LintCode Online Coding Interview Questions - Problem-wise

deepwaterooo

2021 年 10 月 29 日

目录

- #+HTML$_{HEAD}$: <style type="text/css">/*<![CDATA[*/ pre.src { height: 100px; overflow: auto; } *]]>* </style>

Chapter 1

动态规划

1.1   673. Number of Longest Increasing Subsequence

Given an integer array nums, return the number of longest increasing subsequences. Notice that the sequence has to be strictly increasing.

```java
public int findNumberOfLIS(int[] nums) { // dynamic programming
    int n = nums.length;
    int [][] arr = new int[n][2];
    int maxLength = 1;
    for (int i = 0; i < n; i++)
        Arrays.fill(arr[i], 1);
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            if (nums[j] > nums[i]) {
                if (arr[i][0] + 1 > arr[j][0]) {
                    arr[j][0] = arr[i][0] +1;
                    arr[j][1] = arr[i][1];
                    maxLength = Math.max(maxLength, arr[j][0]);
                } else if (arr[i][0] + 1 == arr[j][0])
                    arr[j][1] += arr[i][1];
            }
        }
    }
    int cnt = 0;
    for (int i = 0; i < n; i++)
        if (arr[i][0] == maxLength) cnt += arr[i][1];
    return cnt;
}
```

Chapter 2

树结构：各种新型数据结构

2.1 Create Sorted Array through Instructions

Given an integer array instructions, you are asked to create a sorted array from the elements in instructions. You start with an empty container nums. For each element from left to right in instructions, insert it into nums. The cost of each insertion is the minimum of the following: The number of elements currently in nums that are strictly less than instructions[i]. The number of elements currently in nums that are strictly greater than instructions[i]. For example, if inserting element 3 into nums = [1,2,3,5], the cost of insertion is min(2, 1) (elements 1 and 2 are less than 3, element 5 is greater than 3) and nums will become [1,2,3,3,5]. Return the total cost to insert all elements from instructions into nums. Since the answer may be large, return it modulo $10^9 + 7$

```java
// https://blog.csdn.net/qq_28033719/article/details/112506925
private static int N = 100001;
private static int [] tree = new int [N]; // 拿元素值作为 key 对应 tree 的下标值
public int lowbit(int i) {
    return i & -i;
}
public void update(int i, int v) { // 更新父节点
    while (i <= N) {
        tree[i] += v;
        i += lowbit(i);
    }
}
public int getSum(int i) { // 得到以 i 为下标 1-based 的所有子、叶子节点的和，也就是 [1, i] 的和，1-based
    int ans = 0;
    while (i > 0) {
        ans += tree[i];
        i -= lowbit(i);
    }
    return ans;
}
public int createSortedArray(int[] instructions) {
    int n = instructions.length;
    long res = 0;
    Arrays.fill(tree, 0);
    for (int i = 0; i < n; i++) {
        //              严格小于此数的个数  严格大于此数的个数：为总个数（不含自己） - 小于自己的个数
        res += Math.min(getSum(instructions[i]-1), i-getSum(instructions[i]));
        update(instructions[i], 1);
    }
    return (int)(res % ((int)Math.pow(10, 9) + 7));
}
```

Chapter 3

排序与 recursion

3.1 Reverse Pairs

Given an integer array nums, return the number of reverse pairs in the array. A reverse pair is a pair (i, j) where $0 <= i < j <$ nums.length and nums[i] $> 2 *$ nums[j].

```java
private int mergeSortCount(long [] arr, int bgn, int end) {
    if (bgn >= end) return 0;
    int mid = bgn + (end-bgn)/2;
    int cnt = mergeSortCount(arr, bgn, mid) + mergeSortCount(arr, mid+1, end);
    for (int i = bgn, j = mid+1; i <= mid; i++) {
        while (j <= end && arr[i] > 2*arr[j]) j++;
        cnt += j - (mid+1);
    }
    Arrays.sort(arr, bgn, end+1);
    return cnt;
}
public int reversePairs(int[] nums) {
    int n = nums.length;
    return mergeSortCount(Arrays.stream(nums).mapToLong(i -> i).toArray(), 0, n-1);
}
// bit 的解法: https://www.cnblogs.com/grandyang/p/6657956.html
```

3.2 306. Additive Number

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain at least three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

Given a string containing only digits '0'-'9', write a function to determine if it's an additive number.

