Table of Contents

QuestionList

- 1. Two Sum
- 2. Add Two Numbers
- 3. Longest Substring Without Repeating Characters
- 4. Median of Two Sorted Arrays
- 5. Longest Palindromic Substring
- 6. ZigZag Conversion
- 7. Reverse Integer
- 8. String to Integer (atoi)
- 9. Palindrome Number
- 10. Regular Expression Matching
- 11. Container With Most Water
- 12. Integer to Roman
- 14. Longest Common Prefix
- 16. 3Sum Closest
- 18. 4Sum
- 19. Remove Nth Node From End of List
- 20. Valid Parentheses
- 21. Merge Two Sorted Lists
- 22. Generate Parentheses
- 24. Swap Nodes in Pairs
- 25. Reverse Nodes in k-Group
- 27. Remove Element
- 28. Implement strStr()
- 29. Divide Two Integers
- 30. Substring with Concatenation of All Words
- 31. Next Permutation
- 32. Longest Valid Parentheses
- 33. Search in Rotated Sorted Array
- 34. Search for a Range
- 35. Search Insert Position
- 36. Valid Sudoku
- 37. Sudoku Solver
- 38. Count and Say
- 39. Combination Sum

- 40. Combination Sum II
- 41. First Missing Positive
- 42. Trapping Rain Water
- 43. Multiply Strings
- 44. Wildcard Matching
- 45. Jump Game II
- 46. Permutations
- 47. Permutations II
- 48. Rotate Image
- 50. Pow(x, n)
- 51. N-Queens
- 52. N-Queens II
- 53. Maximum Subarray
- 54. Spiral Matrix
- 55. Jump Game
- 58. Length of Last Word
- 59. Spiral Matrix II
- 60. Permutation Sequence
- 61. Rotate List
- 62. Unique Paths
- 63. Unique Paths II
- 64. Minimum Path Sum
- 65. Valid Number
- 66. Plus One
- 68. Text Justification
- 69. Sqrt(x)
- 70. Climbing Stairs
- 72. Edit Distance
- 73. Set Matrix Zeroes
- 74. Search a 2D Matrix
- 75. Sort Colors
- 76. Minimum Window Substring
- 77. Combinations
- 81. Search in Rotated Sorted Array II
- 82. Remove Duplicates from Sorted List II
- 83. Remove Duplicates from Sorted List

- 84. Largest Rectangle in Histogram
- 85. Maximal Rectangle
- 86. Partition List
- 87. Scramble String
- 89. Gray Code
- 92. Reverse Linked List II
- 93. Restore IP Addresses
- 95. Unique Binary Search Trees II
- 96. Unique Binary Search Trees
- 97. Interleaving String
- 103. Binary Tree Zigzag Level Order Traversal
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 106. Construct Binary Tree from Inorder and Postorder Traversal
- 108. Convert Sorted Array to Binary Search Tree
- 109. Convert Sorted List to Binary Search Tree
- 113. Path Sum II
- 114. Flatten Binary Tree to Linked List
- 115. Distinct Subsequences
- 116. Populating Next Right Pointers in Each Node
- 117. Populating Next Right Pointers in Each Node II
- 118. Pascal's Triangle
- 119. Pascal's Triangle II
- 120. Triangle
- 121. Best Time to Buy and Sell Stock I
- 122. Best Time to Buy and Sell Stock II
- 123. Best Time to Buy and Sell Stock III
- 124. Binary Tree Maximum Path Sum
- 126. Word Ladder II
- 127. Word Ladder
- 128. Longest Consecutive Sequence
- 129. Sum Root to Leaf Numbers
- 130. Surrounded Regions
- 131. Palindrome Partitioning
- 132. Palindrome Partitioning II
- 134. Gas Station
- 135. Candy

- 136. Single Number
- 137. Single Number II
- 138. Copy List with Random Pointer
- 140. Word Break II
- 141. Linked List Cycle
- 142. Linked List Cycle II
- 143. Reorder List
- 144. Binary Tree Preorder Traversal
- 145. Binary Tree Postorder Traversal
- 147. Insertion Sort List
- 148. Sort List
- 149. Max Points on a Line
- 150. Evaluate Reverse Polish Notation
- 151. Reverse Words in a String
- 152. Maximum Product Subarray
- 153. Find Minimum in Rotated Sorted Array
- 154. Find Minimum in Rotated Sorted Array II
- 155. Min Stack
- 156. Binary Tree Upside Down
- 157. Read N Characters Given Read4
- 158. Read N Characters Given Read4 II
- 159. Longest Substring with At Most Two Distinct Characters
- 160. Intersection of Two Linked Lists
- 162. Find Peak Element
- 163. Missing Ranges
- 164. Maximum Gap
- 165. Compare Version Numbers
- 166. Fraction to Recurring Decimal
- 168. Excel Sheet Column Title
- 169. Majority Element
- 171. Excel Sheet Column Number
- 172. Factorial Trailing Zeroes
- 174. Dungeon Game
- 179. Largest Number
- 186. Reverse Words in a String II
- 187. Repeated DNA Sequences

- 188. Best Time to Buy and Sell Stock IV
- 189. Rotate Array
- 190. Reverse Bits
- 191. Number of 1 Bits
- 198. House Robber
- 201. Bitwise AND of Numbers Range
- 202. Happy Number
- 203. Remove Linked List Elements
- 204. Count Primes
- 205. Isomorphic Strings
- 207. Course Schedule
- 208. Implement Trie (Prefix Tree)
- 210. Course Schedule II
- 211. Add and Search Word - Data structure design
- 212. Word Search II
- 213. House Robber II
- 214. Shortest Palindrome
- 216. Combination Sum III
- 217. Contains Duplicate
- 218. The Skyline Problem
- 219. Contains Duplicate II
- 220. Contains Duplicate III
- 222. Count Complete Tree Nodes
- 223. Rectangle Area
- 224. Basic Calculator
- 225. Implement Stack using Queues
- 227. Basic Calculator II
- 228. Summary Ranges
- 229. Majority Element II
- 230. Kth Smallest Element in a BST
- 231. Power of Two
- 232. Implement Queue using Stacks
- 233. Number of Digit One
- 234. Palindrome Linked List
- 237. Delete Node in a Linked List
- 238. Product of Array Except Self

- 239. Sliding Window Maximum
- 240. Search a 2D Matrix II
- 241. Different Ways to Add Parentheses
- 242. Valid Anagram
- 246. Strobogrammatic Number
- 248. Strobogrammatic Number III
- 251. Flatten 2D Vector
- 254. Factor Combinations
- 256. Paint House
- 258. Add Digits
- 259. 3Sum Smaller
- 260. Single Number III
- 261. Graph Valid Tree
- 263. Ugly Number
- 264. Ugly Number II
- 265. Paint House II
- 266. Palindrome Permutation
- 267. Palindrome Permutation II
- 268. Missing Number
- 272. Closest Binary Search Tree Value II
- 273. Integer to English Words
- 274. H-Index
- 275. H-Index II
- 276. Paint Fence
- 278. First Bad Version
- 279. Perfect Squares
- 280. Wiggle Sort
- 281. Zigzag Iterator
- 282. Expression Add Operators
- 284. Peeking Iterator
- 286. Walls and Gates
- 287. Find the Duplicate Number
- 288. Unique Word Abbreviation
- 289. Game of Life
- 290. Word Pattern
- 291. Word Pattern II

- 331. Verify Preorder Serialization of a Binary Tree
- 332. Reconstruct Itinerary
- 335. Self Crossing
- 336. Palindrome Pairs
- 337. House Robber III
- 338. Counting Bits
- 340. Longest Substring with At Most K Distinct Characters
- 342. Power of Four
- 343. Integer Break
- 344. Reverse String
- 345. Reverse Vowels of a String
- 346. Moving Average from Data Stream
- 347. Top K Frequent Elements
- 348. Design Tic-Tac-Toe
- 349. Intersection of Two Arrays
- 350. Intersection of Two Arrays II
- 351. Android Unlock Patterns
- 353. Design Snake Game
- 354. Russian Doll Envelopes
- 355. Design Twitter
- 357. Count Numbers with Unique Digits
- 361. Bomb Enemy
- 363. Max Sum of Rectangle No Larger Than K
- 365. Water and Jug Problem
- 367. Valid Perfect Square
- 368. Largest Divisible Subset
- 371. Sum of Two Integers
- 372. Super Pow
- 373. Find K Pairs with Smallest Sums
- 374. Guess Number Higher or Lower
- 375. Guess Number Higher or Lower II
- 376. Wiggle Subsequence
- 377. Combination Sum IV
- 378. Kth Smallest Element in a Sorted Matrix
- 379. Design Phone Directory
- 381. Insert Delete GetRandom O(1) - Duplicates allowed

- 382. Linked List Random Node
- 383. Ransom Note
- 384. Shuffle an Array
- 385. Mini Parser
- 404. Sum of Left Leaves

1. Two Sum

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

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

Example: Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9, return [0, 1].

UPDATE (2016/2/13):

The return format had been changed to zero-based indices. Please read the above updated description carefully.

This question has a small trap should pay attention, say the following solution

```
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        int[] res = new int[2];
        for(int i=0; i<nums.length ;i++){
            map.put(nums[i], i);
        }
        for(int i=0; i< nums.length; i++){
            int t = target - nums[i];
            if(map.containsKey(t)){
                res[0] = i;
                res[1] = map.get(t);
            }
        }
        return res;
    }
}
```

for this specific case:

[0,4,3,0], 0, ===> the above solution gives [3,3], not [0,3]. to fix this bug.

```
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        int[] res = new int[2];
        for(int i=0; i< nums.length; i++){
            if(map.get(nums[i]) != null){
                res[0] = map.get(nums[i]);
                res[1] = i;
                return res;
            }else{
                map.put(target-nums[i], i); // cross reference t
he key and pos.
            }

        }
        return res;
    }
}
```

2. Add Two Numbers

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

```
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
```

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode res = new ListNode(-1);
        ListNode tail = res;
        int carry = 0;
        while(l1 != null || l2 != null || carry != 0){
            int x1 = l1 == null ? 0 : l1.val;
            int x2 = l2 == null ? 0 : l2.val;
            tail.next = new ListNode((x1+x2+carry)%10);
            carry = (x1+x2+carry)/10;
            tail = tail.next;
            if(l1 != null) l1 = l1.next;
            if(l2 != null) l2 = l2.next;
        }
        return res.next;
    }
}
```

3. Longest Substring Without Repeating Characters

Given a string, find the length of the longest substring without repeating characters.

Examples:

Given "abcabcbb", the answer is "abc", which the length is 3.

Given "bbbbb", the answer is "b", with the length of 1.

Given "pwwkew", the answer is "wke", with the length of 3. Note that the answer must be a substring, "pwke" is a subsequence and not a substring.

> When search for char in map, you need make sure no search pass the current starting index.

```
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        if (s == null || s.length() == 0) {
            return 0;
        }

        HashMap<Character, Integer> map = new HashMap<Character,
 Integer>();
        int lastDupIndex = -1;
        int maxSize = 0;
        char [] chars = s.toCharArray();
        for(int i=0;i<chars.length;i++) {
            Integer lastIndex = map.get(chars[i]);
            if (lastIndex != null && lastDupIndex < lastIndex) {
                lastDupIndex = lastIndex;
            }
            maxSize = Math.max(maxSize, i-lastDupIndex);
            map.put(chars[i], i);
        }

        return maxSize;
    }
}
```

7. Reverse Integer

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

Have you thought about this? Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?

For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

> to protect your code from overflow, when met multiply operation, use divide operation to check first.

```
public class Solution {
    public int reverse(int x) {
        boolean sign = x > 0 ;
        x = Math.abs(x);
        int res = 0;
        while(x > 0){
            if( res > Integer.MAX_VALUE/10) return 0;
            res = res*10 + x%10;
            x = x/10;
        }

        return sign ? res : -res;

    }
}
```

11. Container With Most Water

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0).

Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container.

two pointers.

```
public class Solution {
    public int maxArea(int[] height) {
        if(height == null || height.length <= 1)return 0;
        int i =0, j = height.length-1;
        int max = 0;
        while(i<j){
            max = Math.max(max, (j-i) * Math.min(height[i], heig
ht[j]));
            if( height[i] > height[j]){
                j--;
            }else{
                i++;
            }
        }
        return max;
    }
}
```

12. Integer to Roman

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

> This question belongs to a set of question, how to process numbers.

```
public class Solution {
    public String intToRoman(int n) {
        String[] symbol = new String[]{"M","CM","D","CD","C","XC"
,"L","XL","X","IX","V","IV","I"};
        int[] value = new int[]{1000,900,500,400, 100, 90, 50, 40
, 10, 9, 5, 4, 1};

        int i = 0;
        StringBuilder sb = new StringBuilder();
        for( i=0; n!= 0; i++){
            while(n >= value[i]){
                n -=aa value[i];
                sb.append(symbol[i]);
            }
        }

        return sb.toString();
    }
}
```

```java
public class Solution {
    public String intToRoman(int num) {
        String[] symbol = new String[]{"M","CM","D","CD","C","XC"
,"L","XL","X","IX","V","IV","I"};
        int[] value = new int[]{1000,900,500,400, 100, 90, 50, 40
, 10, 9, 5, 4, 1};

        int k = 0;
        StringBuilder sb = new StringBuilder();
        while(num >= 0 && k < value.length){
            while(num >= value[k]){
                num -= value[k];
                sb.append(symbol[k]);
            }
            k++;
        }

        return sb.toString();
    }
}
```

13. Roman to Integer

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

```java
public class Solution {
    public int romanToInt(String s) {
        Map<String, Integer> roman= new HashMap<>();
        roman.put("M", 1000);
        roman.put("CM", 900);
        roman.put("D", 500);
        roman.put("CD", 400);
        roman.put("C", 100);
        roman.put("XC", 90);
        roman.put("L", 50);
        roman.put("XL", 40);
        roman.put("X", 10);
        roman.put("IX", 9);
        roman.put("V", 5);
        roman.put("IV", 4);
        roman.put("I", 1);

        int result = 0;
        for(int i=0; i< s.length();i++){
            String w = s.substring(i, i<= s.length()-2 ? i+ 2 :
i+1);
            if(roman.containsKey(w)){
                result += roman.get(w);
                if(w.length() == 2)i++;
            }else{
                result += roman.get(s.substring(i, i+1));
            }
        }

        return result;

    }
}
```

```java
public class Solution {
    public int romanToInt(String s) {
        Map<String, Integer> roman= new HashMap<>();
        roman.put("M", 1000);
        roman.put("CM", 900);
        roman.put("D", 500);
        roman.put("CD", 400);
        roman.put("C", 100);
        roman.put("XC", 90);
        roman.put("L", 50);
        roman.put("XL", 40);
        roman.put("X", 10);
        roman.put("IX", 9);
        roman.put("V", 5);
        roman.put("IV", 4);
        roman.put("I", 1);
        int res = 0;
        for(int i=0; i<s.length(); i++){
            String key = s.substring(i,i+1);
            if(i+2 <=s.length()
                &&roman.containsKey(s.substring(i, i+2))){
                key = s.substring(i, i+2);
                i++;
            }
            res += roman.get(key);
        }

        return res;

    }
}
```

14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

```
public class Solution {
    public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) {
            return "";
        }

        if (strs.length == 1) {
            return strs[0];
        }

        int lengthOfCommanPrefix = 0;
        while(true) {
            for(int i=0;i<strs.length;i++) {
                if (strs[i] == null) {
                    return "";
                }

                if (lengthOfCommanPrefix >= strs[i].length()){
                    return strs[i];
                }


                if (i > 0 && (strs[i].charAt(lengthOfCommanPrefi
x) !=  strs[i-1].charAt(lengthOfCommanPrefix))) {
                    return strs[i].substring(0, lengthOfCommanPr
efix);
                }
            }
            lengthOfCommanPrefix ++;
        }

    }
}
```

15. 3Sum

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: The solution set must not contain duplicate triplets.

For example, given array S = [-1, 0, 1, 2, -1, -4],

A solution set is:

```
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

Pay attention to avoid duplicates:

> if nums[i] == nums[i-1], which means if there is a triplet [i-1, l, r] == 0, you need to skip i, same applies to l, l-1 and r, r+1

```java
public class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        if(nums == null || nums.length == 0) return res;

        Arrays.sort(nums);
        for(int i=0; i< nums.length -2; i++){
            if( i > 0 && nums[i] == nums[i-1]) continue;

            int l = i+1;
            int r = nums.length-1;
            while(l < r){
                int sum = nums[i] + nums[l] + nums[r];
                if(sum ==0){
                    List<Integer> t = new ArrayList<>();
                    t.add(nums[i]);
                    t.add(nums[l]);
                    t.add(nums[r]);
                    res.add(t);
                    l++;
                    r--;
                    while(l < r && nums[l] == nums[l-1]) l++;
                    while(l < r && nums[r] == nums[r+1]) r--;
                }else if(sum < 0){
                    l++;
                }else{
                    r--;
                }
            }
        }
        return res;
    }
}
```

16. 3Sum Closest

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

```
    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 class Solution {
    public int threeSumClosest(int[] nums, int target) {
        int minimalDist = Integer.MAX_VALUE;
        int res = 0;
        Arrays.sort(nums);

        for(int i=0;i<nums.length-2; i++){
            int l = i+1;
            int r = nums.length-1;
            while(l < r){
                int sum = nums[i] + nums[l] + nums[r];
                if(sum == target) return target;
                else if(sum > target){//pay attention how target
 and sum is handled here.
                    if((sum - target) < minimalDist){
                        minimalDist = sum - target;
                        res = sum;
                    }
                    r--;
                }else{
                    if((target - sum) < minimalDist){

                        minimalDist = target - sum;
                        res = sum;
                    }
                    l++;
                }
            }
            while(i<nums.length -1 && nums[i] == nums[i+1]) i++;
        }

        return res;
    }

}
```

17. Letter Combinations of a Phone Number

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.

Input:Digit string "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

dfs

```
public class Solution {
    List<String> res = new ArrayList<>();
    String[] dict = new String[]{"","", "abc", "def", "ghi", "jk
l", "mno","pqrs","tuv", "wxyz"};
    public List<String> letterCombinations(String digits) {

        combination(digits, 0, "");

        return res;
    }

    private void combination(String digits, int k, String curren
t){
        if(k == digits.length()){
            if(!current.isEmpty())res.add(current);
            return;
        }
        String s = dict[digits.charAt(k)- '0'];
        for(Character c : s.toCharArray()){
            combination(digits, k+1, current+c);
        }
    }
}
```

18. 4Sum QuestionEditorial Solution My Submissions

Total Accepted: 80129 Total Submissions: 327682 Difficulty: Medium 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: The solution set must not contain duplicate quadruplets.

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]]

> the key point here is how to step over the duplicates.

```
public class Solution {
    // this question is about how to remove duplicates

    List<List<Integer>> res = new ArrayList<>();
    public List<List<Integer>> fourSum(int[] nums, int target) {

            if(nums == null || nums.length == 0) return res;

            Arrays.sort(nums);

            for(int i=0; i< nums.length -3; i++){
                if(i > 0 && nums[i] == nums[i-1]) continue;
                for(int j= i+1; j< nums.length -2; j++){
                    if(j > i+1 && nums[j] == nums[j-1]) continue
;
                    int l = j+1;
                    int r = nums.length-1;

                    while(l < r){
                        int sum = nums[i] + nums[j] + nums[l] +
nums[r];
                        if(sum == target){
                            List<Integer> list = new ArrayList<>
```

```
();
                            list.add(nums[i]);
                            list.add(nums[j]);
                            list.add(nums[l]);
                            list.add(nums[r]);
                            res.add(list);
                            l++;
                            r--;
                            while(l<r && nums[l] == nums[l-1]) l
++;
                            while(l<r && nums[r] == nums[r+1]) r
--;
                        }else if(sum < target){
                            l++;
                        }else{
                            r--;
                        }

                    }
                }
            }
            return res;
    }
}
`
```

20. Valid Parentheses

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

```
public class Solution {
    public boolean isValid(String s) {
            if (s == null || s.length() % 2 == 1) {
                return false;
            }

            Map<Character, Character> parenMap = new HashMap<Cha
racter, Character>();
            parenMap.put(')', '(');
            parenMap.put(']', '[');
            parenMap.put('}', '{');

            char [] parenChars = s.toCharArray();
            Stack<Character> currentParen = new Stack<Character>
();
            for (char parenChar : parenChars) {
                Character targetValue = parenMap.get(parenChar);
                if (targetValue == null) {
                    currentParen.push(parenChar);
                } else {
                    if (currentParen.isEmpty()) {
                        return false;
                    }

                    if (! currentParen.pop().equals(targetValue)
){
                        return false;
                    }
                }
            }

            if (currentParen.isEmpty()) {
                return true;
            }

            return false;
    }
}
```

```
public class Solution {
    public boolean isValid(String s) {
        Map<Character, Character> map = new HashMap<>();
        map.put(')', '(');
        map.put(']','[');
        map.put('}','{');

        Stack<Character> stack = new Stack<>();
        for(char ch : s.toCharArray()){
            if(ch == '(' || ch == '[' || ch == '{'){
                stack.push(ch);
            }else{
                char pair = map.get(ch);
                if(stack.isEmpty() || stack.peek() != pair) retu
rn false;
                else stack.pop();
            }
        }

        return stack.isEmpty();
    }
}
```

22. Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

```
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

```java
public class Solution {

    List<String> res = new ArrayList<>();
    public List<String> generateParenthesis(int n) {
        generate("", n, n);

        return res;
    }

    private void generate(String s, int left, int right){
        if(left == 0 && right == 0){
            res.add(s);
        }
        if(left < 0 || right < 0 || left > right ) return ;

        generate(s+'(', left-1, right);
        generate(s+')', left, right-1);

    }


}
```

```java
public class Solution {
    List<String> res = new ArrayList<>();

    public List<String> generateParenthesis(int n) {
        gen(n,n, new String());
        return res;
    }

    void gen(int l, int r, String cur){
        if(l > r ) return;
        if(l < 0 || r < 0) return;

        if(l==0 && r==0){
            res.add(cur);
            return;
        }

        gen(l-1, r, cur+"(");
        gen(l, r-1, cur+")");
    }
}
```

23. Merge k Sorted Lists

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

- using heap to traverse all nodes in lists

```
public class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        PriorityQueue<ListNode> heap = new PriorityQueue<>(10, n
ew Comparator<ListNode>(){
           @Override
           public int compare(ListNode n1, ListNode n2){
               return n1.val -n2.val;
           }
        });
        for(ListNode node : lists){
            if(node != null)
                heap.offer(node);
        }
        ListNode head = null;
        ListNode tail = null;
        while(heap.size() > 0){
            ListNode node = heap.poll();
            if(head == null){
                head = tail = node;
            }else{
                tail.next = node;
                tail = tail.next;
            }
            if(node.next != null){
                heap.offer(node.next);
            }
        }

        return head;
    }
}
```

- similiar to merge 2 sorted list, using merge sort.

24. Swap Nodes in Pairs

Given a linked list, swap every two adjacent nodes and return its head.

For example, Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode swapPairs(ListNode head) {
        ListNode dummy = new ListNode(-1);

        dummy.next = head;
        ListNode p = dummy;
        while(p.next != null && p.next.next != null){
            ListNode next = p.next;
            ListNode next2 = p.next.next;
            p.next = next2;
            next.next = next2.next;
            next2.next = next;

            p = next;
        }

        return dummy.next;
    }
}
```

26. Remove Duplicates from Sorted Array

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 nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

Related issue: 80. Remove Duplicates from Sorted Array II

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

> another version

```java
public class Solution {
    public int removeDuplicates(int[] nums) {
        if(nums == null) return 0;
        if(nums.length <=1) return nums.length;

        int l=0;
        for(int r =1; r< nums.length; r++){
            if(nums[r] != nums[l])
                nums[++l] = nums[r];
        }
        return l+1;
    }
}
```

27. Remove Element

Given an array and a value, remove all instances of that value in place and return the new length.

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

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example:

Given input array nums = [3,2,2,3], val = 3

Your function should return length = 2, with the first two elements of nums being 2.

two pointers, one static, one moving

```
public class Solution {
    public int removeElement(int[] nums, int val) {
        if(nums == null || nums.length == 0) return nums.length;
        int i = 0;
        while(i < nums.length && nums[i] != val) i++;
        int j = i+1;
        while(j < nums.length ){
            if(nums[j] != val) nums[i++] = nums[j++];
            else j++;
        }

        return i;
    }
}
```

31. Next Permutation

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

```
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1
```

1. find the first pair of index that n[i] < n[i+1], from the end of array.
2. now we are sure from i+1, to the end of array, is a descending sequence, otherwise, we didn't find the correct pair in the first step.
3. from i+1, find the largest index k, where n[k] > n[i],
4. swap k and i, the i+1 to end still is a descending order.
5. reverse i+1, to end.

```java
public class Solution {
    public void nextPermutation(int[] nums) {

        int k = -1;
        for(int i =0; i< nums.length-1; i++){
            if(nums[i] < nums[i+1]) k =i;
        }

        if(k == -1){
            reverse(nums, 0, nums.length-1);
            return;
        }

        int l = k+1;
        for(int i = k+1; i< nums.length; i++){
            if(nums[i] > nums[k]) l = i;
        }

        int tmp = nums[k];
        nums[k] = nums[l];
        nums[l] = tmp;
        reverse(nums, k+1, nums.length-1);
    }

    void reverse(int[] nums, int start, int end){
        while(start < end){
            int tmp = nums[start];
            nums[start] = nums[end];
            nums[end] = tmp;
            start++;
            end--;
        }
    }
}
```

32. Longest Valid Parentheses

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

`The char that makes the string invalid parentheses will cut the string into parts that either is valid parentheses or empty. use a stack to process the string, then the stack is empty, while met ')', calculate the size.`

```
public class Solution {
    public int longestValidParentheses(String s) {
        int invalidPos = -1;
        Stack<Integer> stack = new Stack<>();
        int max = 0;
        for(int i= 0; i< s.length(); i++){
            if(s.charAt(i) == '('){
                stack.push(i);
            }else{
                if(!stack.isEmpty()){
                    stack.pop();
                    int start = stack.isEmpty() ? invalidPos : s
tack.peek();
                    max = Math.max(i-start, max);
                }else{
                    invalidPos = i;
                }
            }
        }

        return max;
    }
}
```

33. Search in Rotated Sorted Array

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).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Related issue:81 Search in Rotated Sorted Array II

> ATTENTION 1: since mid can equal equal to left, so >=.
>
> ATTENTION2 : you need first focus on finding the portion of array that is ordered.

```
public class Solution {
    public int search(int[] nums, int target) {
        int left=0;
        int right = nums.length-1;
        while(left <= right){
            int mid = left + (right -left)/2;
            if(nums[mid] == target) return mid;
            if(nums[mid] >= nums[left]){ // cause mid can equals
 to left
                if(nums[left] <= target && target < nums[mid]){
                    right = mid-1;
                }else{
                    left = mid +1;
                }
            }else{
                if(nums[mid] <= target && target <= nums[right])
{
                    left = mid +1;
                }else{
                    right = mid -1;
                }
            }
        }
        return -1;

    }
}
```

34. Search for a Range

Given a sorted array of integers, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example, Given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4].

