599. Insert into a Cyclic Sorted List ☆

Description Notes >_ Testcase Judge

Given a node from a cyclic linked list which has been sorted, write a function to insert a value into the list such that it remains a cyclic sorted list. The given node can be any single node in the list. Return the inserted new node.

Notice

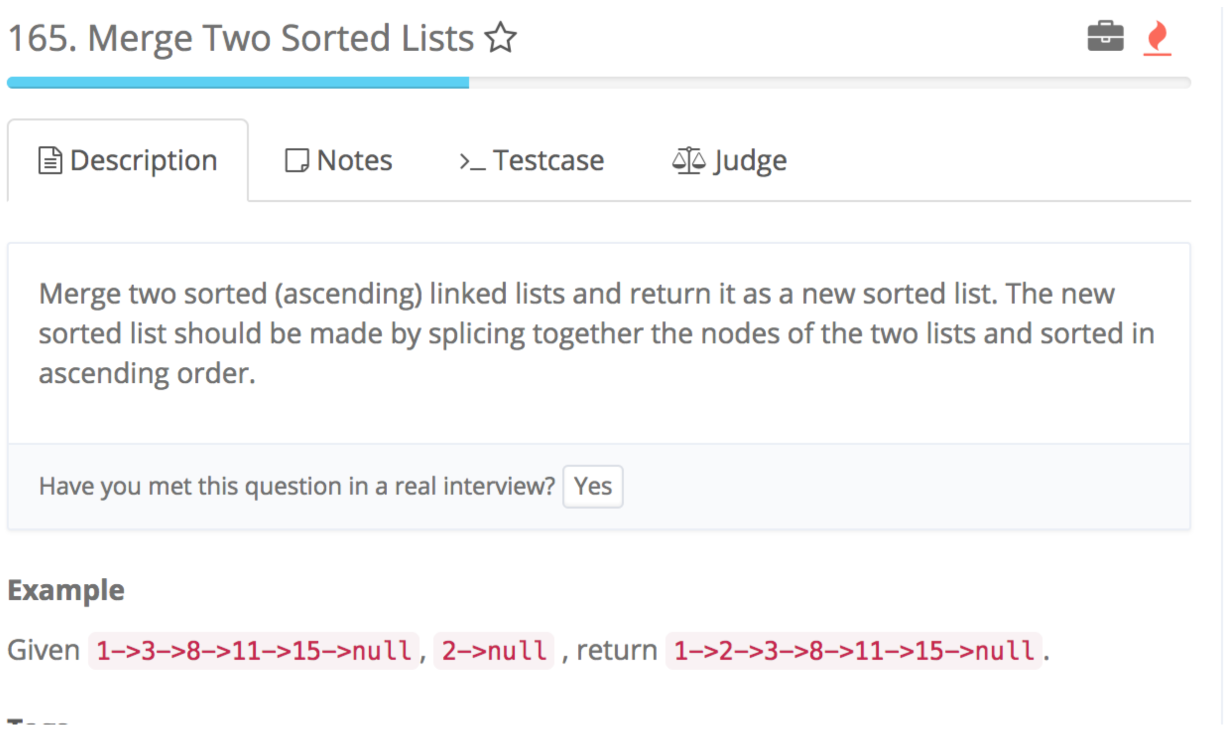
`3->5->1` is a cyclic list, so `3` is next node of `1`.
`3->5->1` is same with `5->1->3`

Have you met this question in a real interview? Yes

Example

Given a list, and insert a value `4`:
`3->5->1`
Return `5->1->3->4`

```
/**
 * Definition for ListNode
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */

public class Solution {
    /**
     * @param node a list node in the list
     * @param x an integer
     * @return the inserted new list node
     */
    public ListNode insert(ListNode node, int x) {
        // Write your code here
       if (node == null) {
            node = new ListNode(x);
            node.next = node;
            return node;
        }

        ListNode p = node;
        ListNode prev = null;
        do {
            prev = p;
            p = p.next;
            if (x <= p.val && x >= prev.val) {
                break;
            }
            if ((prev.val > p.val) && (x < p.val || x > prev.val)) {
                break;
            }
        } while (p != node);

        ListNode newNode = new ListNode(x);
        newNode.next = p;
        prev.next = newNode;
        return newNode;
    }
}
```



165. Merge Two Sorted Lists

Merge two sorted (ascending) linked lists and return it as a new sorted list. The new sorted list should be made by splicing together the nodes of the two lists and sorted in ascending order.