Note: Numbers in the additive sequence cannot have leading zeros, so sequence 1, 2, 03 or 1, 02, 3 is invalid.

```java
public boolean isAdditiveNumber(String num) {
    int n = num.length();
    if (n < 3) return false;
    for (int i = 1; i <= num.length() >> 1; i++)
        for (int j = 1; j + i < num.length(); j++)
            if (isValid(num, num.substring(0, i), num.substring(i, i + j), i + j)) return true;
    return false;
}
private boolean isValid(String num, String first, String second, int index) {
    if (first.length() > 1 && first.startsWith("0")
        || second.length() > 1 && second.startsWith("0")) return false;
    if (index == num.length()) return true; // 如果只有两个数是有效的!!!
```

```java
    long sum = Long.parseLong(first) + Long.parseLong(second);
    if (num.startsWith(sum + "", index)) // 间接检测第三个数
        if (isValid(num, second, sum + "", index + (sum + "").length())) return true;
    return false;
}
```

Chapter 4

others

4.1 Predict the Winner

You are given an integer array nums. Two players are playing a game with this array: player 1 and player 2. Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of 0. At each turn, the player takes one of the numbers from either end of the array (i.e., nums[1] or nums[nums.length - 1]) which reduces the size of the array by 1. The player adds the chosen number to their score. The game ends when there are no more elements in the array. Return true if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return true. You may assume that both players are playing optimally.

```
private int helper( int [] arr, int i, int j) {
    if (i == j) return arr[i];
    else return Math.max(arr[i] - helper(arr, i+1, j), arr[j] - helper(arr, i, j-1));
}
public boolean PredictTheWinner(int[] nums) {
    int n = nums.length;
    if (n == 1) return true;
    return helper(nums, 0, n-1) >= 0;
    // res = false;
    // List<Integer> l = new ArrayList<>();
    // l.add(nums[0]);
    // helper(nums, 1, n-1, nums[0], 0, false, l);
    // l = new ArrayList<>();
    // l.add(nums[n-1]);
    // helper(nums, 0, n-2, nums[n-1], 0, false, l);
    // return res;
}
```

4.2 Sliding Window Median

The median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle values. For examples, if arr = [2,3,4], the median is 3. For examples, if arr = [1,2,3,4], the median is $(2 + 3) / 2 = 2.5$. You are given an integer array nums and an integer k. There is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the median array for each window in the original array. Answers within 10-5 of the actual value will be accepted.

```
public double[] medianSlidingWindow(int[] nums, int k) {
    TreeMap<Integer, Integer> ma = new TreeMap<>();
```

---

[1]DEFINITION NOT FOUND.

```java
        TreeMap<Integer, Integer> mb = new TreeMap<>();
        for (int i = 0; i < k; i++) {
            if (i % 2 == 0) {
                mb.put(nums[i], mb.getOrDefault(nums[i], 0) + 1);
                int n = mb.firstKey();
                if (mb.get(n) == 1) mb.remove(n);
                else mb.put(n, mb.get(n) - 1);
                ma.put(n, ma.getOrDefault(n, 0) + 1);
            } else {
                ma.put(nums[i], ma.getOrDefault(nums[i], 0) + 1);
                int n = ma.lastKey();
                if (ma.get(n) == 1) ma.remove(n);
                else ma.put(n, ma.get(n) - 1);
                mb.put(n, mb.getOrDefault(n, 0) + 1);
            }
        }
        double [] res = new double[nums.length-k+1];
        if (k % 2 == 1) res[0] = ma.lastKey();
        else res[0] =  (double)(((long)(ma.lastKey()) + (long)(mb.firstKey())) / 2.0);
        for (int i = 0; i + k < nums.length; i++) {
            ma.put(nums[i+k], ma.getOrDefault(nums[i+k], 0) + 1);
            int n = ma.lastKey();
            if (ma.get(n) == 1) ma.remove(n);
            else ma.put(n, ma.get(n) - 1);
            mb.put(n, mb.getOrDefault(n, 0) + 1);
            if (ma.containsKey(nums[i])) {
                if (ma.get(nums[i]) == 1) ma.remove(nums[i]);
                else ma.put(nums[i], ma.get(nums[i]) - 1);
                int v = mb.firstKey();
                if (mb.get(v) == 1) mb.remove(v);
                else mb.put(v, mb.get(v) - 1);
                ma.put(v, ma.getOrDefault(v, 0) + 1);
            } else {
                if (mb.get(nums[i]) == 1) mb.remove(nums[i]);
                else mb.put(nums[i], mb.get(nums[i]) - 1);
            }
            if (k % 2 == 1) res[i+1] = ma.lastKey();
            else res[i+1] = (double)(((long)(ma.lastKey()) + (long)(mb.firstKey())) / 2.0);
        }
        return res;
}
```