> requires 2 binary search,

```
public class Solution {
    public int[] searchRange(int[] nums, int target) {

        int[] res = new int[]{-1,-1};
        if(nums == null || nums.length == 0) return res;
        if(nums[0] > target || nums[nums.length-1] < target) ret
urn res;

        int l =0;
        int r = nums.length-1;// eventually l may reach nums.len
gth-1 by `l = mid +1`

        while(l < r){
            int mid = l + (r-l)/2;
            if(nums[mid] < target){
                l = mid+1;
            }else{
                r = mid;
            }
        }

        if(nums[l] != target) return res;
        res[0] = l;

        r = nums.length;// l may reach nums.length since the las
t number is the closing number,
```

```
        //so both r and l == nums.length, and r -1 is the border.

        while(l < r){
            int mid = l + (r-l)/2;
            if(nums[mid]  > target){
                r = mid;
            }else{
                l = mid +1;
            }
        }

        res[1] = r-1;

        return res;
    }
}
```

35. Search Insert Position

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.

```
[1,3,5,6], 5 → 2
[1,3,5,6], 2 → 1
[1,3,5,6], 7 → 4
[1,3,5,6], 0 → 0
```

Think about the left and right change carefully, why return left is alwasy right?

```java
public class Solution {
    public int searchInsert(int[] nums, int target) {
        int left = 0, right = nums.length-1;
        while(left <= right){
            int mid = left + (right-left)/2;
            if(nums[mid] == target) return mid;
            else if(nums[mid] > target){
                right = mid -1;
            }else{
                left = mid +1;
            }
        }

        return left;
    }
}
```

38. Count and Say

The count-and-say sequence is the sequence of integers beginning as follows: 1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.

11 is read off as "two 1s" or 21.

21 is read off as "one 2, then one 1" or 1211.

Given an integer n, generate the nth sequence.

Note: The sequence of integers will be represented as a string.

```
public class Solution {
    public String countAndSay(int n) {
        String res = "1";
        for(int i=1; i< n; i++){
            int count =1;
            StringBuilder sb = new StringBuilder();
            for(int j=1; j < res.length(); j++){
                if(res.charAt(j-1) == res.charAt(j)) count++;
                else{
                    sb.append(count);
                    sb.append(res.charAt(j-1));
                    count =1;
                }
            }
            sb.append(count);
            sb.append(res.charAt(res.length()-1));

            res = sb.toString();
        }

        return res;
    }
}
```

39. Combination Sum

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.

Note: All numbers (including target) will be positive integers.

The solution set must not contain duplicate combinations.

For example, given candidate set [2, 3, 6, 7] and target 7,

A solution set is:

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

Related issue Subset, Subset II, Combination Sum II

question to ask :

- all positive number.
- will the set contains duplicates ?

backtracking.

about this line

```
        dfs(num, i, result, sum+num[i], target);
```

cause the question mention any number can be used many times, so next iteration will need to include this number again, which start from i.

BUT it can not start from 0(i.e. replace i with 0), why ? cause this will cause duplicates,

[2,3,6,7] target is 7, if you start from 0 again, will result in [2,2,3] & [2,3,2(start from 0 again)], which is duplicate.

```
public class Solution {
    List<List<Integer>> res = new ArrayList<>();
    public List<List<Integer>> combinationSum(int[] num, int tar
get) {

        List<Integer> list = new ArrayList<>();
        Arrays.sort(num);
        dfs(num, 0, list, 0, target);

        return res;
    }

    private void dfs(int[] num, int start, List<Integer>result,
int sum, int target){
        if(sum == target){
            res.add(new ArrayList<Integer>(result));
            return;
        }
        for(int i=start; i < num.length;i++){
            if(i > start && num[i] == num[i-1]) continue; // if
the set doesn't contains duplicates, then this line won't be nee
ded.
            if(num[i] + sum > target) break ;

            result.add(num[i]);
            dfs(num, i, result, sum+num[i], target);
            result.remove(result.size() -1);
        }

    }
}
```

40. Combination Sum II

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.

The solution set must not contain duplicate combinations.

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

A solution set is:

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

Related issue Subset, Subset II, Combination Sum

> one number can be used only once(assuming the set includes duplicates), this is why each generation of dfs start with i+1.

```
public class Solution {
    List<List<Integer>> res = new ArrayList<>();
    public List<List<Integer>> combinationSum2(int[] num, int ta
rget) {
        if(num == null || num.length == 0) return res;

        Arrays.sort(num);
        List<Integer> list = new ArrayList<>();
        dfs(num, 0, list, 0, target);

        return res;
    }

    private void dfs(int[] num, int start, List<Integer> list, i
nt sum, int target){
        if(sum == target){
            res.add(new ArrayList<Integer>(list));
            return;
        }

        for(int i=start; i< num.length; i++){
            if(i > start && num[i] == num[i-1]) continue;
            if(num[i] + sum <= target){
                list.add(num[i]);
                dfs(num, i+1, list, sum+num[i], target);
                list.remove(list.size()-1);
            }
        }
    }
}
```

41. First Missing Positive

Given an unsorted integer array, find the first missing positive integer.

For example, Given [1,2,0] return 3,

and [3,4,-1,1] return 2.

Your algorithm should run in O(n) time and uses constant space.

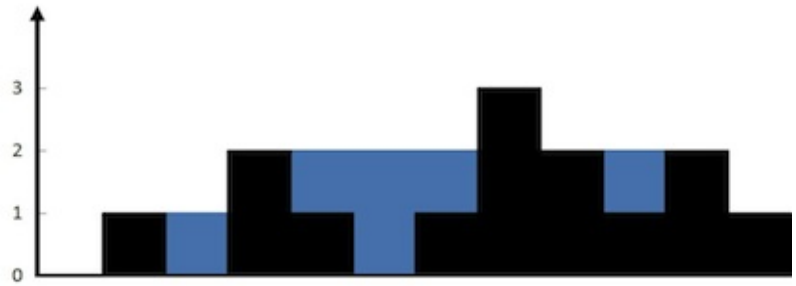
technical speaking, it is a hash table question. but the hash key here is the array index. put the corresponding number in their niche. then swipe the array from the beginning, first not match is what we need,

```
public class Solution {
    public int firstMissingPositive(int[] nums) {
        for(int i=0; i< nums.length; i++){
           if( nums[i] != i+1 && nums[i] > 0 && nums[i] <=nums.l
ength && nums[i] != nums[nums[i]-1]){
               int t = nums[nums[i] -1]; // cache this number fi
rst, other wise won't work.
               nums[nums[i] -1] = nums[i];
               nums[i] = t;
               i--;
           }
        }

        for(int i=0; i< nums.length;i++){
            if(nums[i] != i+1) return i+1;
        }
        return nums.length+1;
    }
}
```

42. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example, Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.



The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. Thanks Marcos for contributing this image!

```java
public class Solution {
    public int trap(int[] height) {

        int peak = 0;
        int max = 0;
        for(int i=0; i< height.length; i++){
            if(height[i] > max){
                max = height[i];
                peak = i;
            }
        }
        int amount  =0;
        int leftMax =0;
        for(int i=0; i<peak; i++){
            if(height[i] >= leftMax){
                leftMax = height[i];
            }else{
                amount += (leftMax - height[i]);
            }
        }
        int rightMax =0;
        for(int i= height.length-1; i> peak; i--){
            if(height[i] >= rightMax){
                rightMax = height[i];
            }else{
                amount += (rightMax - height[i]);
            }
        }

        return amount;
    }
}
```

43. Multiply Strings

Given two numbers represented as strings, return multiplication of the numbers as a string.

Note: The numbers can be arbitrarily large and are non-negative.

Converting the input string to integer is NOT allowed.

You should NOT use internal library such as BigInteger.

one solution is to use add, for each i in num1, do an addition to num2, then build the result as add two string.

second is to notice the property of multiply, which is each postions result is added up by using different postion combination in num1 and num2, which is:

```
  num3[i+j+1] = num1[i] * num2[j] + carry + num3[i+j+1]
```

> why need to add it self ? cause in any [i, j] round, num3[i+j+1] may already been calculated, such as, [i-1, j+1], in this way, there is already a value.

```
public class Solution {
    public String multiply(String num1, String num2) {

        int l1 = num1.length();
        int l2 = num2.length();
        int l3 = l1 + l2;
        int[] num3 = new int[l3];

        for(int i= l1-1; i>=0; i--){
            int carry = 0;
            int j =l2-1;
            for(; j>=0; j--){
                int res = carry + num3[i+j+1] +
                            (num1.charAt(i) - '0') * (num2.charA
t(j) - '0');
                num3[i+j+1] = res%10;
```

```
                carry = res /10;
            }

            num3[i+j+1] = carry; // the carry need to carry to n
ext position. since j is now -1, so nums3[i+j+1] means nums[i+j+
1 - 1]. from backward it is the next position need the carry. SO
 ASSIGNED, NOT APPEND.
        }

        int i=0;
        for(; i< l3; i++){
            if(num3[i] != 0) break;
        }

        if(i == l3)  return "0";
        StringBuilder sb = new StringBuilder();
        while(i < l3){
            sb.append(num3[i++]);
        }

        return sb.toString();

    }
}
```

45. Jump Game II

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

For example: Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

Related Issue : Jump Game

Note: You can assume that you can always reach the last index.

```
public class Solution {
    public int jump(int[] nums) {
        if(nums.length <=1) return 0;
        int start =0, end =0;
        int count =0;

        while(end < nums.length){
            count++;
            int maxEnd = 0;
            for(int i=start; i<=end; i++){
                maxEnd = Math.max(maxEnd, i+nums[i]);
            }
            if(maxEnd >= nums.length -1) return count;
            start = end+1;
            end = maxEnd;
        }

        return count;
    }
}
```

46. Permutations

Given a collection of distinct numbers, return all possible permutations.

For example, [1,2,3] have the following permutations:

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

Related issue Subset, Subset II, Permutations II, Combination Sum II Combinations

put each number in the beginning of the array, then process the begging+1 as a sub issue.

backtracking.

```java
public class Solution {
    List<List<Integer>> res = new ArrayList<>();
    List<Integer> list = new ArrayList<>();
    public List<List<Integer>> permute(int[] nums) {
        permute_(nums, 0, nums.length-1);

        return res;
    }

    private void permute_(int[] nums, int start, int end){
        if(start == end){
            List<Integer> l = new ArrayList<>();
            for(int v : nums) l.add(v);
            res.add(l);
            return;
        }

        for(int i=start; i<=end; i++){
            swap(nums, i, start);
            permute_(nums, start+1, end);
            swap(nums, i, start);
        }
    }

    private void swap(int[] nums, int l, int r){
        if(r == l) return;
        int t = nums[l];
        nums[l] = nums[r];
        nums[r] = t;
    }
}
```

47. Permutations II

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example, [1,1,2] have the following unique permutations:

```
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]
```

> Same as Permutations, except that when you try to swap, you need to make sure you are not swaping a number already processed, if you do, which means there will be duplicate set.

```
public class Solution {
    List<List<Integer>> res = new ArrayList<>();
    List<Integer> list = new ArrayList<>();

    public List<List<Integer>> permuteUnique(int[] nums) {
        if(nums == null || nums.length == 0) return res;

        permute_(nums, 0, nums.length -1);

        return res;
    }

    private void permute_(int[] nums, int start, int end){
        if(start == end){
            List<Integer> t = new ArrayList<>();
            for(int v : nums) t.add(v);
            res.add(t);
            return;
        }
        for(int i= start; i<=end; i++){
```

```
            if(!alowSwap(nums, start, i)) continue;
            swap(nums, i, start);
            permute_(nums, start+1, end);
            swap(nums, i, start);
        }
    }
    private boolean alowSwap(int[] nums, int start, int end){
        int val = nums[end];
        for(int i= start; i<end; i++){
            if(val == nums[i]) return false;
        }
        return true;
    }
    private void swap(int[] nums, int l, int r){
        if(r==l) return ;
        int t = nums[l];
        nums[l] = nums[r];
        nums[r] = t;
    }
}
```

48. Rotate Image

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up: Could you do this in-place?

by using simple example find the rotate index corresponding to 4 location.

```
matrix[i][j] <--- matrix[n-1-j][i];
matrix[n-1-j][i] <--- matrix[n-1-i][n-1-j];
matrix[n-1-i][n-1-j] <--- matrix[j][n-1-i];
matrix[j][n-1-i] <--- matrix[i][j];
```

Then writing the code is easier.

```
public class Solution {
    public void rotate(int[][] matrix) {
        if(matrix == null || matrix.length == 1) return;
        int n = matrix.length;
        for(int i=0; i<= n/2; i++){
            for(int j = i; j< n-1-i; j++){
                int tmp = matrix[i][j];
                matrix[i][j] = matrix[n-1-j][i];
                matrix[n-1-j][i] = matrix[n-1-i][n-1-j];
                matrix[n-1-i][n-1-j] = matrix[j][n-1-i];
                matrix[j][n-1-i] = tmp;
            }
        }

    }
}
```

49. Group Anagrams

Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"], Return:

```
[
  ["ate", "eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

Note: All inputs will be in lower-case.

There are question need to ask during interview : all lower case ? mixed ?

```
public class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> map = new HashMap<>();
        for(String word : strs){

            char[] c = word.toCharArray();
            Arrays.sort(c);
            String hash = new String(c);
            if(map.containsKey(hash)){
                map.get(hash).add(word);
            }else{
                List<String> l = new ArrayList<>();
                l.add(word);
                map.put(hash, l);
            }
        }

        return new ArrayList<List<String>>(map.values());
    }
}
```

```
//sort a string.

            char[] c = word.toCharArray();
            Arrays.sort(c);
            String hash = new String(c);
```

```
public class Solution {
    Map<String, List<String>> map = new HashMap<>();

    public List<List<String>> groupAnagrams(String[] strs) {
        for(String s : strs){
            char[] c = s.toCharArray();
            Arrays.sort(c);
            String key = new String(c);
            if(!map.containsKey(key)){
                map.put(key, new ArrayList<>());
            }
            map.get(key).add(s);
        }

        return new ArrayList<List<String>>(map.values());
    }
}
```

50. Pow(x, n)

Implement pow(x, n).

```
public class Solution {
    public double myPow(double x, int n) {
        if(n < 0){
            return 1/ my_(x, -n);
        }else{
            return my_(x, n);
        }
    }

    private double my_(double x, int n){
        if(n == 0) return 1;
        double v = my_(x, n/2);
        if(n%2 == 0){
            return v*v;
        }else{
            return v*v*x;
        }
    }
}
```

> if you check `n%2 == 1` it will fail on some extreme case for example. x = 2.00000 & n = -2147483648

51. N-Queens

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.

Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

For example, There exist two distinct solutions to the 4-queens puzzle:

```
[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

Related issue: 52 N-Queens II backtacking

```
public class Solution {

    List<List<String>> res = new ArrayList<>();
    public List<List<String>> solveNQueens(int n) {
        char[][] board = new char[n][n];
        for(int i=0; i< n;i++){
            for(int j=0; j< n; j++){
                board[i][j] = '.';
            }
        }

        bt(board, 0, n);
```

```
        return res;
    }

    private void bt(char[][] board, int row, int n){
        if(row == n){
            //save into database.
            List<String> t = new ArrayList<>();
            for(char[] s : board){
                t.add(new String(s));
            }
            res.add(t);
        }

        for(int col=0; col < n; col++){
            if(isValid(board, row, col)){ // if is not valid, th
en there is no solution added.
                board[row][col] = 'Q';
                bt(board, row+1, n);
                board[row][col] = '.';
            }
        }

    }

    private boolean isValid(char[][] board, int row, int col){
        for(int i= row-1; i >=0; i--){
            if(board[i][col] == 'Q' )return false;
        }
        for(int i=row-1, j=col-1; i >=0 && j>=0; i--,j--){
            if(board[i][j] == 'Q') return false;
        }


        for(int i=row-1, j=col+1; i>=0 && j<board.length; i--,j+
+){
            if(board[i][j] == 'Q') return false;
        }

        return true;
    }
```

```
}
```

52. N-Queens II

Follow up for N-Queens problem.

Now, instead outputting board configurations, return the total number of distinct solution

```
public class Solution {
    int count =0;
    public int totalNQueens(int n) {
        char[][] board = new char[n][n];
        for(int i=0; i< n;i++){
            for(int j=0; j< n; j++){
                board[i][j] = '.';
            }
        }

        bt(board, 0, n);
        return count;
    }


    private void bt(char[][] board, int row, int n){
        if(row == n){
            count++;
            return;
        }

        for(int col=0; col < n; col++){
            if(isValid(board, row, col)){ // if is not valid, th
en there is no solution added.
                board[row][col] = 'Q';
                bt(board, row+1, n);
                board[row][col] = '.';
            }
        }

    }
```

```
    private boolean isValid(char[][] board, int row, int col){
        for(int i= row-1; i >=0; i--){
            if(board[i][col] == 'Q' )return false;
        }
        for(int i=row-1, j=col-1; i >=0 && j>=0; i--,j--){
            if(board[i][j] == 'Q') return false;
        }


        for(int i=row-1, j=col+1; i>=0 && j<board.length; i--,j+
+){
            if(board[i][j] == 'Q') return false;
        }

        return true;
    }
}
```

to improve solution, you don't need to save the entire board to get the result, you only need to remember the index of queens in each row.

```
public class Solution {
    int count =0;
    public int totalNQueens(int n) {
        int[] board = new int[n];
        for(int i=0; i< n;i++){
            board[i] = -1;
        }

        bt(board, 0, n);
        return count;
    }


    private void bt(int[] board, int row, int n){
        if(row == n){
            count++;
            return;
```

```
        }

        for(int col=0; col < n; col++){
            if(isValid(board, row, col)){ // if is not valid, th
en there is no solution added.
                board[row] = col;
                bt(board, row+1, n);
                board[row] = -1;
            }
        }

    }

    private boolean isValid(int[] board, int row, int col){
        for(int i= 0; i <row; i++){
            if(board[i] == col || Math.abs(row -i) == Math.abs(c
ol - board[i]) ) return false;
        }

        return true;
    }
}
```

53. Maximum Subarray

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array [−2,1,−3,4,−1,2,1,−5,4],

the contiguous subarray [4,−1,2,1] has the largest sum = 6.

keep two record, max value so far, and current sum. for each number, add to sum, if sum > max, then set the max-so-far as sum, if sum <0, discard all numbers visited, cause it is not used for the following numbers. but the max-so-far is kept.

Related issue: 152 Maximum Product Subarray

classic dp solution

```
public class Solution {
    public int maxSubArray(int[] nums) {
        int[] dp = new int[nums.length];
        dp[0] = nums[0];
        for(int i=1; i< nums.length;i++){
            if(dp[i-1] <=0 && nums[i] >=0){
              dp[i] = nums[i];
              continue;
            }
            int tmp = dp[i-1] + nums[i];
            if(tmp >= 0) dp[i] = tmp;
            else dp[i] = nums[i];


        }
        int max = dp[0];
        for(int v: dp){
            if(v > max) max = v;
        }
        return max;
    }
}
```

to improve above solution, notice that only dp[i-1] and dp[i] is used,.

notice when i ==0, is handles specically.

```
public class Solution {
    public int maxSubArray(int[] nums) {
        int sum =0, max = 0;
        for(int i=0; i< nums.length; i++){
            sum += nums[i];
            if(sum > max || i == 0) max = sum;
            if(sum < 0) sum =0; // if the sum goes below zero, r
eset the sum to zero as it start counting from next item.
        }

        return max;
    }
}
```

54. Spiral Matrix

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

For example, Given the following matrix:

```
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
```

You should return [1,2,3,6,9,8,7,4,5].

Related issue Spiral Matrix II

You surely can used 'peel-onion' to do this, first your will know the peeling stop at n/2, m/2, then another round when there is a row /col left. a better solution is to set stop coordinates.

```java
public class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> res = new ArrayList<>();
        if(matrix == null || matrix.length == 0 ||matrix[0].leng
th == 0) return res;

        int m = matrix.length;
        int n = matrix[0].length;
        int p1 = 0, q1 = 0;
        int p2 = m-1, q2 = n-1;

        while(true){
            for(int k= q1; k<=q2; k++){
                res.add(matrix[p1][k]);
            }
            if(++p1 > p2) break;

            for(int k=p1; k <=p2;k++){
                res.add(matrix[k][q2]);
            }
            if(--q2 < q1) break;

            for(int k=q2; k>=q1;k--){
                res.add(matrix[p2][k]);
            }
            if(--p2 < p1) break;

            for(int k=p2; k>=p1; k--){
                res.add(matrix[k][q1]);
            }
            if(++q1 > q2) break;
        }
        return res;
    }
}
```

55. Jump Game

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:

```
A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false.
```

Related Issue Jump Game II

after entering the loop, you need first check if you can initiate the jump, then update the max distance to jump.

```
public class Solution {
    public boolean canJump(int[] nums) {
        int max = 0;
        for(int i=0; i<nums.length; i++){
            // [0,1] to consider
            if(max >= nums.length-1 || i > max)  break;
            max = Math.max(i+nums[i], max);
        }
        return max >= nums.length-1 ? true : false;
    }
}
```

56. Merge Intervals

Given a collection of intervals, merge all overlapping intervals.

For example, Given [1,3],[2,6],[8,10],[15,18], return [1,6],[8,10],[15,18].

Sort the intervals by its start time, merge two if not first.end < second.start.

```
public class Solution {
    public List<Interval> merge(List<Interval> intervals) {
        if(intervals.size() <=1 ) return intervals;
        ArrayList<Interval> result = new ArrayList<>();

        Collections.sort(intervals, new Comparator<Interval>(){
           @Override
            public int compare(Interval i1, Interval i2){
                if(i1.start == i2.start){
                    return i1.end - i2.end;
                }else{
                    return i1.start - i2.start;
                }
            }
        });


        result.add(intervals.get(0));
        for(int i=1; i< intervals.size(); i++){
            Interval last = result.get(result.size()-1);
            Interval cur = intervals.get(i);
            if(last.end < cur.start){
                result.add(cur);
            }else{
                last.end = Math.max(last.end, cur.end);
            }
        }
        return result;
    }
}
```

57. Insert Interval

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1: Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].

Example 2: Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] in as [1,2], [3,10],[12,16].

This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10].

```
  public class Solution {
      public List<Interval> insert(List<Interval> intervals, Int
erval newInterval) {
          if(intervals.size() ==0 ){
              intervals.add(newInterval);
              return intervals;
          }
          ArrayList<Interval> result = new ArrayList<>();
          int i = 0;
          for(;i<intervals.size();i++){
              Interval cur = intervals.get(i);
              if(cur.end < newInterval.start){
                  result.add(cur);
              }else{
                  break;
              }
          }
          //merge
          for(; i<intervals.size(); i++){
              Interval cur = intervals.get(i);
              if(newInterval.end < cur.start){
                  break;
              }else{
                  newInterval.start = Math.min(newInterval.start
, cur.start);
                  newInterval.end = Math.max(newInterval.end, cu
r.end);
              }
          }
          result.add(newInterval);
          for(;i<intervals.size(); i++){
              result.add(intervals.get(i));
          }

          return result;
      }
  }
```

```
 /**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public List<Interval> insert(List<Interval> intervals, Inter
val newInterval) {
        List<Interval> res = new ArrayList<>();
        if(intervals == null) return res;
        int i =0;
        for(; i< intervals.size(); i++){
            if(newInterval.end < intervals.get(i).start){
                res.add(newInterval);
                break;
            }else if(intervals.get(i).end < newInterval.start){
                res.add(intervals.get(i));
            }else{
                newInterval.start = Math.min(intervals.get(i).st
art, newInterval.start);
                newInterval.end = Math.max(intervals.get(i).end,
 newInterval.end);
            }
        }
        if(i == intervals.size()) res.add(newInterval);
        while(i < intervals.size()) res.add(intervals.get(i++));
        return res;
    }
}
```

58. Length of Last Word

Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example,

Given s = "Hello World",

return 5.

```
public class Solution {
    public int lengthOfLastWord(String s) {
        if(s == null || s.length() == 0) return 0;
        int p = s.length()-1;
        while(p >=0 && s.charAt(p) == ' ') p--;
        if(p < 0) return 0;
        int res = 0;
        while( p >=0 && s.charAt(p) != ' '){
            p--;
            res++;
        }
        return res;
    }
}
```

59. Spiral Matrix II

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

For example, Given n = 3,

You should return the following matrix:

```
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

Related issue Spiral Matrix

```java
public class Solution {
    public int[][] generateMatrix(int n) {
        int mid = n/2;
        int val = 1;
        int[][] res = new int[n][n];

        for(int i=0; i<mid; i++){
            int last = n-i-1;

            for(int k = i; k<last; k++){
                res[i][k] = val++;
            }
            for(int k = i; k<last; k++){
                res[k][last] = val++;
            }
            for(int k =last; k>i;k--){
                res[last][k] = val++;
            }
            for(int k=last; k>i;k--){
                res[k][i]= val++;
            }
        }
        if(n%2 == 1) res[n/2][n/2] = val;
        return res;
    }
}
```

61. Rotate List

Given a list, rotate the list to the right by k places, where k is non-negative.

For example: Given `1->2->3->4->5->NULL and k = 2,`

return `4->5->1->2->3->NULL.`

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if(head == null || head.next == null || k == 0) return h
ead;

        int len = 0;
        ListNode tail = head;
        while(tail.next != null){
            len++;
            tail = tail.next;
        }
        len++;

        k = len - k % len;
        if(k == len) return head;
        ListNode rTail = head;
        while(k-- > 1){
            rTail = rTail.next;
        }
        ListNode newHead = rTail.next;
        rTail.next = null;
        tail.next = head;

        return newHead;


    }
}
```

There is another smart yet simpler solution. after find the last node, link last node to the head, then move forward len - k%len step, you will reach the new tail, break the cycle from there, then you find the new head.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if(head == null || head.next == null || k == 0) return h
ead;

        int len = 0;
        ListNode tail = head;
        while(tail.next != null){
            len++;
            tail = tail.next;
        }
        len++;
        k = len - k % len;
        if(k == len) return head;
        tail.next = head;

        while(k-- >0){
            tail = tail.next;
        }
        head =tail.next;
        tail.next = null;
        return head;

    }
}
```

62. Unique Paths

A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Above is a 3 x 7 grid. How many possible unique paths are there?

Note: m and n will be at most 100.

O(mn), space(mn)

```
public class Solution {
    public int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];
        for(int i=0; i<m;i++){
            dp[i][0] =1;
        }
        for(int i=0; i<n;i++){
            dp[0][i] =1;
        }

        for(int i=1; i<m; i++){
            for(int j=1;j<n;j++){
                dp[i][j] = dp[i-1][j] + dp[i][j-1];
            }
        }
        return dp[m-1][n-1];
    }
}
```

improve o(mn), space(2n)

```
public class Solution {
    public int uniquePaths(int m, int n) {
        int[][] dp = new int[2][n];
        for(int i=0;i<n;i++){
            dp[0][i] = 1;
            dp[1][i] = 1;
        }
        int current =0;

        for(int i=1;i<m;i++){
            int next  = 1- current;
            for(int j=1; j<n;j++){
                dp[next][j] = dp[current][j] + dp[next][j-1];
            }
            current = 1- current; //current will be next.
        }

        return dp[current][n-1];// can use a simple example to f
igure out whether is current or 1-current.