Have you met this question in a real interview? Yes

Example

Given `1->3->8->11->15->null`, `2->null` , return `1->2->3->8->11->15->null`.

```
/**
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int val) {
 *         this.val = val;
 *         this.next = null;
 *     }
 * }
 */


public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode lastNode = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                lastNode.next = l1;
                l1 = l1.next;
            } else {
                lastNode.next = l2;
                l2 = l2.next;
            }
            lastNode = lastNode.next;
        }

        if (l1 != null) {
            lastNode.next = l1;
        } else {
            lastNode.next = l2;
        }

        return dummy.next;
    }
}
```

138. Subarray Sum ☆

Description Notes >_ Testcase Judge

Given an integer array, find a subarray where the sum of numbers is **zero**. Your code should return the index of the first number and the index of the last number.

Notice

There is at least one subarray that it's sum equals to zero.

Have you met this question in a real interview? Yes

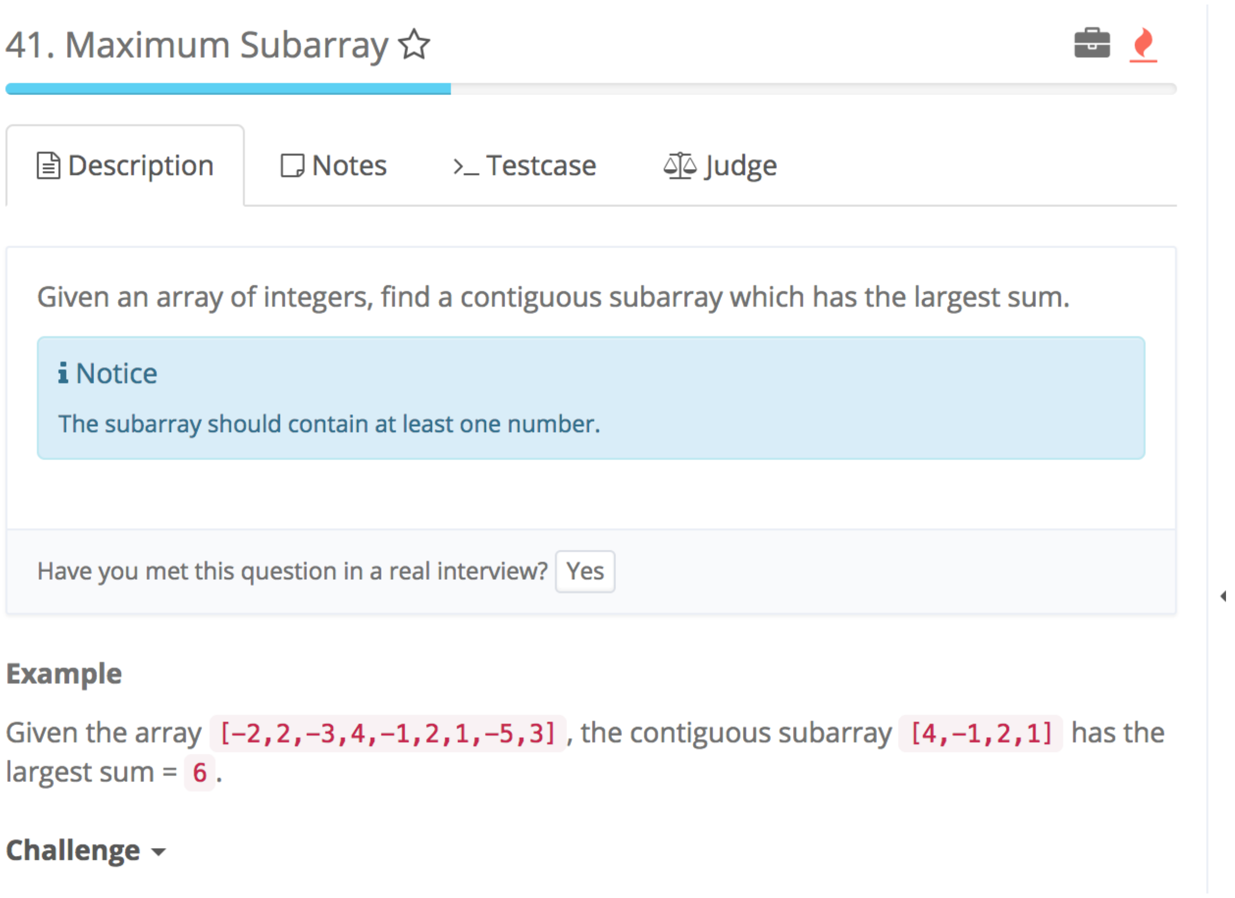
Example

Given `[-3, 1, 2, -3, 4]`, return `[0, 2]` or `[1, 3]`.

