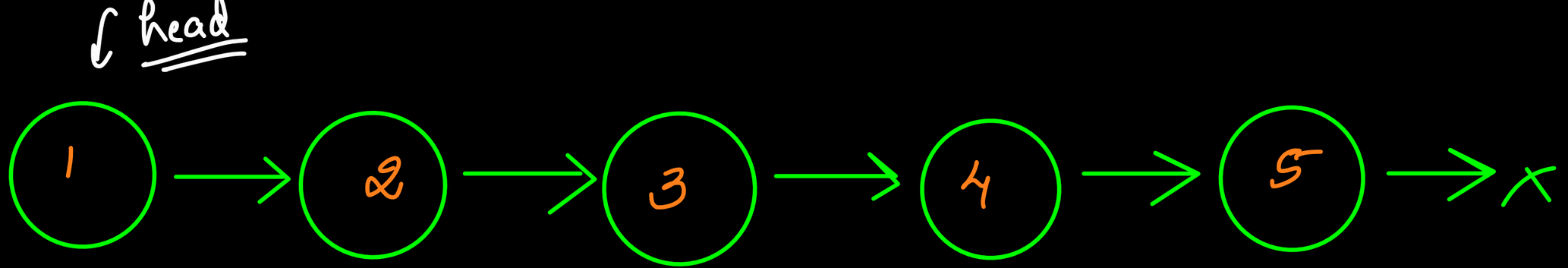
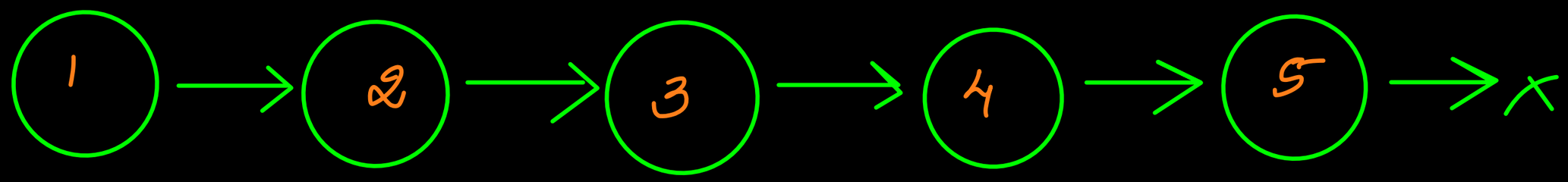


→ Iteration

→ Recursion



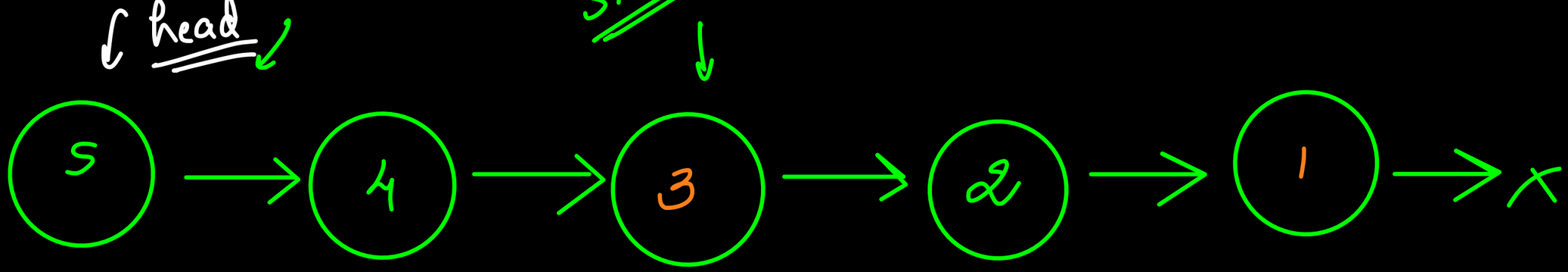
I → Iteration Data Reverse

we will only manipulate the data property of node.