    }
}
```

even better solution

```
public class Solution {
    public int uniquePaths(int m, int n) {
        int[] dp = new int[n];
        for(int i =0; i< n; i++) dp[i] = 1;

        for(int i=1; i < m;i++)
            for(int j=1;j<n;j++)
                dp[j] += dp[j-1];
        return dp[n-1];

    }
}
```

63. Unique Paths II

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example, There is one obstacle in the middle of a 3x3 grid as illustrated below.

```
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
```

The total number of unique paths is 2.

```
public class Solution {
    public int uniquePathsWithObstacles(int[][] grid) {
        if(grid == null || grid.length == 0 || grid[0].length ==
0) return 0;
        int m = grid.length;
        int n = grid[0].length;
        int[][] dp= new int[m][n];
        for(int k=0; k<n;k++){
            if(grid[0][k] == 0) dp[0][k] = 1;
            else break;
        }
        for(int k=0; k<m;k++){
            if(grid[k][0] == 0) dp[k][0] = 1;
            else break;
        }

        for(int i=1; i< m;i++){
            for(int j=1;j<n;j++){
                if(grid[i][j] == 1){
                    dp[i][j] = 0;
                }else{
                    dp[i][j] = dp[i-1][j] + dp[i][j-1];
                }
            }
        }

        return dp[m-1][n-1];
    }
}
```

64. Minimum Path Sum

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.

similar to 62 Unique Path

```
public class Solution {
    public int minPathSum(int[][] grid) {
        if(grid == null || grid.length == 0 || grid[0].length ==
0) return 0;

        int[][] sum = new int[grid.length][grid[0].length];

        sum[0][0] = grid[0][0];

        for(int i=1; i< grid.length; i++){
            sum[i][0] = sum[i-1][0] + grid[i][0];
        }
        for(int j=1; j<grid[0].length; j++){
            sum[0][j] = sum[0][j-1] + grid[0][j];
        }

        for(int i=1; i<grid.length; i++){
            for(int j=1; j<grid[0].length; j++){
                sum[i][j] = Math.min(sum[i-1][j], sum[i][j-1]) +
 grid[i][j];
            }
        }
        return sum[grid.length-1][grid[0].length-1];
    }
}
```

you can also improve this by using only one array, which sum[j-1] is current line's result. corresponding to sum[i][j-1], and sum[j] is sum[i-1][j];

67. Add Binary

Given two binary strings, return their sum (also a binary string).

For example,

a = "11"

b = "1"

Return "100".

```
public class Solution {
    public String addBinary(String a, String b) {
        if(a == null && b == null ) return null;
        if(a == null || b == null) return a == null ? b :a;
        StringBuilder sb = new StringBuilder();

        int p = a.length() - 1;
        int q = b.length() - 1;
        int carry = 0;

        while(p >=0 || q >=0){
            int i1 = p >= 0 ? a.charAt(p--) - '0' : 0;
            int i2 = q >= 0 ? b.charAt(q--) - '0' : 0;

            sb.append((i1+i2+carry)%2);
            carry = (i1+i2+carry)/2;
        }

        if(carry != 0){
            sb.append(carry);
        }

        return sb.reverse().toString();

    }
}
```

> simpler code

```
public class Solution {
    public String addBinary(String a, String b) {
        int carry = 0;
        int al = a.length()-1, bl = b.length()-1;
        StringBuilder sb = new StringBuilder();
        while(al >=0 || bl>=0 || carry > 0){
            int x = al >= 0 ? a.charAt(al--) - '0' : 0;
            int y = bl >= 0 ? b.charAt(bl--) - '0' : 0;

            sb.append((x+y+carry)%2);
            carry = (x+y+carry)/2;
        }

        return sb.reverse().toString();
    }
}
```

68. Text Justification

Given an array of words and a length L, format the text such that each line has exactly L characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly L characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

For example,

```
words: ["This", "is", "an", "example", "of", "text", "justificat
ion."]
L: 16.

Return the formatted lines as:
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

Note: Each word is guaranteed not to exceed L in length.

- if there is only one word in the line , it should be left justified.
- if it is the last line, it should be left justified too.
- if the spaces cannot be evenly distributed between words, left words[Not just one words] shoule contains more space than the right ones. i.e. the extras space should be event distributed from the left.

```
public class Solution {
    List<String > res = new ArrayList<>();
    public List<String> fullJustify(String[] words, int maxWidth)
 {
        int count =words[0].length();
        int start = 0;
        for(int i=1; i<= words.length; i++){
            if( i == words.length){

                justify(words, start, i, maxWidth);
            }else if( count + words[i].length() + 1<= maxWidth )
{
                count += words[i].length() +1;
                continue;
            }else{
                justify(words, start, i, maxWidth);// left inclu
sive, right exclusive;
                start = i;
                count = words[i].length();
            }
        }

        return res;
    }

    void justify(String[] words, int start, int end, int L){

        char[] array = new char[L];
        int k = 0;

        if(end == start +1){
            String w = words[start];
            while(k< L){
                array[k] = k< w.length() ? w.charAt(k) : ' ';
                k++;
            }
            res.add(new String(array));
            return;
        }
```

```
        if(end == words.length){
            for(int i =start; i< end; i++){
                String w = words[i];
                int j =0;
                while(j < w.length()){
                    array[k++] = w.charAt(j++);
                }
                array[k++] = ' ';
            }
            while(k < L) array[k++] = ' ';
            res.add(new String(array));
            return;
        }

        int count = 0;

        for(int i=start; i<end; i++){
            count += words[i].length();
        }
        int spaces = (L - count)/(end -start -1);
        int extras = (L - count)%(end -start -1);

        for(int i= start; i< end; i++){
            String w = words[i];
            int j = 0;
            while(j < w.length()){
                array[k++] = w.charAt(j++);
            }

            if(i == end -1) break;

            j = 0;
            while(j++ < spaces){
                array[k++] = ' ';
            }
            if(extras-- > 0) {
                array[k++] = ' ';
            }
        }
```

```
        res.add(new String(array));

    }
}
```

```
public class Solution {
    public List<String> fullJustify(String[] words, int maxWidth)
 {
        int len = 0;
        List<String> res = new ArrayList<>();
        List<String> list = new ArrayList<>();

        for(int i=0; i<words.length; i++){
            if(len + words[i].length() + list.size() <= maxWidth
){
                list.add(words[i]);
                len += words[i].length();
            }else{
                justify(list, res, len, maxWidth);
                list.clear();
                list.add(words[i]);
                len = words[i].length();
            }
        }

        if(!list.isEmpty()){
            leftJustify(list, res, len, maxWidth);
        }

        return res;
    }
    void leftJustify(List<String> list, List<String> res, int le
n, int maxWidth){
        char[] chars = new char[maxWidth];
        char[] first = list.get(0).toCharArray();
        System.arraycopy(first, 0, chars, 0, first.length);
        if( list.size() == 1){
```

```
            Arrays.fill(chars, first.length, chars.length, ' ');
            res.add(new String(chars));
            return;
        }else{
            int dis = 1;
            int stop = first.length;
            for(int i=1; i< list.size(); i++){
                Arrays.fill(chars, stop, stop+1, ' ');
                stop++;
                char[] word = list.get(i).toCharArray();
                System.arraycopy(word, 0, chars, stop, word.leng
th);
                stop += word.length;
            }
            if(stop < chars.length){
                Arrays.fill(chars, stop, chars.length, ' ');
            }
            res.add(new String(chars));
        }
    }

    void justify(List<String> list, List<String> res, int len, i
nt maxWidth){
        char[] chars = new char[maxWidth];

        char[] first = list.get(0).toCharArray();
        System.arraycopy(first, 0, chars, 0, first.length);


        if (list.size() == 1) {
            Arrays.fill(chars, first.length, chars.length, ' ');
            res.add(new String(chars));
            return;
        } else {
            int dis = (maxWidth - len) / (list.size() - 1);
            int extra = list.size() > 2 ? (maxWidth - len) % (li
st.size() - 1) : 0;
            int stop = first.length;

            for (int i = 1; i < list.size(); i++) {
```

```
                Arrays.fill(chars, stop, stop + dis, ' ');
                stop += dis;
                if(extra > 0 ){
                    Arrays.fill(chars, stop, stop + 1, ' ');
                    stop += 1;
                    extra -= 1;
                }
                char[] word = list.get(i).toCharArray();
                System.arraycopy(word, 0, chars, stop, word.leng
th);
                stop += word.length;

            }
            res.add(new String(chars));
        }
        return;
    }
}
```

69. Sqrt(x)

Implement int sqrt(int x).

Compute and return the square root of x.

> Overflow when mid * mid, use divide instead.

```
public class Solution {
    public int mySqrt(int x) {
        if(x == 0 || x == 1) return x;
        int l =0;
        int r = x;
        while(l <= r){
            int mid = l +(r-l)/2;
            if( x/mid == mid ) return mid;
            else if( x/mid < mid){
                r = mid -1;
            }else{
                l = mid+1;
            }
        }
        return r;
    }
}
```

70. Climbing Stairs

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?

classic dp

```
public class Solution {
    public int climbStairs(int n) {
        // simple dp
        if(n <=2) return n > 0 ? n : 0;
        int[] dp = new int[n+1];
        dp[1] = 1;
        dp[2] = 2;
        for(int i=3; i<=n; i++){
            dp[i] = dp[i-1] + dp[i-2];
        }

        return dp[n];
    }
}
```

constant space.

```
public class Solution {
    public int climbStairs(int n) {
        // simple dp
        if(n <=2) return n > 0 ? n : 0;
        int dp1 = 1, dp2 = 2, res =0;
        for(int i=3; i<=n; i++){
            res = dp1+dp2;
            dp1 = dp2;
            dp2 = res;
        }

        return res;

    }
}
```

less variable

```
public class Solution {
    public int climbStairs(int n) {
        if(n <=2 ) return n;
        int prev2 = 1;
        int prev1 = 2;
        for(int i=3; i<=n; i++){
            prev1 = prev1 + prev2;
            prev2 = prev1 - prev2;
        }
        return prev1;
    }
}
```

71. Simplify Path

Given an absolute path for a file (Unix-style), simplify it.

For example, path = "/home/", => "/home" path = "/a/./b/../../c/", => "/c"

```
public class Solution {
    public String simplifyPath(String path) {
        String[] dirs = path.split("/");
        List<String> list = new LinkedList<>();
        for(String s : dirs){
            if(s == null || s.length() == 0) continue;
            if("..".equals(s)){
                if(!list.isEmpty()) list.remove(list.size()-1);
            }else if(!(".".equals(s))){
                list.add(s);
            }
        }
        if(list.size() == 0) return "/";

        StringBuilder sb = new StringBuilder();
        for(String x : list){
            sb.append('/');
            sb.append(x);
        }
        return sb.toString();

    }
}
```

72. Edit Distance

Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

```
a) Insert a character
b) Delete a character
c) Replace a character
```

DP

```
public class Solution {
    public int minDistance(String word1, String word2) {
        if(word1 == null && word2 == null) return 0;
        if(word1 == null || word2 == null) return word1 == null
? word2.length() : word1.length();
        if(word1.length() == 0 || word2.length() == 0) return wo
rd1.length() == 0 ? word2.length() : word1.length();

        int m = word1.length()+1;
        int n = word2.length()+1;
        int[][] dp = new int[m][n];
        for(int i=0; i<m;i++) dp[i][0] = i;
        for(int i=0;i<n; i++) dp[0][i] = i;
        for(int i=1; i<m; i++){
            for(int j = 1; j<n;j++){
                if(word1.charAt(i-1) == word2.charAt(j-1)){
                    dp[i][j] = dp[i-1][j-1];
                }else{
                    dp[i][j] = Math.min(dp[i-1][j], Math.min(dp[
i-1][j-1], dp[i][j-1])) + 1;
                }
            }
        }

        return dp[m-1][n-1];

    }
}
```

73. Set Matrix Zeroes

Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in place.

click to show follow up.

Follow up:

Did you use extra space?

A straight forward solution using O(mn) space is probably a bad idea.

A simple improvement uses O(m + n) space, but still not the best solution. Could you devise a constant space solution?

constant space.

```
public class Solution {
    public void setZeroes(int[][] matrix) {
        if(matrix == null || matrix.length == 0 || matrix[0].len
gth == 0) return;
        boolean resetFirstRow = false;
        boolean resetFirstColumn = false;
        for(int i=0; i< matrix.length;i++){
            if(matrix[i][0] == 0) resetFirstColumn = true;
        }
        for(int i=0; i< matrix[0].length;i++){
            if(matrix[0][i] == 0) resetFirstRow = true;
        }

        for(int i=1; i<matrix.length;i++){
            for(int j=1; j< matrix[0].length; j++){
                if(matrix[i][j] == 0){
                    matrix[i][0] = 0;
                    matrix[0][j] =0;
                }
            }
        }
```

```
        for(int i=1; i<matrix.length;i++){
            for(int j=1; j< matrix[0].length; j++){
                if(matrix[i][0] == 0 || matrix[0][j] == 0){
                    matrix[i][j] = 0;
                }
            }
        }

        if(resetFirstColumn){
            for(int i=0; i<matrix.length;i++){
                matrix[i][0] = 0;
            }
        }
        if(resetFirstRow){
            for(int i=0;i<matrix[0].length; i++){
                matrix[0][i] = 0;
            }
        }
    }
}
```

74. Search a 2D Matrix

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted from left to right. The first integer of each row is greater than the last integer of the previous row. For example,

Consider the following matrix:

```
[
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
```

Given target = 3, return true.

Show Tags Show Similar Problems

> A linear solution

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if(matrix == null || matrix.length == 0 || matrix[0].len
gth == 0) return false;
        int i = matrix.length -1;
        int j = matrix[0].length -1;
        if(target > matrix[i][j] || target < matrix[0][0] ) retu
rn false;

        int x = 0;
        int y = j;
        while(x <= i&& y>=0){
            if(target == matrix[x][y]) return true;
            else if( target < matrix[x][y]){
                y--;
            }else{
                x++;
            }
        }
        return false;
    }
}
```

> 2 binary search

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int k = matrix[0].length -1;

        // find the row that matrix[r][k] > target && matrix[r-1
][k] < target;

        int t = 0;
        int b = matrix.length -1;
        if(target < matrix[0][0] || target > matrix[matrix.lengt
h-1][matrix[0].length-1]) return false;
        while(t <= b){
            int mid = t + (b-t)/2;
            if(matrix[mid][k] == target) return true;
            if(matrix[mid][k] > target){
                b = mid-1;
            }else{
                t = mid+1;
            }
        }

        int l = t;

        t = 0;
        b = matrix[0].length -1;
        while(t <= b){
            int mid = t + (b-t)/2;
            if(matrix[l][mid] == target) return true;
            if(matrix[l][mid] > target){
                b = mid -1;
            }else{
                t = mid + 1;
            }
        }

        return false;
    }
}
```

> 1 binary search

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int row = matrix.length;
        int col = matrix[0].length;
        int l = 0;
        int r = row * col - 1;

        while(l <=r ){
            int mid = l + (r-l)/2;
            int val = matrix[mid/col][mid%col];
            if(val == target) return true;
            if(val > target){
                r = mid-1;
            }else{
                l = mid+1;
            }
        }

        return false;
    }
}
```

75. Sort Colors

Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

click to show follow up.

Follow up: A rather straight forward solution is a two-pass algorithm using counting sort.

First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

two pointers.

```java
public class Solution {
    public void sortColors(int[] nums) {
        int p = -1; // last index of 0;
        int q = nums.length; // first index of 2.

        int k = p + 1;
        while(k < q){
            switch(nums[k]){
                case 0:
                    swap(nums, ++p, k++);
                break;
                case 1:
                    k++;
                break;
                case 2:
                    swap(nums, k, --q);
                break;
            }
        }
    }

    private void swap(int[] nums, int x, int y){
        int tmp = nums[x];
        nums[x] = nums[y];
        nums[y] = tmp;
    }
}
```

76. Minimum Window Substring

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

For example, S = "ADOBECODEBANC" T = "ABC" Minimum window is "BANC".

Note: If there is no such window in S that covers all characters in T, return the empty string "".

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

> needs two pointers, the front pointer make sure to include all the chars in T, the end pointer make sure to exclude as many chars as possible to make the window smallest.

```
public class Solution {
    // you need to include the
    public String minWindow(String s, String t) {

        Map<Character, Integer> target = new HashMap<>();
        Map<Character, Integer> found = new HashMap<>();

        int minWindowStart = -1;
        int minWindowLen = Integer.MAX_VALUE;

        for(char ch : t.toCharArray()){
            int count   =1;
            if(target.containsKey(ch)){
                count = target.get(ch) + 1;
            }
            target.put(ch, count);
        }

        int start = 0;
        int foundNumber = 0;// this means all letter shows up, y
ou need to shrink window when possible.
```

```
        for(int i=0; i<s.length(); i++){
            // move i to include all chars in target.
            char c = s.charAt(i);
            if(!target.containsKey(c)) continue;

            int k = 1;
            if(found.containsKey(c)){
                k = found.get(c) + 1;
            }
            found.put(c, k);
            //once foundNumber reach t.length(), the number stan
ds. found.get(c) will be updated in the next block
            if(found.get(c) <= target.get(c)){
                foundNumber++;
            }

            if(foundNumber == t.length()){ // you don't decrease
 foundNumber after you found require the number of chars.
                char begin = s.charAt(start);
                // move start as right as possbile, to shrink th
e window size.
                while(!found.containsKey(begin) || found.get(beg
in) > target.get(begin)){

                    if(found.containsKey(begin)){
                        found.put(begin, found.get(begin)-1);
                        //you can NOT decrease foundNumber here.
 there is no go back for foundNumber.
                    }
                    start++;
                    begin = s.charAt(start);
                }

                int len = i-start+1;
                if(len < minWindowLen){
                    minWindowLen = len;
                    minWindowStart = start;
                }
            }
        }
```

```
        if(minWindowStart != -1){
            return s.substring(minWindowStart, minWindowStart +
minWindowLen);
        }
        return "";
    }
}
```

77. Combinations

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

For example, If n = 4 and k = 2, a solution is:

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

backtracking

```java
public class Solution {
    List<List<Integer>> res = new ArrayList<>();
    public List<List<Integer>> combine(int n, int k) {
        if(k == 0) return res;

        bt(1, n, k, new ArrayList<Integer>());
        return res;

    }

    private void bt(int start, int n, int k, List<Integer> list)
{
        if(k == 0){
            List<Integer> t = new ArrayList<>(list);
            res.add(t);
            return;
        }

        for(int i= start; i<=n; i++){
            list.add(i);
            bt(i+1, n, k-1, list);
            list.remove(list.size()-1);
        }
    }


}
```

78. Subsets

Given a set of distinct integers, nums, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example, If nums = [1,2,3], a solution is:

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

BACKTRACKING

is a very important sollution:

- sort the input to be non-descending.
- backtracking/dfs it.

Notice, next iteration is based on current set status, not from the beginning, so the dfs will take input from i+1, not **start+1**.

Related Issue : 90 Subset II

```
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {

        List<List<Integer>> res = new ArrayList<>();
        if(nums == null || nums.length == 0) return res;
        List<Integer> cur = new ArrayList<>();
        Arrays.sort(nums);
        dfs(nums, res, 0, cur);
        return res;
    }

    private void dfs(int[] nums, List<List<Integer>> res, int st
art, List<Integer> cur){
        res.add(new ArrayList<Integer>(cur));
        for(int i=start; i< nums.length; i++){
            cur.add(nums[i]);

            dfs(nums, res, i+1, cur);
            cur.remove(cur.size()-1);
        }
    }
}
```

Solution 1. build a next set based current set, i.e. for each subset in current set, add current number, add it into the result set, as next set.

```
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();

        for(int val : nums){
            List<Integer> v = new ArrayList<>();
            v.add(val);
            List<List<Integer>> r = new ArrayList<>();

            for(List<Integer> l : res){
                List<Integer> t  = new ArrayList<>(l);
                t.addAll(v);
                r.add(t);
            }
            //you cannot update the res inside the for loop, unl
ess you will use an iterator there.
            res.addAll(r);
            res.add(v);
        }
        res.add(new ArrayList<>());
        return res;
    }
}
```

79. Word Search

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example, Given board =

```
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

word = "ABCCED", -> returns true,

word = "SEE", -> returns true,

word = "ABCB", -> returns false.

```
public class Solution {
    public boolean exist(char[][] board, String word) {
        if(board == null || board.length ==0 || board[0].length
== 0) return false;

        boolean[][] visited = new boolean[board.length][board[0]
.length];

        for(int i=0; i< board.length; i++){
            for(int j=0; j< board[0].length; j++){
                if(dfs(board, visited, i, j, word, 0)){
                    return true;
                }
            }
        }
        return false;
```

```
    }

    private boolean dfs(char[][]board, boolean[][] visited, int
x, int y, String word, int k){
        if(word.length() == k) return true;
        if(x < 0 || x >= board.length || y < 0
                || y >= board[0].length || visited[x][y]|| word.
charAt(k) != board[x][y] )return false;

        visited[x][y] = true;

        boolean res = dfs(board, visited, x+1, y, word, k+1)
                        || dfs(board, visited, x-1, y, word, k+1
)
                        || dfs(board, visited, x, y+1, word, k+1
)
                        || dfs(board, visited, x, y-1, word, k+1
);

        visited[x][y] = false;
        return res;
    }
}
```

80. Remove Duplicates from Sorted Array II

Follow up for "Remove Duplicates": What if duplicates are allowed at most twice?

For example, Given sorted array nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of nums being 1, 1, 2, 2 and 3. It doesn't matter what you leave beyond the new length.

Related issue: 26 Remove Duplicates from Sorted Array

```
public class Solution {
    public int removeDuplicates(int[] nums) {
        int i=0;
        int j=1;
        boolean allowDup = true;
        for(;j<nums.length;){
            if(nums[i] == nums[j]){
                if(allowDup){
                    allowDup = false;
                    nums[++i] = nums[j++];
                }else{
                    j++;
                }
            }else{
                allowDup = true;
                nums[++i] = nums[j++];
            }
        }

        return ++i;
    }
}
```

> slightly simpler code.

```java
public class Solution {
    public int removeDuplicates(int[] nums) {
        if(nums == null) return 0;
        if(nums.length <= 2) return nums.length;
        boolean allow = true;
        int l = 0;
        for(int r=1; r<nums.length; r++){
            if(nums[r] == nums[l]){
                if(allow){
                    nums[++l] = nums[r];
                    allow = false;
                }
            }else{// not equal case.
                allow = true;
                nums[++l] = nums[r];
            }
        }
        return l+1;
    }
}
```

```java
public class Solution {
    public int removeDuplicates(int[] nums) {
        if(nums == null) return 0;
        if(nums.length <= 2) return nums.length;

        boolean allow = true;

        int k = 0;
        for(int i =1 ; i< nums.length; i++){

            if(nums[i] != nums[k] || allow){
                nums[++k] = nums[i];
                allow = nums[i] != nums[k-1] ? true : false;

            }
        }
        return ++k;
    }
}
```

81. Search in Rotated Sorted Array II

Follow up for 33 Search in Rotated Sorted Array:

What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.

> ATTENTION: when nums[mid] == nums[left], you cannot gurantee the shape of array.

```
public class Solution {
    public boolean search(int[] nums, int target) {
                int left=0;
        int right = nums.length-1;
        while(left <= right){
            int mid = left + (right -left)/2;
            if(nums[mid] == target) return true;
            if(nums[mid] > nums[left]){
                if(nums[left] <= target && target < nums[mid]){
                    right = mid-1;
                }else{
                    left = mid +1;
                }
            }else if (nums[mid] < nums[left]){
                if(nums[mid] <= target && target <= nums[right])
{
                    left = mid +1;
                }else{
                    right = mid -1;
                }
            }else{
                left++;// move a step ahead. cause  there is no
gaurantee the target will show on which part. cause the shap of
first part[left, mid] is not decided.
            }
        }
        return false;
    }
}
```

82. Remove Duplicates from Sorted List II

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example,

Given 1->2->3->3->4->4->5, return 1->2->5.

Given 1->1->1->2->3, return 2->3.

Keep three pointers, first pointer, tail always points to the tail of found list item. second pointer, prev, point to the head of to-be-found list head, and a third pointer, cur, which points the same as prev, while cur and cur's next value are the same, the move forward cur pointer, if cur and prev is not the same, then we need to cut prev to cur off, then move next section of list.

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode tail = dummy;
        ListNode prev = head;
        ListNode cur = head;

        while(cur != null && cur.next != null){
            while(cur.next != null && cur.val == cur.next.val){
                cur = cur.next;
            }
            if(cur == prev){
                tail.next = cur;
                tail = tail.next;
            }
            prev = cur.next;
            cur = cur.next;
        }
        tail.next = cur;
        return dummy.next;
    }
}
```

83. Remove Duplicates from Sorted List

Given a sorted linked list, delete all duplicates such that each element appear only once.

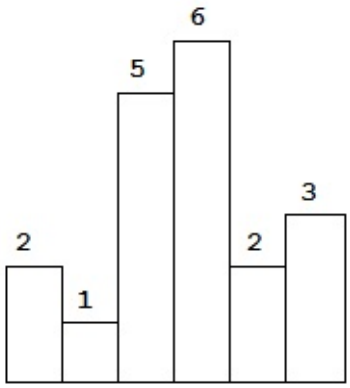
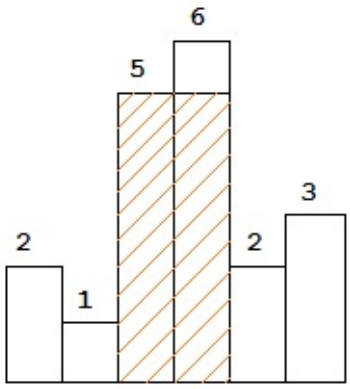
For example,

```
Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.
```

Use two pointers

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if(head == null || head.next == null) return head;
        ListNode h = head;
        ListNode p = head;
        ListNode q = head.next;
        while(q != null){
            if(q.val == p.val){
                p.next = q.next;
            }else{
                p.next = q;
                p = p.next;
            }
            q = q.next;
        }

        return h;
    }
}
```

84. Largest Rectangle in Histogram

> Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.
>
> 
>
>
> Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].
>
> 
>
>
> The largest rectangle is shown in the shaded area, which has area = 10 unit.

```
For example,
Given heights = [2,1,5,6,2,3],
return 10.
```

This is a hard question. To solve This problem in linear time, use a stack to track the index of each number:

- if current value is great-equal than the top of stack. which means it is a increasing sequence,(**The index at top don't need to be the previous of current**), push the current index.

