LintCode Online Coding Interview Questions - Problem-wise

deepwaterooo

April 26, 2015

Contents

Chapter 1

4 Sum, wrong answers

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note

Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a $\leqslant$ b $\leqslant$ c $\leqslant$ d)

The solution set must not contain duplicate quadruplets.

Example

For example, given array S = {1 0 -1 0 -2 2}, and target = 0. A solution set is:

```
(-1, 0, 0, 1)
(-2, -1, 1, 2)
(-2, 0, 0, 2)
```

```
/**
 * @param numbers : Give an array numbersbers of n integer
 * @param target : you need to find four elements that's sum of target
 * @return : Find all unique quadruplets in the array which gives the sum of
 *           zero.
 */
public ArrayList<ArrayList<Integer>> fourSum(int[] numbers, int target) {
    int n = numbers.length;
    HashSet<ArrayList<Integer>> set = new HashSet<ArrayList<Integer>>();
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (n < 4) return res;
    Arrays.sort(numbers);
    int x = 0, y = 0, tmp = 0;
    for (int i = 0; i < n - 3; i++) {
        if (i > 0 && numbers[i] == numbers[i - 1]) continue;
        for (int j = i + 1; j < n - 2; j++) {
            x = j + 1;
            y = n - 1;
            while (x < y) {
                tmp = target - numbers[i] - numbers[j];
                if (numbers[x] + numbers[y] == tmp) {
                    set.add(new ArrayList<Integer>(Arrays.asList(numbers[i], numbers[j]
                    x++;
                    y--;
                }
                else if (numbers[x] + numbers[y] < tmp) x++;
```

```
                else y--;
            }
        }
    }
    res.addAll(set);
    return res;
}
```

Chapter 2

2 Sum: O(1) Space, O(nlogn) Time undone

25% Accepted

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

Note

You may assume that each input would have exactly one solution

Example

numbers=[2, 7, 11, 15], target=9

return [1, 2]

Challenge

2.1 O(1) Space, O(nlogn) Time

2.2 O(n) Space, O(n) Time

```
public int[] twoSum(int[] numbers, int target) { // O(nlogn), O(1)
    Arrays.sort(numbers);
    int [] res = new int[2];
    int i = 0, j = numbers.length - 1;
    while (i < j) {
        if (numbers[i] + numbers[j] == target) {
            res[0] = i + 1;
            res[1] = j + 1;
            return res;
        }
        if (numbers[i] + numbers[j] < target) i++;
        else j--;
    }
    return res;
}
```

Chapter 3

3 Sum

19% Accepted

Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Note

Elements in a triplet (a,b,c) must be in non-descending order. (ie, a $\leqslant$ b $\leqslant$ c)

The solution set must not contain duplicate triplets.

Example

For example, given array S = {-1 0 1 2 -1 -4}, A solution set is:

```
(-1, 0, 1)
(-1, -1, 2)
```

```
public ArrayList<ArrayList<Integer>> threeSum(int[] numbers) {
    int n = numbers.length;
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    HashSet<List<Integer>> set = new HashSet<List<Integer>>();
    Integer [] one = new Integer[3];
    int k = 0;
    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            k = j + 1;
            while (k < n - 1 && numbers[i] + numbers[j] + numbers[k] != 0) k++;
            if (k <= n - 1 && numbers[i] + numbers[j] + numbers[k] == 0) {
                one[0] = numbers[i];
                one[1] = numbers[j];
                one[2] = numbers[k];
                Arrays.sort(one);
                set.add(new ArrayList<Integer>(Arrays.asList(one)));
                k++;
            }
        }
    }
    for(List<Integer> i : set)
        res.add(new ArrayList(i));
    return res;
}
```

Chapter 4

3 Sum Closest

30% Accepted

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers.

Note

You may assume that each input would have exactly one solution.

Example

For example, given array S = {-1 2 1 -4}, and target = 1. The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

```
public int threeSumClosest(int[] numbers ,int target) {
    int n = numbers.length;
    int res = Integer.MAX_VALUE;
    int k = 0;
    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            k = j + 1;
            while (k <= n - 1) {
                if (Math.abs(numbers[i] + numbers[j] + numbers[k] - target) < Math.abs(
                    res = numbers[i] + numbers[j] + numbers[k];
                k++;
            }
        }
    }
    return res;
}
```

Chapter 5

A + B Problem, no idea;;;

60% Accepted
For given numbers a and b in function aplusb, return the sum of them.
Note
You don't need to parse the input and output. Just calculate and return.
Example
If a = 1 and b = 2 return 3
Challenge
Can you do it without + operation?
Clarification
Are a and b both 32-bit integers?

- Yes.

Chapter 6

Anagrams My Submissions

28% Accepted
Given an array of strings, return all groups of strings that are anagrams.
Note
All inputs will be in lower-case
Example

```
Given a string list: ["lint","intl","inlt","code"]
return ["lint","inlt","intl"]
```

```
public String mySort(String s) {
    char [] tmp = s.toCharArray();
    Arrays.sort(tmp);
    return new String(tmp);   //tmp.toString(); doesn't work
}
```

```
public List<String> anagrams(String[] strs) {
    Map<String, List<Integer>> m = new HashMap<String, List<Integer>>();
    ArrayList<String> res = new ArrayList<String>();
    for (int i = 0; i < strs.length; i++) {
        String tmp = mySort(strs[i]);
        if (!m.containsKey(tmp))
            m.put(tmp, new ArrayList<Integer>(Arrays.asList(i)));
        else m.get(tmp).add(i);
    }
    for (String key : m.keySet())
        if (m.get(key).size() > 1)
            for (int i = 0; i < m.get(key).size(); i++)
                res.add(strs[m.get(key).get(i)]);
    return res;
}
```

Chapter 7

Backpack: still feeling difficult for me now ...