assume the Problem was to reverse an array.

```
i=0
j=n-1
while(i<=j){
    Swap(arr,i,j);
    i++;j--;
}
```

n = 5



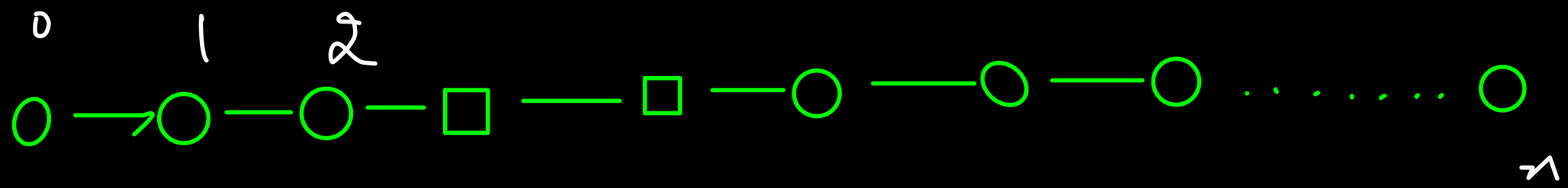
i = 0 1 2 3

In ll, we can't access nodes with indexes.

→ length of ll → O(n)

```
temp = head; i = 0 ; idx = 3
while (i < idx) {
    temp = temp.next;
    i++
}
```

```
t = start.data;
start.data = temp.data
temp.data = t
```



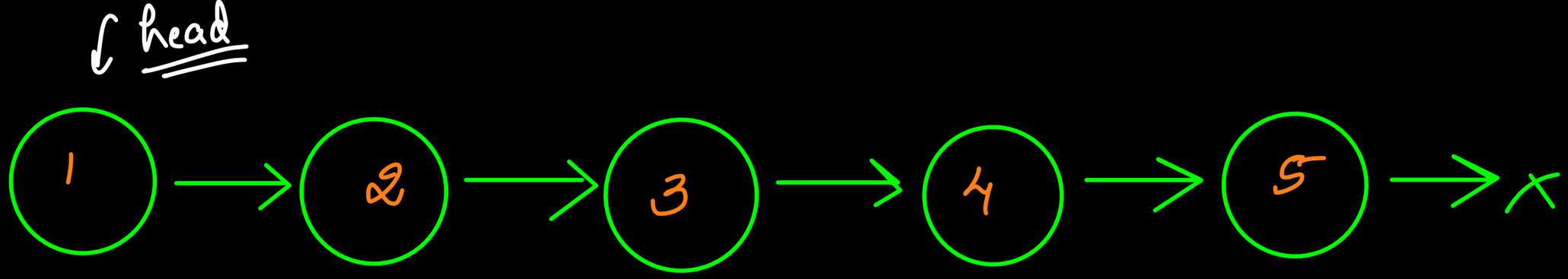
$n^2 \times O(n)$ → $O(n^2)$ ← fine

for every node we are trying to almost visit the whole list.