- If not, this means there is a small rectangle at the top of stack, pop the top of stack, the next top index and current index is the width of a found rectangle, and the height is the previous poped item.

```
public class Solution {
    public int largestRectangleArea(int[] heights) {
        Stack<Integer> stack = new Stack<>();// stack for index
to caculate width.
        int max = 0;
        int i =0;
        while( i <= heights.length){
            int val = i < heights.length ? heights[i] : 0;
            if(stack.empty() || val >= heights[stack.peek()]){
                stack.push(i);
                i++;
            }else{
                int h = stack.pop();
                int w = stack.empty() ? i : i - stack.peek() -1;
                max = Math.max(max, heights[h] * w);
            }
        }

        return max;
    }
}
```

85. Maximal Rectangle

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing all ones and return its area.

This is an extension of **84 Largest Rectangle in Histogram**, you need to convert the 2D matrix, so that each row is a histogram of previous rows. if current row-col is '0', simple treat this row-col in histogram as 0.

Non-Related Issue: 221 Maximal Square

```
public class Solution {
    public int maximalRectangle(char[][] matrix) {
        if(matrix == null || matrix.length == 0 || matrix[0].len
gth == 0) return 0;
        int[][] data = new int[matrix.length][matrix[0].length];
        for(int i=0; i< matrix[0].length; i++){
            data[0][i] = matrix[0][i] == '0' ? 0 :1;
        }

        for(int i=1; i< matrix.length; i++){
            for(int j=0; j < matrix[0].length; j++){
                if(matrix[i][j] != '0'){
                    data[i][j] += data[i-1][j] +  1;
                }else{
                    data[i][j] = 0;
                }
            }
        }
        int max = 0;
        for(int i=data.length-1; i>=0; i--){

            max = Math.max(max, maximalInHistogram(data[i]));
            if(max >= i * data[i].length) break;

        }
        return max;
    }
```

```
    private int maximalInHistogram(int[] data){
        Stack<Integer> stack = new Stack<>();

        int max = 0;
        int i = 0;
        while(i <= data.length){
            int val = i < data.length ? data[i] : 0;
            if(stack.empty() || val >= data[stack.peek()]){
                stack.push(i);
                i++;
            }else{
                int h = stack.pop();
                int w = stack.empty() ? i : i-stack.peek()-1;
                max = Math.max(max, data[h] * w);
            }
        }

        return max;
    }
}
```

86. Partition List

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 and x = 3,
return 1->2->2->4->3->5.
```

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode partition(ListNode head, int x) {
        ListNode gt = new ListNode(-1);
        ListNode lt = new ListNode(-1);
        ListNode t1 = gt;
        ListNode t2 = lt;

        while(head != null){
            if(head.val < x){
                t2.next = head;
                t2 = t2.next;
            }else{
                t1.next = head;
                t1 = t1.next;
            }
            head = head.next;
        }

        t2.next = gt.next;
        t1.next = null;
        return lt.next;
    }
}
```

> the following improvement won't work cause you are CREATING A new reference to the node ltail/mtail points to, then move it around, you are NOT REALLY MOVING ltail/mtail.

```
            ListNode tail = head.val < x ? ltail : mtail;
            tail.next = head;
            tail = tail.next;
```

88. Merge Sorted Array

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note: You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

```
public class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {

        int end = m+n-1;
        int end1 = m -1;
        int end2 = n -1;

        while(end2 >= 0){
            if(end1 >=0){
                nums1[end--] = nums1[end1] > nums2[end2] ? nums1
[end1--] : nums2[end2--];
            }else{
                nums1[end--] = nums2[end2--];
            }
        }

    }
}
```

89. Gray Code QuestionEditorial Solution My Submissions

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:

00 - 0 01 - 1 11 - 3 10 - 2 Note: For a given n, a gray code sequence is not uniquely defined.

For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.

For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

```java
public class Solution {
    public List<Integer> grayCode(int n) {
        List<Integer> list = new ArrayList<>();

        list.add(0);
        if(n <=0) return list;

        int i=0;
        while(i < n){
            int b = 1 << i++;
            int size = list.size();
            for(int j= size-1; j>=0; j--){
                int v = list.get(j);
                list.add(v | b);
            }
        }

        return list;
    }
}
```

```java
public class Solution {
    public List<Integer> grayCode(int n) {
        if(n == 0){
            List<Integer> res = new ArrayList<>();
            res.add(0);
            return res;
        }

        List<Integer> list = grayCode(n-1);
        int k = list.size()-1;
        for(int i=k; i>=0; i--){
            list.add(list.get(i)| 1 << n-1);
        }

        return list;

    }
}
```

90. Subsets II

Given a collection of integers that might contain duplicates, nums, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example, If nums = [1,2,2], a solution is:

[[2], [1], [1,2,2], [2,2], [1,2], []]

Related issue: 78 Subsets

> backtracking, pruning, 回溯法， 剪枝。

```
public class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {

        List<List<Integer>> res = new ArrayList<>();
        if(nums == null || nums.length == 0) return res;
        List<Integer> cur = new ArrayList<>();
        Arrays.sort(nums);
        dfs(nums, res, 0, cur);
        return res;
    }

    private void dfs(int[] nums, List<List<Integer>> res, int st
art, List<Integer> cur){
    //you DONT check if start == nums.length; cause you need to
collect every thing.
        res.add(new ArrayList<Integer>(cur));
        for(int i=start; i< nums.length; i++){
            if( i != start && nums[i] == nums[i-1]) continue;
            cur.add(nums[i]);
            dfs(nums, res, i+1, cur);
            cur.remove(cur.size()-1);
        }
    }
}
```

> if you check the size, then for [1,2,2,2], the result is empty.

```
          if(k == nums.length ){
             res.add(new ArrayList<Integer>(list));
             return;
         }
```

91. Decode Ways

A message containing letters from A-Z is being encoded to numbers using the following mapping:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given an encoded message containing digits, determine the total number of ways to decode it.

For example, Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).

The number of ways decoding "12" is 2.

> DP[i] means how many ways you can decode s.substring(0, i);

```
public class Solution {
    public int numDecodings(String s) {
        if(s == null || s.length() == 0) return 0;
        int[] dp = new int[s.length() +1];
        //dp[i] means how many ways you can decode s.substring(0
..i)
        dp[0] = 1;
        dp[1] = s.charAt(0) == '0' ? 0 : 1;
        for(int i= 2; i<= s.length(); i++){
            int tmp = Integer.parseInt(s.substring(i-1, i));
            if( tmp != 0) dp[i] = dp[i-1];// if tmp == 0, this l
ine won't execute, but the [i-2,i-1] build a number 10, or 20,
            //the result is set to dp[i-2] directly in the next
step.
            tmp = (s.charAt(i-2) - '0') * 10 + s.charAt(i-1) - '
0';
            if(tmp >=10 && tmp <= 26) dp[i] += dp[i-2];
        }

        return dp[s.length()];
    }

}
```

92. Reverse Linked List II

Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example: Given 1->2->3->4->5->NULL, m = 2 and n = 4,

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

Note: Given m, n satisfy the following condition:

1 ≤ m ≤ n ≤ length of list.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseBetween(ListNode head, int m, int n)
{
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode p = dummy;
        int k = 0;
        while( ++k < m){
            if(p != null) p = p.next;
        }
        //TODO:
        ListNode tail = p.next;
        while(++k <= n){
            ListNode tmp = p.next;

            p.next = tail.next;
            tail.next = tail.next.next;
            p.next.next = tmp;


        }

        return dummy.next;


    }
}
```

93. Restore IP Addresses

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example: Given "25525511135",

return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

```
public class Solution {

    List<String> res = new ArrayList<>();
    public List<String> restoreIpAddresses(String s) {

        restore(s, 0, new ArrayList<String>());
        return res;
    }


    private void restore(String s, int start,  List<String> list)
{
        if(list.size() == 4){
            if(start != s.length()) return;

            StringBuilder sb = new StringBuilder();
            for(String ss : list){
                sb.append(".");
                sb.append(ss);
            }
            sb.deleteCharAt(0);
            res.add(sb.toString());
            return;
        }

        for(int k = start+1; k<= s.length() && k <= start+4; k++
){

            String q = s.substring(start, k);
```

```
            if(isValid(q)){
                list.add(q);
                restore(s, k, list);
                list.remove(list.size()-1);
            }
        }

    }

    private boolean isValid(String s){
        if(s.charAt(0) == '0') return s.equals("0");
        return Integer.parseInt(s) >=0 && Integer.parseInt(s) <=
255;
    }
}
```

```
public class Solution {
    List<String> res = new ArrayList<>();
    public List<String> restoreIpAddresses(String s) {
        if(s == null || s.length() < 4) return res;
        restore(s, new ArrayList<>());
        return res;
    }

    void restore(String s, List<String> list){
        if(list.size() == 4 ){
            if(s.length() != 0 ) return;
            StringBuilder sb = new StringBuilder();
            for(String t: list){
                sb.append('.').append(t);
            }
            sb.deleteCharAt(0);
            res.add(sb.toString());
            return;
        }

        int len = Math.min(4, s.length());
        for(int i=1; i<= len; i++){
            String p = s.substring(0, i);
            if(Integer.parseInt(p) > 255 || Integer.parseInt(p)<
0) break;
            if(p.charAt(0) == '0' && p.length() != 1) break;
            list.add(p);
            restore(s.substring(i), list);
            list.remove(list.size()-1);
        }
    }
}
```

94. Binary Tree Inorder Traversal

Given a binary tree, return the inorder traversal of its nodes' values.

For example: Given binary tree {1,#,2,3},

```
   1
    \
     2
    /
   3
```

return [1,3,2].

Note: Recursive solution is trivial, could you do it iteratively?

confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

```
Here's an example:
   1
  / \
 2   3
    /
   4
    \
     5
The above binary tree is serialized as "{1,2,3,#,#,4,#,#,5}".
```

Related issue 230. Kth Smallest Element in a BST

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();

        Stack<TreeNode> stack = new Stack<>();
        TreeNode node = root;
        while(!stack.isEmpty()  || node != null){
            while(node != null){
                stack.push(node);
                node = node.left;
            }

            TreeNode top = stack.pop();
            res.add(top.val);
            node = top.right;
        }

        return res;
    }
}
```

96. Unique Binary Search Trees

Given n, how many structurally unique BST's (binary search trees) that store values 1...n?

For example, Given n = 3, there are a total of 5 unique BST's.

```
   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

This is a dp question. count of 3 =

```
  * if 1 is root, then count[0] * count[2];
  * if 2 is root, then count[1] * count[1];
  * if 3 is root, then count[2] * count[0];
```

from above deduction, so empty subtree is one, if take one node as root, number smaller than root will form left sub tree. and others to form right sub tree.

```
public class Solution {
    public int numTrees(int n) {
        int[] dp = new int[n+1];
        dp[0] =1;
        dp[1]=1;
        for(int i=2; i<= n; i++){
            for(int j=1; j<=i;j++) // take j as root, j-1 nodes
form left subtree, i-j nodes forms right subtree.
                dp[i] += dp[j-1] * dp[i-j];
        }
        return dp[n];
    }
}
```

98. Validate Binary Search Tree

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees. Example 1:

```
  2
 / \
1   3
```

Binary tree [2,1,3], return true.

Example 2:

```
  1
 / \
2   3
```

Binary tree [1,2,3], return false.

> Related issue: 333 Largest BST Subtree

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);

    }

    private boolean isValidBST(TreeNode root, long min, long max)
{
        if(root == null) return true;

        if((root.val > min && root.val < max)){
            return isValidBST(root.left, min, (long)root.val)
                    && isValidBST(root.right, (long)root.val, ma
x);
        }else{
            return false;
        }
    }
}
```

What if `Long` is not available in the system.

> Pre Order traversal

```
public class Solution {
    List<Integer> nodes = new ArrayList<>();
    public boolean isValidBST(TreeNode root) {
         if(root == null) return true;

        boolean leftIsValid = isValidBST(root.left);

        if(nodes.size() > 0 && nodes.get(nodes.size()-1) >= root
.val ){
            return false;
        }

        nodes.add(root.val);

        boolean rightIsValid = isValidBST(root.right);

        return leftIsValid && rightIsValid;
    }

}
```

You notice that you don't really need the array for pre-order traversing, you only need the previous node.

> THIS IS A VERY IMPORTANT TECHNIQUE while traversing he tree, keep tracking of visited nodes.

```
public class Solution {
    TreeNode prev = null;
    public boolean isValidBST(TreeNode root) {
         if(root == null) return true;

        boolean leftIsValid = isValidBST(root.left);
        if(!leftIsValid) return false;
        // you can return from here is left is not valid.
        if(prev != null && prev.val >= root.val ){
            return false;
        }

        prev = root;

        boolean rightIsValid = isValidBST(root.right);

        return leftIsValid && rightIsValid;
    }

}
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isValidBST(TreeNode root) {
        Stack<TreeNode> stack = new Stack<>();
        TreeNode prev = null;
        while(!stack.isEmpty() || root != null){
            while(root != null){
                stack.push(root);
                root = root.left;
            }
            TreeNode top = stack.pop();
            if(prev != null && top.val <= prev.val) return false
;
            else prev = top;
            root = top.right;
        }

        return true;
    }
}
```

99. Recover Binary Search Tree

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note: A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?

> This is a very important skill while traversing tree.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    TreeNode prev = null;
    TreeNode first = null;
    TreeNode second = null;
    public void recoverTree(TreeNode root) {

        findSwap(root);

        int tmp = first.val;
        first.val = second.val;
        second.val = tmp;
    }

    void findSwap(TreeNode root){
        if(root == null) return;

        findSwap(root.left);
        if(prev != null){
            if(first == null && prev.val > root.val){
```

```
                first = prev;
            }
            if(first != null && prev.val > root.val){
                second = root;
            }
        }

        prev = root;

        findSwap(root.right);
    }
}
```

100. Same Tree

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if(p == null && q == null) return true;
        if(p == null || q == null) return false;
        if(p.val != q.val) return false;
        return isSameTree(p.left, q.left)
                && isSameTree(p.right, q.right);
    }
}
```

101 Symmetric Tree

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree [1,2,2,3,4,4,3] is symmetric:

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

But the following [1,2,2,null,3,null,3] is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

Note: Bonus points if you could solve it both recursively and iteratively.

- recursive

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isSymmetric(TreeNode root) {
        if(root == null) return true;
        return symmetric(root.left, root.right);
    }

    boolean symmetric(TreeNode left, TreeNode right){
        if(left == null && right == null) return true;
        if(left == null || right == null) return false;
        if(left.val != right.val) return false;
        return symmetric(left.left, right.right) && symmetric(le
ft.right, right.left);

    }
}
```

- iterative
-

102. Binary Tree Level Order Traversal

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

For example: Given binary tree [3,9,20,null,null,15,7],

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

return its level order traversal as:

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

BFS v.s DFS

> Both use Queue/Stack.BUT BFS is like peeling onion, you need to keep track of each layer. so you need to retrieve the queue size etc.

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if(root == null) return res;
        Queue<TreeNode> queue = new LinkedList<>();

        queue.offer(root);
        while(!queue.isEmpty()){
            List<Integer> l = new ArrayList<>();
            int stop = queue.size();
            for(int i=0; i< stop; i++){
                TreeNode node = queue.poll();
                l.add(node.val);
                if(node.left != null) queue.offer(node.left);
                if(node.right != null) queue.offer(node.right);
            }
            res.add(l);
        }

        return res;
    }
}
```

103. Binary Tree Zigzag Level Order Traversal

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).

For example: Given binary tree [3,9,20,null,null,15,7],

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

return its zigzag level order traversal as:

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

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if(root == null) return res;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        boolean r2l = false;
        while(!queue.isEmpty()){
            int size = queue.size();
            List<Integer> list = new ArrayList<>();
            for(int i=0; i< size; i++){
                TreeNode node = queue.poll();
                list.add(node.val);
                if(node.left != null) queue.offer(node.left);
                if(node.right != null) queue.offer(node.right);
            }
            if(r2l){
                r2l = false;
                Collections.reverse(list);
            }else{
                r2l = true;
            }
            res.add(list);
        }

        return res;
    }
}
```

104. Maximum Depth of Binary Tree

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

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int maxDepth(TreeNode root) {
        if(root == null) return 0;
        return Math.max(maxDepth(root.left), maxDepth(root.right
))+1;
    }

}
```

105. Construct Binary Tree from Preorder and Inorder Traversal

Given preorder and inorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    Map<Integer, Integer> map = new HashMap<>();
    public TreeNode buildTree(int[] preorder, int[] inorder) {

        for(int i =0; i< inorder.length; i++){
            map.put(inorder[i], i);
        }

        return build(preorder,0, preorder.length-1, inorder, 0,
inorder.length-1);
    }

    private TreeNode build(int[] preorder, int ps, int pe, int[]
 inorder, int is, int ie ){
        if(ps > pe || is > ie){
            return null;
        }

        TreeNode root = new TreeNode(preorder[ps]);
        int mid = map.get(preorder[ps]);
```

```
        int count = mid - is;
        root.left = build(preorder, ps+1, ps+count, inorder, is,
 mid - 1);
        root.right = build(preorder, ps+count+1, pe, inorder, mi
d+1, ie);

        return root;

    }
}
```

106. Construct Binary Tree from Inorder and Postorder Traversal

Given inorder and postorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

> same as previous one.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    Map<Integer, Integer> map = new HashMap<>();

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        for(int i = 0; i< inorder.length; i++){
            map.put(inorder[i], i);
        }

        return build(inorder, 0, inorder.length-1, postorder, 0,
 postorder.length -1);
    }

    private TreeNode build(int[] inorder, int i0, int i1, int[]
postorder, int p0, int p1){

        if(i0 > i1 || p0 > p1) return null;

        TreeNode root = new TreeNode(postorder[p1]);

        int mid = map.get(postorder[p1]);
        int count = i1 -mid; // right sub-tree size
        root.left = build(inorder, i0, mid-1, postorder, p0, p1
- count-1);
        root.right = build(inorder, mid+1, i1, postorder, p1-cou
nt, p1-1);

        return root;
    }
}
```

107. 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).

For example: Given binary tree [3,9,20,null,null,15,7],

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

return its bottom-up level order traversal as:

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

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if(root == null) return res;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while(!queue.isEmpty()){
            List<Integer> list = new ArrayList<>();
            int size = queue.size();
            for(int i=0; i< size; i++){
                TreeNode node = queue.poll();
                list.add(node.val);
                if(node.left != null) queue.offer(node.left);
                if(node.right != null) queue.offer(node.right);
            }
            res.add(list);
        }

        Collections.reverse(res);

        return res;
    }
}
```

108. Convert Sorted Array to Binary Search Tree

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        if(nums == null) return null;
        return subArrayBuilder(nums, 0, nums.length - 1);

    }

    private TreeNode subArrayBuilder(int[] nums, int left, int r
ight){
        if(left > right) return null;
        if(left == right) return new TreeNode(nums[left]);

        int mid = left + (right-left)/2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = subArrayBuilder(nums, left, mid-1);
        root.right = subArrayBuilder(nums, mid+1, right);

        return root;
    }

}
```

109. Convert Sorted List to Binary Search Tree

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

> Top-down approach will take n^2 time, cause each time you need to find the root in the linked list, then recursively construct the two parts.
>
> Bottom-Up approach, you need to keep track of from where to build the sub tree and for how many nodes this sub tree has. so the current is used for this purpose.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    ListNode cur = null;
    public TreeNode sortedListToBST(ListNode head) {
        int size = getLen(head);
        cur = head;

        return getBST(size);
```

```
    }

    private TreeNode getBST(int size){
        if(size <=0) return null;
        TreeNode left = getBST(size/2);
        TreeNode root = new TreeNode(cur.val);
        cur = cur.next;
        TreeNode right = getBST(size -1 - size/2);
        root.left = left;
        root.right = right;
        return root;
    }
    private int getLen(ListNode head){
        int l = 0;
        while(head != null){
            head = head.next;
            l++;
        }

        return l;
    }
}
```

1. Balanced Binary Tree `````` 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.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isBalanced(TreeNode root) {
        return getHeightOrDelta(root) != -1;
    }

    int getHeightOrDelta(TreeNode root){
        if(root == null) return 0;
        int l = getHeightOrDelta(root.left);
        if(l == -1) return -1;
        int r = getHeightOrDelta(root.right);
        if(r == -1) return -1;
        if(Math.abs(l - r) >1 ) return -1;

        return 1 + Math.max(l, r);
    }
}
```

1. Minimum Depth of Binary Tree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

The take away here is that if any node is empty, there is no path, so cannot count as a minium depth.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int minDepth(TreeNode root) {
        if(root == null) return 0;
        int l = minDepth(root.left);
        int r = minDepth(root.right);
        if( l !=0 && r != 0){
            return Math.min(l,r) + 1;
        }else if(l == 0){
            return r +1;
        }else{
            return l+1;
        }


    }
}
```

1. Path Sum

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example: Given the below binary tree and sum = 22,

```
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \      \
        7    2      1
```

return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

Solution.

if you reach to a node which is empty, which means there is no such path, when you reach to a leaf node, check wether it is a valid path. otherwise continue to next level.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean hasPathSum(TreeNode root, int sum) {
        if(root == null) return false;
        if(root.left == null && root.right == null){
            return root.val == sum;
        }
        return hasPathSum(root.left, sum-root.val) || hasPathSum
(root.right, sum-root.val);

    }
}
```

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean hasPathSum(TreeNode root, int sum) {
        if(root == null) return false;
        return has(root, 0, sum);
    }

    boolean has(TreeNode root, int cur, int target){
        if(root.left == null && root.right == null){
            return target == cur+root.val;
        }
        cur += root.val;

        boolean res = false;
        if(root.left != null){
            res = has(root.left, cur, target);
        }
        if(res ) return true;

        if(root.right != null){
            res = has(root.right, cur,target);
        }

        return res;

    }
}
```

113. Path Sum II

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example: Given the below binary tree and sum = 22,

```
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \    / \
        7    2  5   1
```

return

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<List<Integer>> res = new ArrayList<>();
    public List<List<Integer>> pathSum(TreeNode root, int sum) {

        getSum(root, new ArrayList<Integer>(), 0, sum);
```

```
        return res;
    }

    private void getSum(TreeNode node, List<Integer> list, int c
urrent, int sum){
        if(node == null) return;
        current += node.val;
        list.add(node.val);

        if(node.left == null && node.right == null){
            if(current == sum){
                res.add(list);
            }else{
                return;
            }
        }
        if(node.left != null){
            getSum(node.left, new ArrayList<Integer>(list), curr
ent, sum);
        }
        if(node.right != null){
            getSum(node.right, new ArrayList<Integer>(list), cur
rent, sum);

        }

    }

}
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<List<Integer>> res = new ArrayList<>();

    public List<List<Integer>> pathSum(TreeNode root, int sum) {
        if(root == null) return res;
        build(root, new ArrayList<Integer>(), 0, sum);
        return res;
    }

    void build(TreeNode root, List<Integer> list, int cur, int t
arget){
        if(root.left == null && root.right == null){
            if(cur + root.val == target){
                list.add(root.val);
                res.add(new ArrayList<>(list));
                list.remove(list.size()-1);
            }
            return;
        }
        list.add(root.val);
        if(root.left != null) build(root.left, list, cur+root.va
l, target);
        if(root.right != null) build(root.right, list, cur + roo
t.val, target);
        list.remove(list.size()-1);
    }
}
```

114. Flatten Binary Tree to Linked List

Given a binary tree, flatten it to a linked list in-place.

For example, Given

```
        1
       / \
      2   5
     / \   \
    3   4   6
```

The flattened tree should look like:

```
1
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

> thing to notice:
>
> - 1 if right sub tree turned into a linked list,and its tail is not empty, this will be the tail of this sub list,
> - 2 if right sub tree turns to a empty list, then if left sub tree is not empty, then this sub tree's tail will become the tail of this tree.
> - other wise the root itself is the tail.

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void flatten(TreeNode root) {
        flat(root);
    }

    private TreeNode flat(TreeNode root){
        if(root == null) return null;

        TreeNode ltail = flat(root.left);
        TreeNode rtail = flat(root.right);
        if(root.left != null){
            ltail.right = root.right;
            root.right = root.left;
            root.left = null;

        }
        if(rtail != null) return rtail;
        if(ltail != null) return ltail;
        return root;
    }
}
```

a more concise solution in c++

```
/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root: a TreeNode, the root of the binary tree
     * @return: nothing
     */
    TreeNode* prev;
    void flatten(TreeNode *root) {
        // write your code here
        if(root == nullptr) return;
        flatten(root->right);
        flatten(root->left);
        root->right = prev;
        root->left = nullptr;
        prev = root;
    }
};
```

116. Populating Next Right Pointers in Each Node

Given a binary tree

```
struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

Note:

You may only use constant extra space. You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children). For example, Given the following perfect binary tree,

```
         1
       /  \
      2    3
     / \  / \
    4  5  6  7
```

After calling your function, the tree should look like:

```
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \  / \
    4->5->6->7 -> NULL
```

```java
/**
 * Definition for binary tree with next pointer.
 * public class TreeLinkNode {
 *     int val;
 *     TreeLinkNode left, right, next;
 *     TreeLinkNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void connect(TreeLinkNode root) {
        if(root == null) return;
        /*if(root.left != null){
            do not need to check here. if root.right exist, then
 left for sure exist.
        }*/
        if(root.right != null){
            root.left.next = root.right;
            root.right.next = root.next == null ?
                                null : root.next.left;
        }
        connect(root.left);
        connect(root.right);
    }
}
```

117 Populating Next Right Pointers in Each Node II

Follow up for problem "Populating Next Right Pointers in Each Node".

What if the given tree could be any binary tree? Would your previous solution still work?

Note:

You may only use constant extra space. For example, Given the following binary tree,

```
         1
       /  \
      2    3
     / \    \
    4   5    7
```

After calling your function, the tree should look like:

```
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \    \
    4-> 5 -> 7 -> NULL
```

> using a queue

```
/**
 * Definition for binary tree with next pointer.
 * public class TreeLinkNode {
 *     int val;
 *     TreeLinkNode left, right, next;
 *     TreeLinkNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void connect(TreeLinkNode root) {
        if(root == null) return;

        Queue<TreeLinkNode> q = new LinkedList<>();

        q.add(root);

        while(!q.isEmpty()){
            int size = q.size();
            for(int i=0; i< size; i++){
                TreeLinkNode node = q.poll();
                node.next = (i == size-1)? null : q.peek();
                if(node.left != null) q.add(node.left);
                if(node.right != null) q.add(node.right);
            }
        }
    }
}
```

- keep track of next level's first.
- keep track of next level's last visited.
- move current along its next chain.