Tags

Related Problems

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    public ArrayList<Integer> subarraySum(int[] nums) {
        // write your code here

        int len = nums.length;

        ArrayList<Integer> ans = new ArrayList<Integer>();
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();

        map.put(0, -1);

        int sum = 0;
        for (int i = 0; i < len; i++) {
            sum += nums[i];

            if (map.containsKey(sum)) {
                ans.add(map.get(sum) + 1);
                ans.add(i);
                return ans;
            }

            map.put(sum, i);
        }

        return ans;
    }
}
```



41. Maximum Subarray ☆

Description | Notes | >_ Testcase | Judge

Given an array of integers, find a contiguous subarray which has the largest sum.

Notice

The subarray should contain at least one number.

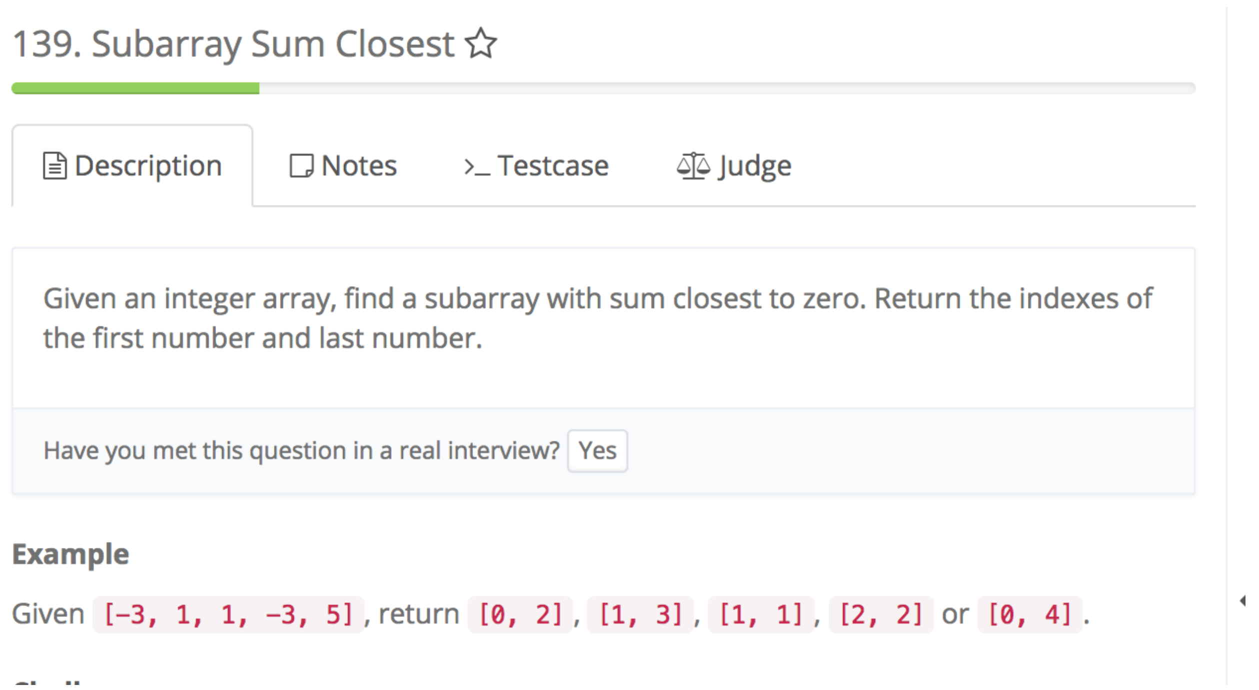
Have you met this question in a real interview? Yes

Example

Given the array `[-2,2,-3,4,-1,2,1,-5,3]`, the contiguous subarray `[4,-1,2,1]` has the largest sum = `6`.

Challenge ▾