4.3 Construct Binary Tree from Preorder and Postorder Traversal

Given two integer arrays, preorder and postorder where preorder is the preorder traversal of a binary tree of distinct values and postorder is the postorder traversal of the same tree, reconstruct and return the binary tree. If there exist multiple answers, you can return any of them.

```java
public TreeNode constructFromPrePost(int[] preorder, int[] postorder) {
    int n = preorder.length;
    TreeNode r = new TreeNode(preorder[0]);
    if (n == 1) return r;
    Stack<TreeNode> s = new Stack<>();
    s.push(r);
    int idx = 0;
    for (int i = 1; i < n; i++) {
        TreeNode cur = new TreeNode(preorder[i]);
        if (s.peek().left == null) s.peek().left = cur;
        else s.peek().right = cur;
        s.push(cur);
        while (idx < n && postorder[idx] == s.peek().val) {
            s.pop();
            ++idx;
        }
    }
```

```
        return r;
}
```

4.4 Path Sum III

Given the root of a binary tree and an integer targetSum, return the number of paths where the sum of the values along the path equals targetSum. The path does not need to start or end at the root or a leaf, but it must go downwards (i.e., traveling only from parent nodes to child nodes).

```java
private int solve(TreeNode r, int t, int value) {
    if (r == null) return 0;
    if (value + r.val == t)
        return 1 + solve(r.left, 0, 0) + solve(r.right, 0, 0);
    return solve(r.left, t, value + r.val) + solve(r.right, t, value + r.val);
}
public int pathSum(TreeNode root, int targetSum) {
    if (root == null) return 0;
    return solve(root, targetSum, 0) + pathSum(root.left, targetSum) + pathSum(root.right, targetSum);
}
```

4.5 Critical Connections in a Network

- There are n servers numbered from 0 to n - 1 connected by undirected server-to-server connections forming a network where connections[i] = [ai, bi] represents a connection between servers ai and bi. Any server can reach other servers directly or indirectly through the network.

- A critical connection is a connection that, if removed, will make some servers unable to reach some other server.

- Return all critical connections in the network in any order.

```java
static class Eg {
    int u, v, next;
    // int w;
    boolean cut;
    // int num;
}
public Eg[] egs;
public int cnt;
public int [] fir; // 边的出发点
int [] low;
int [] dfn;
int recdfn;
void tarjanAddEg(int u, int v, int w) {
    egs[cnt] = new Eg();
    egs[cnt].u = u;
    egs[cnt].v = v;
    // egs[cnt].w = w;
    egs[cnt].cut = false;
    // egs[cnt].num = 0;
    egs[cnt].next = fir[u]; // ?
    fir[u] = cnt++;         // ?
}
private void initTarjan(int nodeSize, int edgeSize) {
    cnt = 0;
    egs = new Eg [edgeSize];
    low = new int [nodeSize];
    dfn = new int [nodeSize];
    fir = new int [edgeSize];
    Arrays.fill(fir, -1);
}
```

```java
private void tarjan(int u, int fa) { // fa: father
    low[u] = ++recdfn;
    dfn[u] = recdfn;
    int have = 0;
    for (int i = fir[u]; i != -1; i = egs[i].next) {
        int v = egs[i].v;
        if (have == 0 && v == fa) { // 走过你来时的路
            have++;
            continue;
        }
        if (dfn[v] == 0) { // dfs 过程中还未经过该点
            tarjan(v, u);
            low[u] = Math.min(low[u], low[v]);
            if (dfn[u] < low[v]) { // 连通世外桃源与外界的路
                // 当 dfn[x] < low[y] 的时候:
                // --- 我们发现从 yy 节点出发, 在不经过 (x,y)(x,y) 的前提下, 不管走哪一条边, 我们都无法抵达 xx 节点, 或者比 xx 节点更早出现的节点
                // --- 此时我们发现 yy 所在的子树似乎形成了一个封闭圈, 那么 (x,y)(x,y) 自然也就是桥了.
                egs[i].cut = true;
                egs[i^1].cut = true; // ???
            }
        } else {
            low[u] = Math.min(low[u], dfn[v]); // 取已访问的节点的 dfs 序的最小值
        }
    }
}
private boolean findEdgeCut(int l, int r) {
    Arrays.fill(low, 0);
    Arrays.fill(dfn, 0);
    recdfn = 0;
    tarjan(l, l);
    for (int i = l; i <= r; i++) {
        if (dfn[i] == 0) return false;
    }
    return true;
}
public List<List<Integer>> criticalConnections(int n, List<List<Integer>> connections) {
    initTarjan(n, connections.size()*2);
    for (List<Integer> eg : connections) {
        tarjanAddEg(eg.get(0), eg.get(1), 1);
        tarjanAddEg(eg.get(1), eg.get(0), 1);
    }
    // boolean ans = findEdgeCut(0, n-1);
    Arrays.fill(low, 0);
    Arrays.fill(dfn, 0);
    recdfn = 0;
    tarjan(0, 0);
    List<List<Integer>> res = new ArrayList<>();
    int l = connections.size();
    for (int i = 0; i < l * 2; i += 2) { // i += 2 skipped egs[i^1] ?
        Eg eg = egs[i];
        if (eg != null && eg.cut) {
            List<Integer> t = new ArrayList<>();
            t.add(eg.u);
            t.add(eg.v);
            res.add(t);
        }
    }
    return res;
}
```

