```java
public class Solution {
  private Deque<Integer> s1;
  private Deque<Integer> s2;
  public Solution() {
    s1 = new ArrayDeque<>();
    s2 = new ArrayDeque<>();
  }
  public Integer poll() {
    if (s1.isEmpty() && s2.isEmpty()) {
      return null;
    }
    shuffle(s1, s2);
    return s2.pollFirst();
  }

  public void offer(int element) {
    s1.offerFirst(element);
  }

  public Integer peek() {
    if (s1.isEmpty() && s2.isEmpty()) {
      return null;
    }
    shuffle(s1, s2);
    return s2.peekFirst();
  }

  public int size() {
    return s1.size() + s2.size();
  }

  public boolean isEmpty() {
    return s1.isEmpty() && s2.isEmpty();
  }
  public void shuffle(Deque<Integer> s1, Deque<Integer> s2) {
    if (s2.isEmpty()) {
      while (!s1.isEmpty()) {
        s2.offerFirst(s1.pollFirst());
      }
    }
  }
}

```

```java
//最小栈 easy 版本 双端队列
class MinStack {

    /** initialize your data structure here. */
    Deque<Integer> mStack;
    Deque<Integer> minStack;
    public MinStack() {
        mStack = new LinkedList<Integer>();
        minStack = new LinkedList<Integer>();
        minStack.push(Integer.MAX_VALUE);

    }

    public void push(int x) {
        mStack.push(x);
        minStack.push(Math.min(minStack.peek(),x));
    }

    public void pop() {
        mStack.pop();
        minStack.pop();
    }

    public int top() {
        return mStack.peek();
    }

    public int getMin() {
        return minStack.peek();
    }
}

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack obj = new MinStack();
 * obj.push(x);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */


#
链接：https://leetcode-cn.com/problems/min-stack/solution/di-yi-bian-xian-ba-da-an-chao-hui-by-gavin-131/

// 最小栈 mid 版本 双端队列
public class Solution {
  private Deque<Integer> stack;
  private Deque<Integer> minStack;
  public Solution() {
    stack = new LinkedList<Integer>();
    minStack = new LinkedList<Integer>();
  }
  public Integer min() {
    if (minStack.isEmpty()){
      return -1;
    }
    return minStack.peekFirst();
  }
  public void push(int value){
    stack.offerFirst(value);
    if(minStack.isEmpty()|| value <= minStack.peekFirst()){
      minStack.offerFirst(value);
    }
  }
  public Integer pop(){
    if(stack.isEmpty()){
      return -1;
    }
    Integer result = stack.pollFirst();
```

```
70        if (minStack.peekFirst().equals(result)){
71          minStack.pollFirst();
72        }
73        return result;
74      }
75      public Integer top(){
76        if(stack.isEmpty()){
77          return -1;
78        }
79        return stack.peekFirst();
80      }
81    }
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode reverse(ListNode head) {
    if (head == null || head.next == null) {
      return head;
    }
    ListNode prev = null;
    ListNode curr = head;
    while (curr != null) {
      ListNode next = curr.next;
      curr.next = prev;
      prev = curr;
      curr = next;
    }
    return prev;
  }
}
```

```
1
2  # shuffle stacks
3  class Solution {
4      public void shuffle(Deque<Integer> stack1, Deque<Integer> stack2) {
5        while (!stack1.isEmpty()) {
6          stack2.push(stack1.pop());
7        }
8      }
9  }
10
```

```java
public class Solution {
  public ListNode generate(int n) {
    ListNode head = new ListNode(0);
    ListNode cur = head;
    for (int i = 1 ; i < n ; i++){
      cur.next = new ListNode(i);
      cur = cur.next;
    }
    return head;
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public int count(ListNode head) {
    ListNode cur = head;
    int i =0;
    while (cur != null){
      i++;
      cur = cur.next;
    }
    return i ;
  }
}

```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode reverse(ListNode head) {
    if(head ==null || head.next == null){
      return head;
    }
    ListNode curr = reverse(head.next);
    head.next.next = head;
    head.next = null;
    return curr;
  }
}

```