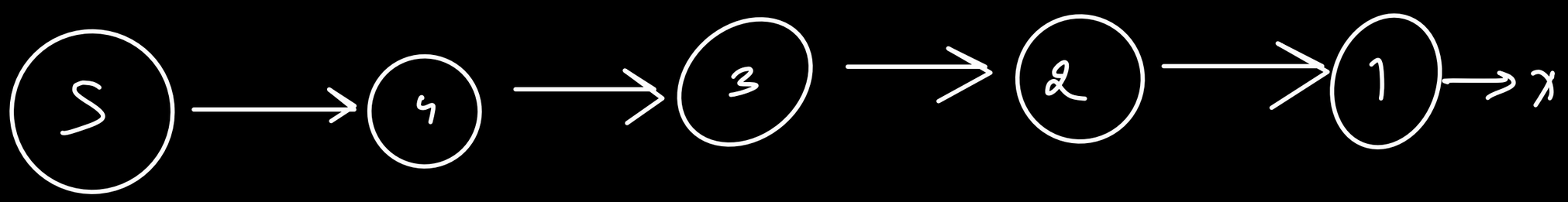
Span → $O(1)$

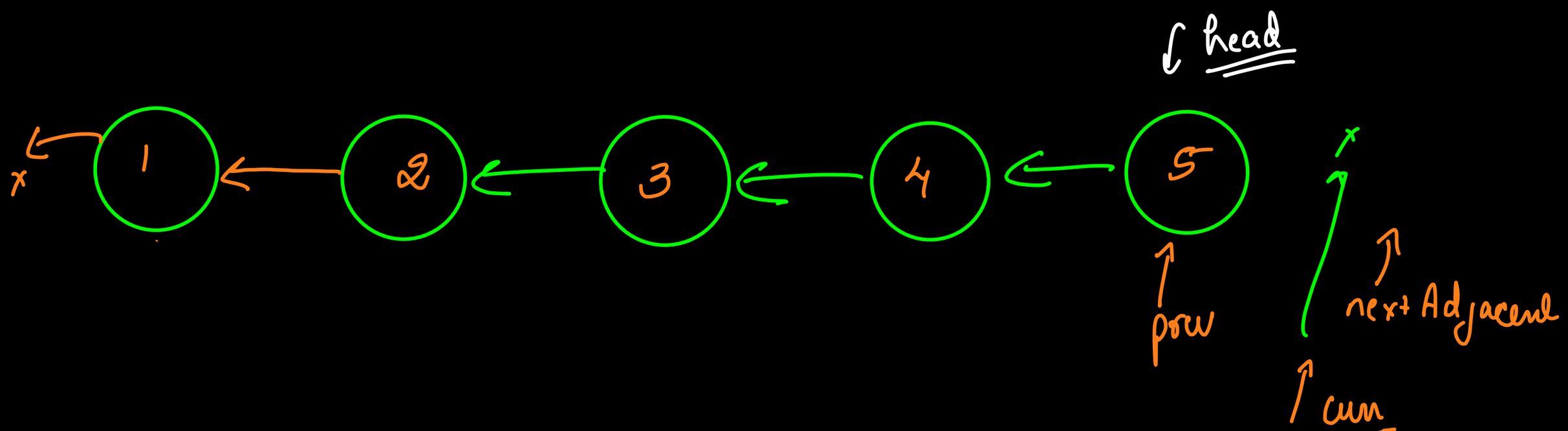
$1 \approx 5 \times 10^8$

TLE



Iteration → Pointer Reverse





prev = null

curr = head

nextAdjacent = curr.next

while (curr != null) {

curr.next = prev

prev = curr

curr = nextAdjacent

if (curr != null)

nextAdjacent = curr.next

}

curr.next = prev

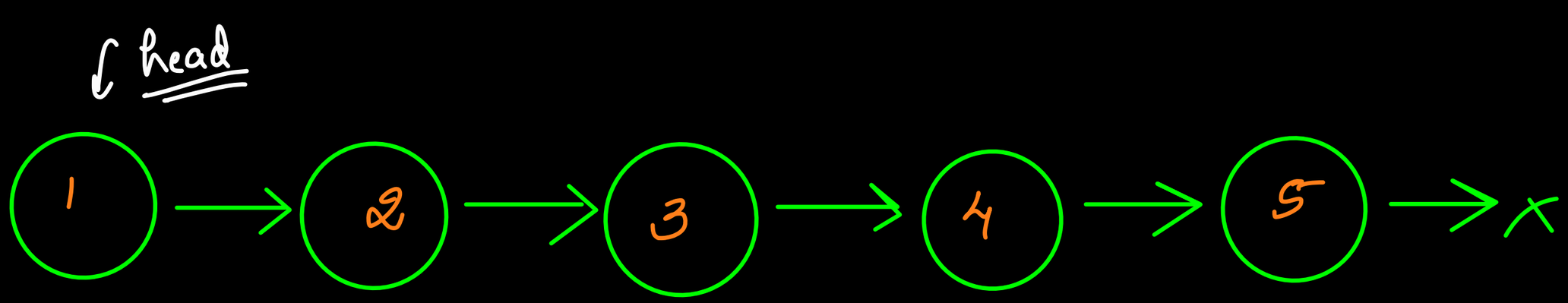
Here we are going to every node once

time → O(n)

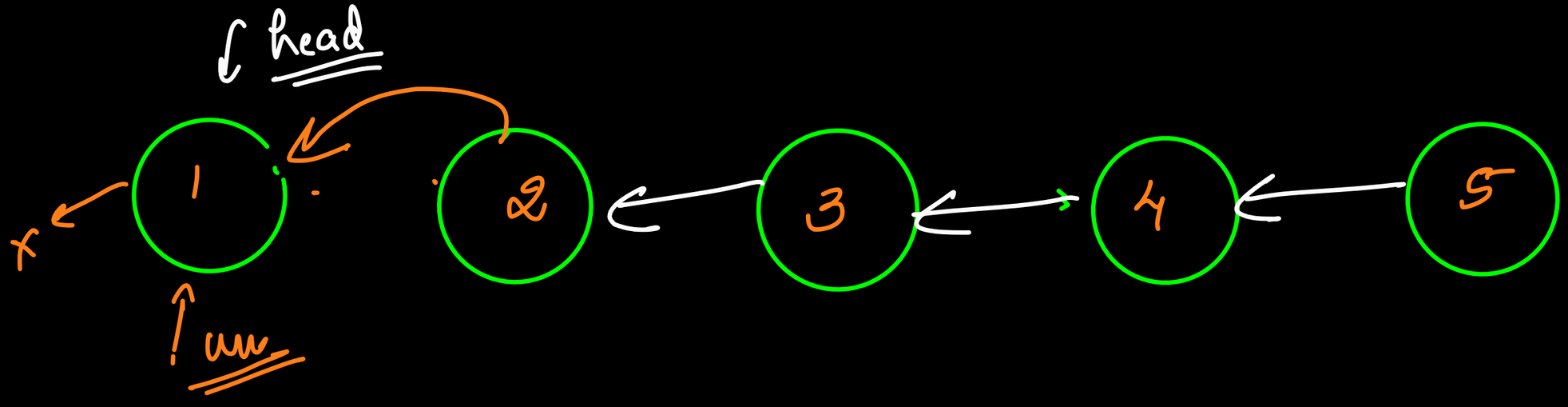
Space → O(1)