```
/**
 * Definition for binary tree with next pointer.
 * public class TreeLinkNode {
 *     int val;
 *     TreeLinkNode left, right, next;
 *     TreeLinkNode(int x) { val = x; }
```

```
 * }
 */
public class Solution {
    public void connect(TreeLinkNode root) {
        if(root == null) return;

        //keep track of first node in next level,
        TreeLinkNode first = null;
        //and previous node in current level
        TreeLinkNode last = null;
        //and current visiting node
        TreeLinkNode current = root;
        root.next = null;

        while(current != null){
            //retrieve next first if is not set.
            if(first == null){
                if(current.left != null) first = current.left;
                else first = current.right;
            }

            if(current.left != null){
                if(last != null) last.next = current.left;
                // IT is not else, if last is not null, then ste
p ahead last
                // to next which is current.left,
                //if last is null, then current.left become last;

                last = current.left;
            }
            if(current.right != null){
                if(last != null) last.next = current.right;
                last = current.right;
            }

            if(current.next != null){
                current = current.next;
            }else{
                current = first;
                first = last = null;
```

```
                }

            }
        }
    }
```

121. Best Time to Buy and Sell Stock

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 1:

```
Input: [7, 1, 5, 3, 6, 4]
Output: 5
```

max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)

Example 2:

```
Input: [7, 6, 4, 3, 1]
Output: 0
```

In this case, no transaction is done, i.e. max profit = 0.

Related issue

- **122. Best Time to Buy and Sell Stock II**
- **123. Best Time to Buy and Sell Stock III**
- **188. Best Time to Buy and Sell Stock IV**

> classic solution, will time limit exceeded. O(n^2)

```
public class Solution {
    public int maxProfit(int[] prices) {
        int res = 0;
        if(prices == null || prices.length <= 1) return res;
        int[] dp = new int[prices.length];

        dp[0] = 0;
        for(int i=1; i<dp.length; i++){
            for(int j=0; j<i; j++){
                dp[i] = Math.max(prices[i]-prices[j], dp[i]);
            }
            res = Math.max(dp[i], res);
        }

        return res;
    }
}
```

> However, you only need to keep track of the current minimal value ever found so far, use current value minus the min value, you get a profit, if current value is even smaller than the current minimal value, this is potentially a buy-in time, which will be sold later.
>
> O(n)

```java
public class Solution {
    public int maxProfit(int[] prices) {
        int res = 0;
        if(prices == null || prices.length <=1 )return res;
        int min = prices[0];
        for(int i=1; i< prices.length; i++){
            if(prices[i] > min){
                res = Math.max(res, prices[i] - min);
            }else{
                min = prices[i];
            }
        }

        return res;

    }
}
```

122. Best Time to Buy and Sell Stock II

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).

Related issue:

- **121 Best Time to Buy and Sell Stock II**
- **123 Best Time to Buy and Sell Stock III**

```
public class Solution {
    public int maxProfit(int[] prices) {
        int res = 0;
        if(prices == null || prices.length <= 1) return res;

        for(int i=1; i< prices.length; i++){
            if(prices[i] > prices[i-1]) res += prices[i] - price
s[i-1];
        }

        return res;
    }

}
```

123. Best Time to Buy and Sell Stock III

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).

first thought is to use binary search, for each i:

- find the maximal profit between [0..i]
- find the maximal profit between [i+1..n]
- then find the maximal sum for i postion.

> This approach require *O(n^2)*

second approach is to do it in two pass:

- first find all maximal profit between [0..i], where i reach to n. in this process, you need to keep track of the mininal buy-in price. **if i sell at price[i] what is the maximal profit.**
- second find all maximal profit between[n..0] i.e. from the end to the begin. **if I buy-in at price[i], what will be the maximal profit,**, in this process, you need to keep the maximal sell price .

Related issue

- **121. Best Time to Buy and Sell Stock I**
- **122. Best Time to Buy and Sell Stock II**

```
public class Solution {
    public int maxProfit(int[] prices) {

        int res = 0;
        if(prices == null || prices.length <=1) return res;

        int[] p = new int[prices.length];

        int minBuyInPrice= prices[0];
        p[0] = 0;
        for(int i=1; i<p.length; i++){
            minBuyInPrice = Math.min(prices[i], minBuyInPrice);
            p[i] = Math.max(prices[i] - minBuyInPrice, p[i-1]);

        }

        int[] q = new int[prices.length];
        q[q.length-1] = 0;
        int maxSellOutPrice = prices[q.length-1];
        for(int i= q.length-2; i>=0; i--){
            maxSellOutPrice = Math.max(prices[i], maxSellOutPric
e);
            q[i] = Math.max(q[i+1], maxSellOutPrice- prices[i]);
        }

        for(int i=0; i<q.length; i++){
            res = Math.max(res, q[i] + p[i]);
        }

        return res;
    }
}
```

125. Valid Palindrome

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,

"A man, a plan, a canal: Panama" is a palindrome.

"race a car" is not a palindrome.

Note: Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

```
public class Solution {
    public boolean isPalindrome(String s) {
        int i=0, j = s.length()-1;
        while( i < j){
            while(i<s.length() && !Character.isLetterOrDigit(s.c
harAt(i))) i++;
            if(i==s.length()) return true;
            while(j >=0 && !Character.isLetterOrDigit(s.charAt(j
))) j--;
            if(j < 0 ) return true;

            if(Character.toLowerCase(s.charAt(i))
                != Character.toLowerCase(s.charAt(j))) return fa
lse;
            i++;
            j--;
        }
        return true;
    }
}
```

127. Word Ladder

Given two words (beginWord and endWord), and a dictionary's word list, find the length of shortest transformation sequence from beginWord to endWord, such that:

Only one letter can be changed at a time Each intermediate word must exist in the word list For example,

Given: beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog", return its length 5.

Note: Return 0 if there is no such transformation sequence. All words have the same length. All words contain only lowercase alphabetic characters.

```
public class Solution {
    public int ladderLength(String beginWord, String endWord, Se
t<String> wordList) {

        wordList.add(endWord);
        Queue<String> queue = new LinkedList<>();
        queue.add(beginWord);
        int len = 0;

        while(!queue.isEmpty()){
            int size = queue.size();
            len++;
            for(int i=0; i< size; i++){
                String s = queue.poll();
                if(s.equals(endWord)) return len;
                for(String n : getNext(s, wordList)){
                    queue.offer(n);
                }
```

```
            }
        }

        return 0;
    }

    private Set<String> getNext(String s, Set<String> wordList){
        char[] chars = s.toCharArray();
        Set<String> res = new HashSet<String>();
        for(int i=0; i< chars.length; i++){
            char c = chars[i];
            for(char x ='a'; x <= 'z'; x++){
                if(c == x) continue;
                chars[i] = x;
                String next = new String(chars);
                if(wordList.contains(next)){
                    res.add(next);
                    wordList.remove(next);
                }
                chars[i] = c;
            }
        }
        return res;
    }
}
```

128. Longest Consecutive Sequence

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example,

Given [100, 4, 200, 1, 3, 2],

The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

Your algorithm should run in O(n) complexity.

```java
public class Solution {
    public int longestConsecutive(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for(int i : nums){
            set.add(i);
        }
        int max = 0;
        for(int val : nums){
            if(set.contains(val)){
                set.remove(val);
                int low = val -1;
                while(set.contains(low)){
                    set.remove(low);
                    low--;
                }
                int up = val + 1;
                while(set.contains(up)){
                    set.remove(up);
                    up++;
                }
                max = Math.max(max, (up - low - 1));
            }
        }

        return max;
    }
}
```

```java
public class Solution {
    public int longestConsecutive(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for(int i : nums){
            set.add(i);
        }
        int max = 0;
        for(int i=0; i< nums.length; i++){
            int low = nums[i] -1;
            while(set.contains(low)){
                set.remove(low);
                low--;
            }

            int up = nums[i] + 1;
            while(set.contains(up)){
                set.remove(up);
                up++;
            }
            max = Math.max(max, (up - low - 1));
        }

        return max;
    }
}
```

129. Sum Root to Leaf Numbers

> Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.
>
> An example is the root-to-leaf path 1->2->3 which represents the number 123.
>
> Find the total sum of all root-to-leaf numbers.>
>
> For example,

```
    1
   / \
  2   3
```

> The root-to-leaf path 1->2 represents the number 12.
>
> The root-to-leaf path 1->3 represents the number 13.
>
> Return the sum = 12 + 13 = 25.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    int res = 0;
    public int sumNumbers(TreeNode root) {

        if(root == null) return res;
        bSum(root, 0);
        return res;
    }

    private void bSum(TreeNode root, int value){
        if(root.left == null && root.right == null){
            res += value*10 + root.val;
            return;
        }

        if(root.left != null) bSum(root.left, value*10 + root.va
l);
        if(root.right != null) bSum(root.right, value*10 + root.
val);
    }

}
```

Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,

```
X X X X
X O O X
X X O X
X O X X
```

After running your function, the board should be:

```
X X X X
X X X X
X X X X
X O X X
```

bfs/dfs.

```
public class Solution {
    public void solve(char[][] board) {
        if(board == null || board.length == 0|| board[0].length==
0) return;
        class Node{
            int x;
            int y;
            Node(int i, int j){
                x = i;
                y = j;
            }
        }

        ArrayList<Node> queue = new ArrayList<>();

        int m = board.length;
        int n = board[0].length;
        for(int i=0; i< m; i++){
```

```
            if(board[i][0] == 'O'){
                queue.add(new Node(i,0));
            }
            if(board[i][n-1] == 'O'){
                queue.add(new Node(i, n-1));
            }
        }
        for(int i=1; i<n-1; i++){
            if(board[0][i] == 'O') queue.add(new Node(0, i));
            if(board[m-1][i] == 'O') queue.add(new Node(m-1, i))
;
        }

        int k =0;
        while(k < queue.size()){
            Node node = queue.get(k);
            int x = node.x;
            int y = node.y;

            board[node.x][node.y] = 'U';
            k++;
            if(x > 0 && board[x-1][y] == 'O'){
                queue.add(new Node(x-1,y));
            }
            if(x<m-1 && board[x+1][y] == 'O'){
                queue.add(new Node(x+1, y));
            }
            if(y >0 && board[x][y-1] == 'O'){
                queue.add(new Node(x, y-1));
            }
            if(y < n-1 && board[x][y+1] == 'O'){
                queue.add(new Node(x, y+1));
            }
        }
        for(int i=0;i<m;i++){
            for(int j=0;j<n;j++){
                if(board[i][j] == 'O') board[i][j] = 'X';
                if(board[i][j] == 'U') board[i][j] = 'O';
            }
        }
```

```
        }
    }
```

Or you can get a recursive solution. which go ove the border of the board, if find a 'O', then dfs from this point. mark itself and its neighbor as 'U' recursively.

131. Palindrome Partitioning

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example, given s = "aab", Return

```
[
  ["aa","b"],
  ["a","a","b"]
]
```

> Backtracking

```java
public class Solution {
    List<List<String>> res = new ArrayList<>();
    public List<List<String>> partition(String s) {
        if(s == null || s.length() == 0) return res;
        part(s, 0, new ArrayList<String>());
        return res;
    }

    void part(String s, int start, List<String> list){
        if(s.length() == start){
            res.add(new ArrayList<String>(list));
            return;
        }

        for(int i=start+1; i<= s.length(); i++){
            String st = s.substring(start, i);
            if(isP(st)){
                list.add(st);
                part(s, i, list);
                list.remove(list.size()-1);
            }
        }
    }

    boolean isP(String s){
        int l = 0;
        int r = s.length()-1;
        while(l < r){
            if(s.charAt(l) != s.charAt(r)) return false;
            l++;
            r--;
        }
        return true;
    }


}
```

133. Clone Graph

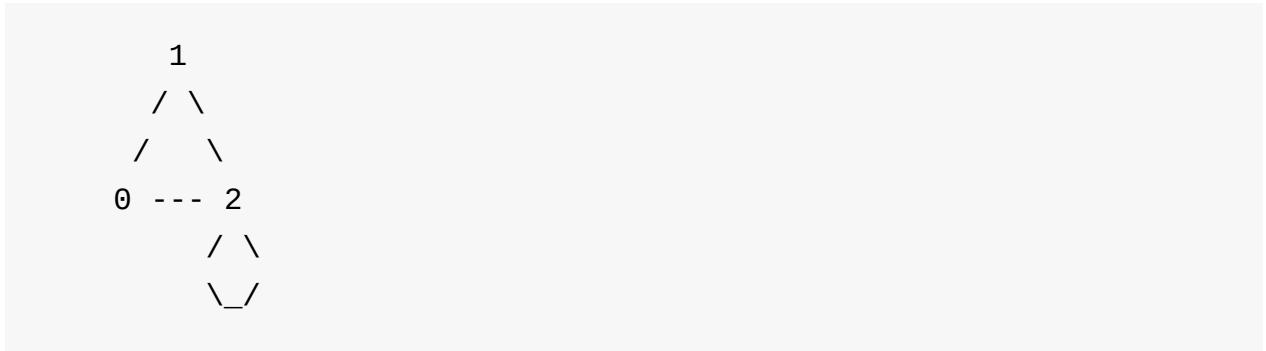
Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

OJ's undirected graph serialization: Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node. As an example, consider the serialized graph **{0,1,2#1,2#2,2}**.

The graph has a total of three nodes, and therefore contains three parts as separated by #.

First node is labeled as 0. Connect node 0 to both nodes 1 and 2. Second node is labeled as 1. Connect node 1 to node 2. Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle. Visually, the graph looks like the following:



> 1 map node to new node.
>
> 2 connection new nodes neighbors by getting old node's.

```
/**
 * Definition for undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     List<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new A
rrayList<UndirectedGraphNode>(); }
 * };
```

```
 */
public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode no
de) {
        if(node == null) return null;
        List<UndirectedGraphNode> nodes = new ArrayList<>();
        Map<UndirectedGraphNode,UndirectedGraphNode> map = new H
ashMap<>();

        nodes.add(node);
        map.put(node, new UndirectedGraphNode(node.label));

        int start =0;
        while(start < nodes.size()){
            UndirectedGraphNode n = nodes.get(start++);
            for(UndirectedGraphNode nn : n.neighbors){
                if(!map.containsKey(nn)){
                    map.put(nn, new UndirectedGraphNode(nn.label
));
                    nodes.add(nn);
                }
            }
        }

        for(UndirectedGraphNode n : nodes){
            for(UndirectedGraphNode nn : n.neighbors){
                map.get(n).neighbors.add(map.get(nn));
            }
        }

        return map.get(node);

    }
}
```

> The following solution failed cause this test case:
>
> Input:{0,1,5#1,2,5#2,3#3,4,4#4,5,5#5}
>
> Output:{0,1,5#1,2,5#2,3#3,4,4#4,5,5,5,5#5}
>
> Expected:{0,1,5#1,2,5#2,3#3,4,4#4,5,5#5}

```
/**
 * Definition for undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     List<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new A
rrayList<UndirectedGraphNode>(); }
 * };
 */
public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode no
de) {

        if(node == null ) return node;
        // node < - > copy
        Map<UndirectedGraphNode,UndirectedGraphNode> map = new H
ashMap<>();
        List<UndirectedGraphNode> list = new ArrayList<>();
        list.add(node);

        int k= 0;
        while(k < list.size()){
            UndirectedGraphNode cur = list.get(k++);
            UndirectedGraphNode copy = new UndirectedGraphNode(c
ur.label);
            map.put(cur, copy);
            Set<UndirectedGraphNode> keys = map.keySet();
            for(UndirectedGraphNode n : cur.neighbors){
                // if the neighbors is like 4,4,,4. then you add
 all the same instance into the list, multiple times.
                // But how possible ?
                if(!keys.contains(n)){
                    list.add(n);
```

```
                }
            }
        }

        for(UndirectedGraphNode n : list){
            for(UndirectedGraphNode nn : n.neighbors){
                map.get(n).neighbors.add(map.get(nn));
            }
        }

        return map.get(node);
    }
}
```

134. Gas Station

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

> Two problems for this :
>
> - if sum(gas) - sum(cost) >= 0, then there is a point you can start and complete.
> - if at certain point k, if sum(gas[0..k]) - sum(cost[0..k]) <0 then k is not possible to begin with, then reset the current sum to 0 , k + 1 will be the result.

```
public class Solution {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int station = -1;
        int current =0;
        int tank = 0;
        for(int i=0; i< gas.length; i++){
            tank += gas[i] - cost[i];
            current += gas[i] - cost[i];
            if(current < 0){
                current = 0;
                station = i;
            }
        }
        if (tank < 0) return -1;
        return station+1;
    }
}
```

135. Candy

There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy.

Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

Two passes:

- left to right, if a[i] > a[i-1], increase the candy of i by 1, if not set it as 1.
- right to left, if a[i-1] > a[i], BUT count is less, then increase candy of i.

```
public class Solution {
    public int candy(int[] ratings) {
        if(ratings == null || ratings.length == 0) return 0;
        int[] count = new int[ratings.length];
        count[0] = 1;
        for(int i=1;i<ratings.length;i++){
            if(ratings[i] > ratings[i-1]){
                count[i] = count[i-1] + 1;
            }else{
                count[i] = 1;
            }
        }
        int res = count[ratings.length-1];

        for(int i=ratings.length-1; i >0;i--){
            if(ratings[i-1] > ratings[i] && count[i-1] <= count[
i]){
                count[i-1] = count[i] + 1;
            }
            res += count[i-1];
        }

        return res;

    }
}
```

138. Copy List with Random Pointer

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

Return a deep copy of the list.

```
/**
 * Definition for singly-linked list with a random pointer.
 * class RandomListNode {
 *     int label;
 *     RandomListNode next, random;
 *     RandomListNode(int x) { this.label = x; }
 * };
 */
public class Solution {
    public RandomListNode copyRandomList(RandomListNode head) {
        RandomListNode p = head;
        while(p != null){
            RandomListNode next = p.next;
            RandomListNode dup = new RandomListNode(p.label);
            dup.next = next;
            p.next =dup;
            p =next;
        }


        p = head;
        while(p != null){
            if(p.random != null){
                p.next.random = p.random.next;
            }
            p = p.next.next;
        }

        RandomListNode dummy = new RandomListNode(-1);
        RandomListNode dupTail = dummy;
        p = head;
```

```
        while(p != null){
            RandomListNode next = p.next.next;
            dupTail.next = p.next;
            p.next = next;
            dupTail = dupTail.next;
            p = next;
        }

        return dummy.next;

    }
}
```

136. Single Number I

Given an array of integers, every element appears twice except for one. Find that single one.

Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Related Issue Single Number II Single Number III

```
// Xor operation
public class Solution {
    public int singleNumber(int[] nums) {
        int res = 0;
        for(int v : nums){
            res ^= v;
        }
        return res;
    }
}
```

139. Word Break

Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

For example, given

s = "leetcode",

dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".

DP. set the begining to be a dummy char, and the set default value to be true.

dp[i] = dp[k] && substring(k+1, i+1) in the dict. k = [0..i)

```
public class Solution {
    public boolean wordBreak(String s, Set<String> wordDict) {
        boolean[] res = new boolean[s.length() +1];
        String t = "*" + s;
        res[0] = true;

        for(int i=1; i< t.length(); i++){
            for(int k = 0; k< i; k++){
                res[i] = res[k] && wordDict.contains(t.substring
(k+1, i+1)); // this is i+1 cause we are calculating res[i], so
charAt(i) should be included;
                if(res[i]) break;
            }
        }

        return res[res.length-1];
    }
}
```

140. Word Break II

Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where each word is a valid dictionary word.

Return all such possible sentences.

For example, given

```
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].
```

A solution is ["cats and dog", "cat sand dog"].

Time limit exceed

```java
public class Solution {
    List<String> res = new ArrayList<>();

    public List<String> wordBreak(String s, Set<String> wordDict)
 {
        if(s == null || s.length() == 0) return res;
        List<String> list = new ArrayList<String>();
        break2(s, list, wordDict);
        return res;
    }

    void break2(String s, List<String> list, Set<String> dict){
        if(s.length() == 0){
            StringBuilder sb = new StringBuilder();
            sb.append(list.get(0));
            for(int i=1; i< list.size(); i++){
                sb.append(' ');
                sb.append(list.get(i));
            }
            res.add(sb.toString());
            return;
        }

        for(int i=1; i<= s.length(); i++){
            String sub = s.substring(0, i);
            if(dict.contains(sub)){
                list.add(sub);
                break2(s.substring(i), list, dict);
                list.remove(list.size()-1);
            }
        }
    }
}
```

Understanding KMP algorithms

The core the KMP algorithm is the calculation of next array. Before looking into how to calculate next array, it is better to understand what to deal with string comparison between text and pattern.

in the context of brute force to solve the problem "find pattern in text", each time there is a mismatch between text[i+k] and pattern[k], pointer to text is set to i+1, and pointer to pattern reset to 0, until find a match(k == pattern.length), or i+k> tet.length, there is no pattern in text.

This naive approach wastes a lot comparison that is unnecessary, next array in KMP is trying to improve this mismatch-comparison situation. everytime there is a mismatch(j as the index), pattern is moved toward text end by *LEN*, this *LEN* means that there is a substring pattern[0..LEN] matches PATTERN[j-LEN..j-1], so that it can bypass unnecessary comparison.

the *LEN* is next[j] : there is a length next[j] substring in pattern which appears in the begin and end of pattern(prefix and suffix).

the relationship between next array and max prefix-suffix is that during KMP algorithm, there is no need to consider current mismatch char, and there is no need to keep last max prefix-suffix, cause if match, the algorithm return. so the values in max prefix-suffix is shift right by one.

to Calculate max prefix-suffix array.

```
int[] max = new int[target.length()];
max[0] = 0;
for(int i=1; i< target.length(); i++){
     int k = max[i-1];
     while(k>0 && target.charAt(k) != target.charAt(i)) k = max[
k-1];
     max[i] = target.charAt(k) == target.charAt(i) ? k+1 : k;
}
```

Calculate the next array is relatively simple. say we already know next[0..j], where next[j]==k(there is a substringP[0..k-1], length k, is a prefix-suffix, **P[j] not included**), to calculate next[j+1]:

1. if P[k] == P[j], next[j+1] = next[j]+1 = k+1;
2. if P[k] != P[j], we need to trace back to k == next[k], until
    i. P[next[k]] == P[j], then P[j] = next[k] +1;
    ii. K == -1, P[j] == 0; where -1 is next[0];

```
int[] next = new int[N]; // N is pattern size;
next[0] = -1;
int k =-1;
int j=0;
while(j < N-1){
  if(k==-1 || P[k] == P[j]){
    ++k; ++j;
    next[j] = k;
  }else{
    k = next[k];
  }
}
```

141. Linked List Cycle

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

Follow up: Can you solve it without using extra space?

- use hash map to save node.
- Two pointers with different speed.

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null || head.next == null) return false;
        ListNode p = head, q = head.next;
        while(p != null && q != null){
            if(p == q) return true;
            p = p.next;
            q = q.next;
            if(q != null) q = q.next;
        }
        return false;
    }
}
```

142. Linked List Cycle II

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

Note: Do not modify the linked list.

still two pointers, but DON'T give any pointer a head-start, cause in reality there is no such thing. in the while loop, run the pointers first, then check

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode detectCycle(ListNode head) {
        if(head == null || head.next == null) return null;
        ListNode slow = head;
        ListNode fast = head;

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

```
                }


            }

            return null;



        }
    }
```

143. Reorder List

Given a singly linked list L: L0→L1→…→Ln-1→Ln, reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes' values.

For example, Given {1,2,3,4}, reorder it to {1,4,2,3}.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void reorderList(ListNode head) {
        if(head == null) return ;
        ListNode p = head;
        ListNode q = head.next;
        while(q != null && q.next != null){
            p = p.next;
            q = q.next;
            if(q != null) q = q.next;
        }

        q = p.next;
        p.next = null;
        ListNode newHead = null;
        while(q != null){
            ListNode tmp = q.next;
            q.next = newHead;
            newHead = q;
            q = tmp;
        }
```

```
        p = head;
        q = newHead;
        ListNode dummy = new ListNode(-1);
        ListNode tail = dummy;

        while(p != null || q != null){
            if(p != null){
                tail.next = p;
                p = p.next;
                tail = tail.next;
            }
            if(q!= null){
                tail.next = q;
                q = q.next;
                tail = tail.next;
            }

        }

        head = dummy.next;

    }
}
```

144. Binary Tree Preorder Traversal

Given a binary tree, return the preorder traversal of its nodes' values.

For example: Given binary tree {1,#,2,3},

```
   1
    \
     2
    /
   3
```

return `[1,2,3]`.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        if(root == null) return res;
        Stack<TreeNode> stack = new Stack<>();
        stack.push(root);

        while(!stack.isEmpty()){
            TreeNode node = stack.pop();
            res.add(node.val);
            if(node.right != null) stack.push(node.right);
            if(node.left != null) stack.push(node.left);
        }

        return res;
    }
}
```

145. Binary Tree Postorder Traversal

Given a binary tree, return the postorder traversal of its nodes' values.

For example: Given binary tree {1,#,2,3},

```
   1
    \
     2
    /
   3
return [3,2,1].
```

Note: Recursive solution is trivial, could you do it iteratively?

to get post-order, you will need 2 stacks to get the result. in the question. the result array is the second stack.

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        if(root == null) return res;
        Stack<TreeNode> stack1 = new Stack<>();
        stack1.push(root);
        while(!stack1.isEmpty()){
            TreeNode node = stack1.pop();

            res.add(node.val);
            if(node.left != null) stack1.push(node.left);
            if(node.right != null) stack1.push(node.right);

        }

        Collections.reverse(res);

        return res;

    }
}
```

146. LRU Cache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

set(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

```
public class LRUCache {

    private int mSize;
    private int mCapacity;
    private Node start;
    private Node end;
    Map<Integer, Node> map = new HashMap<>();
    public LRUCache(int capacity) {
        mSize = 0;
        mCapacity = capacity;
        start = new Node();
        end = new Node();
        start.next = end;
        end.prev = start;

    }

    public int get(int key) {
        if(map.containsKey(key)){
            Node node = map.get(key);
            moveToHead(node);
            return node.val;
        }else{
            return -1;
        }
```

```
    }
    private void moveToHead(Node node){
        if(node.prev == start) return;

        node.prev.next = node.next;
        node.next.prev = node.prev;
        insertToHead(node);
    }
    private void insertToHead(Node node){
        Node next = start.next;
        node.next = next;
        next.prev = node;

        start.next = node;
        node.prev = start;


    }
    public void set(int key, int value)
    {

        if(map.containsKey(key)){
            moveToHead(map.get(key));
            Node node = map.get(key);
            node.val = value;
        }else{
            Node node = new Node(key , value);
            if(mSize >= mCapacity){
                clearCache();
            }
            insertToHead(node);
            mSize++;
            map.put(key, node);
        }


    }

    private void clearCache(){
        while(mSize >= mCapacity){
```

```
            Node tail = end.prev;
            if(tail == start) break;
            tail.prev.next = end;
            end.prev = tail.prev;

            map.remove(tail.key);
            mSize--;
        }
    }

    class Node{
        int key;
        int val;
        Node prev, next;
        Node(int key, int val){
            this.key = key;
            this.val = val;
            prev = next =null;
        }
        Node(){};
    }
}
```

150. Evaluate Reverse Polish Notation

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, /. Each operand may be an integer or another expression.