17% Accepted

Given n items with size A[i], an integer m denotes the size of a backpack. How full you can fill this backpack?

Note

You can not divide any item into small pieces.

Example

If we have 4 items with size [2, 3, 5, 7], the backpack size is 11, we can select 2, 3 and 5, so that the max size we can fill this backpack is 10. If the backpack size is 12. we can select [2, 3, 7] so that we can fulfill the backpack.

You function should return the max size we can fill in the given backpack.

Chapter 8

Balanced Binary Tree

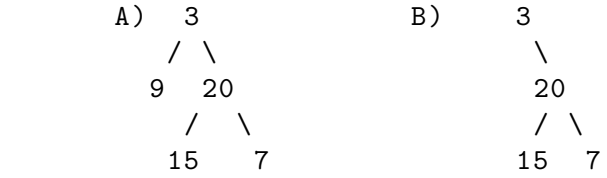
46% Accepted

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Example

Given binary tree A={3,9,20,#,#,15,7}, B={3,#,20,15,7}



The binary tree A is a height-balanced binary tree, but B is not.

Chapter 9

Best Time to Buy and Sell Stock Show Result

45% Accepted

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Example

Given an example [3,2,3,1,2], return 1

```
public int maxProfit(int[] prices) {
    if (prices.length == 0) return 0;
    int n = prices.length;
    int [] buy = new int[n];
    buy[0] = prices[0];
    for (int i = 1; i < n; i++)
        buy[i] = Math.min(buy[i - 1], prices[i]);
    int [] sell = new int[n];
    sell[n - 1] = prices[n - 1];
    int res = Integer.MIN_VALUE;
    for (int i = n - 2; i >= 0; i--) {
        sell[i] = Math.max(sell[i + 1], prices[i]);
        res = Math.max(res, sell[i] - buy[i]);
    }
    return res;
}
```

Chapter 10

Best Time to Buy and Sell Stock II

62% Accepted

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example

Given an example [2,1,2,0,1], return 2

Chapter 11

Best Time to Buy and Sell Stock III

25% Accepted

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete at most two transactions.

Note

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example

Given an example [4,4,6,1,1,4,2,5], return 6

Chapter 12

Binary Representation

6% Accepted

Given a (decimal - e g 3.72) number that is passed in as a string,return the binary representation that is passed in as a string.If the number can not be represented accurately in binary, print “ERROR”

Example

n = 3.72, return ERROR

n = 3.5, return 11.1

Chapter 13

Binary Search My Submissions: arr.length > Integer.MAX_VALUE ? don't understand···

27% Accepted

Binary search is a famous question in algorithm.

For a given sorted array (ascending order) and a target number, find the first index of this number in O(log n) time complexity.

If the target number does not exist in the array, return -1.

Example

If the array is [1, 2, 3, 3, 4, 5, 10], for given target 3, return 2.

Challenge

If the count of numbers is bigger than MAXINT, can your code work properly?

```
/**
 * @param nums: The integer array.
 * @param target: Target to find.
 * @return: The first position of target. Position starts from 0.
 */
public int binarySearch(int[] nums, int target) {
    if (target < nums[0] || target > nums[nums.length - 1]) return -1;
    int bgn = 0, end = nums.length - 1;
    if (bgn == end - 1) {
        if (target == nums[bgn]) return bgn;
        else if (target == nums[end]) return end;
        else return -1;
    }
    while (bgn < end - 1) {
        int mid1 = bgn + (end - bgn) / 2;
        int mid2 = mid1 + 1;
        if (target < nums[mid1]) end = mid1 - 1;
        else if (target > nums[mid2]) bgn = mid2 + 1;
        else if (target == nums[mid1]) end = mid1;
        else if (target == nums[mid2] && nums[mid1] < nums[mid2]) bgn = mid2;
    }
    if (bgn == end - 1) {
        if (target == nums[bgn]) return bgn;
```

```
        else if (target == nums[end]) return end;
        else return -1;
    } else if (bgn == end)
        return nums[bgn] == target ? bgn : -1;
    else return -1;
}
```

Chapter 14

Binary Tree Inorder Traversal: Iterative undone···

37% Accepted
Given a binary tree, return the inorder traversal of its nodes' values.
Example
Given binary tree {1,#,2,3},

```
 1
  \
   2
  /
 3
```

return [1,3,2].
Challenge
Can you do it without recursion?

```
public void inorderTraversal(TreeNode root, ArrayList<Integer> res) {
    if (root == null) return;
    inorderTraversal(root.left, res);
    res.add(root.val);
    inorderTraversal(root.right, res);
}

public ArrayList<Integer> inorderTraversal(TreeNode root) {
    ArrayList<Integer> res = new ArrayList<Integer>();
    inorderTraversal(root, res);
    return res;
}
```

Chapter 15

Binary Tree Level Order Traversal Show Result My Submissions

33% Accepted

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

Example

Given binary tree {3,9,20,#,#,15,7},

```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

Challenge

Using only 1 queue to implement it.