→ to maintain access to memory list

head = prev



I → Recursion → Pointer Reverse



Everything beyond 2 has been removed.

we want next of 2 to be equal to 1

↳ cur.next

curr.next.next = curr

curr.next = nil

f (curr) = f (curr.next) ⟵ newHead

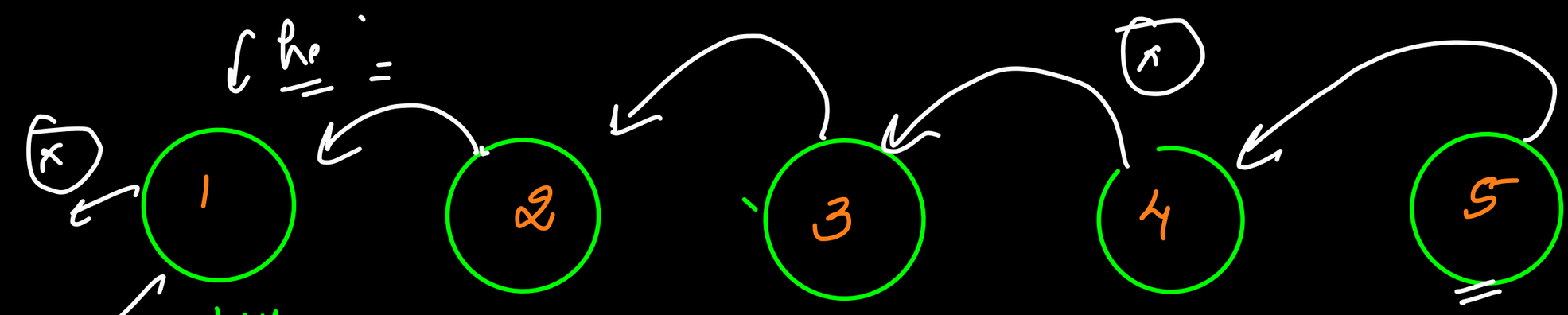
returns the ll starting from curr node

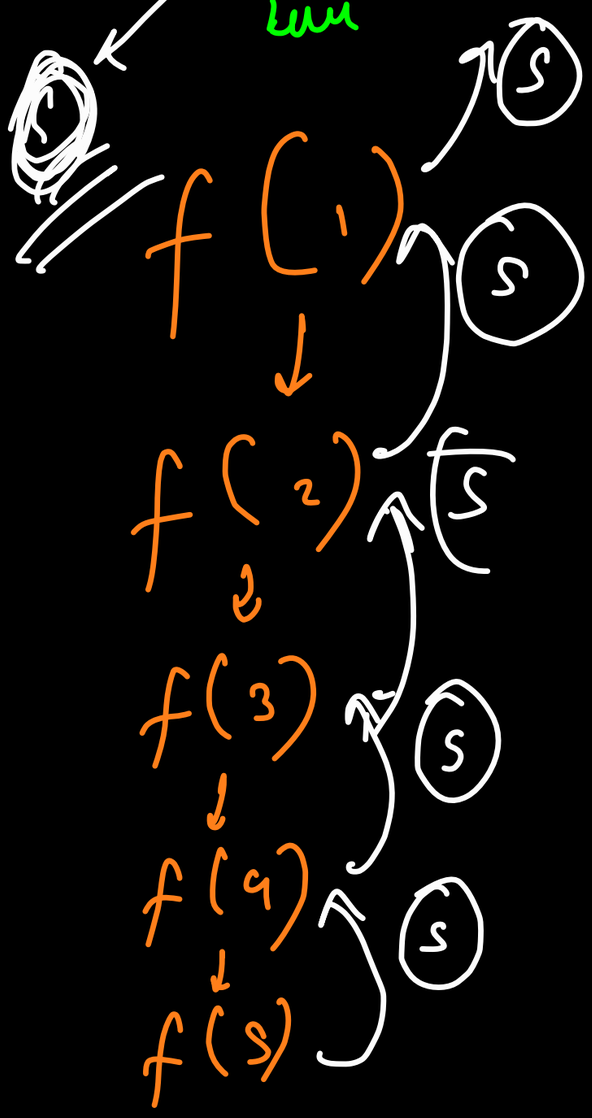
curr.next.next = curr

curr.next = null

