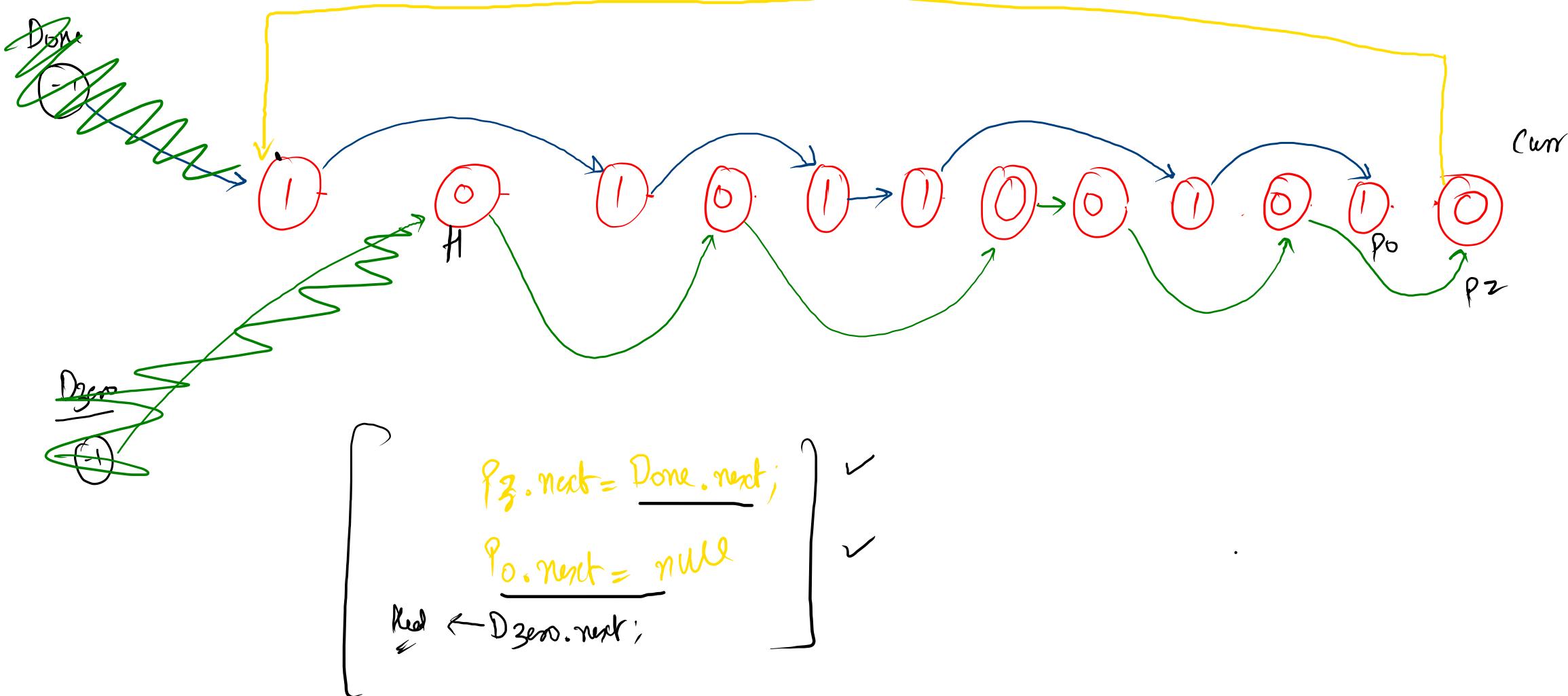
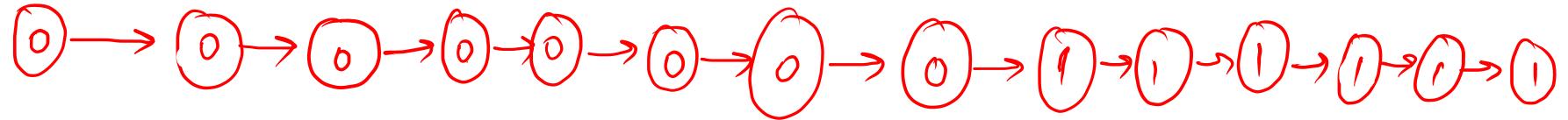


