Linked List + Stack [Day 15]

09 October 2024 16:03

Linked List + Stack [Day 15]

141. Linked List Cycle

Solved

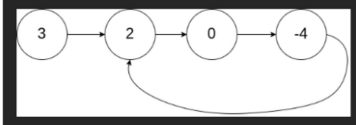
Easy Topics Companies

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

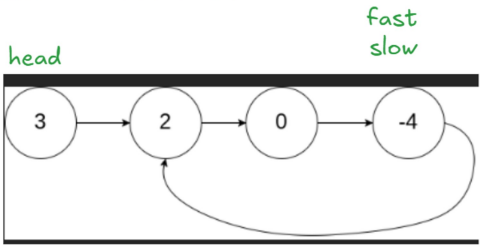
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:

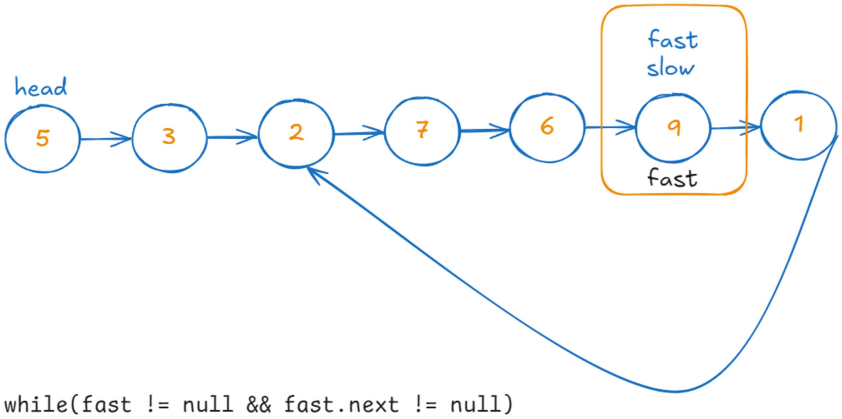


Input: head = [3,2,0,-4], pos = 1



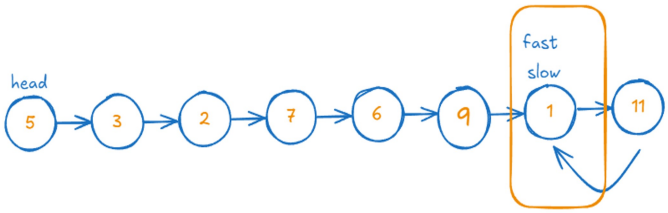

```
public boolean hasCycle(ListNode head) {

}
```

return True/false




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

```java
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null || head.next == null){
            return false;
        }
        ListNode slow = head;
        ListNode fast = head;

        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;

            if(slow == fast){
                return true;
            }
        }
        return false;
    }
}
```