return newHead;

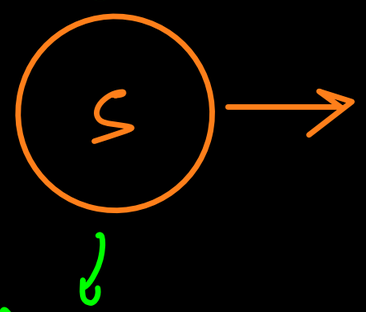
we assume everything works correctly for curr.next & the f" f returns the ll starting from curr.next.






```
if (curr.next == null)
    return curr;
```

2 <u>leaf</u>



dll with a single node is already <u>returned</u>

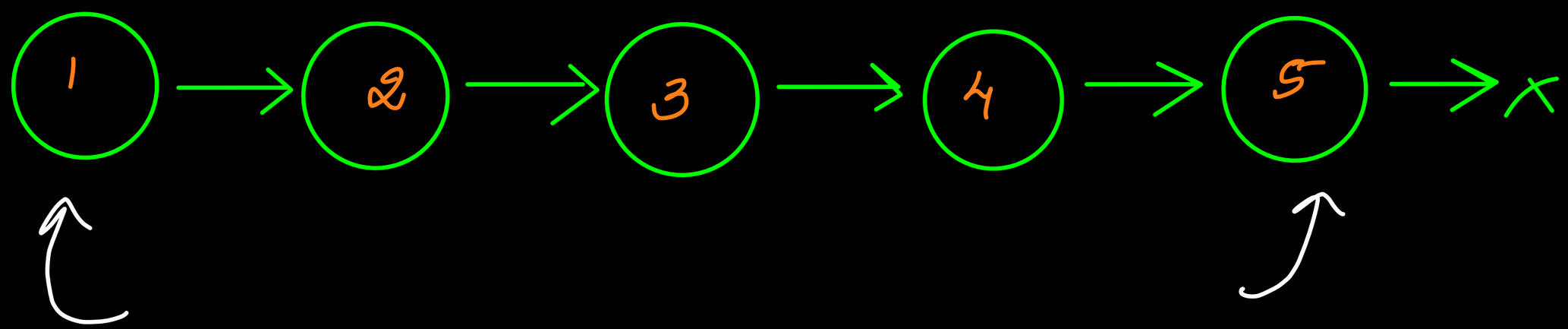
<u>Global level</u>

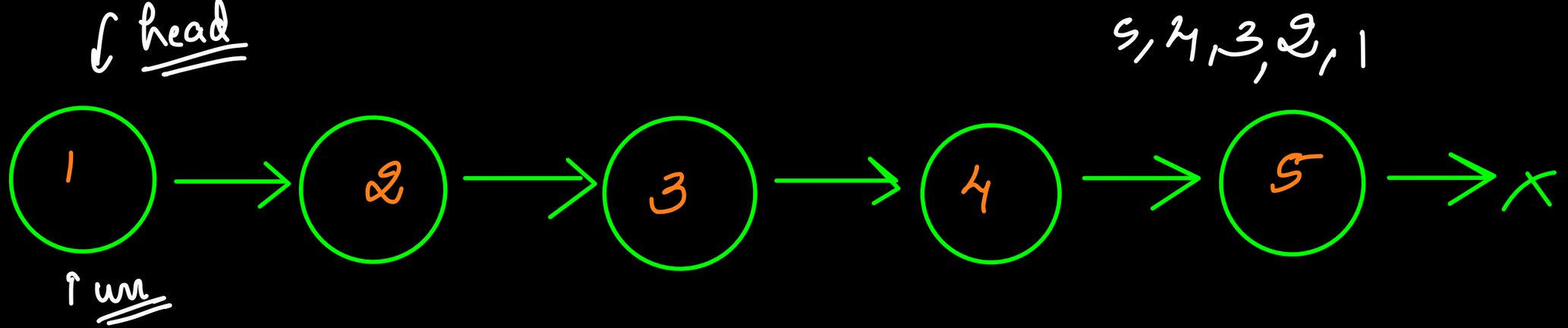
<u>Time → O(n)</u>

Space → O(n)