```
public ArrayList<ArrayList<Integer>> levelOrder(TreeNode root) {
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (root == null) return  res;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(null);
    q.add(root);
    TreeNode curr = root;
    TreeNode prev = null;
    int n = 0;
    while (!q.isEmpty()) {
        prev = curr;
        curr = q.poll();
        if (curr == null) {
            if (prev != curr) {
                res.add(new ArrayList<Integer>());
```

```
                q.add(curr);
                continue;
            } else {
                res.remove(res.size() - 1);
                return res;
            }
        }
        if (curr != null) {
            n = res.size() - 1;
            res.get(n).add(curr.val);
            if (curr.left != null) q.add(curr.left);
            if (curr.right != null) q.add(curr.right);
        }
    }
    return res;
}
```

Chapter 16

Binary Tree Level Order Traversal II

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

Example

Given binary tree {3,9,20,#,#,15,7},

```
    3
   / \
  9  20
    /  \
   15   7
```

return its bottom-up level order traversal as:

```
[
  [15,7],
  [9,20],
  [3]
]
```

```
public ArrayList<ArrayList<Integer>> levelOrderButtom(TreeNode root) {
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (root == null) return  res;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(null);
    q.add(root);
    TreeNode curr = root;
    TreeNode prev = null;
    int n = 0;
    while (!q.isEmpty()) {
        prev = curr;
        curr = q.poll();
        if (curr == null) {
            if (prev != curr) {
                res.add(new ArrayList<Integer>());
                q.add(curr);
                continue;
            } else {
                res.remove(res.size() - 1);
                ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>
```

```
                for (int i = res.size() - 1; i >= 0; i--)
                    result.add(new ArrayList<Integer>(res.get(i)));
                return result;
            }
        }
        if (curr != null) {
            n = res.size() - 1;
            res.get(n).add(curr.val);
            if (curr.left != null) q.add(curr.left);
            if (curr.right != null) q.add(curr.right);
        }
    }
    return res;
}
```

Chapter 17

Binary Tree Maximum Path Sum: some part missing….

23% Accepted
Given a binary tree, find the maximum path sum.
The path may start and end at any node in the tree.
Example
Given the below binary tree,

```
    1
   / \
  2   3
```

Return 6.

Chapter 18

Binary Tree Zigzag Level Order Traversal

26% Accepted

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

Example

Given binary tree {3,9,20,#,#,15,7},

```
    3
   / \
  9  20
    /  \
   15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

```
public ArrayList<ArrayList<Integer>> zigzagLevelOrder(TreeNode root) {
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (root == null) return  res;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(null);
    q.add(root);
    TreeNode curr = root;
    TreeNode prev = null;
    int n = 0;
    int cnt = 0;
    while (!q.isEmpty()) {
        prev = curr;
        curr = q.poll();
        if (curr == null) {
            if (prev != curr) {
                res.add(new ArrayList<Integer>());
                q.add(curr);
                ++cnt;
                continue;
```

```
            } else {
                res.remove(res.size() - 1);
                return res;
            }
        }
        if (curr != null) {
            n = res.size() - 1;
            if (cnt % 2 == 1)
                res.get(n).add(curr.val);
            else
                res.get(n).add(0, curr.val);
            if (curr.left != null) q.add(curr.left);
            if (curr.right != null) q.add(curr.right);
        }
    }
    return res;
}
```

Chapter 19

Climbing Stairs

40% Accepted
You are climbing a stair case. It takes n steps to reach to the top.
Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?
Example
Given an example n=3 , 1+1+1=2+1=1+2=3
return 3

```
public int climbStairs(int n) {
    int [] res = new int[n];
    res[0] = 1;
    if (n < 2) return 1;
    res[1] = 2;
    for (int i = 2; i < n; i++)
        res[i] = res[i - 1] + res[i - 2];
    return res[n - 1];
}
```

Chapter 20

Combination Sum

26% Accepted

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

For example, given candidate set 2,3,6,7 and target 7, A solution set is:

```
[7]
[2, 2, 3]
```

Note

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, ⋯, ak) must be in non-descending order. (ie, a1 ⩽ a2 ⩽⋯⩽ ak).

The solution set must not contain duplicate combinations.

Example

given candidate set 2,3,6,7 and target 7,

A solution set is:

```
[7]
[2, 2, 3]
```

```
public void combinationSum(int [] candidates, int gap, int idx,
                           List<List<Integer>> res, List<Integer> path) {
    if (gap == 0)
        res.add(new ArrayList<Integer>(path));
    for (int i = idx; i < candidates.length; i++) {
        if (candidates[i] <= gap) {
            path.add(candidates[i]);
            combinationSum(candidates, gap - candidates[i], i, res, path);
            path.remove(path.size() - 1);
        }
    }
}

public List<List<Integer>> combinationSum(int[] candidates, int target) {
    int n = candidates.length;
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    List<Integer> path = new ArrayList<Integer>();
```

```
    Arrays.sort(candidates);
    combinationSum(candidates, target, 0, res, path);
    return res;
}
```

Chapter 21

Combination Sum II

24% Accepted

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, ⋯, ak) must be in non-descending order. (ie, a1 ⩽ a2 ⩽⋯⩽ ak).

The solution set must not contain duplicate combinations.

Example

For example, given candidate set 10,1,6,7,2,1,5 and target 8,

A solution set is:

```
[1,7]
[1,2,5]
[2,6]
[1,1,6]
```

```
public void combinationSum2(int [] candidates, int gap, int idx,
                            List<List<Integer>> res, List<Integer> path,
                            boolean [] used) {
    if (gap == 0)
        res.add(new ArrayList<Integer>(path));
    for (int i = idx; i < candidates.length; i++) {
        if (i > 0 && candidates[i] == candidates[i - 1] && !used[i - 1]) continue;
        if (candidates[i] <= gap) {
            used[i] = true;
            path.add(candidates[i]);
            combinationSum2(candidates, gap - candidates[i], i + 1, res, path, used);
            path.remove(path.size() - 1);
            used[i] = false;
        }
    }
}

public List<List<Integer>> combinationSum2(int[] candidates, int target) {
    int n = candidates.length;
```

```
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    List<Integer> path = new ArrayList<Integer>();
    Arrays.sort(candidates);
    boolean [] used = new boolean[n];
    combinationSum2(candidates, target, 0, res, path, used);
    return res;
}
```