142. Linked List Cycle II

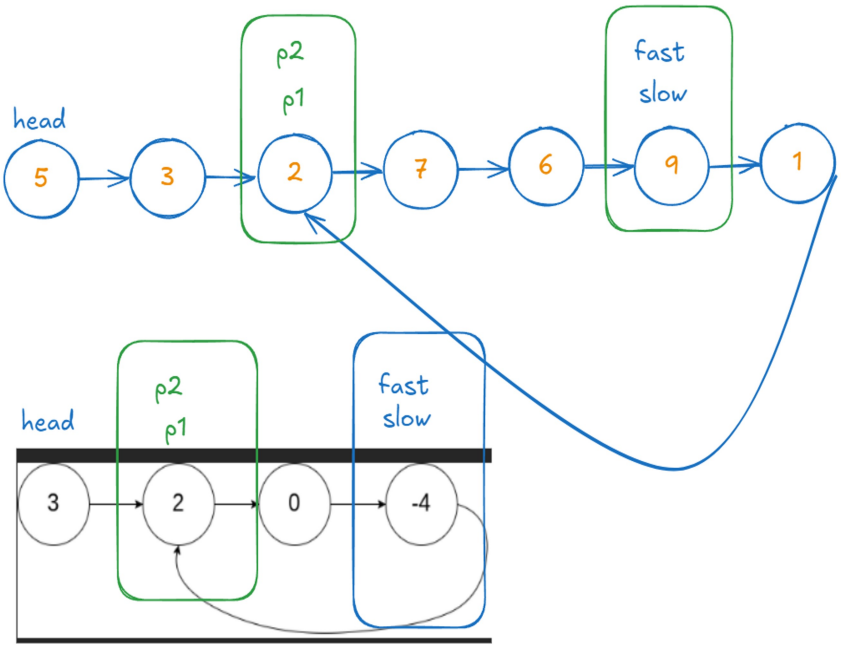
Solved

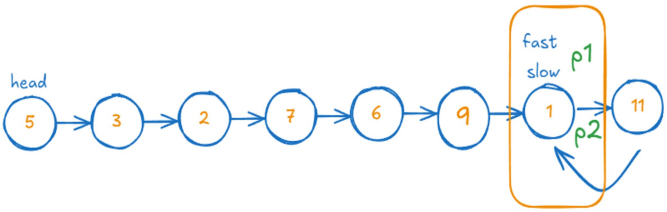
Medium · Topics · Companies

Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter.**

Do not modify the linked list.





```java
public ListNode detectCycle(ListNode head) {
    if(head == null || head.next == null){
        return null;
    }
    ListNode slow = head;
    ListNode fast = head;

    while(fast != null && fast.next != null){
        slow = slow.next;
        fast = fast.next.next;

        if(slow == fast){
            ListNode p1 = head;
            ListNode p2 = slow;
            while(p1 != p2){
                p1 = p1.next;
                p2 = p2.next;
            }
            return p1;
        }
    }
```

21. Merge Two Sorted Lists

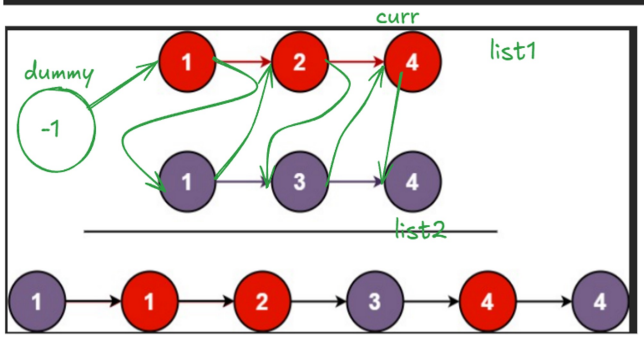
Solved

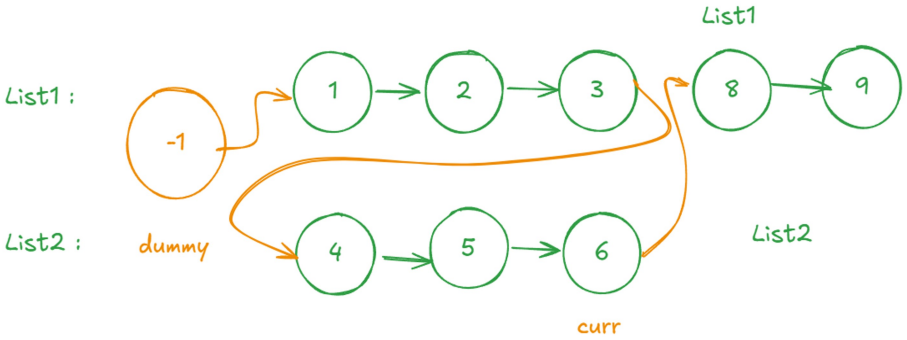
Easy | Topics | Companies

You are given the heads of two sorted linked lists `list1` and `list2`.