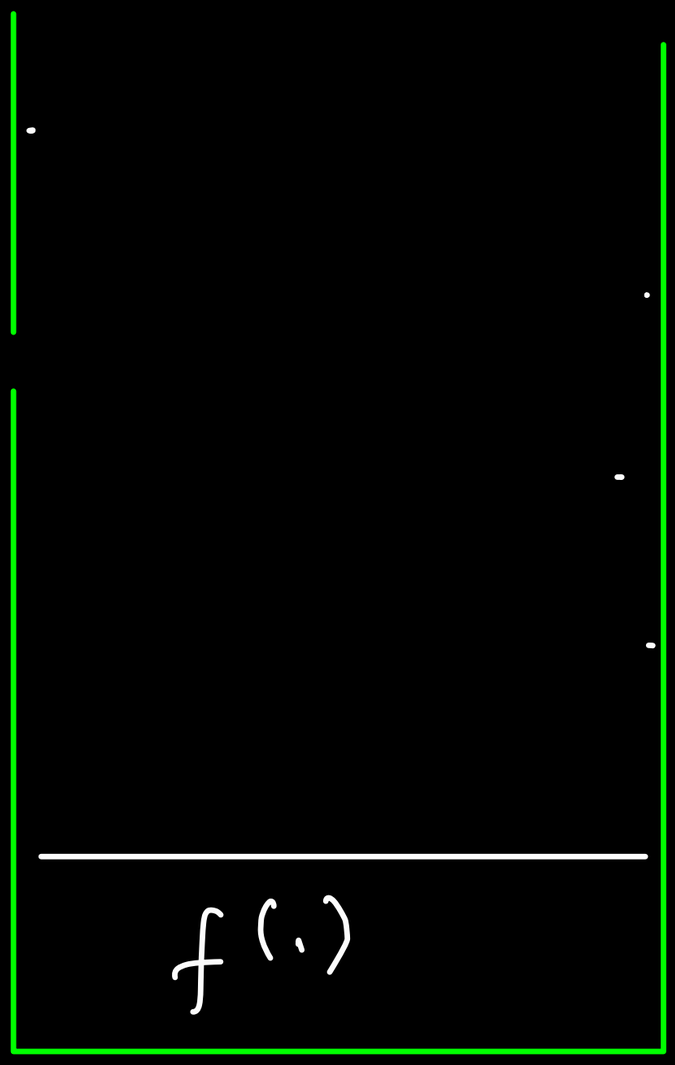
IV → Data Recursive ?

head





5, 4, 3, 2, 1



$f(curr) = f(curr.next)$

2

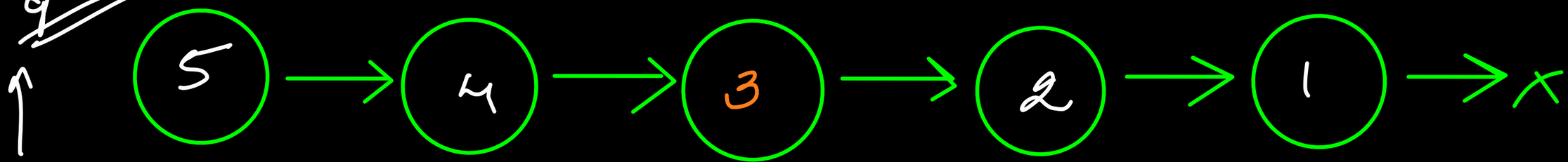
recursively manipulate the data

In singly LL, it is easy to go forward.