Chapter 5

LintCode Online Coding Interview Questions - Problem-wise

5.1 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   b   c   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)
```

```java
/**
 * @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], numbers[x], numbers[y])));
                    x++;
                    y--;
                }
                else if (numbers[x] + numbers[y] < tmp) x++;
```

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

5.2   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

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

5.4   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;
}
```

5.5   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 ≤ b ≤ 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)
```

```java
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;
}
```

5.6   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).

```java
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 - target))
                    res = numbers[i] + numbers[j] + numbers[k];
                k++;
            }
        }
    }
    return res;
}
```

5.7   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.

5.8   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"]
```

```java
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;
}
```

5.9   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.

5.10   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}

```
A)   3              B)    3
    / \                    \
   9   20                   20
      /  \                  /  \
    15    7               15    7
```
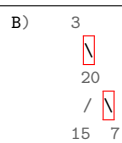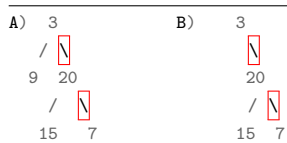
The binary tree A is a height-balanced binary tree, but B is not.

5.11   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

```java
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;
}
```

5.12   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

5.13   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

5.14   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

5.15   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;
}
```

5.16 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;
}
```

5.17 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.

```java
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;
}
```

5.18   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]
]
```

```java
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;
}
```

5.19 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
```
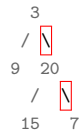
Return 6.

5.20 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]
]
```

```java
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;
}
```

5.21   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

```java
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];
}
```

5.22 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, $\cdots$, ak) must be in non-descending order. (ie, a1   a2 $\cdots$ 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]
```

```java
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;
}
```

5.23　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]
```

```java
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;
}
```

5.24　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]]
```

```java
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;
}
```

5.25   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.

```java
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;
}
```

5.26   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.

```java
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;
}
```

5.27 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"

5.28 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

```java
public int findMin(int[] num) {
    int n = num.length;
```

```java
    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];
}
```

5.29 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

```java
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];
}
```

5.30   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)

```java
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;
}
```

5.31   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.

```java
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;
}
```

5.32 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.

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

5.33 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.

```java
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;
    }
}
```

5.34   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]].

5.35   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?

```java
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;
}
```

5.36 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?

```java
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;
}
```

5.37 Max Tree: 14/16 TLE

24% Accepted

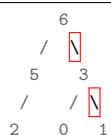Given an integer array with no duplicates. A max tree building on this array is defined as follow:

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

```
        6
      /   \
     5     3
    /     / \
   2     0   1
```

Challenge

O(n) time complexity

5.38   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:

```
    1
   / \
  2   3
     / \
    4   5
```

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);
}
```

5.39   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);
```

```java
    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;
}
```

5.40 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.

5.41 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) ?

```java
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],
```

---

[1]DEFINITION NOT FOUND.

```
                         max[1][i - 1] + min[1][i]);
            res = Math.max(res, tmp);
        }
        res = Math.max(res, max[1][0] + min[1][1]);
        return res;
}
```

应该可以把代码再精减一下的。

5.42 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

5.43 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?

```java
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;
}
```

5.44 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]

```java
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;
}
```

5.45 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

```java
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;
}
```

5.46 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

```java
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();
    }
}
```

5.47   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.

```java
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];
}
```

5.48   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

```java
public boolean checkPowerOf2(int n) {
    if (n <= 0) return false;
    while (n > 0) {
```

```
        if (n & 1 == 1) return false;
        else n >>= 1;
    }
    return true;
}
```

5.49 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)?

```java
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;
}
```

5.50 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]

```java
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;
}
```

5.51   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.

```java
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;
}
```

5.52   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.

```java
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;
}
```

5.53   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]

```java
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;
}
```

5.54   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].

```java
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;
}
```

5.55   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].

```java
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;
}
```

5.56   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;
}
```

5.57   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;
}
```

5.58   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;
```

```java
        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;
}
```

Chapter 6

test

6.1 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;
}
```