Some examples:

```
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9

["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6
```

Stack.

```
public class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();
        String ops = "+-/*";
        for(String s : tokens){
            if(ops.indexOf(s) == -1){
                stack.push(Integer.valueOf(s));
            }else{
                if(stack.size() < 2){
                    throw new RuntimeException("Invalid expressi
on");
                }
                int second = stack.pop();
                int first = stack.pop();
                int res = 0;
                char op = s.charAt(0);
                switch(op){
                    case '+':
                        res = first + second;
                        break;
                    case '-':
                        res = first - second;
```

```
                        break;
                    case '*':
                        res = first * second;
                        break;
                    case '/':
                        //exceptions.
                        res = first/second;
                        break;
                    default:
                        throw new RuntimeException("Invalid oper
ation");
                }
                stack.push(res);
            }

        }
        return stack.pop();
    }
}
```

151. Reverse Words in a String

Given an input string, reverse the string word by word.

For example, Given s = "the sky is blue", return "blue is sky the".

Update (2015-02-12): For C programmers: Try to solve it in-place in O(1) space.

click to show clarification.

Clarification: What constitutes a word?

A sequence of non-space characters constitutes a word.

Could the input string contain leading or trailing spaces?

Yes. However, your reversed string should not contain leading or trailing spaces.

How about multiple spaces between two words?

Reduce them to a single space in the reversed string.

```
public class Solution {
    public String reverseWords(String s) {
        StringBuilder sb = new StringBuilder();
        for(int i= s.length()-1; i>=0; i--){
            while(i>=0 && s.charAt(i) == ' ') i--;
            if(i < 0) break;
            /* if sb is empty, which means it is first word met,
 simple skip, and if it is not, when we reach here, the string i
s not done processing, so append a space.
            */
            if(sb.length() != 0) sb.append(' ');
            StringBuilder w = new StringBuilder();
            /* Add the string in reverse order. */
            while(i>=0 && s.charAt(i) != ' ') w.append(s.charAt(
i--));
            sb.append(w.reverse());
        }

        return sb.toString();
    }
}
```

152. Maximum Product Subarray

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array [2,3,-2,4],

the contiguous subarray [2,3] has the largest product = 6.

Subscribe to see which companies asked this question

Related issue: 53. Maximum Subarray

```
public class Solution {
    public int maxProduct(int[] nums) {
        if(nums == null || nums.length == 0)
            return 0;

        int max = nums[0];
        int min = nums[0];
        int product = nums[0];
        for(int i = 1; i< nums.length; i++){
            int t1 = max * nums[i];
            int t2 = min * nums[i];
            max = Math.max(Math.max(t1, t2), nums[i]);
            min= Math.min(Math.min(t1, t2), nums[i]);

            product = Math.max(product, max);
        }
        return product;
    }
}
```

153. Find Minimum in Rotated Sorted Array

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.

Related issue: 154 Find Minimum in Rotated Sorted Array II

```
public class Solution {
    public int findMin(int[] nums) {
        int l = 0;
        int r = nums.length-1;
        while(l <= r){
            int mid = l + (r-l)/2;
            if(l == r) break;
            if(nums[l] < nums[r]) break;
            if(nums[mid] >= nums[l]){
                l = mid +1;
            }else{
                r = mid;
            }

        }
        return nums[l];
    }
}
```

154. Find Minimum in Rotated Sorted Array II

Follow up for "Find Minimum in Rotated Sorted Array": What if duplicates are allowed?

Would this affect the run-time complexity? How and why? 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.

```
public class Solution {
    public int findMin(int[] nums) {
        int l = 0;
        int r = nums.length-1;
        while(l<= r){
            if( l == r) break;
            if(nums[l] < nums[r])break;
            int mid = l + (r-l)/2;
            if(nums[mid] > nums[l]){
                l = mid+1;
            }else if(nums[mid] < nums[l]){
                r = mid;
            }else{
                l++;
            }
        }

        return nums[l];
    }
}
```

155 Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Solution. using two stacks, push value as usually, but at mean time, push the top of another stack, minStack, or the current value onto minStack, whichever is smaller.

```
class MinStack {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> minStack = new Stack<>();

    public void push(int x) {
        if(stack.empty()){
            stack.push(x);
            minStack.push(x);
        }else{
            stack.push(x);
            if(minStack.peek() > x){
                minStack.push(x);
            }else{
                minStack.push(minStack.peek());
            }
        }
    }

    public void pop() {
        if(stack.empty()){
            //throw exception
        }else{
            stack.pop();
            minStack.pop();
```

```
        }
    }

    public int top() {
        if(stack.empty()){
            //throw Exception();
            return -1;
        }else{
            return stack.peek();
        }
    }

    public int getMin() {
        if(minStack.empty()){
            //throw Exception
            return -1;
        }else{
            return minStack.peek();
        }
    }
}
```

156. Binary Tree Upside Down

Given a binary tree where all the right nodes are either leaf nodes with a sibling (a left node that shares the same parent node) or empty, flip it upside down and turn it into a tree where the original right nodes turned into left leaf nodes. Return the new root.

For example: Given a binary tree {1,2,3,4,5},

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

return the root of the binary tree [4,5,2,#,#,3,1].

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

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode upsideDownBinaryTree(TreeNode root) {
        if(root == null) return root;
        if(root.left == null && root.right == null) return root;

        TreeNode newRoot = upsideDownBinaryTree(root.left);
        TreeNode newLeft = root.right;
        TreeNode oldLeft = root.left;

        root.left = null;
        root.right = null;
        oldLeft.left = newLeft;
        oldLeft.right = root;

        return newRoot;
    }
}
```

157. Read N Characters Given Read4

The API: int read4(char *buf) reads 4 characters at a time from a file.

The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.

By using the read4 API, implement the function int read(char *buf, int n) that reads n characters from the file.

Note: The read function will only be called once for each test case.

```
/* The read4 API is defined in the parent class Reader4.
      int read4(char[] buf); */

public class Solution extends Reader4 {
    /**
     * @param buf Destination buffer
     * @param n   Maximum number of characters to read
     * @return    The number of characters read
     */
    public int read(char[] buf, int n) {
        int index =0;
        char[] r4 = new char[4];
        while(index < n){
            int c = read4(r4);
            for(int i=0; i<c && index < n; i++){
                buf[index++] = r4[i];
            }
            if(c < 4) break;
        }

        return index;
    }
}
```

158. Read N Characters Given Read4 II - Call multiple times

The API: int read4(char *buf) reads 4 characters at a time from a file.

The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.

By using the read4 API, implement the function int read(char *buf, int n) that reads n characters from the file.

Note: The read function may be called multiple times.

```java
/* The read4 API is defined in the parent class Reader4.
      int read4(char[] buf); */

public class Solution extends Reader4 {
    /**
     * @param buf Destination buffer
     * @param n   Maximum number of characters to read
     * @return    The number of characters read
     */
    Queue<Character> queue = new LinkedList<>();
    public int read(char[] buf, int n) {

        int index = 0;
        while(!queue.isEmpty() && index < n){
            buf[index++] = queue.poll();
        }

        if(index == n) return n;

        char[] r4 = new char[4];

        while(index < n){
            int c = read4(r4);
            int i =0;
            for(; i< c && index < n; i++){
                buf[index++] = r4[i];
            }
            while(i < c){
                queue.offer(r4[i++]);
            }
            if(c < 4) break;// nothing to read from the stream/f
ile.
        }

        return index;
    }
}
```

160. Intersection of Two Linked Lists

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

```
A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:     b1 → b2 → b3
```

begin to intersect at node c1.

Notes:

- If the two linked lists have no intersection at all, return null.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode
 headB) {
        if(headA == null || headB == null) return null;
```

```
        int a = 0;
        int b = 0;
        ListNode ha = headA;
        ListNode hb = headB;
        while(ha != null || hb != null){
            if(ha != null){
                ha = ha.next;
                a++;
            }
            if(hb != null){
                hb = hb.next;
                b++;
            }
        }

        int delta = b - a;
        ha = headA;
        hb = headB;
        if(delta > 0){
            while(delta-- > 0){
                hb = hb.next;
            }
        }else if(delta < 0){
            while(delta++ <0){
                ha = ha.next;
            }
        }
        // ping ha and hb to the right pos in the list, so that
both has k distance to the end.

        while(ha != null){
            if(ha == hb) return ha;
            ha = ha.next;
            hb = hb.next;
        }

        return null;

    }
```

```
}
```

161. One Edit Distance

Given two strings S and T, determine if they are both one edit distance apart.

```
public class Solution {
    public boolean isOneEditDistance(String s, String t) {
        if(Math.abs(s.length() - t.length()) > 1) return false;

        if(s.length() == t.length()){
            int count =0;
            int i = 0;
            while(i < s.length()){
                if(s.charAt(i) != t.charAt(i)) count++;
                if(count > 1) break;
                i++;
            }
            return count == 1;
        }else{
            return s.length() > t.length() ? isOneDelete(s, t) :
 isOneDelete(t,s);
        }
    }

    private boolean isOneDelete(String longer, String shorter){
        int i=0;
        for(; i< shorter.length(); i++){
            if(longer.charAt(i) != shorter.charAt(i))break;
        }

        if(i == shorter.length()) return true;
        return shorter.substring(i, shorter.length()).equals(lon
ger.substring(i+1, longer.length()));
    }
}
```

```
public class Solution {
    public boolean isOneEditDistance(String s, String t) {
        int sl = s.length();
        int tl = t.length();
        int delta = Math.abs(sl-tl);

        if(delta > 1) return false;
        else if( delta == 0){
            int i = 0;
            while( i < s.length() && s.charAt(i) == t.charAt(i))
{
                i++;
            }
            if(i == sl) return false;
            else return s.substring(i+1).equals(t.substring(i+1)
);
        }else{
            return sl > tl ? isOneDelete(s, t) : isOneDelete(t,
s);
        }
    }

    boolean isOneDelete(String l, String s){
        int i=0;
        for(; i< s.length(); i++){
            if(l.charAt(i) != s.charAt(i)) break;
        }

        if(i == s.length()) return true;

        return s.substring(i).equals(l.substring(i+1));
    }
}
```

165. Compare Version Numbers

Compare two version numbers version1 and version2.

If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . character.

The . character does not represent a decimal point and is used to separate number sequences.

For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

```
0.1 < 1.1 < 1.2 < 13.37
```

```
public class Solution {
    public int compareVersion(String version1, String version2)
{
        String[] v1 = version1.split("\\.");
        String[] v2 = version2.split("\\.");
        Comparator<String> comp = new Comparator<String>(){
            @Override
            public int compare(String s1, String s2){
                int p = 0;
                int q = 0;
                while(p < s1.length() && s1.charAt(p) == '0') p+
+;
                while(q < s2.length() && s2.charAt(q) == '0') q+
+;
                if(p >= s1.length() && q >= s2.length()) return 0
;
                else if(p >= s1.length() || q >= s2.length()){
                    if(p >= s1.length()) return -1;
                    else return 1;
```

```
                }else{
                    String ss1 = s1.substring(p);
                    String ss2 = s2.substring(q);
                    if(ss1.length() > ss2.length()) return 1;
                    else if(ss1.length() < ss2.length()) return -
1;
                    else return ss1.compareTo(ss2);
                }
            }
        };

        int i =0;
        for(; i< v1.length || i<v2.length; i++){
            String vs1 = i< v1.length ? v1[i] : "";
            String vs2 = i< v2.length ? v2[i] : "";

            int res = Objects.compare(vs1, vs2, comp);
            if(res != 0) return res > 0 ? 1 : -1;
        }

        return 0;


    }
}
```

167 Two Sum II - Input array is sorted

Given an array of integers that is already sorted in ascending order, 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.

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

Input: numbers={2, 7, 11, 15}, target=9 Output: index1=1, index2=2

```
public class Solution {
    public int[] twoSum(int[] num, int target) {
        int l =0;
        int r = num.length -1;
        while(l < r){
            if(num[l] + num[r] == target) return new int[]{l+1,r+
1};// convert into 1-based.
            else if(num[l] + num[r] > target){
                r--;
            }else{
                l++;
            }
        }

        return new int[0];
    }
}
```

168. Excel Sheet Column Title

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

```
    1 -> A
    2 -> B
    3 -> C
    ...
    26 -> Z
    27 -> AA
    28 -> AB
```

The solution

1. turn 1-based col number to 0-based in each round.
2. get the aplabetic representation from left to right, by using mod operation.
3. reduce the number using / operation. disgard the remainder. use quotient for next round.

```
public class Solution {
    public String convertToTitle(int n) {
        StringBuilder sb = new StringBuilder();
        while(n >0 ){
            sb.append((char)('A' + (--n)%26));
            n = n/26;
        }
        return sb.reverse().toString();

    }
}
```

169. Majority Element

Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

related question: Majority Element II

brute force solution is to count. one O(n) solution is to keep track of the major number from the beginning, in this way the major number may vary during the process, but at the end, only the real major number will remain. for each pair of number, if they are same increase count of major number, if not, this pair will be removed.

```
public class Solution {
    public int majorityElement(int[] nums) {
        int count = 0;
        int major = 0;
        for(int val : nums){
            if(count == 0){
                major = val;
                count =1;
                continue;
            }
            if(major == val) count++;
            else count--;
        }
        return major;

    }
}
```

170. Two Sum III - Data structure design

Design and implement a TwoSum class. It should support the following operations: add and find.

add - Add the number to an internal data structure. find - Find if there exists any pair of numbers which sum is equal to the value.

For example,

```
add(1); add(3); add(5);
find(4) -> true
find(7) -> false
```

```
public class TwoSum {
    Map<Integer, Integer> map = new HashMap<>();
    // Add the number to an internal data structure.
    public void add(int number) {
        if(map.containsKey(number)){
            map.put(number, map.get(number) + 1);
        }else{
            map.put(number, 1);
        }
    }

    // Find if there exists any pair of numbers which sum is equ
al to the value.
    public boolean find(int value) {
        for(Integer key : map.keySet()){
            int target = value -key;
            if(map.containsKey(target)){
                if(target == key && map.get(key) <2) continue;
                return true;
            }
        }
        return false;
    }
}


// Your TwoSum object will be instantiated and called as such:
// TwoSum twoSum = new TwoSum();
// twoSum.add(number);
// twoSum.find(value);
```

171. Excel Sheet Column Number

Related to question Excel Sheet Column Title

Given a column title as appear in an Excel sheet, return its corresponding column number.

link to 168 Excel Sheet Column Title

For example:

```
    A -> 1
    B -> 2
    C -> 3
    ...
    Z -> 26
    AA -> 27
    AB -> 28
```

```java
public class Solution {
    public int titleToNumber(String s) {
        int res = 0;
        for(int i=0; i<s.length(); i++){
            res = res*26 + (s.charAt(i) - 'A' + 1);
        }
        return res;
    }
}
```

174. Dungeon Game

The demons had captured the princess (P) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (K) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess. The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately. Some of the rooms are guarded by demons, so the knight loses health (negative integers) upon entering these rooms; other rooms are either empty (0's) or contain magic orbs that increase the knight's health (positive integers). In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.

Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.

For example, given the dungeon below, the initial health of the knight must be at least 7 if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN.

```
//TODO: add the table
```

The trick here is to go from dungeon place holds princess, and at each point, the knight's minimum hp should be either 1 or more(if the dungeon cell contains demons). depends on later need, this why you have to go from bottom right to top-left.

```
public class Solution {
    public int calculateMinimumHP(int[][] dungeon) {
        int m = dungeon.length;
        int n = dungeon[0].length;
        int[][] hp = new int[m][n];

        hp[m-1][n-1] = Math.max(1, 1 - dungeon[m-1][n-1]);
        for(int k=n-2; k>=0; k--){
            hp[m-1][k] = Math.max(1, hp[m-1][k+1] - dungeon[m-1]
[k]);
        }
        for(int k = m-2; k>=0; k--){
            hp[k][n-1] = Math.max(1, hp[k+1][n-1] - dungeon[k][n-
1]);
        }
        for(int i= m-2; i>=0; i--){
            for(int j = n-2; j>=0; j--){
                hp[i][j] = Math.max(1, Math.min(hp[i+1][j], hp[i
][j+1]) - dungeon[i][j]);
            }
        }

        return hp[0][0];
    }
}
```

172. Factorial Trailing Zeroes

Given an integer n, return the number of trailing zeroes in n!.

Note: Your solution should be in logarithmic time complexity.

> there should no multiply anywhere in the code, other wise there will be overflow.

consider 125, there are multiple fives in the number to end up trailing zeros. so you need to find such number recursively.

```
public class Solution {
    public int trailingZeroes(int n) {
        int res = 0;

        while(n >= 5){
            res += n/5;
            n /= 5;
        }
        return res;
    }
}
```

173. Binary Search Tree Iterator

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

Note: next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree.

This is another version of in order traversal of tree.

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */

public class BSTIterator {
    TreeNode node;
    Stack<TreeNode> stack = new Stack<>();

    public BSTIterator(TreeNode root) {
        node = root;
    }

    /** @return whether we have a next smallest number */
    public boolean hasNext() {
        return !stack.isEmpty() || node != null;
    }

    /** @return the next smallest number */
    public int next() {
      while(node != null){
```

```
            stack.push(node);
            node = node.left;
        }
        TreeNode top = stack.peek();
        stack.pop();
        node = top.right;

        return top.val;

    }
}

/**
 * Your BSTIterator will be called like this:
 * BSTIterator i = new BSTIterator(root);
 * while (i.hasNext()) v[f()] = i.next();
 */
```

179. Largest Number

Given a list of non negative integers, arrange them such that they form the largest number.

For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330.

Note: The result may be very large, so you need to return a string instead of an integer.

> Notice for Java
>
> - int[] and Comparator is illegal.
> - String compareTo and Comparator's compare.

```
public class Solution {
    public String largestNumber(int[] nums) {
        Integer[] ints = new Integer[nums.length];
        int i=0;
        for(int v : nums){
            ints[i++] =v;
        }
        Comparator<Integer> comp = new Comparator<Integer>(){
            @Override
            public int compare(Integer s1, Integer s2){
                String c1 = Integer.toString(s1) + Integer.toStr
ing(s2);
                String c2 = Integer.toString(s2) + Integer.toStr
ing(s1);

                return 0 - c1.compareTo(c2);
            }
        };

        Arrays.sort(ints, comp);

        if(ints[0] == 0) return "0";

        StringBuilder sb = new StringBuilder();
        for(int v : ints){
            sb.append(v);
        }

        return sb.toString();
    }
}
```

187. Repeated DNA Sequences

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

For example,

Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",

Return: ["AAAAACCCCC", "CCCCCAAAAA"].

```
 public class Solution {
    public List<String> findRepeatedDnaSequences(String s) {
        List<String> res = new ArrayList<>();
        Map<Integer, Integer> keys = new HashMap<>();// use hash
map to track if the sequence showed, and whether is duplicated.

        for(int i=10; i<= s.length(); i++){
            String seq = s.substring(i-10, i);
            int key = hashKey(seq);
            if(keys.containsKey(key)){
                if(keys.get(key) == 1){
                    res.add(seq);
                }
                keys.put(key, 2);
            }else{
                keys.put(key, 1);
            }
        }
        return res;
    }

    private int hashKey(String s){
        int res = 0;
        for(int i = s.length()-1; i>=0; i--){
```

```
            char ch = s.charAt(i);
            res <<= 2;
            switch (ch){
                case 'A':
                    res |= 0b00;
                    break;
                case 'C':
                    res |= 0b01;
                    break;
                case 'G':
                    res |= 0b10;
                    break;
                case 'T':
                    res |= 0b11; //0x11 means hexdecimal, so it
is 0b00010001.. which is not 3. in binary.
                    break;
            }

        }
        return res;
    }
}
```

188. Best Time to Buy and Sell Stock IV

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 k transactions.

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

what is the current day's k transactions profit, what is previous day's k-transactions profit. this it local k transactions profit, get the max of this IS global.

For each day i, [1..n], what is the maximal profit after j transaction, [1..k]

- at each day i, there will be a profit(or not, depends wether p[i] > p[i-1]) for **transaction j**, use a local array to keep track of **transaction j**'s profit built along the days. **local[j] = Math.max(global[j-1] + profit, local[j] + profit)**
- update global profit.

```
public class Solution {
    public int maxProfit(int k, int[] prices) {
        int res = 0;
        if(prices == null || prices.length <=1 ) return res;
        if(k >= prices.length) return getAllProfit(prices);
        int[] local = new int[k+1];
        int[] global = new int[k+1]; // for each day i, transact
ion count is j, what is the maximal value.
        for(int i=1; i < prices.length; i++){
            int profit = prices[i] - prices[i-1];
            for(int j = k; j>=1 ; j--){
                local[j] = Math.max(global[j-1] + profit, local[
j] + profit); // nothing to do with local[j-1];
                global[j] = Math.max(local[j], global[j]);
            }
        }

        return global[k];
    }

    int getAllProfit(int[] p){
        int res = 0;
        for(int i=1; i<p.length; i++){
            if(p[i] > p[i-1]){
                res += p[i] - p[i-1];
            }
        }
        return res;
    }
}
```

189. Rotate Array

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

Note: Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

- Solution 1. rotate one by one, make a copy of last element, shift all element 1 distance to right, put last element back into the first position, continue k steps.
- Solution 2. use extra space to save the last k elements from array. move first n-k elements to the right by k steps. copy the extra array back into first k elements.
- reverse 3 times.

```java
public class Solution {
    public void rotate(int[] nums, int k) {
        int n = nums.length;
        if(n <=1) return; // no need to rotate.
        k = k % n;
        if(k == 0) return;

        swap(nums, 0, n-1);
        swap(nums, 0, k-1);
        swap(nums, k, n-1);


    }

    void swap(int[] nums, int l, int r){
        while(l<r){
            int t = nums[l];
            nums[l] = nums[r];
            nums[r] = t;
            l++;
            r--;
        }
    }
}
```

190. Reverse Bit

Reverse bits of a given 32 bits unsigned integer.

For example, given input 43261596 (represented in binary as 00000010100101000001111010011100), return 964176192 (represented in binary as 00111001011110000010100101000000).

Follow up:

If this function is called many times, how would you optimize it?

the solution is of constant time ,which is 32, to improve it on large scale, your code need to cache certain result, and reduce the number of operation, i.e. from 32 to smaller number.

```
public class Solution {
    // you need treat n as an unsigned value
    public int reverseBits(int n) {
        int res = 0;
        int x = 31;
        while(x >= 0){
            res = res ^ ((n & 0x1) << (x--));
            n >>= 0x1;
        }

        return res;
    }
}
```

constant time of 8, calculate the reserse result of each 4 bits number.

```
public class Solution {
    // you need treat n as an unsigned value
    int[]  cache = new int[]{0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,
15};
    public int reverseBits(int n) {
        int res = 0;
        int mask = 0xF;
        for(int i=0; i<8;i++){
            res = res << 4;
            res |= cache[mask&n];
            n >>= 4;
        }

        return res;
    }
}
```

1. House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police

DP: while robbing i-th house, the max value is either rob the i-th house and the max value **BEFORE** the previous houses, or don't rob i-th house, and max value of robbing all previous houses

```
public class Solution {
    public int rob(int[] nums) {
        if(nums == null || nums.length == 0) return 0;
        if(nums.length < 2) return nums[0];

        int[] res = new int[nums.length];
        res[0] = nums[0];
        res[1] = nums[0] > nums[1] ? nums[0] : nums[1];
        for(int i=2; i< nums.length; i++){
            res[i] = Math.max(res[i-2] + nums[i], res[i-1]);
        }

        return res[nums.length-1];

    }
}
```

using constant space. you only need prev max, current max, in the next round, prev + nums[i] will be the current, and max(prev, current) will the prev.

```java
public class Solution {
    public int rob(int[] nums) {
        if(nums ==null || nums.length == 0) return 0;
        if(nums.length == 1) return nums[0];

        int prev=0; // res[i-1]
        int cur=0; // res[i];


        for(int i=0; i< nums.length;i++){
            int m = prev, n = cur;
            cur = prev + nums[i];
            prev = Math.max(m,n);
        }

        return Math.max(cur,prev);
    }
}
```

```java
public class Solution {
    public int rob(int[] nums) {
        if(nums == null || nums.length == 0) return 0;
        if(nums.length == 1) return nums[0];

        int prev2 = nums[0];
        int prev1 = nums[0] > nums[1] ? nums[0] : nums[1];

        for(int i=2; i< nums.length; i++){
            int tmp = prev1;
            prev1 = Math.max(nums[i]+prev2, prev1);
            prev2 = tmp;
        }
        return prev1;
    }
}
```

199. Binary Tree Right Side View

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

For example: Given the following binary tree,

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

You should return [1, 3, 4].

> BFS, instead of saving all the numbers in each level, save the last one.

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        if(root == null) return res;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        while(!queue.isEmpty()){
            int size = queue.size();
            for(int i=1; i<= size; i++){
                TreeNode node = queue.poll();
                if(node.left != null) queue.offer(node.left);
                if(node.right != null) queue.offer(node.right);
                if(i == size) res.add(node.val);
            }
        }
        return res;
    }
}
```

200. Number of Islands

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
11110
11010
11000
00000
Answer: 1
```

Example 2:

```
11000
11000
00100
00011
Answer: 3
```

This is a classic dfs search question.

```
public class Solution {
    public int numIslands(char[][] grid) {
        int count = 0;
        if(grid.length==0 || grid[0].length == 0) return count;
        boolean[][] visited = new boolean[grid.length][grid[0].l
ength];
        for(int i=0; i< visited.length;i++){
            for(int j=0; j< visited[0].length; j++){
                visited[i][j] = false;
            }
        }
        for(int i=0; i< grid.length; i++){
            for(int j=0; j< grid[0].length;j++){
                if(grid[i][j] == '1' && !visited[i][j]){
                    count++;
                    dfs(grid, visited, i,j);
                }
            }
        }
        return count;
    }

    private void dfs(char[][] grid, boolean[][] visited, int i,
int j){
        if(i<0 || i>= grid.length || j <0 || j>=grid[0].length |
| grid[i][j] == '0' || visited[i][j]){
            return;
        }
        visited[i][j] = true;
        dfs(grid, visited, i,j+1);
        dfs(grid, visited, i,j-1);
        dfs(grid, visited, i+1,j);
        dfs(grid, visited, i-1,j);
    }
}
```

you can avoid the visited matrix by set the value of grid[i][j] to '2' for instance

201. Bitwise AND of Numbers Range

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

For example, given the range [5, 7], you should return 4.

> If two numbers cross the boundary of 2^x, i.e. m < 2^x < n, this means there is no common 1s exist in all the number. so the real question is to find the left-most common 1s, if any.

```
public class Solution {
    public int rangeBitwiseAnd(int m, int n) {
        int res = 0;
        while( m != n){
            m >>= 1;
            n >>= 1;
            res++;
        }

        return m << res;
    }
}
```

202.Happy Number

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example: 19 is a happy number

```
1^2 + 9^2 = 82
8^2 + 2^2 = 68
6^2 + 8^2 = 100
1^2 + 0^2 + 0^2 = 1
```

Will this method ever terminate? yes, the number square sum will not increase for ever. if you get the maximum 3-digit number, 999, 3*9^2 = 243, so the number will go up, then go down.