while coming back in recursion, we can easily traverse the LL in reverse order

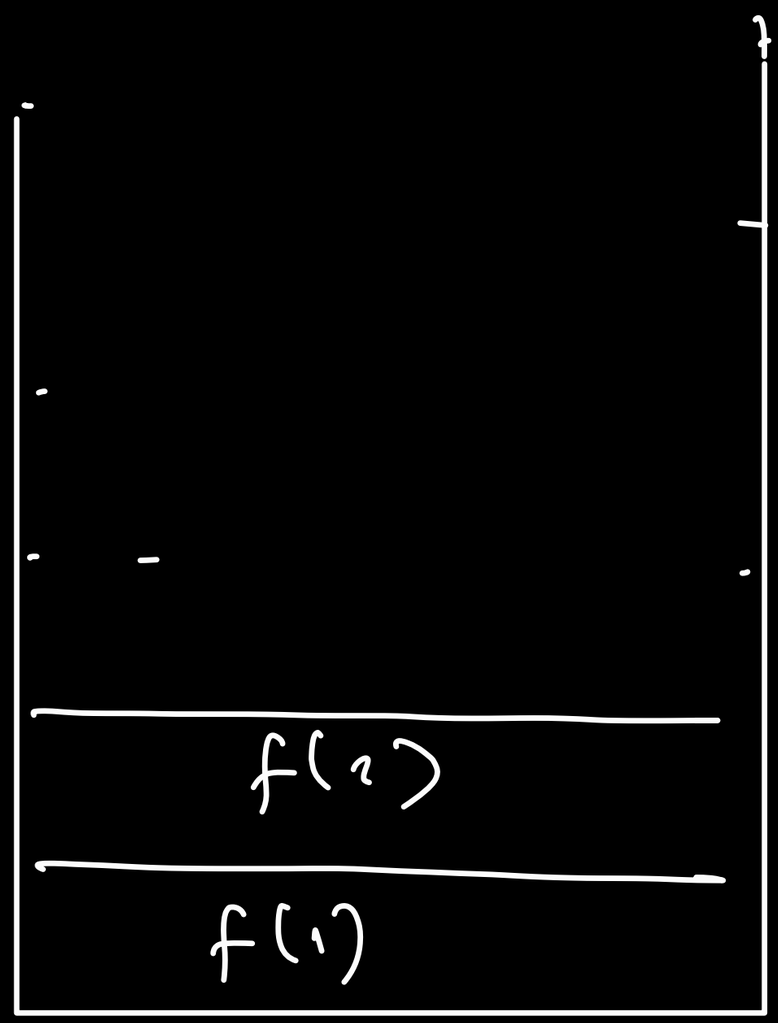
singly

global head



curr

start



$$f(curr) = f(curr.next)$$

↓

Swap the data

start = start.next

return

Time → O(N)

Space → O(N)

if (curr == null)

return

f(curr)

2

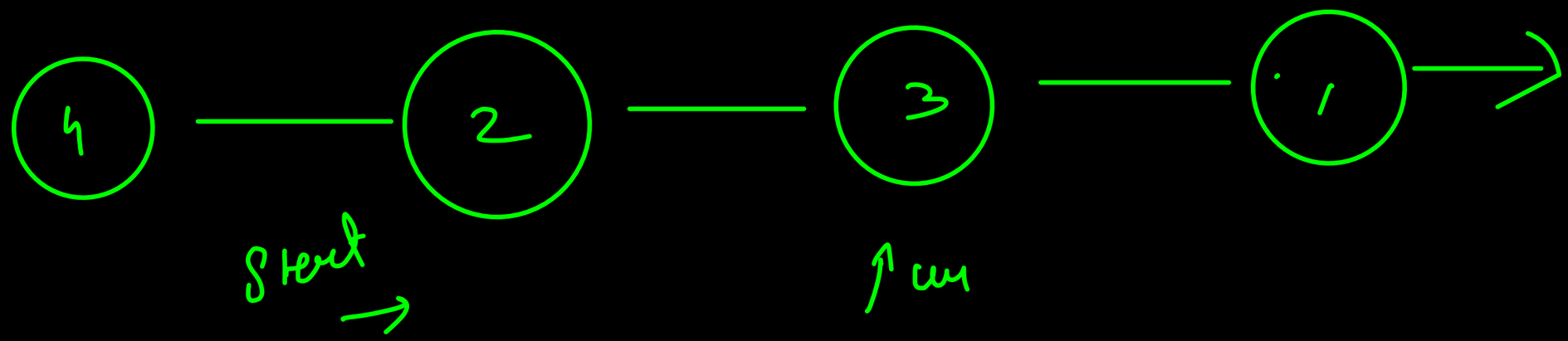
means add, by data

and returns true when

ll is found.