```
// Version 1: Greedy

public class Solution {
    public int maxSubArray(int[] A) {
        if (A == null || A.length == 0){
            return 0;
        }

        int max = Integer.MIN_VALUE, sum = 0;
        for (int i = 0; i < A.length; i++) {
            sum += A[i];
            max = Math.max(max, sum);
            sum = Math.max(sum, 0);
        }

        return max;
    }
}

// Version 2: Prefix Sum

public class Solution {
    public int maxSubArray(int[] A) {
        if (A == null || A.length == 0){
            return 0;
        }

        int max = Integer.MIN_VALUE, sum = 0, minSum = 0;
        for (int i = 0; i < A.length; i++) {
            sum += A[i];
            max = Math.max(max, sum - minSum);
            minSum = Math.min(minSum, sum);
        }

        return max;
    }
}



public class Solution {
    /**
     * @param nums: a list of integers
     * @return: A integer indicate the sum of minimum subarray
     */
    public int maxSubArray(int[] nums) {
        // write your code
        if(nums.length == 0){
            return 0;
        }
        int n = nums.length;
        int[] global = new int[2];
        int[] local = new int[2];
        global[0] = nums[0];
        local[0] = nums[0];
        for(int i = 1; i < n; i ++) {
            local[i % 2] = Math.max(nums[i], local[(i - 1) % 2] + nums[i]);
            global[i % 2] = Math.max(local[i % 2], global[(i - 1) % 2]);
        }
        return global[(n-1) % 2];
    }
}
```



139. Subarray Sum Closest ☆

Description | Notes | >_ Testcase | Judge

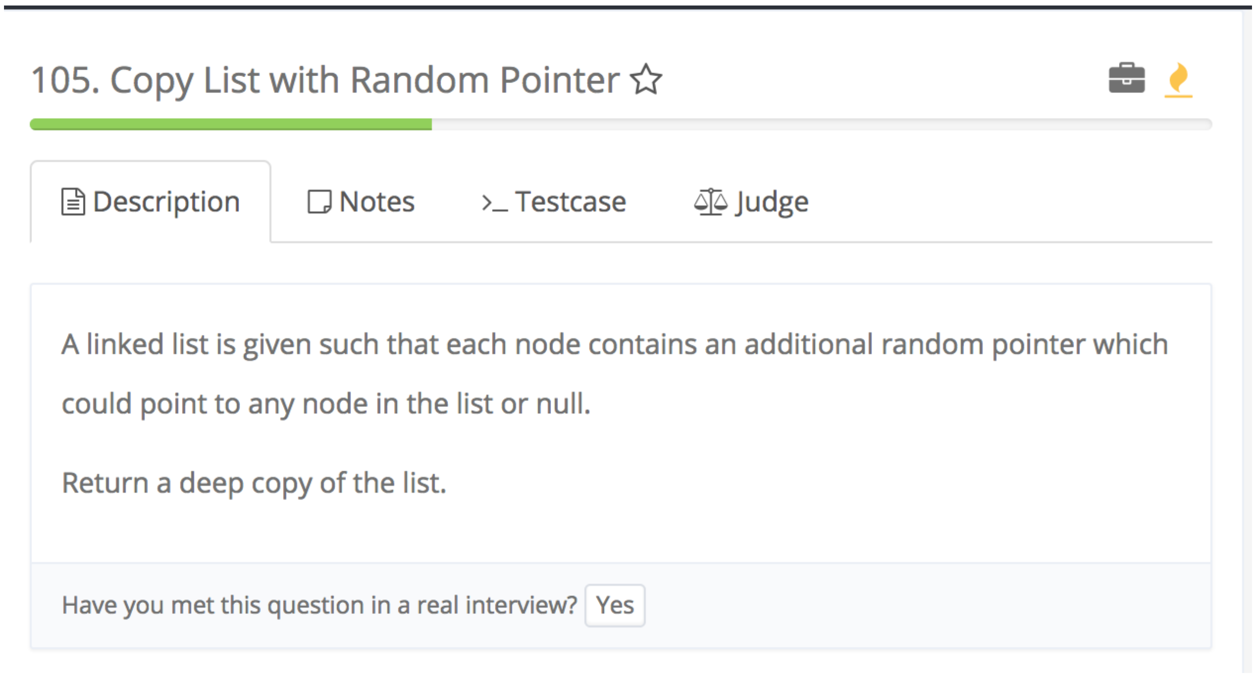
Given an integer array, find a subarray with sum closest to zero. Return the indexes of the first number and last number.

Have you met this question in a real interview? Yes

Example

Given `[-3, 1, 1, -3, 5]`, return `[0, 2]`, `[1, 3]`, `[1, 1]`, `[2, 2]` or `[0, 4]`.