Chapter 22

Combinations

31% Accepted
Given two integers n and k, return all possible combinations of k numbers out of 1 ⋯ n.
Example
For example,
If n = 4 and k = 2, a solution is:

```
[[2,4],[3,4],[2,3],[1,2],[1,3],[1,4]]
```

```
public void combine(int n, int k, int idx, List<Integer> src, List<Integer> path,
                    List<List<Integer>> res) {
    if (path.size() == k) {
        List<Integer> one = new ArrayList<Integer>(path);
        Collections.sort(one);
        res.add(new ArrayList(one));
        return;
    }
    for (int i = idx; i < n; i++) {
        path.add(src.get(i));
        combine(n, k, i + 1, src, path, res);
        path.remove(path.size() - 1);
    }
}

public List<List<Integer>> combine(int n, int k) {
    List<Integer> src = new ArrayList<Integer>();
    for (int i = 0; i < n; i++)
        src.add(i + 1);
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    List<Integer> path = new ArrayList<Integer>();
    combine(n, k, 0, src, path, res);
    return res;
}
```

Chapter 23

Compare Strings

32% Accepted
Compare two strings A and B, determine whether A contains all of the characters in B.
The characters in string A and B are all Upper Case letters.
Example
For A = ”ABCD”, B = ”ABC”, return true.
For A = ”ABCD” B = ”AABC”, return false.

```
public boolean compareStrings(String a, String b) {
    if (b == null) return a == null;
    if (a.length() < b.length()) return false;
    Map<Character, Integer> bm = new HashMap<Character, Integer>();
    Map<Character, Integer> am = new HashMap<Character, Integer>();
    for (int i = 0; i < b.length(); i++) {
        if (!bm.containsKey(b.charAt(i)))
            bm.put(b.charAt(i), 1);
        else bm.put(b.charAt(i), bm.get(b.charAt(i)) + 1);
    }
    for (int i = 0; i < a.length(); i++) {
        if (!am.containsKey(a.charAt(i)))
            am.put(a.charAt(i), 1);
        else am.put(a.charAt(i), am.get(a.charAt(i)) + 1);
    }
    for (Character key : bm.keySet())
        if (!am.containsKey(key) || am.get(key) < bm.get(key)) return false;
    return true;
}
```

Chapter 24

Convert Sorted List to Binary Search Tree: bottom-up undone~

25% Accepted

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

```
public int getSize(ListNode head) {
    int cnt = 0;
    while (head != null) {
        ++cnt;
        head = head.next;
    }
    return cnt;
}
public ListNode getKthNode(ListNode head, int n) {
    if (n == 0) return head;
    if (n < 0 || head == null) return null;
    int cnt = 0;
    while (head != null && cnt < n) {
        ++cnt;
        head = head.next;
    }
    return head;
}

public TreeNode sortedListToBST(ListNode head) {
    if (head == null) return null;
    if (head.next == null) return new TreeNode(head.val);
    int n = getSize(head);
    TreeNode root = new TreeNode(getKthNode(head, (n - 1) / 2).val);
    root.right = sortedListToBST(getKthNode(head, (n - 1) / 2).next);
    if (n  > 2) {
        ListNode leftT = getKthNode(head, (n - 1) / 2 - 1);
        if (leftT != null)
            leftT.next = null;
        root.left = sortedListToBST(head);
    }
    return root;
```

```
}
```

Chapter 25

Delete Digits: Tidious, work on it later···

13% Accepted

Given string A representative a positive integer which has N digits, remove any k digits of the number, the remaining digits are arranged according to the original order to become a new positive integer. Make this new positive integers as small as possible.

N <= 240 and k <=N,

Example

Given an integer A = 178542, k = 4

return a string "12"

Chapter 26

Find Minimum in Rotated Sorted Array

34% Accepted
Suppose a sorted array is rotated at some pivot unknown to you beforehand.
(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
Find the minimum element.
You may assume no duplicate exists in the array.
Example
Given [4,5,6,7,0,1,2] return 0

```
public int findMin(int[] num) {
    int n = num.length;
    if (n == 1) return num[0];
    if (n == 2) return Math.min(num[0], num[1]);
    int bgn = 0, end = n - 1;
    while (bgn < end) {
        int mid = bgn + (end - bgn) / 2;
        if (num[mid] < num[bgn] && num[bgn] > num[end])
            end = mid;
        else if (num[mid] > num[end])
            bgn = mid + 1;
        else if (num[mid] < num[end]) {
            if (num[end] < num[bgn])
                bgn = mid + 1;
            else end = mid;
        }
    }
    return num[bgn];
}
```

Chapter 27

Find Minimum in Rotated Sorted Array II: still feel so wired with this one···

35% Accepted
Suppose a sorted array is rotated at some pivot unknown to you beforehand.
(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
Find the minimum element.
The array may contain duplicates.
Example
Given [4,4,5,6,7,0,1,2] return 0

```
public int findMin(int[] num) {
    int n = num.length;
    if (n == 1) return num[0];
    if (n == 2) return Math.min(num[0], num[1]);
    int bgn = 0, end = n - 1;
    while (bgn < end) {
        int mid = bgn + (end - bgn) / 2;
        if (num[mid] < num[bgn] && num[bgn] >= num[end])
            end = mid;
        else if (num[mid] > num[end])
            bgn = mid + 1;
        else if (num[mid] < num[end]) {
            if (num[end] < num[bgn])
                bgn = mid + 1;
            else end = mid;
        } else if (num[mid] == num[end]) {
            if (num[bgn] != num[end]) {
                end = mid;
            } else {
                int i = mid + 1;
                while (i < end && num[i] == num[i - 1]) i++;
                if (i == end) end = mid - 1;
                else bgn = mid + 1;
            }
        }
    }
    return num[bgn];
```

```
}
```

Chapter 28

Find Peak Element

42% Accepted
There is an integer array which has the following features:

- The numbers in adjacent positions are different.
- A[0] < A[1] && A[A.length - 2] > A[A.length - 1].