1->0->1->0->0->1->1->1->1->1->null



H

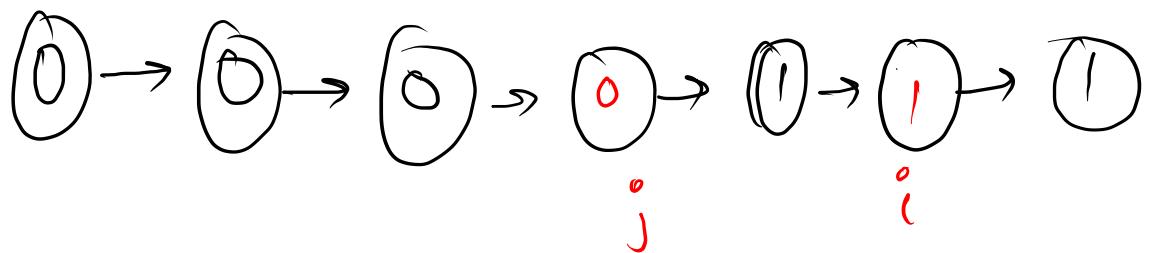


j

i

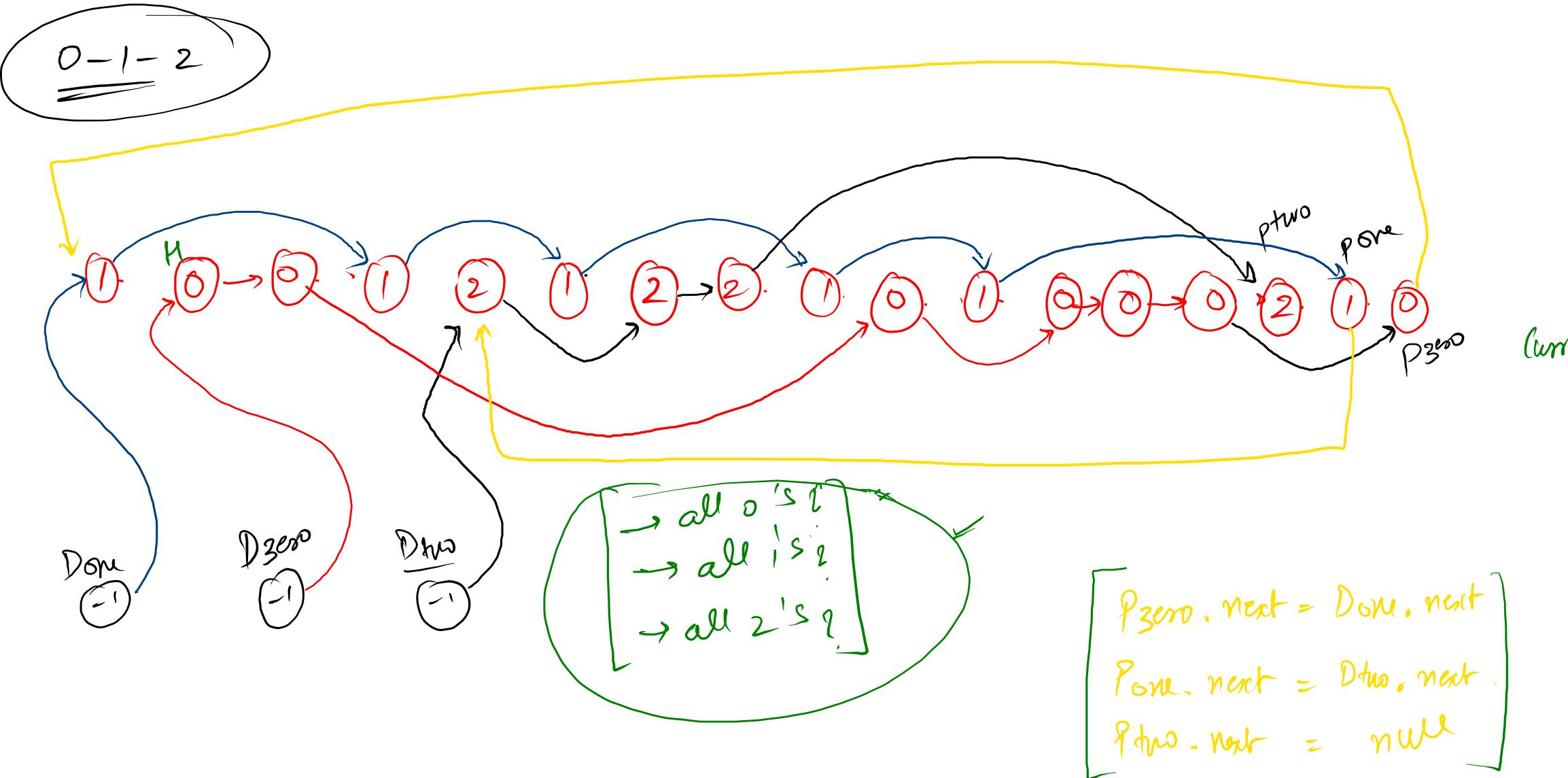
{ while (i != null) {
 if (i.val == 1) {
 i = i.next;
 } else {
 swap
 i = i.next;
 j = j.next;
 }
}

H

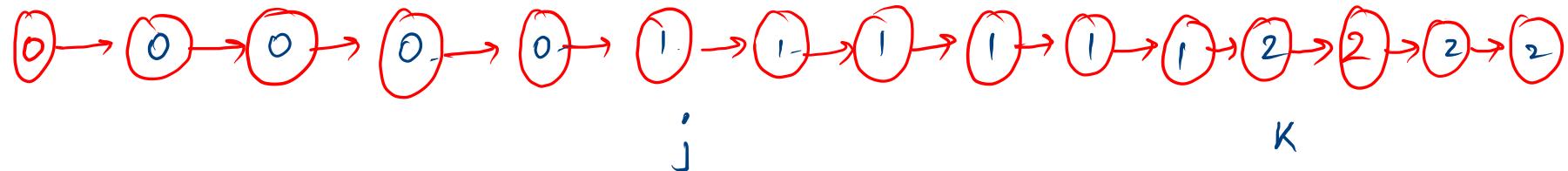


j

i



H



if ($i.\text{val} == 2$) {

$i = i.\text{next}$

}

if ($i.\text{val} == 1$) {

swap(i, K);

$i = i.\text{next}$;

$K = K.\text{next}$;

}

if ($i.\text{val} == 0$) {

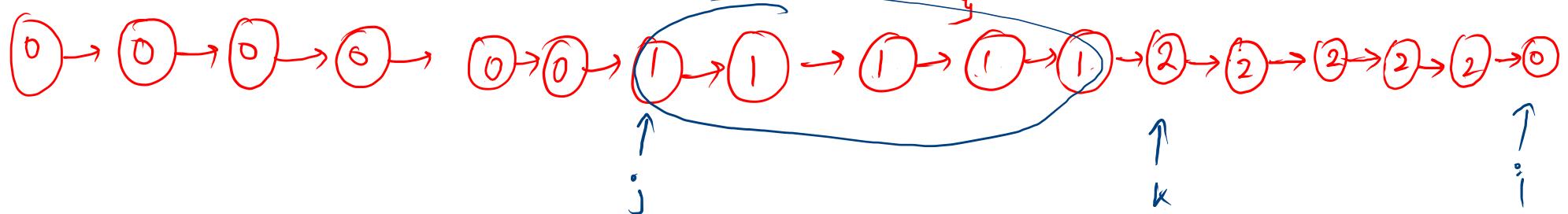
swap(i, j);

swap(i, K);

$i = i.\text{next}$;