```java
public class Solution {
    public boolean isHappy(int n) {
        Set<Integer> set = new HashSet<>();
        while(true){
            if(n == 1 ) return true;

            if(set.contains(n)){
                return false;
            }else{
                set.add(n);
            }
            int res =0;
            while(n > 0){
                res += (n%10) * (n%10);
                n /= 10;
            }
            n = res;
        }
    }
}
```

203. Remove Linked List Elements

Remove all elements from a linked list of integers that have value val.

Example

Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6

Return: 1 --> 2 --> 3 --> 4 --> 5

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(val - 1);

        dummy.next =head;
        head = dummy;
        // if head.next is the val, remove it, don't update head
, continue to check new next.
        // otherwise udpate it.
        while(head.next != null){
            if(head.next.val == val){
                head.next = head.next.next;
            }else
                head = head.next;
        }

        return dummy.next;
    }
}
```

204. Count Primes

Count the number of prime numbers less than a non-negative number, n.

> To verify a number is prime, you need to divide n by all the number less than n, to see if remainder is 0, in this case, for each number you need to calculate in such way, so the total complexity in time is O(n^2).
>
> There is a simple way, for each number less than n, you only need to visit it once to tell is a primer or not
>
> Start with 2, for all the 2*k* < *n, they are not prime, continue to 2, since all number less than 3 is not factor of 3, so 3 is prime, continue to 4, since already visited when 2*k, so not, continue this to n.

```
public class Solution {
    public int countPrimes(int n) {
        if(n <= 2) return 0;
        boolean[] isNotPrime = new boolean[n];
        int count =0;
        for(int i= 2; i<n; i++){
            if(!isNotPrime[i]){
                count++;
                for(int k = 2; k*i < n; k++){
                    isNotPrime[k*i] = true;
                }
            }
        }
        return count;

    }
}
```

205. Isomorphic Strings

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

For example,

Given "egg", "add", return true.

Given "foo", "bar", return false.

Given "paper", "title", return true.

Note: You may assume both s and t have the same length.

you can use two maps to map the chars from each string,however, you can use the index information saved inside two strings, if two chars are 'equal', they should have same index.

```
public class Solution {
    public boolean isIsomorphic(String s, String t) {
        if(s.length() != t.length()) return false;
        int[] m1 = new int[256];
        int[] m2 = new int[256];
        for(int i=0; i< s.length(); i++){
            if(m1[s.charAt(i)] != m2[t.charAt(i)]) return false;

            m1[s.charAt(i)] = i+1;
            m2[t.charAt(i)] = i+1;
        }

        return true;
    }
}
```

206 Reverse Linked List

Reverse a singly linked list.

A linked list can be reversed either iteratively or recursively. Could you implement both?

- iterative

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode newHead = null;
        while(head != null){
            ListNode tmp = head.next;
            head.next = newHead;
            newHead = head;
            head = tmp;
        }
        return newHead;

    }
}
```

- recursive to reverse a linked list recursively, we need to keep track of two nodes, the recursive function should return new head, and before jump into the recursive function, we need keep track of the tail of new list.

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseList(ListNode head) {
        if(head == null || head.next == null){
            return head;
        }
        //get new tail.
        ListNode tail = head.next;
        //remove old head.
        head.next = null;
        ListNode newHead = reverseList(tail);
        tail.next = head;

        return newHead;
    }
```

207. Course Schedule

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses?

For example:

2, [[1,0]]

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

2, [[1,0],[0,1]]

There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Note:

The input prerequisites is a graph represented by a list of edges, not adjacency matrices. Read more about how a graph is represented.

click to show more hints.

> Hints: This problem is equivalent to finding if a cycle exists in a directed graph. If a cycle exists, no topological ordering exists and therefore it will be impossible to take all courses.
>
> Topological Sort via **DFS** - A great video tutorial (21 minutes) on Coursera explaining the basic concepts of Topological Sort.
>
> Topological sort could also be done via **BFS**.

```
public class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisit
```

```
es) {
        int[] indegree = new int[numCourses];
        Map<Integer, List<Integer>> map = new HashMap<>();

        for(int i=0; i < prerequisites.length; i++){
            int p = prerequisites[i][1];
            int q = prerequisites[i][0];
            if(map.containsKey(p)){
                map.get(p).add(q);
            }else{
                List<Integer> list = new ArrayList<>();
                list.add(q);
                map.put(p, list);
            }

            indegree[q]++;

        }
        //each course in this queue has no dependency on other
 courses.
        Queue<Integer> queue = new LinkedList<>();

        for(int i=0; i<numCourses; i++){
            if(indegree[i] == 0) queue.offer(i);
        }
        int res = 0;
        while(!queue.isEmpty()){
            int c = queue.poll();
            res++;

            if(map.containsKey(c)){
                List<Integer> dep = map.get(c);
                for(int cc : dep){
                    indegree[cc]--;
                    if(indegree[cc] == 0) queue.offer(cc);
                }
            }
        }

        return res == numCourses;
```

```
    }
}
```

DFS solution, you still need to build the map for the edges, but this time, used a visited[] array to represent if the course is finished or not. for each course:

- mark it as visited
- visits it all dependent courses, if you reach any course twice, then there is a cycle.
- mark it as not visited, visit next course.

> it has three states: unvisited (=0), visited(=1), searching(=-1). The third states is to detect the existence of cycle, while the 2nd state indicate that the vertex is already checked and there is no cycle, there is no need to check this vertex again, since we already verify that there is no circle from this vertex.

```
//taken from web
class Solution {
public:
    bool dfs(int v, vector<int> &visit, vector<set<int> > &gr){
        /* if this line maps to BFS, it means that I already che
cked all not
         * coming into this nodes , there is no cycle, all the n
odes depends
         *  on this node, their in degree decrease by 1.
         *
         *  in other cases, if a node whose dominate node visite
d later,
         *  this node is already marked as 1, then the dominate
node is
         *  visited later, when this node is trying to visited d
ependent
         *  node, it found that the node is already marked as vi
sited and
         *  validated, which means this node will not result in
cycle, then
         *  the dominate node will also return as validated and
visted.
         *  example:
```

```
         *  1 -> 2,  3 -> 2, 4 -> 2
         *  1. visit 1 first, then its dependents, 3, then 3's d
ependents 2, all
         *  is marked 1 , as visited and validated
         *  2. skip 2, 3 cause they are already processed.
         *  3, visit 4, and its dependents 2, which is already m
arked as good.
        if (visit[v] == 1){return true;}
        visit[v] = -1;
        for (auto it = gr[v].begin(); it != gr[v].end(); it++){
            if (visit[*it] == -1 || ! dfs(*it, visit, gr)){
                return false;
            }
        }
        visit[v] = 1;
        return true;
    }

    bool canFinish(int numCourses, vector<pair<int, int>>& prere
quisites) {
        int plen = prerequisites.size();
        vector<int> visit(numCourses,0);
        vector<set<int> > gr(numCourses);

        for (int i=0;i<plen;i++){
            gr[prerequisites[i].second].insert(prerequisites[i].
first);
        }

        for (int i=0;i<numCourses;i++){
            if (!dfs(i, visit, gr)) return false;
        }
        return true;
    }
};
```

208. Implement Trie (Prefix Tree)

Implement a trie with insert, search, and startsWith methods.

```
class TrieNode {
    // Initialize your data structure here.
    Map<Character, TrieNode> children = new HashMap<>();
    public boolean hasWord = false;
    public TrieNode() {

    }

    public void insert(String word, int k){
        if(k == word.length()){
            hasWord = true;
            return;
        }
        Character c = word.charAt(k);
        if(!children.containsKey(c)){
            children.put(c, new TrieNode());
        }
        ((TrieNode)children.get(c)).insert(word, k+1);
    }
    public TrieNode find(String word, int k){
        if(k == word.length()){
            return this;
        }
        Character c = word.charAt(k);
        TrieNode tn = (TrieNode)children.get(c);
        if(tn == null) return null;
        else return tn.find(word, k+1);
    }
}

public class Trie {
    private TrieNode root;

    public Trie() {
```

```
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    public void insert(String word) {
        root.insert(word, 0);
    }

    // Returns if the word is in the trie.
    public boolean search(String word) {
        TrieNode tn = root.find(word, 0);
        return tn != null && tn.hasWord;
    }

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    public boolean startsWith(String prefix) {
        return root.find(prefix, 0) != null;
    }
}

// Your Trie object will be instantiated and called as such:
// Trie trie = new Trie();
// trie.insert("somestring");
// trie.search("key");
```

A clear/ better solution

```
class TrieNode {
    // Initialize your data structure here.
    TrieNode[] children = new TrieNode[26];
    String word = "";
    public TrieNode() {

    }
}

public class Trie {
    private TrieNode root;
```

```
    public Trie() {
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    public void insert(String word) {
        TrieNode node = root;
        for(char ch : word.toCharArray()){
            if(node.children[ch - 'a'] == null){
                node.children[ch-'a'] = new TrieNode();
            }
            node = node.children[ch - 'a'];
        }

        node.word = word;
    }

    // Returns if the word is in the trie.
    public boolean search(String word) {
        TrieNode node = root;
        for(char ch : word.toCharArray()){
            if(node.children[ch - 'a'] == null) return false;
            node = node.children[ch - 'a'];
        }
        return node.word.equals(word);
    }

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    public boolean startsWith(String prefix) {
        TrieNode node = root;
        for(char ch : prefix.toCharArray()){
            if(node.children[ch - 'a'] == null) return false;
            node = node.children[ch - 'a'];
        }
        return true;
    }
}
```

```
// Your Trie object will be instantiated and called as such:
// Trie trie = new Trie();
// trie.insert("somestring");
// trie.search("key");
```

209. Minimum Size Subarray Sum

Given an array of n positive integers and a positive integer s, find the minimal length of a subarray of which the sum ≥ s. If there isn't one, return 0 instead.

For example, given the array [2,3,1,2,4,3] and s = 7, the subarray [4,3] has the minimal length under the problem constraint.

> Two pointers

```
public class Solution {
    public int minSubArrayLen(int s, int[] nums) {
        if(nums == null || nums.length == 0) return 0;

        int len = nums.length +1;
        int start =0;
        int end = 0;
        int sum = 0;
        while(end < nums.length){// if end ever reach length,
                                 //which means start already move
d to a point you need to break;
            while(sum < s && end < nums.length){
                sum += nums[end++];
            }
            while(sum >= s){
                len = Math.min(len, end - start);
                sum -= nums[start++];
            }
        }

        return len == nums.length + 1 ? 0 : len;
    }
}
```

```java
public class Solution {
    public int minSubArrayLen(int s, int[] nums) {
        int len = nums.length + 1;
        int sum = 0;
        int k = 0;
        for(int i=0; i<nums.length; ){
            if (sum < s) sum += nums[i++]; // i++ has to be here
 , next block needs its value be updated.

            while(sum >=s){
                len = Math.min(len, i - k);
                sum -= nums[k++];
            }

        }

        return len == nums.length+1 ? 0 : len;
    }
}
```

210. Course Schedule II

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

For example:

```
2, [[1,0]]
```

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1]

```
4, [[1,0],[2,0],[3,1],[3,2]]
```

There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is[0,2,1,3].

Note: The input prerequisites is a graph represented by a list of edges, not adjacency matrices. Read more about how a graph is represented.

```
public class Solution {
    public int[] findOrder(int numCourses, int[][] prerequisites
) {
        int[] indegree = new int[numCourses];
        Map<Integer, List<Integer>> map = new HashMap<>();

        for(int[] edge : prerequisites){
```

```
            int x = edge[1]; // x decide y, y has indegree.
            int y = edge[0];
            if(!map.containsKey(x)){
                map.put(x, new ArrayList<>());
            }
            map.get(x).add(y);
            indegree[y]++;
        }

        Queue<Integer> queue = new LinkedList<>();
        int[] res = new int[numCourses];
        int k = 0;
        for(int i=0; i< indegree.length; i++){
            if(indegree[i] == 0){
                queue.offer(i);
            }
        }

        while(!queue.isEmpty()){
            int node = queue.poll();
            res[k++] = node;
            List<Integer> list = map.get(node);
            if(list == null) continue;
            for(int end : list){
                indegree[end]--;
                if(indegree[end] == 0) queue.offer(end);
            }
        }

        if( k == numCourses) return res;
        return new int[0];
    }
}
```

212. Word Search II

Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

For example, Given words = ["oath","pea","eat","rain"] and board =

```
[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]
```

Return ["eat","oath"].

Note:

You may assume that all inputs are consist of lowercase letters a-z.

this question requires usage of Trie, 208. Implement Trie (Prefix Tree) and 79 Word Search

```
public class Solution {
class TrieNode {
    // Initialize your data structure here.
    TrieNode[] children = new TrieNode[26];
    String word = "";
    public TrieNode() {

    }
}

class Trie {
```

```
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    public void insert(String word) {
        TrieNode node = root;
        for(char ch : word.toCharArray()){
            if(node.children[ch - 'a'] == null){
                node.children[ch-'a'] = new TrieNode();
            }
            node = node.children[ch - 'a'];
        }

        node.word = word;
    }

    // Returns if the word is in the trie.
    public boolean search(String word) {
        TrieNode node = root;
        for(char ch : word.toCharArray()){
            if(node.children[ch - 'a'] == null) return false;
            node = node.children[ch - 'a'];
        }
        return node.word.equals(word);
    }

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    public boolean startsWith(String prefix) {
        TrieNode node = root;
        for(char ch : prefix.toCharArray()){
            if(node.children[ch - 'a'] == null) return false;
            node = node.children[ch - 'a'];
        }
        return true;
    }
}
```

```
    private Trie dict = new Trie();
    private Set<String> res = new HashSet<>();
    public List<String> findWords(char[][] board, String[] words)
 {
        if(board == null || board.length == 0 || board[0].length
 == 0) return new ArrayList<String>(res);
        for(String s : words){
            dict.insert(s);
        }
        boolean[][] visited = new boolean[board.length][board[0]
.length];
        for(int i =0 ; i< board.length; i++){
            for(int j =0; j< board[0].length; j++){
                dfs(board, visited, i, j, "");
            }
        }

        return new ArrayList<String>(res);
    }

    private void dfs(char[][] board, boolean[][] visited, int x,
int y, String current){
        if(x < 0 || y < 0 || x >= board.length || y >= board[0].
length || visited[x][y]) return;

        current += board[x][y];
        if(!dict.startsWith(current)) return;
        if(dict.search(current)){
            res.add(current);
        }
        visited[x][y] = true;

        dfs(board, visited, x+1, y, current);
        dfs(board, visited, x-1, y, current);
        dfs(board, visited, x, y+1, current);
        dfs(board, visited, x, y-1, current);

        visited[x][y] = false;
    }
```

```
}
```

214 Shortest Palindrome

Given a string S, you are allowed to convert it to a palindrome by adding characters in front of it. Find and return the shortest palindrome you can find by performing this transformation.

For example:

Given "aacecaaa", return "aaacecaaa".

Given "abcd", return "dcbabcd".

```
public class Solution {
    public String shortestPalindrome(String s) {
        if(s.length() <=1) return s;
        String reverse = (new StringBuilder(s)).reverse().toStri
ng();
        String target = s + "*" + reverse;
        int[] next = new int[target.length()];
        next[0] = 0;
        for(int i=1; i< target.length(); i++){
            int k = next[i-1];
            while(k>0 && target.charAt(k) != target.charAt(i)) k
 = next[k-1];
            next[i] = target.charAt(k) == target.charAt(i) ? k+1
 : k;
        }

        return reverse.substring(0, reverse.length()-next[target
.length()-1]) + s;
    }
}
```

215. Kth Largest Element in an Array

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example,

Given [3,2,1,5,6,4] and k = 2, return 5.

Note: You may assume k is always valid, 1 ≤ k ≤ array's length.

Solution first. remain a min-heap of size k, if size exceed k, pop the top value.

```
public class Solution {
    public int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>()
;
        for(int i : nums){
            pq.add(i);
            if(pq.size() > k ) pq.poll();
        }

        return pq.poll();
    }
}
```

Java's PriorityQueue implementation allows duplicates instance of same value been added,

> To have non-duplicate PriorityQueue, you need to override PriorityQueue's offer/add function.

216. Combination Sum III

Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Example 1:

Input: k = 3, n = 7

Output:

[[1,2,4]]

Example 2:

Input: k = 3, n = 9

Output:

[[1,2,6], [1,3,5], [2,3,4]]

> backtracking

```
public class Solution {

    List<List<Integer> > res = new ArrayList<>();
    public List<List<Integer>> combinationSum3(int k, int n) {
        sum3(1, 0, n, 0, k, new ArrayList<Integer>());
        return res;
    }


    private void sum3(int val, int curSum, int n, int count, int
 k, List<Integer> list){
        if(count > k || curSum > n) return;

        if(count == k && curSum == n){
            List<Integer> l = new ArrayList<>(list);
            res.add(l);
            return;
        }

        for(int i=val; i<10 ; i++){
            list.add(i);
            sum3(i+1, curSum+i, n, count+1, k, list);
            list.remove(list.size()-1);

        }

    }
}
```

> better form

```
public class Solution {
    List<List<Integer>> res = new ArrayList<>();

    public List<List<Integer>> combinationSum3(int k, int n) {
        if(k > n) return res;

        sum3(1, 9, 0, n, 0, k, new ArrayList<Integer>());
        return res;

    }

    void sum3(int start, int stop, int cur, int target, int coun
t, int k, List<Integer> list){
        if(count > k) return;
        if(count == k){
            if(cur == target){
                res.add(new ArrayList<Integer>(list));
            }
            return;
        }

        for(int i=start; i<=stop; i++){
            if(cur + i > target) break;
            list.add(i);
            sum3(i+1, stop, cur+i, target, count+1, k, list);
            list.remove(list.size()-1);
        }
    }
}
```

217. Contains Duplicate

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

links to related : 219 Contains Duplicate II

```java
public class Solution {
    public boolean containsDuplicate(int[] nums) {
        Set<Integer> set = new HashSet<>();
        for(int val : nums){
            if(set.contains(val)) return true;
            set.add(val);
        }
        return false;
    }
}
```

219. Contains Duplicate II

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the difference between i and j is at most k.

related links : 217 Contains Duplicate

```
public class Solution {
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        Map<Integer,Integer> map = new HashMap<>();
        for(int i=0;i< nums.length; i++){
            int val = nums[i];
            if(map.containsKey(val)){
                if(i - map.get(val) <= k) return true;
            }
            map.put(val, i);
        }
        return false;

    }
}
```

220. Contains Duplicate III

Given an array of integers, find out whether there are two distinct indices i and j in the array such that the difference between nums[i] and nums[j] is at most t and the difference between i and j is at most k.

1. there is no need for the window to be fully sorted, so uses a treeset to partially sort the window.
2. if the next index exceed the window side, remove the frist window element.

```
public class Solution {
    public boolean containsNearbyAlmostDuplicate(int[] nums, int
 k, int t) {
        if(k < 1 || t <0 || nums == null || nums.length < 2) ret
urn false;
        SortedSet<Long> set = new TreeSet<>();
        for(int j=0; j<nums.length; j++){

            SortedSet<Long> sub = set.subSet((long)nums[j] -t , (
long)nums[j] + t+1);
            if(!sub.isEmpty()) return true;


            if(j>=k){
                set.remove((long)nums[j-k]);
            }

            set.add((long)nums[j]);
        }

        return false;
    }
}
```

221. Maximal Square

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing all 1's and return its area.

For example, given the following matrix:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Return 4.

DP, if `matrix[i][j]` belongs to a square and it is the bottom-right element, in this case, the left, top, and top-left neighbours of this are all required to bottom-right element of another square.

```
public class Solution {
    public int maximalSquare(char[][] matrix) {
        if(matrix == null || matrix.length == 0 ) return 0;
        int[][] dp = new int[matrix.length][matrix[0].length];

        int edge = 0;
        for(int i =0 ;i < matrix.length; i++){
            dp[i][0] = matrix[i][0] == '0' ? 0 : 1;
            edge = Math.max(dp[i][0], edge);
        }
        for(int i=0; i< matrix[0].length; i++){
            dp[0][i] = matrix[0][i] == '0' ? 0 : 1;
            edge = Math.max(dp[0][i], edge);
        }

        for(int i=1; i< matrix.length; i++){
            for(int j= 1; j < matrix[0].length; j++){
                if(matrix[i][j] == '1'){
                        dp[i][j] = Math.min(dp[i-1][j-1], Math.m
in(dp[i-1][j], dp[i][j-1])) + 1;
                }
                edge = Math.max(edge, dp[i][j]);

            }
        }

        return edge * edge;
    }
}
```

222. Count Complete Tree Nodes

Given a complete binary tree, count the number of nodes.

Definition of a complete binary tree from Wikipedia: In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

> for complete tree, if h(l) == h(r) then then the nodes are 2^h -1;

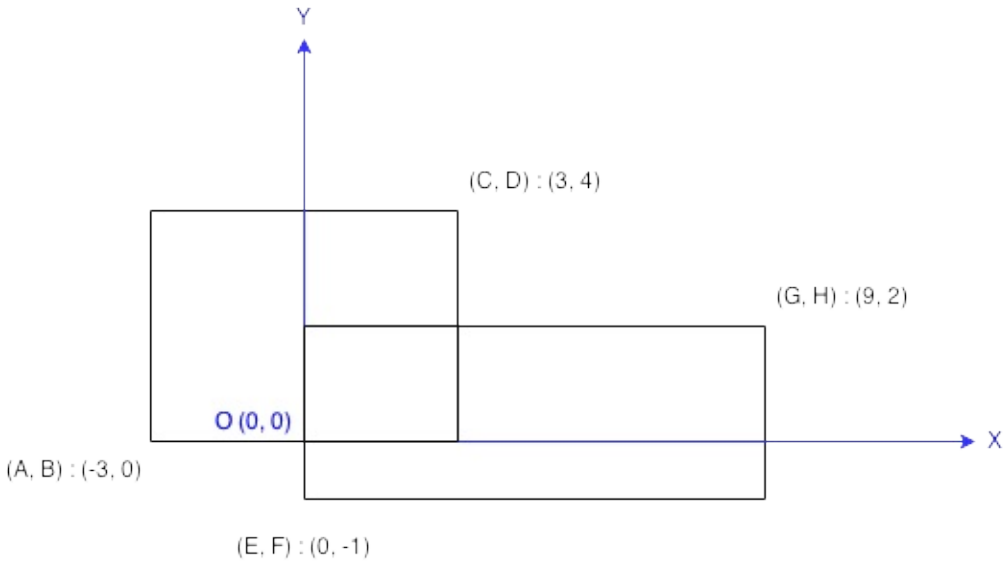
```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int countNodes(TreeNode root) {

        if(root == null) return 0;
        int l = getHeight(root, true);
        int r = getHeight(root, false);

        if(l == r) return (2 <<(l-1)) -1;
        return countNodes(root.left) + countNodes(root.right) +1
;

    }

    private int getHeight(TreeNode root, boolean toLeft){
        if(root ==null) return 0;
        int h = 1;
        if(toLeft){
            while(root.left != null){
```

```
                root = root.left;
                h++;
            }
        }else{
            while(root.right != null){
                root = root.right;
                h++;
            }
        }
        return h;
    }
}
```

223. Rectangle Area My Submissions

Find the total area covered by two rectilinear rectangles in a 2D plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.



Rectangle Area Assume that the total area is never beyond the maximum possible value of int.

```
public class Solution {
    public int computeArea(int A, int B, int C, int D, int E, int
 F, int G, int H) {
        int a1 = (C-A) * (D-B) ;
        int a2 = (G-E) * (H-F) ;
        if (E >= C || F >= D || B >= H || A >= G) return a1+a2;

        int intersect = (Math.min(C, G) - Math.max(A, E)) * (Mat
h.min(D, H) - Math.max(B, F));

        return  (a1+a2 - intersect);
    }
}
```

the following python solution is not suitable for java in the case of large number but no overlap. a pythonic solution:

```
class Solution:
    # @param {integer} A
    # @param {integer} B
    # @param {integer} C
    # @param {integer} D
    # @param {integer} E
    # @param {integer} F
    # @param {integer} G
    # @param {integer} H
    # @return {integer}
    def computeArea(self, A, B, C, D, E, F, G, H):
        sums = (C - A) * (D - B) + (G - E) * (H - F)
        return sums - max(min(C, G) - max(A, E), 0) * max(min(D,
 H) - max(B, F), 0)
```

225. Implement Stack using Queues

```
push(x) -- Push element x onto stack.
pop() -- Removes the element on top of the stack.
top() -- Get the top element.
empty() -- Return whether the stack is empty.
```

Notes:

You must use only standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.

Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.

You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

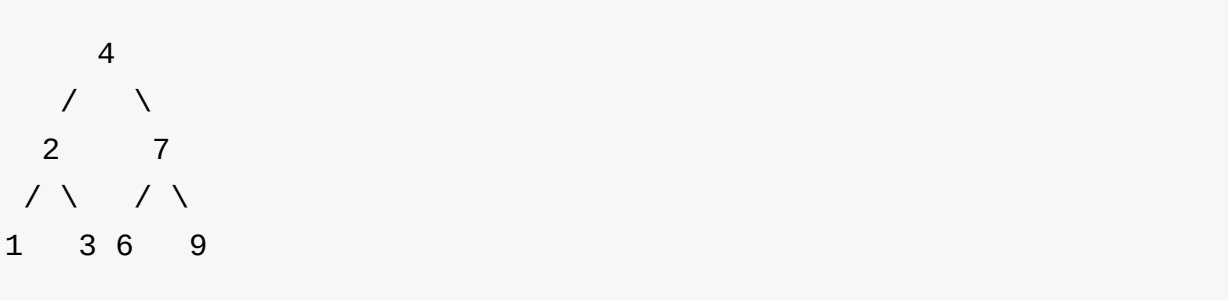
Update (2015-06-11):

The class name of the Java function had been updated to MyStack instead of Stack.

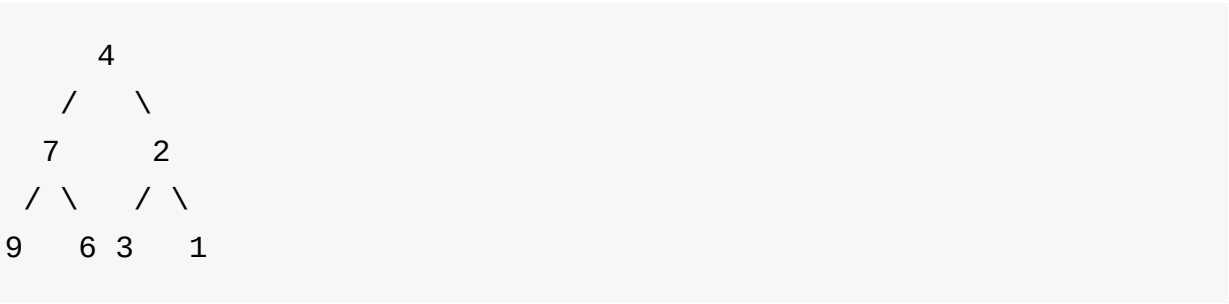
```
class MyStack {

    Queue<Integer> q1 = new LinkedList<>();
    Queue<Integer> q2 = new LinkedList<>();

    // Push element x onto stack.
    public void push(int x) {
        q2.offer(x);
        while(!q1.isEmpty()){
            q2.offer(q1.poll());
        }
        Queue<Integer> q;
        q = q1;
        q1 = q2;
        q2 = q;
    }

    // Removes the element on top of the stack.
    public void pop() {
        q1.poll();
    }

    // Get the top element.
    public int top() {
        return q1.peek();
    }

    // Return whether the stack is empty.
    public boolean empty() {
        return q1.isEmpty();
    }


}
```

226. Invert Binary Tree

Invert a binary tree.



to



> Trivia: This problem was inspired by this original tweet by Max Howell: Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off.