We define a position P is a peek if A[P] > A[P-1] && A[P] > A[P+1].
Find a peak element in this array. Return the index of the peak.
Note
The array may contains multiple peeks, find any of them.
Example
[1, 2, 1, 3, 4, 5, 7, 6]
return index 1 (which is number 2) or 6 (which is number 7)
Challenge
Time complexity O(logN)

```
public int findPeak(int[] a) {
    int n = a.length;
    if (n < 3) return -1;
    if (n == 3) return (a[0] < a[1] && a[1] > a[2]) ? 1 : -1;
    int bgn = 0, end = n - 1;
    while (bgn < end) {
        int mid = bgn + (end - bgn) / 2;
        if (a[mid] > a[mid - 1] && a[mid] > a[mid + 1]) return mid;
        if (a[mid] > a[mid - 1]) bgn = mid;
        else end = mid;
    }
    return bgn;
}
```

Chapter 29

First Bad Version

31% Accepted

The code base version is an integer and start from 1 to n. One day, someone commit a bad version in the code case, so it caused itself and the following versions are all failed in the unit tests. You can determine whether a version is bad by the following interface:

```
Java:
    public VersionControl {
        boolean isBadVersion(int version);
    }
C++:
    class VersionControl {
    public:
        bool isBadVersion(int version);
    };
Python:
    class VersionControl:
        def isBadVersion(version)
```

Find the first bad version.

Note

You should call isBadVersion as few as possible.

Please read the annotation in code area to get the correct way to call isBadVersion in different language. For example, Java is VersionControl.isBadVersion.

Example

Given n=5

Call isBadVersion(3), get false

Call isBadVersion(5), get true

Call isBadVersion(4), get true

return 4 is the first bad version

Challenge

Do not call isBadVersion exceed O(logn) times.

```
public int findFirstBadVersion(int n) {
    if (VersionControl.isBadVersion(1)) return 1;
    if (!VersionControl.isBadVersion(n)) return -1;
    if (VersionControl.isBadVersion(n) && !VersionControl.isBadVersion(n - 1)) return n
    int bgn = 2, end = n - 1;
    while (bgn < end) {
```

```
        int mid = bgn + (end - bgn) / 2;
        if (VersionControl.isBadVersion(mid)) end = mid;
        else bgn = mid + 1;
    }
    return (VersionControl.isBadVersion(bgn)) ? bgn : -1;
}
```

Chapter 30

Heapify: O(n) time complexity ? think about it……

29% Accepted

Given an integer array, heapify it into a min-heap array.

For a heap array A, A[1] is the root of heap, and for each A[i], A[i * 2 + 1] is the left child of A[i] and A[i * 2 + 2] is the right child of A[i].

Example

Given [3,2,1,4,5], return [1,2,3,4,5] or any legal heap array.

Challenge

O(n) time complexity

Clarification

What is heap?

Heap is a data structure, which usually have three methods: push, pop and top. where ”push” add a new element the heap, ”pop” delete the minimum/maximum element in the heap, ”top” return the minimum/maximum element.

What is heapify?

Convert an unordered integer array into a heap array. If it is min-heap, for each element A[i], we will get A[i * 2 + 1] >= A[i] and A[i * 2 + 2] >= A[i].

What if there is a lot of solutions?

Return any of them.

```
public void heapify(int[] A) {
    Arrays.sort(A);
}
```

[1]DEFINITION NOT FOUND.

Chapter 31

Implement Queue by Stacks

41% Accepted

As the title described, you should only use two stacks to implement a queue's actions.

The queue should support push(element), pop() and top() where pop is pop the first(a.k.a front) element in the queue.

Both pop and top methods should return the value of first element.

Example

For push(1), pop(), push(2), push(3), top(), pop(), you should return 1, 2 and 2

Challenge

implement it by two stacks, do not use any other data structure and push, pop and top should be O(1) by AVERAGE.

```
public static class Solution {
    private Stack<Integer> stack1;
    private Stack<Integer> stack2;
    public Solution() {
        stack1 = new Stack<Integer>();
        stack2 = new Stack<Integer>();
    }
    public void push(int element) {
        while (!stack2.isEmpty()) {
            int tmp = stack2.pop();
            stack1.push(tmp);
        }
        stack1.push(element);
        while (!stack1.isEmpty()) {
            int tmp = stack1.pop();
            stack2.push(tmp);
        }
    }
    public int pop() {
        int tmp = stack2.pop();
        return  tmp;
    }
    public int top() {
        int tmp = stack2.peek();
        return tmp;
    }
```

```
}
```

Chapter 32

Insert Interval: got blocked here⋯.

22% Accepted

Given a non-overlapping interval list which is sorted by start point.

Insert a new interval into it, make sure the list is still in order and non-overlapping (merge intervals if necessary).

Example

Insert [2, 5] into [[1,2], [5,9]], we get [1, 9].

Insert [3, 4] into [[1,2], [5,9]], we get [[1,2], [3,4], [5,9]].

Chapter 33

Linked List Cycle

51% Accepted
Given a linked list, determine if it has a cycle in it.
Example
Given -21->10->4->5, tail connects to node index 1, return true
Challenge
Follow up:
Can you solve it without using extra space?