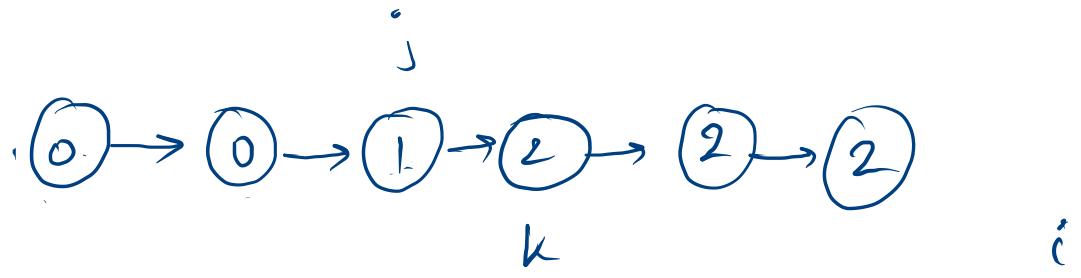
$j = j.\text{next}$;

$K = K.\text{next}$;



- </> Segregate 01 Node Of Linkedlist Over Swapping Nodes
</> Segregate 01 Node Of Linkedlist By Swapping Data
</> Segregate 012 Node Of Linkedlist Over Swapping Nodes
</> Segregate 012 Node Of Linkedlist By Swapping Data

● Easy	10	✓ Auth	0	✓ Public	✓ Sol	21
● Easy	10	✓ Auth	0	✓ Public	✓ Sol	22
● Easy	10	✓ Auth	0	✓ Public	✓ Sol	23
● Easy	10	✓ Auth	0	✓ Public	✓ Sol	24



```

ListNode i = head, j = head, k = head;
while(i != null){
    if(i.val == 2){
        i = i.next;
    }else if(i.val == 1){
        swap(i,k);
        i = i.next;
        k = k.next;
    }else{
        swap(i,j);
        swap(j,i); →
        k = k.next;
        j = j.next;
    }
}
return head;

```

The code implements a three-way partitioning of a linked list. It uses three pointers: *i*, *j*, and *k*. The algorithm moves through the list, and based on the value of the current node *i*, it performs one of three actions:

- If *i* has value 2, it simply moves to the next node.
- If *i* has value 1, it swaps the values of *i* and *k*, then moves *i* to the next node and increments *k*.
- If *i* has value 0, it swaps the values of *i* and *j*, then moves *i* to the next node and increments both *k* and *j*.

Annotations in the code include:

- A bracket under *i = head*, *j = head*, and *k = head*.
- A bracket under the entire while loop body.
- A bracket under the three if statements.
- A bracket under the three swap operations.
- A handwritten note ~~swap(j,i);~~ with an arrow pointing to the swap(i,j) line.
- Checkmarks next to *k = k.next;* and *j = j.next;*


```

public static ListNode segregate012(ListNode head) {
    if(head == null || head.next == null) return head;

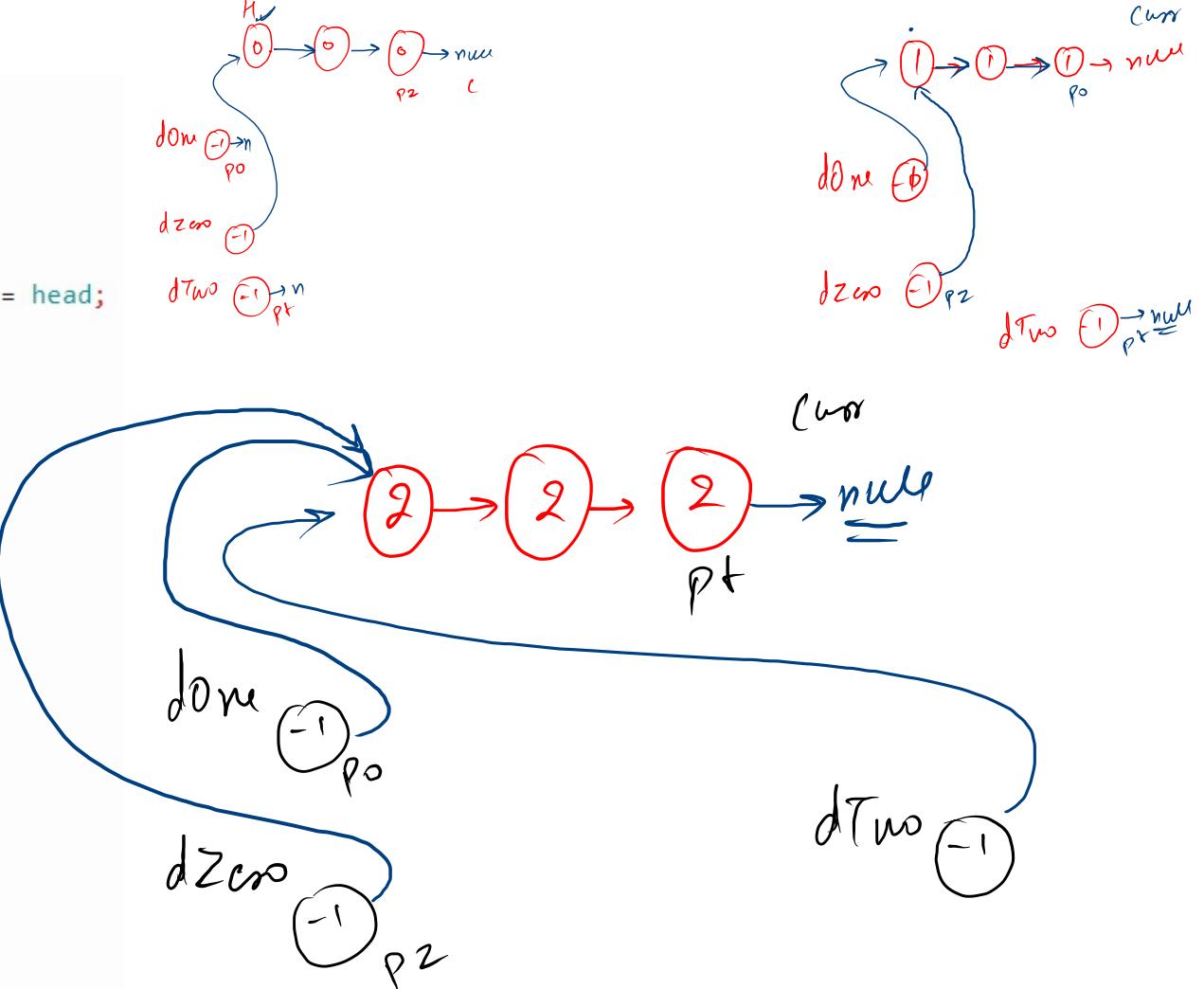
    ListNode dOne = new ListNode(-1);
    ListNode dZero = new ListNode(-1);
    ListNode dTwo = new ListNode(-1);
    ListNode po = dOne, pz = dZero, pt = dTwo, curr = head;

    while(curr != null){
        if(curr.val == 1){
            po.next = curr;
            po = po.next;
        }else if(curr.val == 0){
            pz.next = curr;
            pz = pz.next;
        }else{
            pt.next = curr;
            pt = pt.next;
        }
        curr = curr.next;
    }

    pz.next = dOne.next;
    po.next = dTwo.next;
    pt.next = null;

    return dZero.next;
}