```
class Pair {
    int sum;
    int index;
    public Pair(int s, int i) {
        sum = s;
        index = i;
    }
}

public class Solution {
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    public int[] subarraySumClosest(int[] nums) {
        int[] res = new int[2];
        if (nums == null || nums.length == 0) {
            return res;
        }

        int len = nums.length;
        if(len == 1) {
            res[0] = res[1] = 0;
            return res;
        }
        Pair[] sums = new Pair[len+1];
        int prev = 0;
        sums[0] = new Pair(0, 0);
        for (int i = 1; i <= len; i++) {
            sums[i] = new Pair(prev + nums[i-1], i);
            prev = sums[i].sum;
        }
        Arrays.sort(sums, new Comparator<Pair>() {
           public int compare(Pair a, Pair b) {
               return a.sum - b.sum;
           }
        });
        int ans = Integer.MAX_VALUE;
        for (int i = 1; i <= len; i++) {

            if (ans > sums[i].sum - sums[i-1].sum) {
                ans = sums[i].sum - sums[i-1].sum;
                int[] temp = new int[]{sums[i].index - 1, sums[i - 1].index - 1};
                Arrays.sort(temp);
                res[0] = temp[0] + 1;
                res[1] = temp[1];
            }
        }

        return res;
    }
}


/*
问：为什么需要一个 (0,0) 的初始 Pair?
答：
我们首先需要回顾一下，在 subarray 这节课里，我们讲过一个重要的知识点，叫做 Prefix Sum
比如对于数组 [1,2,3,4]，他的 Prefix Sum 是 [1,3,6,10]
分别表示 前1个数之和，前2个数之和，前3个数之和，前4个数之和
这个时候如果你想要知道 子数组 从下标  1 到下标 2 的这一段的和(2+3)，就用前 3个数之和 减去
前1个数之和 = PrefixSum[2] - PrefixSum[0] = 6 - 1 = 5
你可以看到这里的 前 x 个数，和具体对应的下标之间，存在 +-1 的问题
第 x 个数的下标是 x - 1，反之 下标 x 是第 x + 1 个数
那么问题来了，如果要计算 下标从 0~2 这一段呢？也就是第1个数到第3个数，因为那样会访问
到 PrefixSum[-1]
所以我们把 PrefixSum 整体往后面移动一位，把第0位空出来表示前0个数之和，也就是0. =>
[0,1,3,6,10]
那么此时就用 PrefixSum[3] - PrefixSum[0] ，这样计算就更方便了。
此时，PrefixSum[i] 代表 前i个数之和，也就是 下标区间在 0 ~ i-1 这一段的和

那么回过头来看看，为什么我们需要一个 (0,0) 的 pair 呢？
因为 这个 0,0 代表的就是前0个数之和为0
一个 n 个数的数组， 变成了 prefix Sum 数组之后，会多一个数出来
*/
```



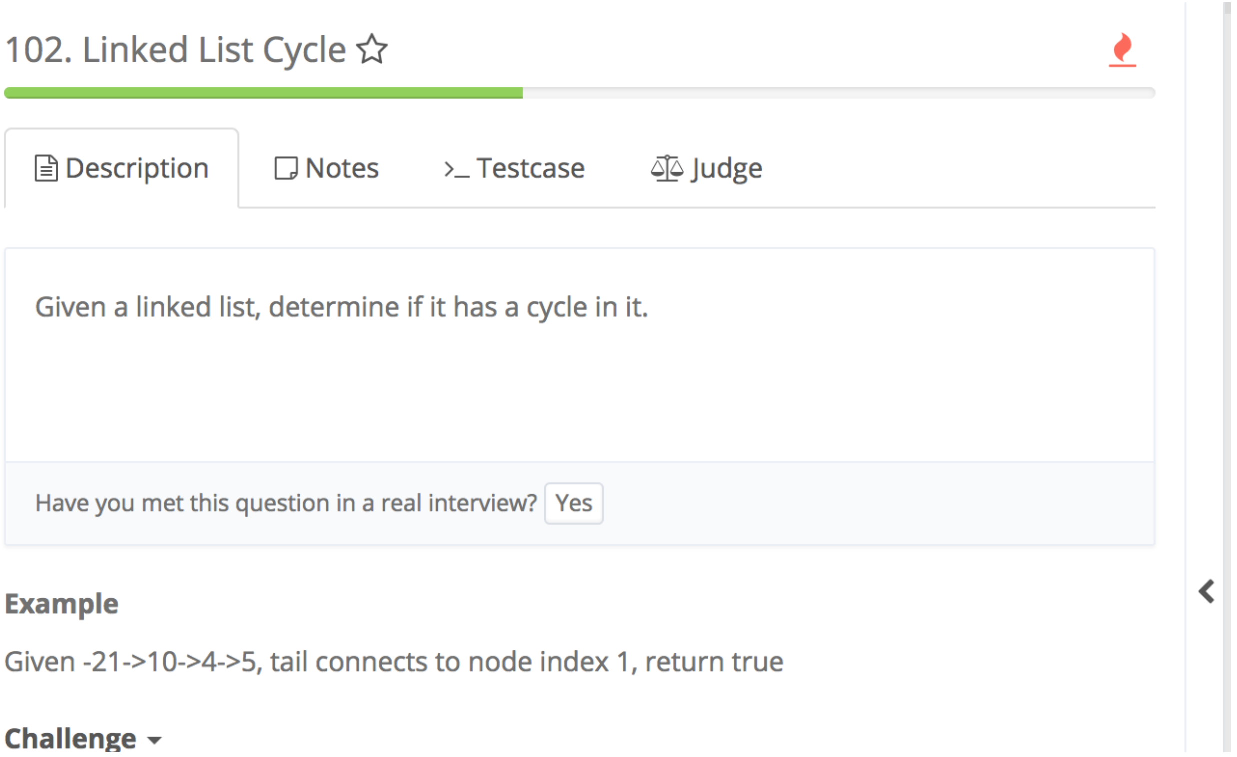
105. Copy List with Random Pointer

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

Have you met this question in a real interview? Yes

