Question 1: How could we implement a queue by using two stacks?

Stack1|| 1 2 3 4 5 6 ..... n

Stack2|| n n-1 n-2 .... 6 5 4 3 2 1

```
Add()   : TIme = O(1)
Remove()or Pop():   O(n)

Amortized time complexity = O(1)
```

Why??????

1st time call Pop() : n + n +1

2nd time call Pop(): 1

3rd time... 1

4th 1

...

n 1

$$\frac{1 \times (2n+1) + (n-1) \times 1}{n} == 3n / n = O(3) = O(1)$$

Question 3: How to sort numbers with three (or two) stacks (taught in Class 1)

S1|| 3 4

S2|| **1** 2 2 2 ||

int **global_min** = 2

int **counter_of_min** = 3

method1:

```
while (S2.size() > size before this iteration) {
}
```

method 2:

```
while (!s2.empty() && S2.top() >= global_min) {
}
```

Key points:

1. When you want to de-reference a ListNode, make sure it is not a NULL pointer

```
ListNode* p = new ListNode(10);          // p = 0XFFFF0001
...p->value
p.value
p.next.next
   null
```

2. Never ever lost the control of the head pointer of the LinkedLIst,

E1 ← E2 E3 → E4 → … En→ NULL

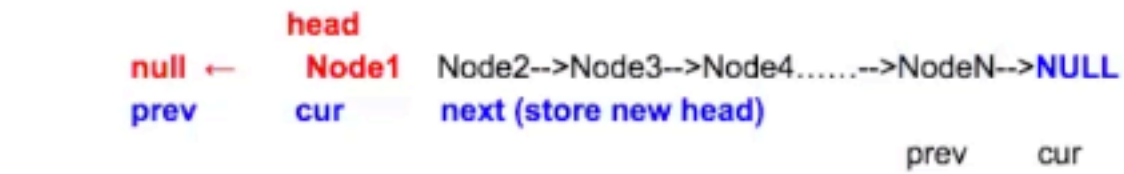
head sub-section **tail**

常见考题： **No.1 interview question on linkedlist: how to reverse a linked list**

Node1->Node2-->Node3-->Node4……-->NodeN-->NULL
head

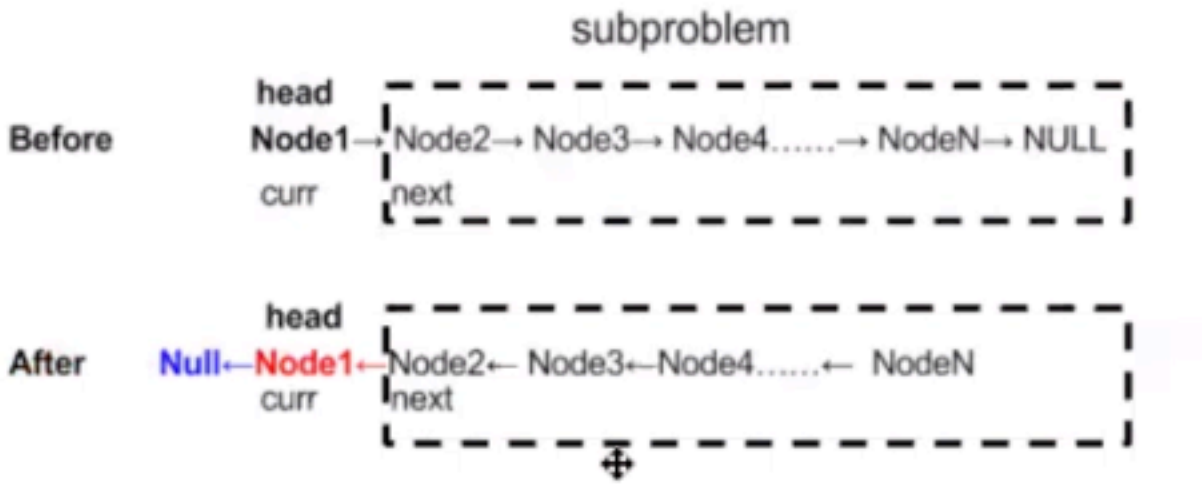
reversed: NULL <--Node1<-Node2<--Node3<--Node4……<--NodeN
head

Before




```
00 public ListNode reverseList(ListNode head){
01     if (head == null || head.next == null) {
02         return head;
03     }
04
05     ListNode prev = null;
06     ListNode next = null;
07     ListNode curNode = head;
08
09     while (curNode != null) {
10         next = curNode.next;    // store the new head of sub-list
11         curNode.next = prev;    // reverse happens here!!!
12         prev = curNode;         // move prev by 1 step
13         curNode = next;         // move cur by 1 step
14     }
15     return prev;
16 }
```

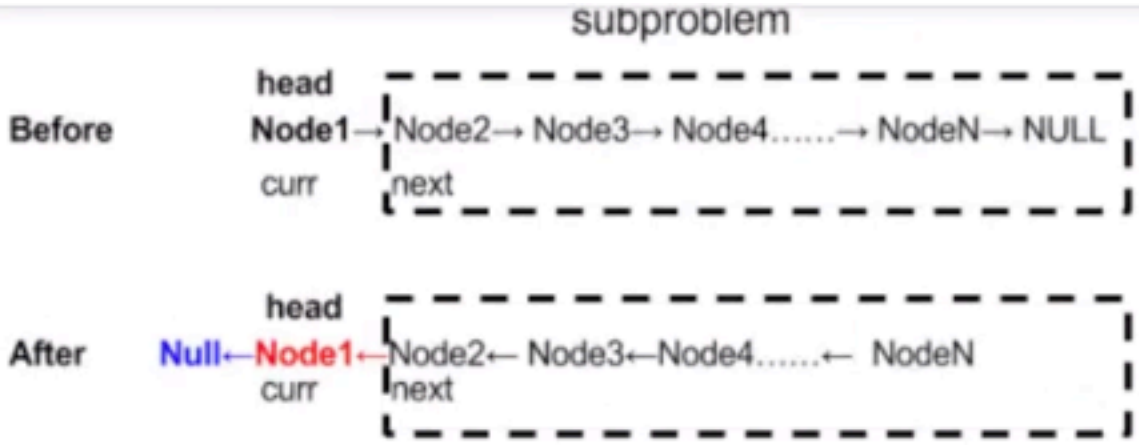
Method 2:



除了subproblem外几处不同？

(1) next→ next = curr; // subproblem head 指向current node;

(2) curr → next = null; // current node's next is set to Null;



除了subproblem外几处不同？

(1) next→ next = curr; // subproblem head 指向current node;

(2) curr → next = null; // current node's next is set to Null;

Solution:

```
// Recursive way:
00 public ListNode reverseList(ListNode head) {
        If (head == null || head.next == null) {
                Return head;                    // base case
        }
        ListNode newHead = reverseList(head.next);   // newHead =N1000
        Head.next.next = head;
        Head.next = null;
        Return newHead;
}
```

Solution:

```
// Recursive way:
00 public ListNode reverseList(ListNode head = N3) {
01      If (head == null || head.next == null) {
02              Return head;                    // base case
03      }
04      ListNode newHead = reverseList(head.next);   // newHead =N1000

05      Head.next.next = head;
06      Head.next = null;
07      Return newHead;
08 }
```

Example step-by-step

1st call R-func: **Node1** → Node2 → Node3 → NULL line04 bp...
head nextNode

2nd call R-func: **Node1** → Node2 → Node3 → NULL line04 bp...
head nextNode

3rd call R-func: **Node1** → Node2 → Node3 → NULL return N3 as newHead
head