```



Node Swap

```
public static ListNode segregate012(ListNode head) {
    if(head == null || head.next == null) return head;

    ListNode dOne = new ListNode(-1);
    ListNode dZero = new ListNode(-1);
    ListNode dTwo = new ListNode(-1);
    ListNode po = dOne, pz = dZero, pt = dTwo, curr = head;

    while(curr != null){
        if(curr.val == 1){
            po.next = curr;
            po = po.next;
        }else if(curr.val == 0){
            pz.next = curr;
            pz = pz.next;
        }else{
            pt.next = curr;
            pt = pt.next;
        }
        curr = curr.next;
    }

    po.next = dTwo.next;
    pz.next = dOne.next;
    pt.next = null;

    return dZero.next;
}
```

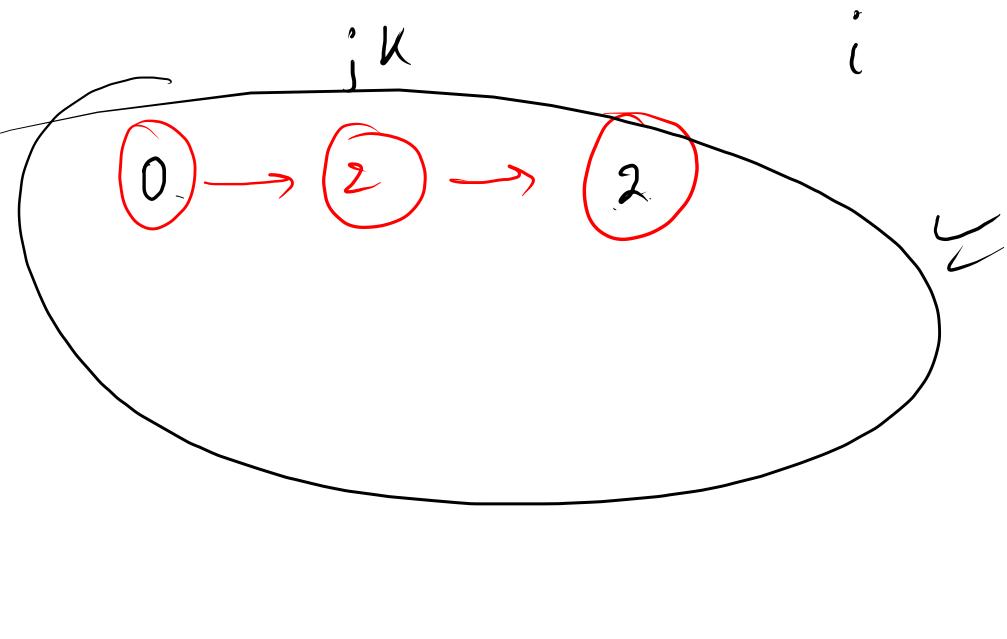
Data Swap

```
public static ListNode segregate012(ListNode head) {

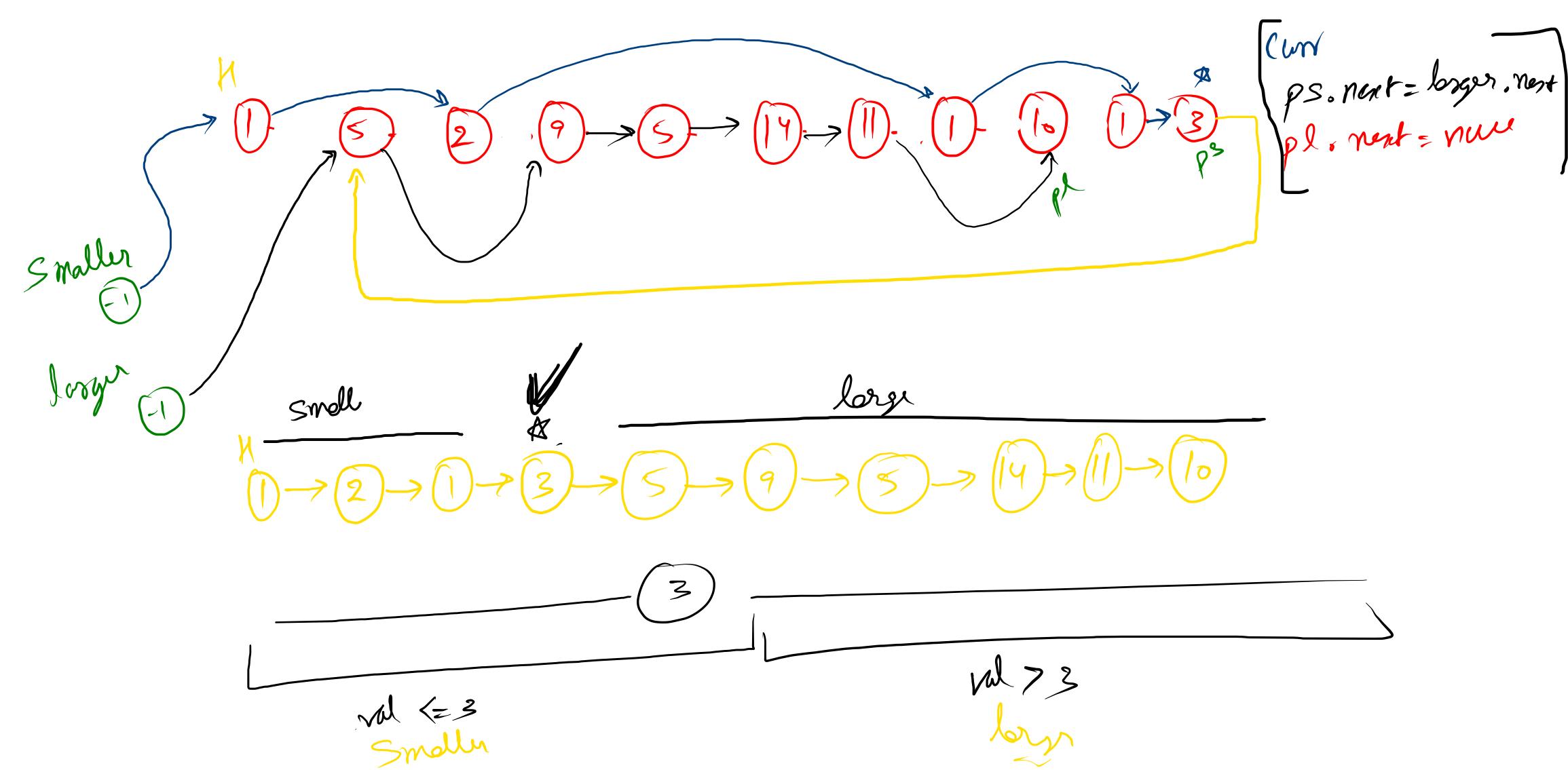
    ListNode i = head, j = head, k = head;

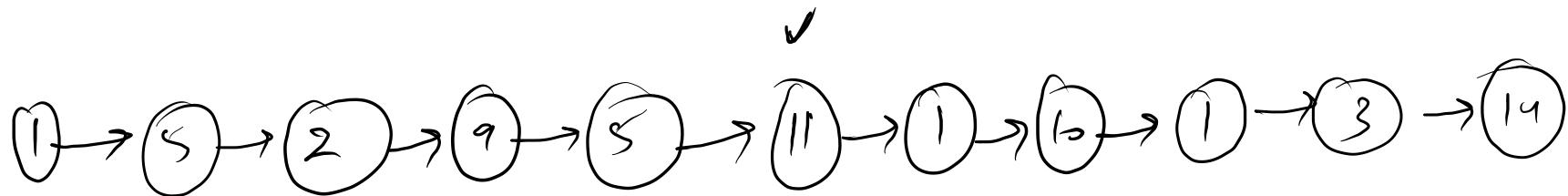
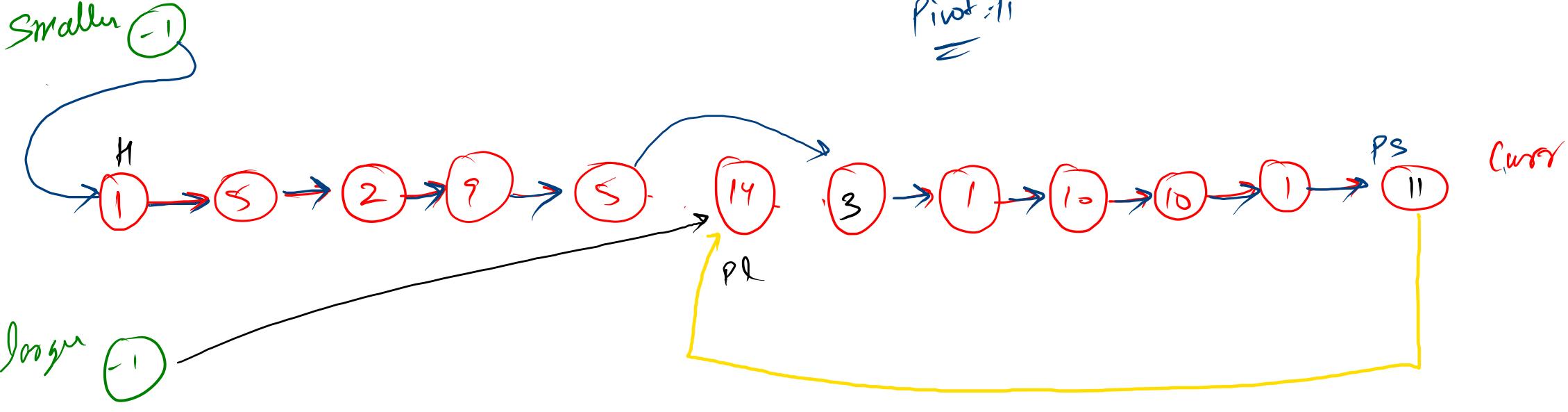
    while(i != null){
        if(i.val == 2){
            i = i.next;
        }else if(i.val == 1){
            swap(i,k);
            i = i.next;
            k = k.next;
        }else{
            swap(i,j);
            if(i.val == 1){
                swap(i,k);
            }
            i = i.next;
            k = k.next;
            j = j.next;
        }
    }

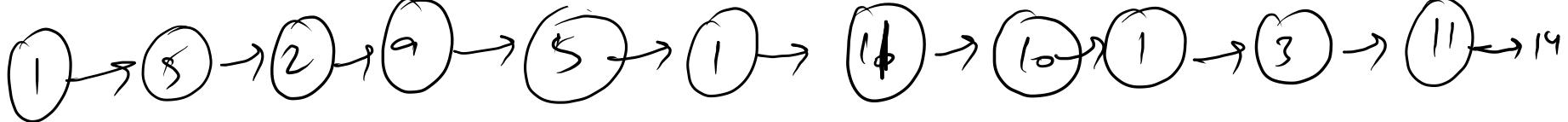
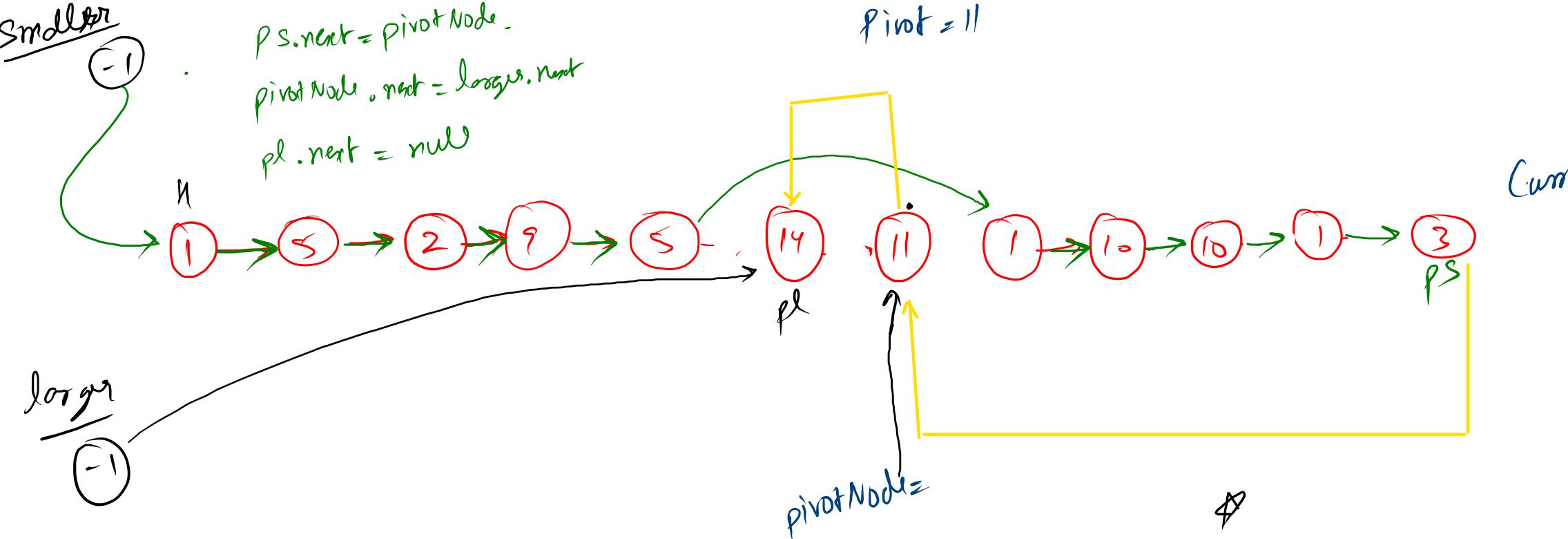
    return head;
}
```