```
//HashMap version
public class Solution {
    public RandomListNode copyRandomList(RandomListNode head) {
        if (head == null) {
            return null;
        }

        HashMap<RandomListNode, RandomListNode> map = new HashMap<RandomListNode,
RandomListNode>();
        RandomListNode dummy = new RandomListNode(0);
        RandomListNode pre = dummy, newNode;
        while (head != null) {
            if (map.containsKey(head)) {
                newNode = map.get(head);
            } else {
                newNode = new RandomListNode(head.label);
                map.put(head, newNode);
            }
            pre.next = newNode;

            if (head.random != null) {
                if (map.containsKey(head.random)) {
                    newNode.random = map.get(head.random);
                } else {
                    newNode.random = new RandomListNode(head.random.label);
                    map.put(head.random, newNode.random);
                }
            }

            pre = newNode;
            head = head.next;
        }

        return dummy.next;
    }
}

/*第一遍扫的时候巧妙运用next指针， 开始数组是1->2->3->4 。 然后扫描过程中 先建立copy节
点 1->1`->2->2`->3->3`->4->4`, 然后第二遍copy的时候去建立边的copy， 拆分节点, 一边扫描一
边拆成两个链表，这里用到两个dummy node。第一个链表变回 1->2->3 , 然后第二变成 1`->2`-
>3` */
//No HashMap version
public class Solution {
    private void copyNext(RandomListNode head) {
        while (head != null) {
            RandomListNode newNode = new RandomListNode(head.label);
            newNode.random = head.random;
            newNode.next = head.next;
            head.next = newNode;
            head = head.next.next;
        }
    }

    private void copyRandom(RandomListNode head) {
        while (head != null) {
            if (head.next.random != null) {
                head.next.random = head.random.next;
            }
            head = head.next.next;
        }
    }

    private RandomListNode splitList(RandomListNode head) {
        RandomListNode newHead = head.next;
        while (head != null) {
            RandomListNode temp = head.next;
            head.next = temp.next;
            head = head.next;
            if (temp.next != null) {
                temp.next = temp.next.next;
            }
        }
        return newHead;
    }

    public RandomListNode copyRandomList(RandomListNode head) {
        if (head == null) {
            return null;
        }
        copyNext(head);
        copyRandom(head);
        return splitList(head);
    }
}
```


102. Linked List Cycle ☆

Description | Notes | >_ Testcase | Judge

Given a linked list, determine if it has a cycle in it.

Have you met this question in a real interview? Yes

Example

Given -21->10->4->5, tail connects to node index 1, return true

Challenge

```
/**
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int val) {
 *         this.val = val;
 *         this.next = null;
 *     }
 * }
 */

public class Solution {
    public Boolean hasCycle(ListNode head) {
        if (head == null || head.next == null) {
            return false;
        }

        ListNode fast, slow;
        fast = head.next;
        slow = head;
        while (fast != slow) {
            if(fast==null || fast.next==null)
                return false;
            fast = fast.next.next;
            slow = slow.next;
        }
        return true;
    }
}
```

98. Sort List ☆

Description Notes >_ Testcase Judge

Sort a linked list in O(*n* log *n*) time using constant space complexity.

Have you met this question in a real interview? Yes

Example

Given `1->3->2->null`, sort it to `1->2->3->null`.