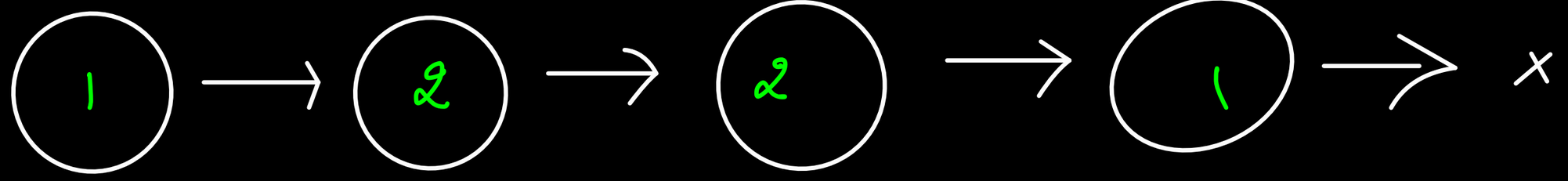
=

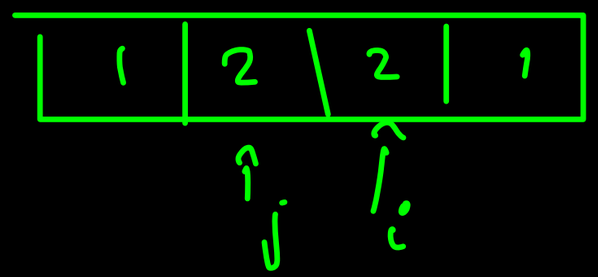
```
{
  result = f(curr.next)

  if (result == false) {
    Snap
    respons = (start == curr) or
              (start.next == curr)
    Start = start.next
    return respons
  } else {
    return result;
  }
}
```



$f(1) \rightarrow f(2) \rightarrow f(3)$

Data





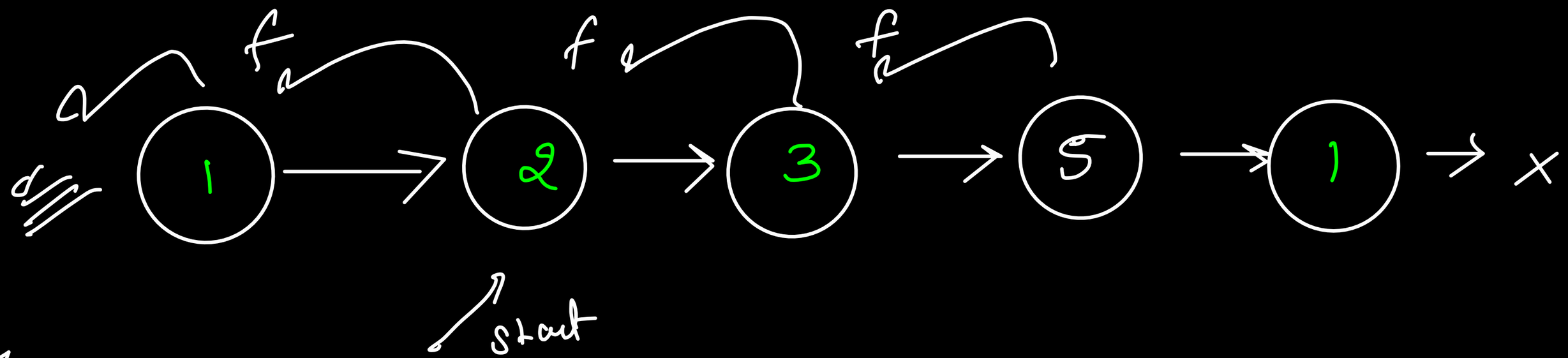

```
f(curr)
if(curr == null)
    return true;
r = f(curr.next)

isEqual = (start.data == curr.data)
if(!isEqual)
    return false;
else {
    start = start.next
    return r and isEqual;
}
```

global
↓
conf

```
f(curr) {
    if (curr == null) return [true, true]

    r = f(curr.next);
    if ( r[1] == false)
        return r;

    if( start == curr  or  start.next == curr) {
        result = [r and (start.data == curr.data), false]

    } else {

        result = [r and (start.data == curr.data), true];

    }
    return result;
}
```

[,]
↳ whole ll is palin
↳ whole we want to do comp

JOIN THE DARKSIDE



Time

```
f (cur) {
    if (cur == null) return true;
    res = f (cur.next);
    if ( res == false) return res;
    if ( start.data == cur.data ) {
        start = start.next ;
        return true;
    } else {
        return false;
    }
}
```

O(n)
O(n)