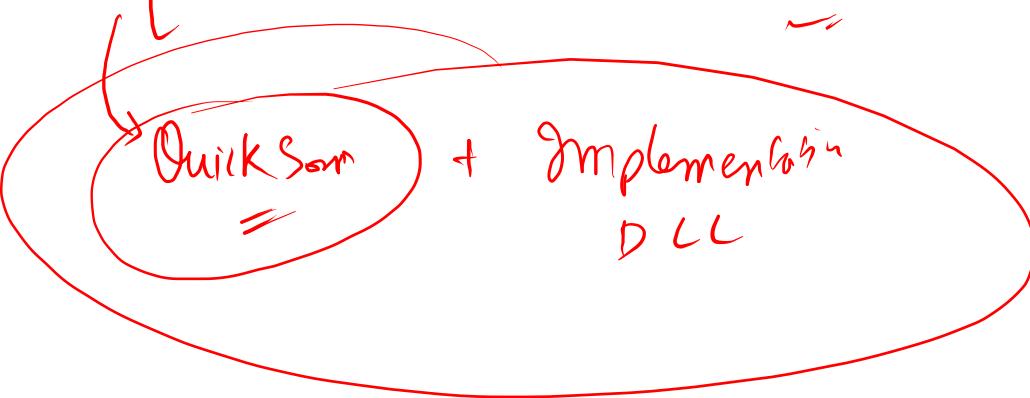
```
ListNode i = head , j = head , k = head;  
  
while(i != null){  
    if(i.val == 2){  
        i = i.next;  
    }else if(i.val == 1){  
        swap(i,k);  
        i = i.next;  
        k = k.next;  
    }else{  
        swap(i,j);  
        if(i.val == 1){  
            swap(i,k);  
        }  
        i = i.next;  
        k = k.next;  
        j = j.next;  
    }  
}  
  
return head;
```







<input checked="" type="checkbox"/>	Segregate 01 Node Of Linkedlist Over Swapping Nodes	Easy	10	✓ Auth	0	✓ Public	✓ Sol	21
<input checked="" type="checkbox"/>	Segregate 01 Node Of Linkedlist By Swapping Data	Easy	10	✓ Auth	0	✓ Public	✓ Sol	22
<input checked="" type="checkbox"/>	Segregate 012 Node Of Linkedlist Over Swapping Nodes	Easy	10	✓ Auth	0	✓ Public	✓ Sol	23
<input checked="" type="checkbox"/>	Segregate 012 Node Of Linkedlist By Swapping Data	Easy	10	✓ Auth	0	✓ Public	✓ Sol	24
<input checked="" type="checkbox"/>	Segregate Node Of Linkedlist Over Last Index.	Easy	10	✓ Auth	0	✓ Public	✓ Sol	25
<input checked="" type="checkbox"/>	Segregate Node Of Linkedlist Over Pivot Index	Medium	10	✓ Auth	0	✓ Public	✓ Sol	26