```java
/**
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int val) {
 *         this.val = val;
 *         this.next = null;
 *     }
 * }
 */

// version 1: Merge Sort
public class Solution {
    private ListNode findMiddle(ListNode head) {
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            fast = fast.next.next;
            slow = slow.next;
        }
        return slow;
    }

    private ListNode merge(ListNode head1, ListNode head2) {
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;
        while (head1 != null && head2 != null) {
            if (head1.val < head2.val) {
                tail.next = head1;
                head1 = head1.next;
            } else {
                tail.next = head2;
                head2 = head2.next;
            }
            tail = tail.next;
        }
        if (head1 != null) {
            tail.next = head1;
        } else {
            tail.next = head2;
        }

        return dummy.next;
    }

    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode mid = findMiddle(head);

        ListNode right = sortList(mid.next);
        mid.next = null;
        ListNode left = sortList(head);

        return merge(left, right);
    }
}

// version 2: Quick Sort 1
public class Solution {
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode mid = findMedian(head); // O(n)

        ListNode leftDummy = new ListNode(0), leftTail = leftDummy;
        ListNode rightDummy = new ListNode(0), rightTail = rightDummy;
        ListNode middleDummy = new ListNode(0), middleTail = middleDummy;
        while (head != null) {
            if (head.val < mid.val) {
                leftTail.next = head;
                leftTail = head;
            } else if (head.val > mid.val) {
                rightTail.next = head;
                rightTail = head;
            } else {
                middleTail.next = head;
                middleTail = head;
            }
            head = head.next;
        }

        leftTail.next = null;
        middleTail.next = null;
        rightTail.next = null;

        ListNode left = sortList(leftDummy.next);
        ListNode right = sortList(rightDummy.next);

        return concat(left, middleDummy.next, right);
    }
    private ListNode findMedian(ListNode head) {
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

    private ListNode concat(ListNode left, ListNode middle, ListNode right) {
        ListNode dummy = new ListNode(0), tail = dummy;

        tail.next = left; tail = getTail(tail);
        tail.next = middle; tail = getTail(tail);
        tail.next = right; tail = getTail(tail);
        return dummy.next;
    }

    private ListNode getTail(ListNode head) {
        if (head == null) {
           return null;
        }

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

450. Reverse Nodes in k-Group ☆

Description Notes >_ Testcase Judge

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.
Only constant memory is allowed.