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

Chapter 34

Linked List Cycle II

35% Accepted
Given a linked list, return the node where the cycle begins. If there is no cycle, return null.
Example
Given -21->10->4->5, tail connects to node index 1，返回 10
Challenge
Follow up:
Can you solve it without using extra space?

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

Chapter 35

Max Tree: 14/16 TLE

24% Accepted

Given an integer array with no duplicates. A max tree building on this array is defined as follow:

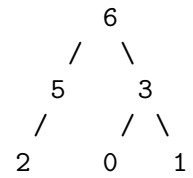
The root is the maximum number in the array

The left subtree and right subtree are the max trees of the subarray divided by the root number.

Construct the max tree by the given array.

Example

Given [2, 5, 6, 0, 3, 1], the max tree is



Challenge

O(n) time complexity

Chapter 36

Maximum Depth

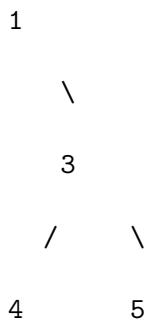
68% Accepted

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example

Given a binary tree as follow:



The maximum depth is 3

```
public void maxDepth(TreeNode root, int cnt, List<Integer> res) {
    if (root == null) return;
    if (root.left == null && root.right == null) {
        if (cnt > res.get(0)) res.set(0, cnt);
        return;
    }
    maxDepth(root.left, cnt + 1, res);
    maxDepth(root.right, cnt + 1, res);
}

public int maxDepth(TreeNode root) {
    if (root == null) return 0;
    List<Integer> res = new ArrayList<Integer>();
    res.add(1);
    maxDepth(root, 1, res);
    return res.get(0);
}
```

Chapter 37

Maximum Subarray

35% Accepted

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

Note

The subarray should contain at least one number

Example

For 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

Can you do it in time complexity O(n)?

```
public int maxSubArray(ArrayList<Integer> nums) {
    int n = nums.size();
    int [] res = new int[n];
    res[0] = nums.get(0);
    int result = res[0];
    for (int i = 1; i < n; i++) {
        res[i] = Math.max(nums.get(i), res[i - 1] + nums.get(i));
        result = Math.max(result, res[i]);
    }
    return result;
}
```

Chapter 38

Maximum Subarray Difference: I think I lost the other direction

21% Accepted
Given an array with integers.
Find two non-overlapping subarrays A and B, which |SUM(A) - SUM(B)| is the largest.
Return the largest difference.
Note
The subarray should contain at least one number
Example
For [1, 2, -3, 1], return 6
Challenge
O(n) time and O(n) space.

Chapter 39

Maximum Subarray II

23% Accepted

Given an array of integers, find two non-overlapping subarrays which have the largest sum.

The number in each subarray should be contiguous.

Return the largest sum.

Note

The subarray should contain at least one number

Example

For given [1, 3, -1, 2, -1, 2], the two subarrays are [1, 3] and [2, -1, 2] or [1, 3, -1, 2] and [1], they both have the largest sum 7.

Challenge

Can you do it in time complexity O(n) ?

```
public int maxTwoSubArrays(ArrayList<Integer> nums) {
    int n = nums.size();
    int [][] max = new int[2][n];
    max[0][0] = nums.get(0);
    max[1][0] = nums.get(0);
    int [][] min = new int[2][n];
    min[0][n - 1] = nums.get(n - 1);
    min[1][n - 1] = nums.get(n - 1);
    int res = Integer.MIN_VALUE;
    for (int i = 1; i < n; i++) {
        max[0][i] = Math.max(nums.get(i), max[0][i - 1] + nums.get(i));
        max[1][i] = Math.max(max[0][i], max[1][i - 1]);
    }
    for (int i = n - 2; i > 0; i--) {
        min[0][i] = Math.max(nums.get(i), min[0][i + 1] + nums.get(i));
        min[1][i] = Math.max(min[0][i], min[1][i + 1]);
        int tmp = Math.max(max[1][i] + min[1][i + 1],
                           max[1][i - 1] + min[1][i]);
        res = Math.max(res, tmp);
    }
    res = Math.max(res, max[1][0] + min[1][1]);
    return res;
}
```

[1] DEFINITION NOT FOUND.

应该可以把代码再精减一下的。

Chapter 40

Maximum Subarray III: this one is crazy, should consider recursive ways···

19% Accepted

Given an array of integers and a number k, find k non-overlapping subarrays which have the largest sum.

The number in each subarray should be contiguous.

Return the largest sum.

Note

The subarray should contain at least one number

Chapter 41

Merge Sorted Array

32% Accepted
Merge two given sorted integer array A and B into a new sorted integer array.
Example
A=[1,2,3,4]
B=[2,4,5,6]
return [1,2,2,3,4,4,5,6]
Challenge
How can you optimize your algorithm if one array is very large and the other is very small?

```
public ArrayList<Integer> mergeSortedArray(ArrayList<Integer> A, ArrayList<Integer> B)
    int m = A.size();
    int n = B.size();
    ArrayList<Integer> res = new ArrayList<Integer>();
    int i = 0, j = 0;
    while (i < m || j < n) {
        while (i < m && j < n) {
            if (A.get(i) <= B.get(j))
                res.add(A.get(i++));
            else res.add(B.get(j++));
        }
        if (i == m && j == n) return res;
        if (i == m) while (j < n) res.add(B.get(j++));
        else while (i < m) res.add(A.get(i++));
        return res;
    }
    return res;
}
```

Chapter 42

Merge Sorted Array II

40% Accepted

Given two sorted integer arrays A and B, merge B into A as one sorted array.

Note

You may assume that A has enough space (size that is greater or equal to m + n) to hold additional elements from B. The number of elements initialized in A and B are mand n respectively.