 Quick Sort + Implementation
 =
 DLL

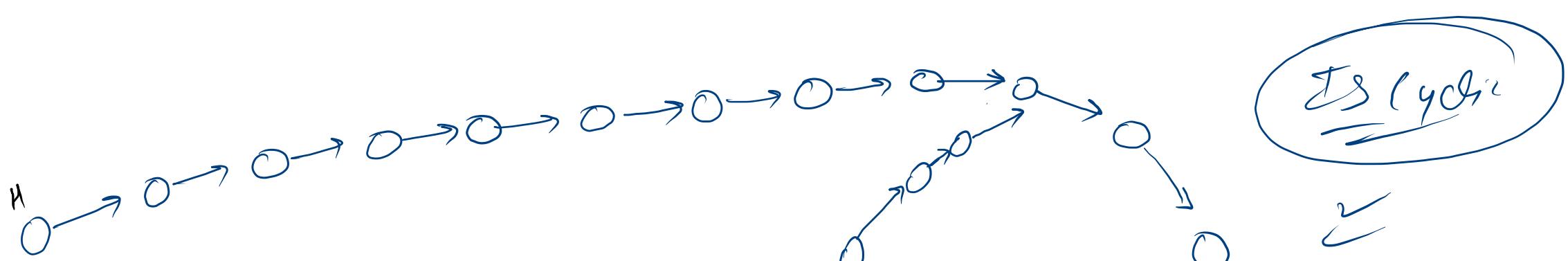
Is Cycle Present In Linkedlist

Cycle Node In Linkedlist

Intersection Node In Two Linkedlist Using Difference Method

Intersection Node In Two Linkedlist Using Floyd Cycle Me...

<input checked="" type="checkbox"/> Is Cycle Present In Linkedlist	● Easy	10	✓ Auth	0	<input type="checkbox"/> Public	✓ Sol	15
<input type="checkbox"/> Cycle Node In Linkedlist	● Easy	10	✓ Auth	0	<input type="checkbox"/> Public	✓ Sol	16
<input type="checkbox"/> Intersection Node In Two Linkedlist Using Difference Method	● Easy	10	✓ Auth	0	<input type="checkbox"/> Public	✓ Sol	17
<input type="checkbox"/> Intersection Node In Two Linkedlist Using Floyd Cycle Me...	● Easy	10	✓ Auth	0	<input type="checkbox"/> Public	✓ Sol	18



```
while (fast != null && fast.next != null)
```

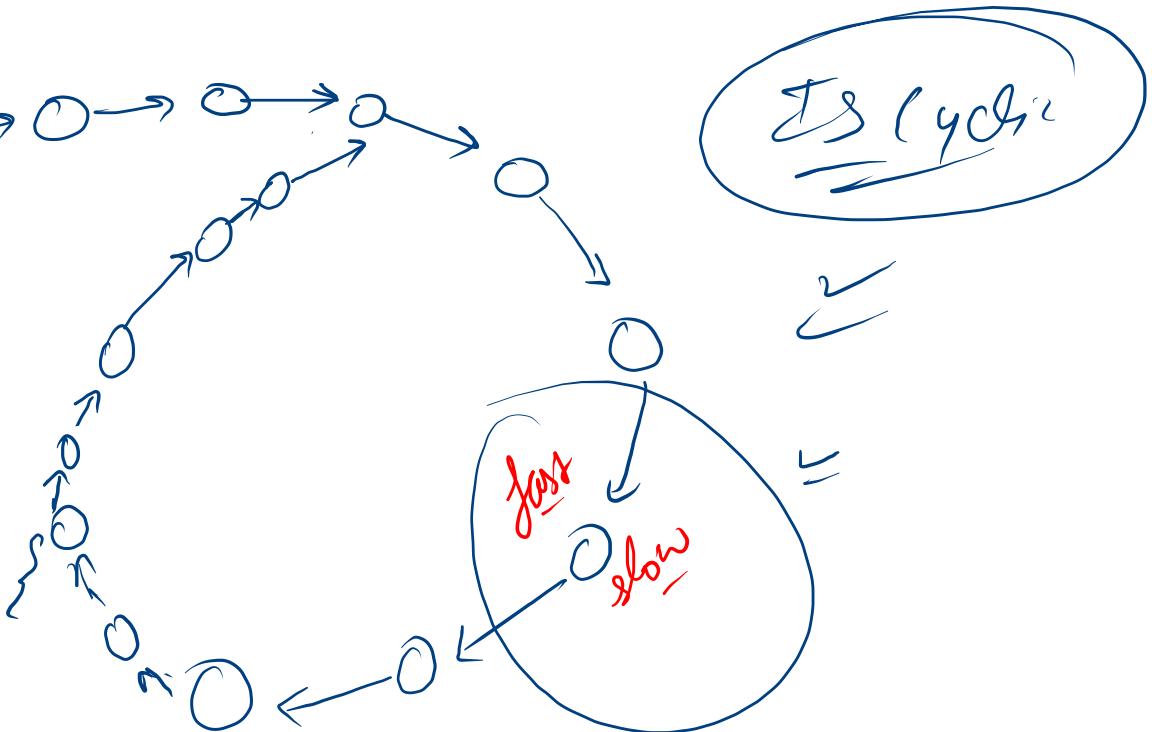
```
    fast = fast.next.next;
```

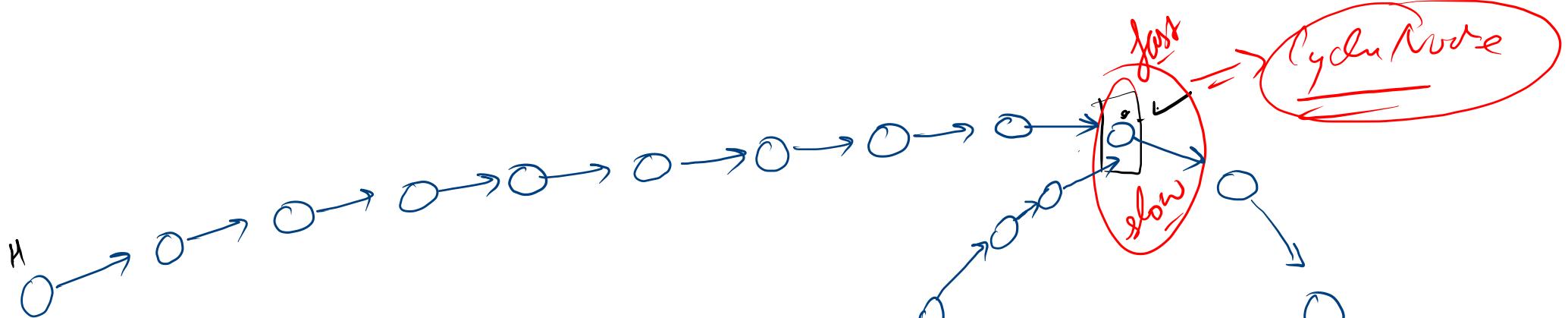
```
    slow = slow.next;
```

```
    if (fast == slow) {
```

```
        return true;
```

```
}
```





```
while (fast != null && fast.next != null)
```