Have you met this question in a real interview? Yes

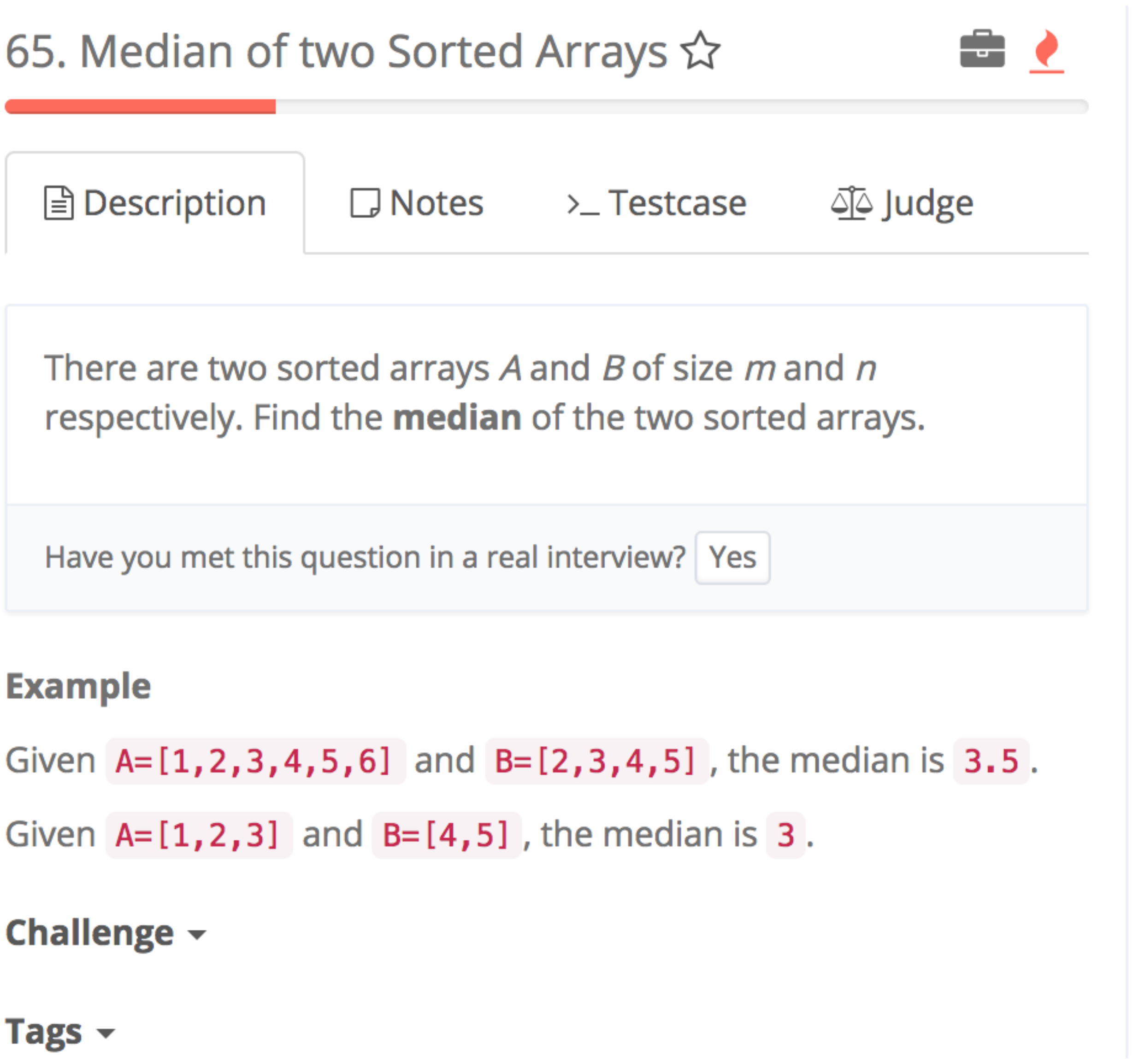
Example

Given this linked list: `1->2->3->4->5`

For k = 2, you should return: `2->1->4->3->5`

For k = 3, you should return: `3->2->1->4->5`

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */


public class Solution {
    /**
     * @param head a ListNode
     * @param k an integer
     * @return a ListNode
     */
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;

        head = dummy;
        while (true) {
            head = reverseK(head, k);
            if (head == null) {
                break;
            }
        }

        return dummy.next;
    }

    // head -> n1 -> n2 ... nk -> nk+1
    // =>
    // head -> nk -> nk-1 .. n1 -> nk+1
    // return n1
    public ListNode reverseK(ListNode head, int k) {
        ListNode nk = head;
        for (int i = 0; i < k; i++) {
            if (nk == null) {
                return null;
            }
            nk = nk.next;
        }

        if (nk == null) {
            return null;
        }
        // reverse
        ListNode n1 = head.next;
        ListNode nkplus = nk.next;

        ListNode prev = null;
        ListNode curt = n1;
        while (curt != nkplus) {
            ListNode temp = curt.next;
            curt.next = prev;
            prev = curt;
            curt = temp;
        }

        // connect
        head.next = nk;
        n1.next = nkplus;
        return n1;
    }
}
```



65. Median of two Sorted Arrays

There are two sorted arrays *A* and *B* of size *m* and *n* respectively. Find the **median** of the two sorted arrays.

Have you met this question in a real interview? Yes

Example

Given `A=[1,2,3,4,5,6]` and `B=[2,3,4,5]`, the median is `3.5`.

Given `A=[1,2,3]` and `B=[4,5]`, the median is `3`.

Challenge

Tags

```
public class Solution {
    public double findMedianSortedArrays(int A[], int B[]) {
        int len = A.length + B.length;
        if (len % 2 == 1) {
            return findKth(A, 0, B, 0, len / 2 + 1);
        }
        return (
            findKth(A, 0, B, 0, len / 2) + findKth(A, 0, B, 0, len / 2 + 1)
        ) / 2.0;
    }

    // find kth number of two sorted array
    public static int findKth(int[] A, int A_start,
                              int[] B, int B_start,
                              int k){
                    if (A_start >= A.length) {
                            return B[B_start + k - 1];
                    }
                    if (B_start >= B.length) {
                            return A[A_start + k - 1];
                    }

                    if (k == 1) {
                            return Math.min(A[A_start], B[B_start]);
                    }

                    int A_key = A_start + k / 2 - 1 < A.length
                            ? A[A_start + k / 2 - 1]
                            : Integer.MAX_VALUE;
                    int B_key = B_start + k / 2 - 1 < B.length
                            ? B[B_start + k / 2 - 1]
                            : Integer.MAX_VALUE;

                    if (A_key < B_key) {
                            return findKth(A, A_start + k / 2, B, B_start, k - k / 2);
                    } else {
                            return findKth(A, A_start, B, B_start + k / 2, k - k / 2);
                    }
            }
}
```