Example

A = [1, 2, 3, empty, empty] B = [4,5]

After merge, A will be filled as [1,2,3,4,5]

```
public void mergeSortedArray(int[] a, int m, int[] b, int n) {
    int i = m - 1, j = n - 1, k = m + n - 1;
    while (i >= 0 && j >= 0) {
        if (a[i] <= b[j]) a[k--] = b[j--];
        else a[k--] = a[i--];
    }
    while (j >= 0) a[k--] = b[j--];
    return;
}
```

Chapter 43

Merge Two Sorted Lists Show Result My Submissions

39% Accepted

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

Example

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

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(Integer.MIN_VALUE);
    ListNode curr = dummy;
    ListNode one = null;
    ListNode two = null;
    for ( one = l1, two = l2; one != null && two != null; curr = curr.next) {
        int a = one.val;
        int b = two.val;
        if (a <= b) {
            curr.next = one;
            one = one.next;
        } else {
            curr.next = two;
            two = two.next;
        }
    }
    if (one == null) curr.next = two;
    else curr.next = one;
    return dummy.next;
}
```

Chapter 44

Min Stack

25% Accepted
Implement a stack with min() function, which will return the smallest number in the stack.
It should support push, pop and min operation all in O(1) cost.
Note
min operation will never be called if there is no number in the stack
Example
Operations: push(1), pop(), push(2), push(3), min(), push(1), min() Return: 1, 2, 1

```
public static class Solution {
    Stack<Integer> s;
    Stack<Integer> t;
    public Solution() {
        s = new Stack<Integer>();
        t = new Stack<Integer>();
    }
    public void push(int x) {
        if (t.isEmpty() || (!t.isEmpty() && x <= t.peek().intValue()))
            t.push(x);
        s.push(x);
    }
    public int pop() {
        int tmp = s.pop().intValue();
        if (!t.isEmpty() && tmp == t.peek().intValue())
            t.pop();
        return tmp;
    }
    public int min() {
        return t.peek();
    }
}
```

Chapter 45

Minimum Path Sum

35% Accepted

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

Note

You can only move either down or right at any point in time.

```
public int minPathSum(int[][] grid) {
    int m = grid.length;
    int n = grid[0].length;
    int res[][] = new int[m][n];
    res[0][0] = grid[0][0];
    for (int i = 1; i < n; i++) res[0][i] = res[0][i - 1] + grid[0][i];
    for (int j = 1; j < m; j++) res[j][0] = res[j - 1][0] + grid[j][0];
    for (int i = 1; i < m; i++)
        for (int j = 1; j < n; j++)
            res[i][j] = Math.min(res[i - 1][j], res[i][j - 1]) + grid[i][j];
    return res[m - 1][n - 1];
}
```

Chapter 46

O(1) Check Power of 2

22% Accepted
Using O(1) time to check whether an integer n is a power of 2.
Example
For n=4, return true
For n=5, return false
Challenge
O(1) time

```
public boolean checkPowerOf2(int n) {
    if (n <= 0) return false;
    while (n > 0) {
        if (n & 1 == 1) return false;
        else n >>= 1;
    }
    return true;
}
```

Chapter 47

Partition Array

23% Accepted

Given an array "nums" of integers and an int "k", Partition the array (i.e move the elements in "nums") such that,

- All elements < k are moved to the left
- All elements >= k are moved to the right

Return the partitioning Index, i.e the first index "i" nums[i] >= k.

Note

You should do really partition in array "nums" instead of just counting the numbers of integers smaller than k.

If all elements in "nums" are smaller than k, then return "nums.length"

Example

If nums=[3,2,2,1] and k=2, a valid answer is 1.

Challenge

Can you partition the array in-place and in O(n)?

```
public int partitionArray(ArrayList<Integer> nums, int k) {
    int n = nums.size();
    if (n == 0) return 0;
    int i = 0, j = n - 1;
    while (i < j) {
        while (j >= 0 && nums.get(j) >= k) j--;
        while (i < n && nums.get(i) < k) i++;
        if (i == n) return n;
        if (j == -1) return 0;
        else if (i < j) {
            int tmp = nums.get(i);
            nums.set(i++, nums.get(j));
            nums.set(j--, tmp);
        }
    }
    System.out.println(nums);
    return i;
}
```

Chapter 48

Recover Rotated Sorted Array

27% Accepted
Given a rotated sorted array, recover it to sorted array in-place.
Example
[4, 5, 1, 2, 3] -> [1, 2, 3, 4, 5]
Challenge
In-place, O(1) extra space and O(n) time.
Clarification
What is rotated array:

- For example, the orginal array is [1,2,3,4], The rotated array of it can be [1,2,3,4], [2,3,4,1], [3,4,1,2], [4,1,2,3]

```
public int getMinIdx(ArrayList<Integer> a) {
    int n = a.size();
    if (n == 1) return 0;
    if (n == 2) return a.get(0) < a.get(1) ? 0 : 1;
    int bgn = 0, end = n - 1;
    while (bgn < end - 1) {
        int mid = bgn + (end - bgn) / 2;
        if (a.get(mid) < a.get(bgn) && a.get(bgn) > a.get(end))
            end = mid;
        else if (a.get(mid) > a.get(bgn) && a.get(bgn) > a.get(end))
            bgn = mid + 1;
        else if (a.get(mid) > a.get(bgn) && a.get(bgn) < a.get(end))
            end = mid - 1;
    }
    if (bgn == end) return bgn;
    if (bgn == end - 1) return a.get(bgn) < a.get(end) ? bgn : end;
    return -1;
}

public void recoverRotatedSortedArray(ArrayList<Integer> nums) {
    int n = nums.size();
    int tmp = 0;
    if (n < 2) return;
    if (n == 2) {
        if (nums.get(0) > nums.get(0)) {
```

```
            tmp = nums.get(0);
            nums.set(0, nums.get(1));
            nums.set(1, tmp);
        }
        return;
    }
    int i = 0, j = getMinIdx(nums);
    if (j == 0) return;
    int cnt = n - j;
    while (cnt > 0) {
        tmp = nums.get(n - 1);
        nums.remove(n - 1);
        nums.add(0, tmp);
        --cnt;
    }
    return;
}
```

Chapter 49

Nth to Last Node in List

51% Accepted
Find the nth to last element of a singly linked list.
The minimum number of nodes in list is n.
Example
Given a List 3->2->1->5->null and n = 2, return node whose value is 1.