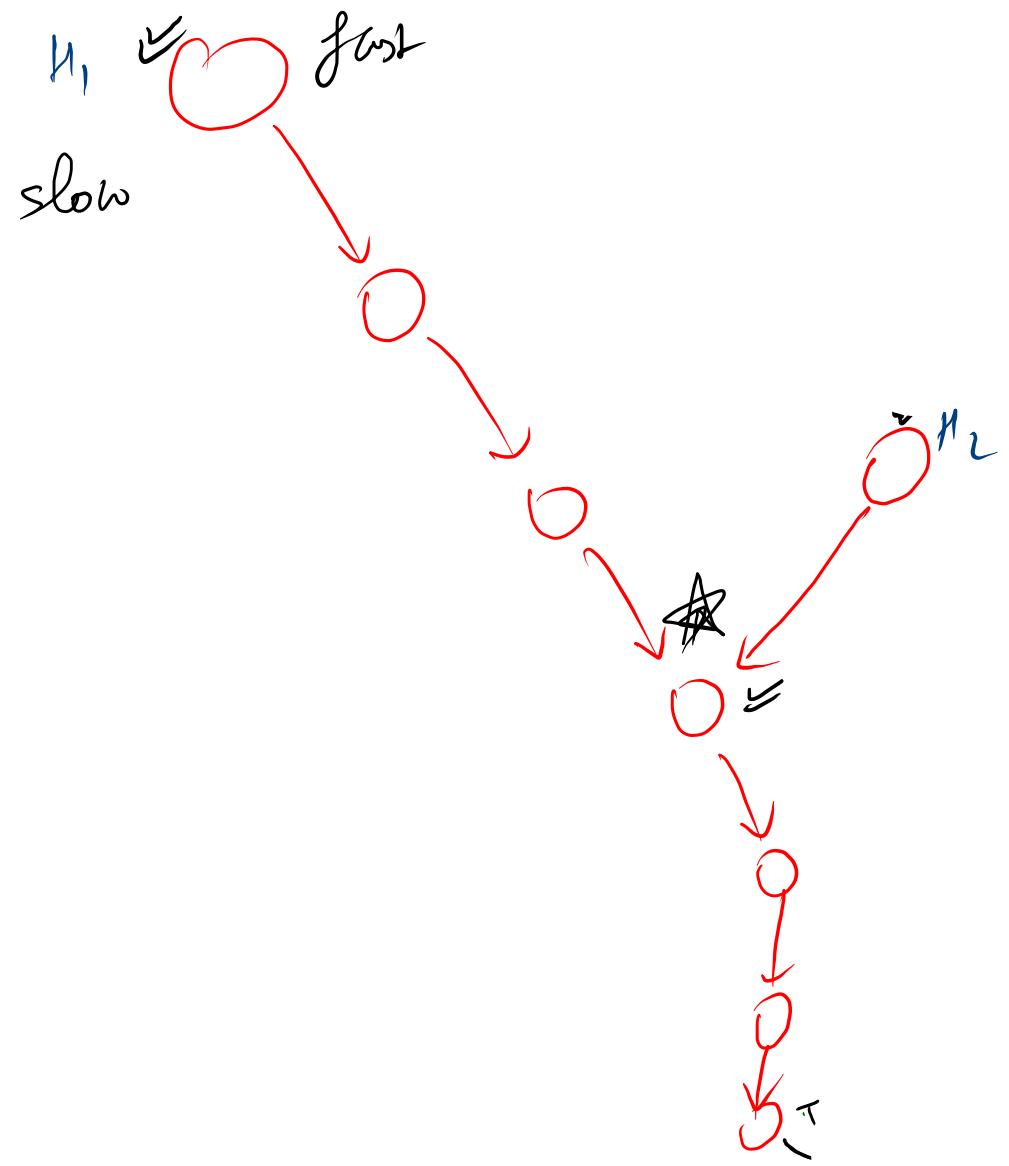
```
    fast = fast.next.next;
```

```
    slow = slow.next;
```

```
    if (fast == slow){  
        break;  
    }
```

```
    if (fast != slow){  
        return null  
    }
```

fast = head;



	</> Is Cycle Present In Linkedlist			✓ Auth		□ Public	✓ Sol	
	</> Cycle Node In Linkedlist			✓ Auth		□ Public	✓ Sol	
	</> Intersection Node In Two Linkedlist Using Difference Method			✓ Auth		□ Public	✓ Sol	
	</> Intersection Node In Two Linkedlist Using Floyd's Cycle Method			✓ Auth		□ Public	✓ Sol	
	</> Remove Duplicate From Sorted Linkedlist			✓ Auth		✓ Public	✓ Sol	
	</> Remove All Duplicates From Sorted Linkedlist			✓ Auth		✓ Public	✓ Sol	
	</> Segregate 01 Node Of Linkedlist Over Swapping Nodes			✓ Auth		✓ Public	✓ Sol	
	</> Segregate 01 Node Of Linkedlist By Swapping Data			✓ Auth		✓ Public	✓ Sol	
	</> Segregate 012 Node Of Linkedlist Over Swapping Nodes			✓ Auth		✓ Public	✓ Sol	
	</> Segregate 012 Node Of Linkedlist By Swapping Data			✓ Auth		✓ Public	✓ Sol	
	</> Segregate Node Of Linkedlist Over Last Index.			✓ Auth		✓ Public	✓ Sol	
	</> Segregate Node Of Linkedlist Over Pivot Index			✓ Auth		✓ Public	✓ Sol	
	</> Quicksort In Linkedlist			✓ Auth		□ Public	✓ Sol	

?