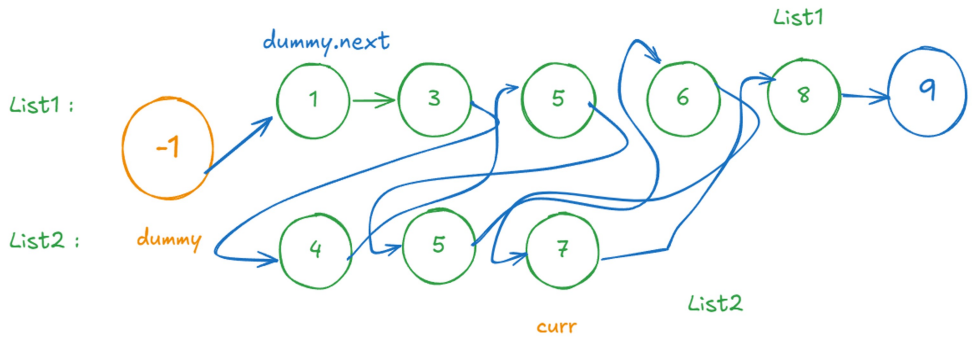
Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list.*





return dummy.next



1 -- 3 -- 4 -- 5 -- 5 -- 6 -- 7 -- 8 -- 9

return dummy.next

```java
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(-1);
        ListNode curr = dummy;

        while(list1 != null && list2 != null){
            if(list1.val <= list2.val){
                curr.next = list1;
                list1 = list1.next;
            }else{
                curr.next = list2;
                list2 = list2.next;
            }
            curr = curr.next;
        }
        if(list1 != null){
            curr.next = list1;
        }else{
            curr.next = list2;
        }
```

328. Odd Even Linked List

Solved

Medium | Topics | Companies

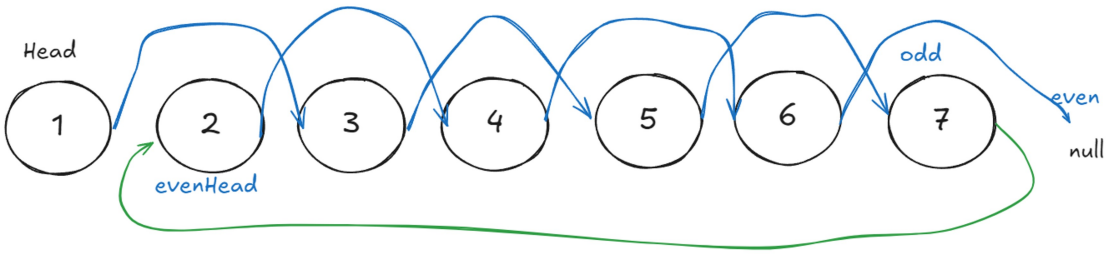
Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

```java
class Solution {
    public ListNode oddEvenList(ListNode head) {

    }
}
```



odd.next = odd.next.next; even.next = even.next.next;

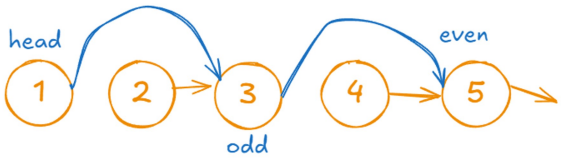
odd = odd.next even= even.next

when loops break - odd.next = evenHead;

1 -- 3 -- 5 -- 7 -- 2 -- 4 -- 6 -- null

```java
class Solution {
    public ListNode oddEvenList(ListNode head) {
        if(head == null || head.next == null){
            return head;
        }
        ListNode odd = head;
        ListNode even = head.next;
        ListNode evenHead = even;

        while(even != null && even.next != null){
            odd.next = odd.next.next;
            odd = odd.next;

            even.next = even.next.next;
            even = even.next;
        }
        odd.next = evenHead;

        return head;
    }
```

```java
class Solution {
    public ListNode oddEvenList(ListNode head) {
        ListNode odd=head;
        ListNode even=head.next;
        ListNode Evehead=even;
        while(odd!=null && odd.next!=null){
            odd.next=odd.next.next;
            odd=odd.next;
        }
        while(even!=null && even.next!=null){
            even.next=even.next.next;
            even=even.next;
        }
        odd.next=Evehead;
        return head;

    }
}
```