```
ListNode nthToLast(ListNode head, int n) {
    int cnt = 0;
    ListNode curr = head;
    while (cnt < n && curr != null) {
        ++cnt;
        curr = curr.next;
    }
    if (cnt == n && curr == null) return head;
    if (cnt < n) return null;
    ListNode prev = head;
    while (curr != null) {
        prev = prev.next;
        curr = curr.next;
    }
    return prev;
}
```

Chapter 50

Partition List: MLE

32% Accepted

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,

Given 1->4->3->2->5->2->null and x = 3,

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

```
public ListNode partition(ListNode head, int x) {
    ListNode left = new ListNode(Integer.MIN_VALUE);
    ListNode right = new ListNode(Integer.MIN_VALUE);
    ListNode one = left;
    ListNode two = right;
    for(;head != null; head = head.next) {
        if (head.val < x) {
            one.next = head;
            one = one.next;
        } else {
            two.next = head;
            two = two.next;
        }
    }
    one.next = right.next;
    return left.next;
}
```

Chapter 51

Product of Array Exclude Itself

26% Accepted
Given an integers array A.
Define B[i] = A^1 * ··· * A[i-1] * A[i+1] * ··· * A[n-1], calculate B without divide operation.
Example
For A=[1, 2, 3], B is [6, 3, 2]

```
public ArrayList<Long> productExcludeItself(ArrayList<Integer> a) {
    int n = a.size();
    ArrayList<Long> res = new ArrayList<Long>(n);
    if (n == 1) return res;
    long bgn = 1, end = 1;
    for (int i = 0; i < n; i++) {
        bgn = 1; end = 1;
        for (int j = 0; j < i; j++)
            bgn *= a.get(j);
        for (int k = i + 1; k < n; k++)
            end *= a.get(k);
        res.add(bgn * end);
    }
    return res;
}
```

Chapter 52

Remove Duplicates from Sorted Array

33% Accepted

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,

Given input array A = [1,1,2],

Your function should return length = 2, and A is now [1,2].

```
public int removeDuplicates(int[] nums) {
    int n = nums.length;
    if (n == 0) return 0;
    int i = 0;
    for (int j = 1; j < n; j++) {
        if (nums[j] == nums[i]) continue;
        nums[++i] = nums[j];
    }
    return i + 1;
}
```

Chapter 53

Remove Duplicates from Sorted Array II

29% Accepted
Follow up for "Remove Duplicates":
What if duplicates are allowed at most twice?
For example,
Given sorted array A = [1,1,1,2,2,3],
Your function should return length = 5, and A is now [1,1,2,2,3].

```
public int removeDuplicates(int[] nums) {
    int n = nums.length;
    if (n == 0) return 0;
    int i = 0;
    int cnt = 1;
    for (int j = 1; j < n; j++) {
        if (nums[j] == nums[i]) {
            if (cnt < 2) {
                ++cnt;
                nums[++i] = nums[j];
            }
            continue;
        } else {
            nums[++i] = nums[j];
            cnt = 1;
        }
    }
    return i + 1;
}
```

Chapter 54

Remove Duplicates from Sorted List

39% Accepted
Given a sorted linked list, delete all duplicates such that each element appear only once.
Example
Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.

```
public static ListNode deleteDuplicates(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode prev = head;
    ListNode curr = head.next;
    for( ;curr != null; curr = curr.next) {
        if (curr.val != prev.val) {
            prev.next = curr;
            prev = prev.next;
        }
    }
    prev.next = null;
    return head;
}
```

Chapter 55

Remove Element

45% Accepted
Given an array and a value, remove all occurrences of that value in place and return the new length.
The order of elements can be changed, and the elements after the new length don't matter.
Example
Given an array [0,4,4,0,0,2,4,4], value=4
return 4 and front four elements of the array is [0,0,0,2]

```
public int removeElement(int[] a, int elem) {
    int n = a.length;
    int i = -1;
    for (int j = 0; j < n; j++) {
        if (a[j] == elem) continue;
        a[++i] = a[j];
    }
    return i + 1;
}
```

Chapter 56

Remove Nth Node From End of List: don't know if there is bug, run 15/15 forever···

40% Accepted
Given a linked list, remove the nth node from the end of list and return its head.
Note
The minimum number of nodes in list is n.
Example
Given linked list: 1->2->3->4->5->null, and n = 2.
After removing the second node from the end, the linked list becomes 1->2->3->5->null.
Challenge
O(n) time

```
ListNode removeNthFromEnd(ListNode head, int n) {
    int cnt = 0;
    ListNode curr = head;
    while (cnt < n && curr != null) {
        ++cnt;
        curr = curr.next;
    }
    if (cnt == n && curr == null) return head.next;
    if (cnt < n) return null;
    ListNode prev = head;
    ListNode slow = null;
    while (curr != null) {
        slow = prev;
        prev = prev.next;
        curr = curr.next;
    }
    slow.next = prev.next;
    return head;
}
```