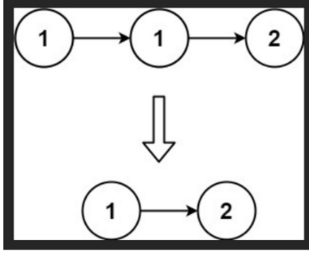
83. Remove Duplicates from Sorted List

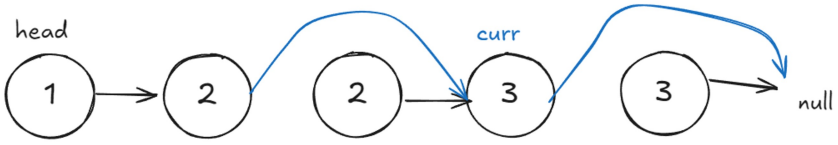
Solved

Easy Topics Companies

Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once.* Return *the linked list* ***sorted*** *as well.*

Example 1:





curr.val == curr.next.val -- if they are same : curr.next = curr.next.next
else : curr = curr.next;

```java
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if(head == null || head.next == null){
            return head;
        }
        ListNode curr = head;
        while(curr != null && curr.next != null){
            if(curr.val == curr.next.val){
                curr.next = curr.next.next;
            }else{
                curr = curr.next;
            }
        }
        return head;
    }
}
```

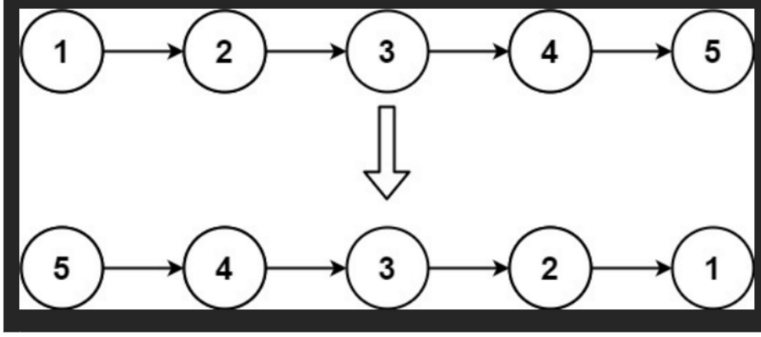
206. Reverse Linked List

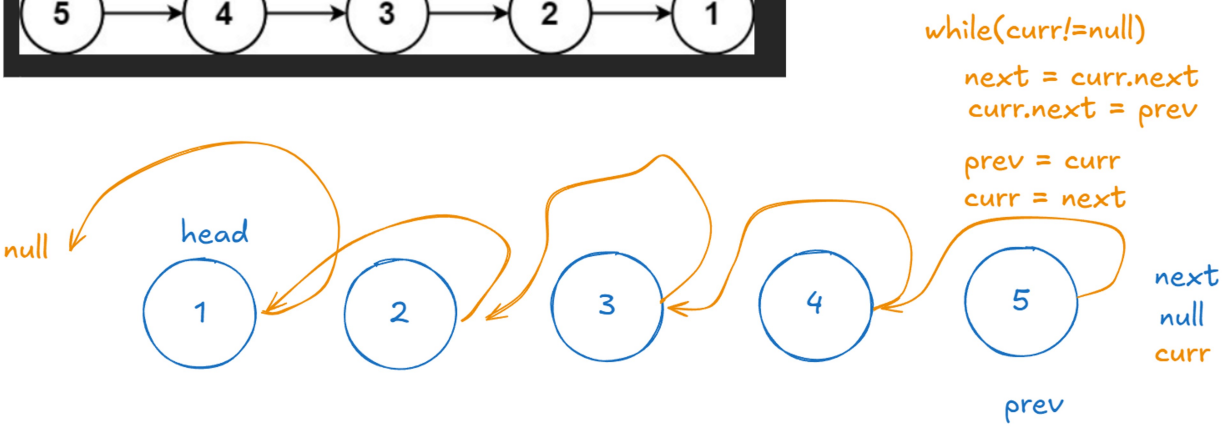
Solved

Easy | Topics | Companies

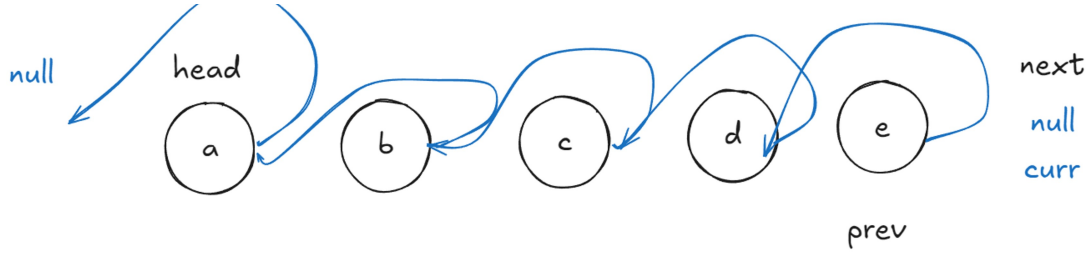
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:





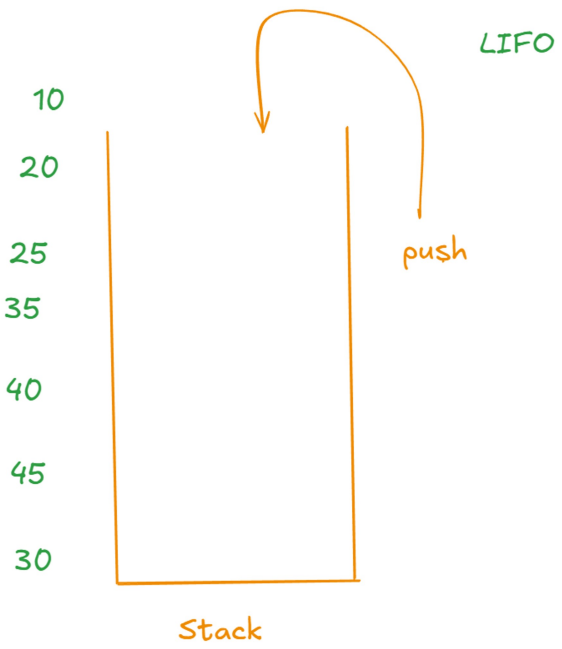
return 5 -- 4 -- 3 -- 2 -- 1 -- null



e - d - c - b - a - null

```java
*/
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;
        ListNode next = null;

        while(curr != null){
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

Introduction To Stack.....



push -- adding an element into stack
pop -- remove an element out of stack
peek -- top of the stack
isEmpty -- returns boolean, whether your stack is empty or not.

Stack st = new Stack(5); // Capcity

push(10)

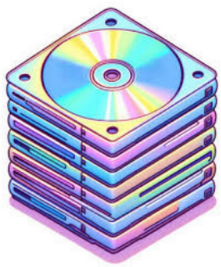
push(20)

size = 2

push(30)

st.peek() --> 30

top = st.pop() --> 30



push(25)

push(35)

push(40)

push(45)

4 times Pop()

st.isEmpty() --> false

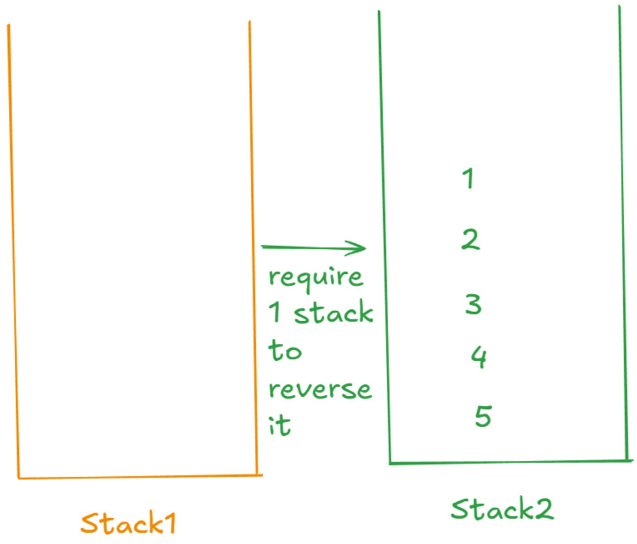
st.pop()

st.pop()

Underflowed

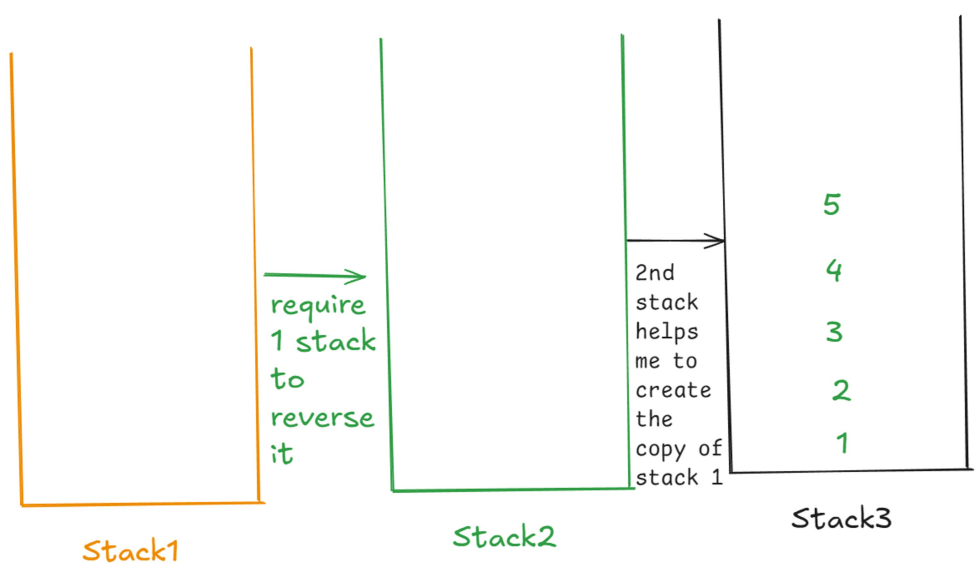
OverFlowed

Reverse the Stack Element using another Stack.




```
while(!stack1.isEmpty()){
    stack2.push(stack1.pop())
}
```

Copy the Stack



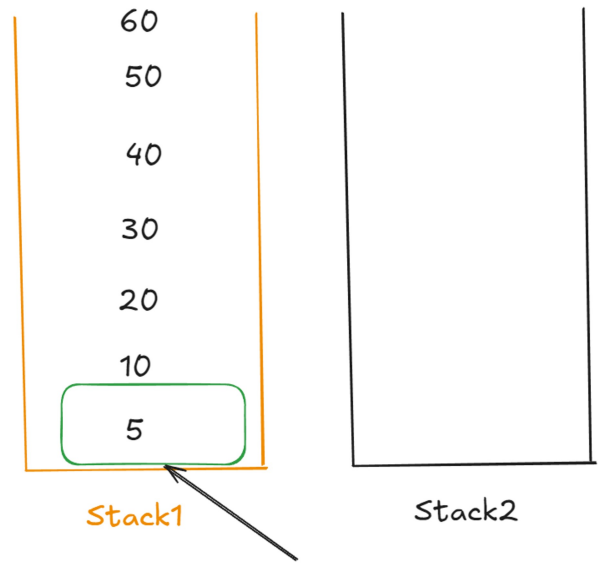

```
while(!stack1.isEmpty()){
    stack2.push(stack1.pop())
}
```

```
while(!stack2.isEmpty()){
    stack3.push(stack2.pop())
}
```

Change Status

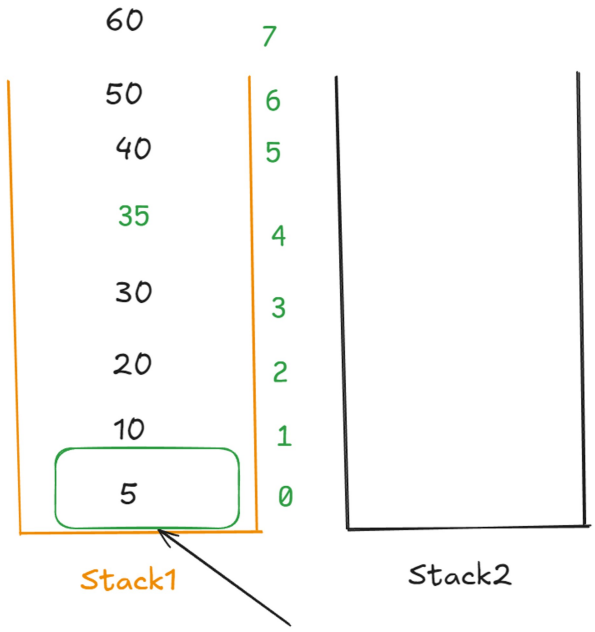
Completed

Attendance Code: 39C81D1F




```
while(!stack1.isEmpty()){
    Stack2.push(Stack1.pop());
}

st.push(5);

while(!stack2.isEmpty()){
    Stack1.push(Stack2.pop());
}
```

Insert at bottom(5)



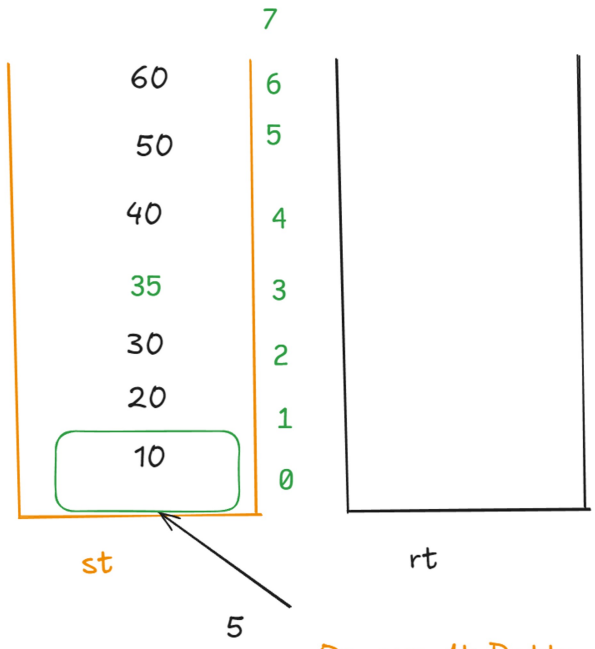

```
for(int i = 0 ; i< size - index; i++){
    Stack2.push(Stack1.pop());
}


Stack1.push(35);

while(!stack2.isEmpty()){
    Stack1.push(Stack2.pop());
}
```

size - idx == pop()

Insert at particular - Index : 4 , 7-4 = 3

idx < 0 || idx > size -- invalid

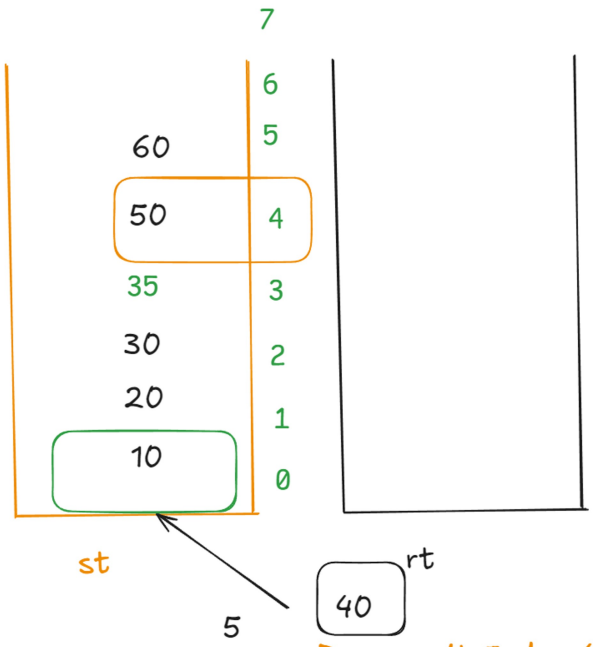
Remove At Bottom




```
while(st.size() > 1){
    rt.push(st.pop())
}

st.pop()

while(!rt.isEmpty()){
    st.push(rt.pop())
}
```

Remove At Bottom

Remove At Specific Index(4)



7 - 4 - 1 = 2

```
for(int i = 0 ; i< size - index - 1; i++){
    rt.push(st.pop());
}

 st.pop()

while(!rt.isEmpty()){
    st.push(rt.pop())
}
```

Remove At Index 4 : where the element is 40