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode invertTree(TreeNode root) {
        invert(root);
        return root;
    }

    void invert(TreeNode root){
        if(root == null) return;
        TreeNode l = root.left;
        root.left = root.right;
        root.right = l;
        invert(root.left);
        invert(root.right);
    }
}


//solution 2:
public class Solution {
    public TreeNode invertTree(TreeNode root) {
        if(root == null) return root;
        TreeNode invertLeft = invertTree(root.left);
        root.left = invertTree(root.right);
        root.right = invertLeft;
        return root;
    }
}
```

228. Summary Ranges

Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given [0,1,2,4,5,7], return ["0->2","4->5","7"].

Solution 1, Extend the original array to include last number twice, this is deal-breaker.

```
public class Solution {
    public List<String> summaryRanges(int[] nums) {
        List<String> res = new ArrayList<>();
        if(nums == null || nums.length == 0) return res;
        int[] extend = new int[nums.length+1];
        for(int i=0; i< nums.length; i++){
            extend[i] = nums[i];
        }
        extend[nums.length] = extend[nums.length-1];
        int start = extend[0];
        for(int i=1; i< extend.length; i++){
            if(extend[i] != extend[i-1] +1){
                int end = extend[i-1];
                if(start == end){
                    res.add(Integer.toString(start));
                }else{
                    res.add(start + "->" + end);
                }

                start = extend[i];
            }
        }

        return res;

    }
}
```

without extending, special check in the ending index

```java
public class Solution {
    public List<String> summaryRanges(int[] nums) {

        List<String> res = new ArrayList<>();
        if(nums == null || nums.length == 0) return res;

        int start = nums[0];
        for(int i=1; i<=nums.length; i++){
            if(i == nums.length){
                if(start == nums[nums.length-1]){
                    res.add(Integer.toString(start));
                }else{
                    res.add(start + "->" + nums[nums.length-1]);
                }
                break;
            }
            if(nums[i] != nums[i-1] +1){
                int end = nums[i-1];
                if(start == end){
                    res.add(Integer.toString(start));
                }else{
                    res.add(start + "->" + end);
                }
                start = nums[i];
            }
        }

        return res;
    }
}
```

229. Majority Element II

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times. The algorithm should run in linear time and in O(1) space.

```java
public class Solution {
    public List<Integer> majorityElement(int[] nums) {
        List<Integer> res = new ArrayList<>();

        int major1 = 0, count1 =0;
        int major2 = 0, count2 =0;

        for(int val : nums){
            if(major1 == val){
                count1++;
            }else if(major2 == val){
                count2++;
            }else if(count1==0){
                major1 = val;
                count1 =1;
                continue;
            }else if(count2 == 0){
                major2 = val;
                count2 =1;
                continue;
            }else{
                count1--;
                count2--;
            }
        }
        count1 =0;
        count2 =0;
        for(int val : nums){
            if(val == major1) count1++;
            else if(val == major2) count2++;// this else is very
 important
        }

        if(count1 > nums.length/3) res.add(major1);
        if(count2 > nums.length/3 ) res.add(major2);
        return res;
    }
}
```

230. Kth Smallest Element in a BST

Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.

Note:

You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Follow up:

What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

Hint:

Try to utilize the property of a BST.

What if you could modify the BST node's structure?

The optimal runtime complexity is O(height of BST).

Related issue: 94. Binary Tree Inorder Traversal

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int kthSmallest(TreeNode root, int k) {
        Stack<TreeNode> stack = new Stack<>();
        TreeNode node = root;
        while(!stack.isEmpty() || node != null){
            while(node != null){
                stack.push(node);
                node = node.left;
            }

            TreeNode top = stack.pop();
            if(--k == 0){
                node = top;
                break;
            }else{
                node = top.right;
            }
        }
        return node.val;
    }
}
```

234. Palindrome Linked List

Given a singly linked list, determine if it is a palindrome.

Follow up: Could you do it in O(n) time and O(1) space?

//by spliting the linked list

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isPalindrome(ListNode head) {
        if(head == null || head.next==null) return true;

        ListNode slow = head;
        ListNode fast = head.next;

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

        fast = slow.next;
        slow.next = null;

        fast = reverse(fast);
        slow = head;

        while(slow != null && fast != null){
            if(slow.val != fast.val) return false;
            slow = slow.next;
```

```
            fast = fast.next;
        }

        return true;
    }

    ListNode reverse(ListNode head){
        ListNode newHead = null;
        while(head != null){
            ListNode tmp = head.next;
            head.next =newHead;
            newHead = head;
            head = tmp;
        }
        return newHead;
    }
}
```

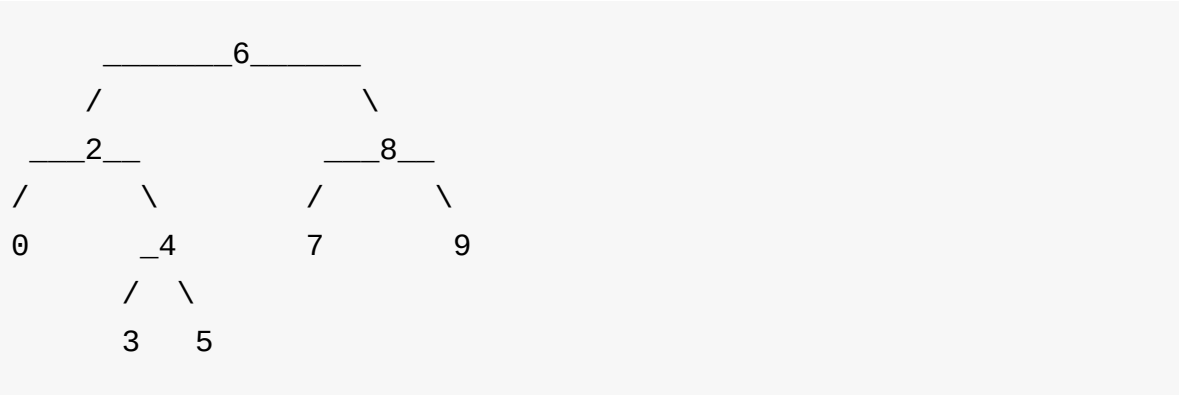
//recursively

```java
public class Solution {
    ListNode left;

    public boolean isPalindrome(ListNode head) {
        left = head;

        boolean result = helper(head);
        return result;
    }

    public boolean helper(ListNode right){

        //stop recursion
        if (right == null)
            return true;

        //if sub-list is not palindrome,  return false
        boolean x = helper(right.next);
        if (!x)
            return false;

        //current left and right
        boolean y = (left.val == right.val);

        //move left to next
        left = left.next;

        return y;
    }
}
```

235. Lowest Common Ancestor of a Binary Search Tree

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself).”



For example, the lowest common ancestor (LCA) of nodes 2 and 8 is 6. Another example is LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

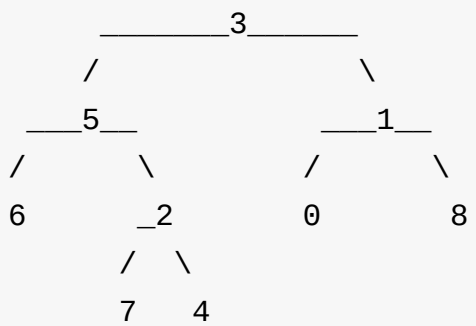
```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode
 p, TreeNode q) {
        if(root == null) return null;
        if(root.val == p.val || root.val == q.val) return root;
        if(root.val > p.val && root.val > q.val){
            return lowestCommonAncestor(root.left, p,q);
        }else if(root.val < p.val && root.val < q.val){
            return lowestCommonAncestor(root.right, p,q);
        }else{
            return root;
        }
    }
}
```

```
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode
 p, TreeNode q) {
        if(root == null) return null;

        if((root.val >= p.val && root.val <= q.val)
                || (root.val <= p.val && root.val >= q.val)) ret
urn root;
        else{
          if(root.val < p.val && root.val < q.val) root = root.r
ight;
          else root = root.left;

          return lowestCommonAncestor(root, p, q);
        }
    }
}
```

236. Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself).”



For example, the lowest common ancestor (LCA) of nodes 5 and 1 is 3. Another example is LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode
 p, TreeNode q) {
        if(root == null) return root;
        if(root == p || root == q) return root;
        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);
        //PLEASE NOTE, IF LEFT AND RIGHT BOTH NOT NULL, MEANS TH
E NODE IS FOUND IN BOTH SUB TREE.
        if(left != null && right != null) return root;
        return left == null ? right : left;

    }


}
```

237. Delete Node in a Linked List

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

Delete the node next to it.

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

238. Product of Array Except Self

Given an array of n integers where n > 1, nums, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

Solve it without division and in O(n).

For example, given [1,2,3,4], return [24,12,8,6].

Follow up: Could you solve it with constant space complexity? (Note: The output array does not count as extra space for the purpose of space complexity analysis.)

```
public class Solution {
    public int[] productExceptSelf(int[] nums) {
        int[] res = new int[nums.length];
        res[0] = 1;

        for(int i=1;i<nums.length; i++){
            res[i] = res[i-1] * nums[i-1];
        }
        int r = 1;
        for(int i= nums.length-1; i>=0;i--){
            res[i] *= r;
            r *= nums[i];
        }
        return res;
    }
}
```

240. Search a 2D Matrix II

rite an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right. Integers in each column are sorted in ascending from top to bottom. For example,

Consider the following matrix:

```
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
```

Given target = 5, return true.

Given target = 20, return false.

```
//O(m+n)
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if(matrix.length == 0 || matrix[0].length == 0) return f
alse;
        int m = matrix.length-1;
        int n = matrix[0].length-1;

        int x = 0;
        int y = n;

        while(x <= m && y >=0){
            if(matrix[x][y] == target) return true;
            else if(matrix[x][y] > target) y--;
            else x++;
        }
        return false;
    }
}
```

O(lgm + logn) solution

Wrong thinking, you cannot do a binary search row-wise, and then a binary search on col-wise, THIS IS TOTAL WRONG THINKING. the correct one is divide the panel to 4 parts

Check this copy paste solution.

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0 || matrix[0].le
ngth == 0) {
            return false;
        }

        int m = matrix.length;
        int n = matrix[0].length;

        return helper(matrix, 0, m - 1, 0, n - 1, target);
```

```
    }

    private boolean helper(int[][] matrix, int rowStart, int row
End, int colStart, int colEnd, int target) {
        if (rowStart > rowEnd || colStart > colEnd) {
            return false;
        }

        int rowMid = rowStart + (rowEnd - rowStart) / 2;
        int colMid = colStart + (colEnd - colStart) / 2;

        if (matrix[rowMid][colMid] == target) {
            return true;
        }

        if (matrix[rowMid][colMid] > target) {
            return helper(matrix, rowStart, rowMid - 1, colStart
, colMid - 1, target) ||
                helper(matrix, rowMid, rowEnd, colStart, colMid
- 1, target) ||
                helper(matrix, rowStart, rowMid - 1, colMid, col
End, target);
        } else {
            return helper(matrix, rowMid + 1, rowEnd, colMid + 1
, colEnd, target) ||
                helper(matrix, rowMid + 1, rowEnd, colStart, col
Mid, target) ||
                helper(matrix, rowStart, rowMid, colMid + 1, col
End, target);
        }
    }
}
```

Given a string of numbers and operators, return all possible results from computing all the different possible ways to group numbers and operators. The valid operators are +, - and *.

Example 1

```
Input: "2-1-1".

((2-1)-1) = 0
(2-(1-1)) = 2
Output: [0, 2]
```

Example 2

```
Input: "2*3-4*5"

(2*(3-(4*5))) = -34
((2*3)-(4*5)) = -14
((2*(3-4))*5) = -10
(2*((3-4)*5)) = -10
(((2*3)-4)*5) = 10
Output: [-34, -14, -10, -10, 10]
```

Classic divide and conquer.

```
public class Solution {
    public List<Integer> diffWaysToCompute(String input) {
        List<Integer> list  = new ArrayList<>();
        for(int i=0; i< input.length(); i++){
            if(Character.isDigit(input.charAt(i))) continue;
            List<Integer> l = diffWaysToCompute(input.substring(0
, i));
            List<Integer> r = diffWaysToCompute(input.substring(
i+1));
            for(Integer li : l){
                for(Integer ri : r){
                    switch (input.charAt(i)){
                        case '+':
                            list.add(li+ri);
                            break;
                        case '-':
                            list.add(li - ri);
                            break;
                        case '*':
                            list.add(li * ri);
                            break;
                        //throws
                    }
                }
            }
        }
        if(list.isEmpty()){
            list.add(Integer.parseInt(input));
        }
        return list;
    }
}
```

242. Valid Anagram

Given two strings s and t, write a function to determine if t is an anagram of s.

For example, s = "anagram", t = "nagaram", return true.

s = "rat", t = "car", return false.

Note: You may assume the string contains only lowercase alphabets.

Follow up: What if the inputs contain unicode characters? How would you adapt your solution to such case?

```
public class Solution {
    public boolean isAnagram(String s, String t) {
        if(s.length() != t.length()) return false;

        int[] set = new int[256];

        int i =0, j =0;
        for(; i<s.length()&&j<t.length(); i++,j++){
            set[s.charAt(i)]++;
            set[t.charAt(j)]--;
        }

        for(int count : set){
            if(count != 0) return false;
        }
        return true;

    }
}
```

243. Shortest Word Distance

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

For example, Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given word1 = “coding”, word2 = “practice”, return 3. Given word1 = "makes", word2 = "coding", return 1.

```
 public class Solution {
    public int shortestDistance(String[] words, String word1, St
ring word2) {
        int l = -1, r= -1;
        int shortest =words.length;
        for(int i=0; i< words.length; i++){
            if(word1.equals(words[i])){
                l = i;
            }else if(word2.equals(words[i])){
                r = i;
            }
            if(l >=0 && r >=0){
                shortest = Math.min(shortest, Math.abs(l-r));
            }
        }
        return shortest;
    }
}
```

244. Shortest Word Distance II

This is a follow up of Shortest Word Distance. The only difference is now you are given the list of words and your method will be called repeatedly many times with different parameters. How would you optimize it?

Design a class which receives a list of words in the constructor, and implements a method that takes two words word1 and word2 and return the shortest distance between these two words in the list.

For example, Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given word1 = “coding”, word2 = “practice”, return 3. Given word1 = "makes", word2 = "coding", return 1.

```
public class WordDistance {
    Map<String, TreeSet<Integer>> map = new HashMap<>();
    public WordDistance(String[] words) {
        for(int i =0; i< words.length; i++){
            TreeSet<Integer> ts;
            if(map.containsKey(words[i])){
                ts = map.get(words[i]);
            }else{
                ts = new TreeSet<Integer>();
                map.put(words[i], ts);
            }
            ts.add(i);
        }
    }

    public int shortest(String word1, String word2) {
        TreeSet<Integer> ts1 = map.get(word1);
        TreeSet<Integer> ts2 = map.get(word2);
        int s = Integer.MAX_VALUE;
        for(Integer i : ts1){
            Integer f = ts2.floor(i);
            if(f != null) s = Math.min(s, i - f);
            Integer h = ts2.higher(i);
            if(h != null) s = Math.min(s, h - i);
        }
        return s;
    }
}

// Your WordDistance object will be instantiated and called as s
uch:
// WordDistance wordDistance = new WordDistance(words);
// wordDistance.shortest("word1", "word2");
// wordDistance.shortest("anotherWord1", "anotherWord2");
```

> O(n) solution, you can look into the arrays more carefully you only need to compare each one at a time, and move the smaller pointer ahead.
>
> To further step improve the performance, if you have a string only show up once, then you can use above solution to achieve a log(k) solution.

```
public class WordDistance {
    Map<String, List<Integer>> map = new HashMap<>();
    public WordDistance(String[] words) {
        for(int i =0; i< words.length; i++){
            List<Integer> ts;
            if(map.containsKey(words[i])){
                ts = map.get(words[i]);
            }else{
                ts = new ArrayList<Integer>();
                map.put(words[i], ts);
            }
            ts.add(i);
        }
    }

    public int shortest(String word1, String word2) {
        List<Integer> ts1 = map.get(word1);
        List<Integer> ts2 = map.get(word2);
        int s = Integer.MAX_VALUE;
        int i = 0;
        int k = 0;
        while(i < ts1.size() && k < ts2.size()){
            s = Math.min(s, Math.abs(ts1.get(i) - ts2.get(k)));
            if(ts1.get(i) > ts2.get(k)){
                k++;
            }else{
                i++;
            }
        }

        return s;
    }
}

// Your WordDistance object will be instantiated and called as s
uch:
// WordDistance wordDistance = new WordDistance(words);
// wordDistance.shortest("word1", "word2");
// wordDistance.shortest("anotherWord1", "anotherWord2");
```

245. Shortest Word Distance III

This is a follow up of Shortest Word Distance. The only difference is now word1 could be the same as word2.

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

word1 and word2 may be the same and they represent two individual words in the list.

For example, Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given word1 = “makes”, word2 = “coding”, return 1. Given word1 = "makes", word2 = "makes", return 3.

```
public class Solution {
    public int shortestWordDistance(String[] words, String word1
, String word2) {
        if(word1.equals(word2)){
            return shortest(words, word1);
        }else{
            int s = Integer.MAX_VALUE;
            int l = -1;
            int r = -1;
            for(int i=0 ; i< words.length; i++){
                if(word1.equals(words[i])) l=i;
                else if(word2.equals(words[i])) r=i;

                if(l>=0 && r>=0) s = Math.min(s, Math.abs(l - r)
);
            }
            return s;
        }
    }
    int shortest(String[] w, String needle){
        int k = -1;
        int s = Integer.MAX_VALUE;
        for(int i=0; i< w.length; i++){
            if(needle.equals(w[i])){
                if(k>=0) s = Math.min(s, i-k);
                k = i;
            }
        }
        return s;
    }
}
```

> folloing is a better solution. you use l and r to track:
>
> - if word1 == word2, then l is current index, and r will be previous index.
> - if not equal, then l and r is for differnt word

```
public class Solution {
    public int shortestWordDistance(String[] words, String word1
, String word2) {
        int l=-1, r= -1;
        int s = Integer.MAX_VALUE;
        for(int i=0; i< words.length; i++){
            if(word1.equals(words[i])){
                l = i;
                if( l >=0 && r >=0){
                    //if word1 == word2, then r is the last inde
x, and l is current index;
                    s = l ==r ? s : Math.min(s, Math.abs(l-r));
                }
            }
            if(word2.equals(words[i])){
                r = i;
                if(l >=0 && r >=0){
                    s = l ==r ? s : Math.min(s, Math.abs(l-r));
                }
            }
        }

        return s;
    }
}
```

246. Strobogrammatic Number

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Write a function to determine if a number is strobogrammatic. The number is represented as a string.

For example, the numbers "69", "88", and "818" are all strobogrammatic.

```
public class Solution {
    public boolean isStrobogrammatic(String num) {
        HashMap<Character, Character> dict = new HashMap<>();
        dict.put('0','0');
        dict.put('1','1');
        dict.put('8','8');
        dict.put('6','9');
        dict.put('9','6');
        int i=0;
        int j = num.length()-1;
        while(i<=j){
            char f = num.charAt(i);
            char b = num.charAt(j);
            if(dict.containsKey(f) && dict.containsKey(b) && dic
t.get(f) == b){
                i++;j--;
            }else{
                return false;
            }
        }
        return true;

    }
}
```

247. Strobogrammatic Number II

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Find all strobogrammatic numbers that are of length = n.

For example, Given n = 2, return ["11","69","88","96"].

```
public class Solution {
    int target;
    public List<String> findStrobogrammatic(int n) {
        target = n;
        return find(n);
    }

    List<String> find(int n){
        List<String> res = new ArrayList<>();
        if(n == 0){
            res.add("");
            return res;
        }
        if(n == 1){
            res.add("1");
            res.add("0");
            res.add("8");
            return res;
        }

        List<String> prev = find(n-2);

        for(String s : prev){
            if(n != target) res.add("0" + s + "0");
            res.add("1" + s + "1");
            res.add("8" + s + "8");
            res.add("6" + s + "9");
            res.add("9" + s + "6");
        }

        return res;
    }
}
```

> iterative

```java
public class Solution {
    public List<String> findStrobogrammatic(int n) {
        List<String> res;
        if( (n&1) == 0){
            List<String> l0 = new ArrayList<>();
            l0.add("");
            res = l0;
        }else{
            List<String> l1 = new ArrayList<>();
            l1.add("1");
            l1.add("0");
            l1.add("8");
            res = l1;
        }

        int i = ((n&1) == 0) ? 2 : 3;
        for(; i<= n; i+=2){
            List<String> tmp = new ArrayList<>();
            for(String s : res){
                if(i != n) tmp.add("0" + s +"0");
                tmp.add("1" + s +"1");
                tmp.add("9" + s +"6");
                tmp.add("6" + s +"9");
                tmp.add("8" + s +"8");
            }

            res = tmp;
        }
        return res;
    }
}
```

249. Group Shifted Strings

Given a string, we can "shift" each of its letter to its successive letter, for example: "abc" -> "bcd". We can keep "shifting" which forms the sequence:

```
"abc" -> "bcd" -> ... -> "xyz"
```

Given a list of strings which contains only lowercase alphabets, group all strings that belong to the same shifting sequence.

For example, given: ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"], Return:

```
[
  ["abc","bcd","xyz"],
  ["az","ba"],
  ["acef"],
  ["a","z"]
]
```

Note: For the return value, each inner list's elements must follow the lexicographic order.

things to consider: 1 does each group allow duplicates, such as ["a", "a"]. 2 how to calculate unique hash key.

> For Java to do a mod operation, you need to do (26 + a)%26, if a == -25, then without 26+, you get a%26 == -25. This is not the case in Python.

```
public class Solution {
    public List<List<String>> groupStrings(String[] strings) {
        Map<String,   List<String>> map = new HashMap<>();

        for(String s : strings){
            boolean added =false;
            for(String k : map.keySet()){
                if(k.length() != s.length()) continue;
                if(isShift(k, s)){
                    added = true;
                    map.get(k).add(s);
                }
            }

            if(!added){
                List<String> l = new ArrayList<>();
                l.add(s);
                map.put(s, l);
            }
        }

        return new ArrayList<List<String>>(map.values());
    }

    boolean isShift(String k, String s){
        int prev = (26 + s.charAt(0) - k.charAt(0))%26;
        for(int i=1; i< k.length(); i++){
            int cur = (26 + s.charAt(i) - k.charAt(i))%26;
            if(cur != prev) return false;
            prev = cur;
        }
        return true;
    }
}
```

```
public class Solution {
    public List<List<String>> groupStrings(String[] strings) {
        Map<String, List<String>> table = new HashMap<>();
        for(String s : strings){
            String key = getHashKey(s);
            if(table.containsKey(key)){
                table.get(key).add(s);
            }else{
                List<String> l = new ArrayList<>();
                l.add(s);
                table.put(key, l);
            }
        }
        List<List<String>> result = new ArrayList<>();
        for(Map.Entry e : table.entrySet()){
            List<String> l = (List<String>)e.getValue();
            Collections.sort(l);
            result.add(l);
        }
        return result;
    }

    private String getHashKey(String s){
        if(s.length() == 0) return "0";
        StringBuilder sb = new StringBuilder();
        for(int i=1; i< s.length();i++){
            int diff =((s.charAt(i) - s.charAt(i-1)) + 26)%26;
            sb.append(diff);
        }
        return sb.toString();
    }
}
```

> This solution is bit faster

```java
public class Solution {
    public List<List<String>> groupStrings(String[] strings) {
        Map<String,   List<String>> map = new HashMap<>();

        for(String s : strings){
            String hash = getHash(s);
            if(map.containsKey(hash)){
                map.get(hash).add(s);
            }else{
                List<String> l = new ArrayList<>();
                l.add(s);
                map.put(hash, l);
            }
        }

        return new ArrayList<List<String>>(map.values());
    }

    String getHash(String s){
        StringBuilder sb = new StringBuilder();
        for(int i=0; i< s.length(); i++){
            sb.append((s.charAt(i) - s.charAt(0) + 26) % 26);
            sb.append('.');// to make sure there is no overlap.
        }

        return sb.toString();
    }
}
```

250. Count Univalue Subtrees

Given a binary tree, count the number of uni-value subtrees.

A Uni-value subtree means all nodes of the subtree have the same value.

For example: Given binary tree,

```
              5
             / \
            1   5
           / \   \
          5   5   5
```

return 4.

> This is a set of questions regarding trees, two pieces of information is needed, the solution is in the return of recursion, return a struct which contains both information.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    class CNode{
        boolean isUnique;
        int val;
        int size;
        CNode(){
            isUnique = true;
            size = 0;
            val = 0;
```

```
        }
    }
    public int countUnivalSubtrees(TreeNode root) {
        CNode c = count(root);
        return c.size;
    }

    CNode count(TreeNode root){
        CNode c = new CNode();
        if( root == null) return c;
        CNode left = count(root.left);
        CNode right = count(root.right);
        if(!left.isUnique || !right.isUnique){
            c.size = left.size + right.size;
            c.isUnique = false;
        }else{
            boolean lUnique = left.size == 0 || left.val == root
.val;
            boolean rUnique = right.size == 0 || right.val == ro
ot.val;
            if( lUnique && rUnique){
                c.size = left.size + right.size + 1;

                c.isUnique = true;
            }else{
                c.size = left.size + right.size;
                c.isUnique = false;
            }
        }
        c.val = root.val;
        return c;
    }
}
```

251. Flatten 2D Vector

Implement an iterator to flatten a 2d vector.

For example, Given 2d vector =

```
[
  [1,2],
  [3],
  [4,5,6]
]
```

By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,2,3,4,5,6].

Attention:

1. filter out empty list when saving the incoming data;
2. in hasNext() function, make sure both pointers are valid;
3. after retrieve data in next(), update pointers.

```
public class Vector2D implements Iterator<Integer> {
    List<List<Integer>> vector = new ArrayList<>();
    int p =0;// which list;
    int q =0;// which number in list;
    public Vector2D(List<List<Integer>> vec2d) {
        for(List<Integer> l : vec2d){
            if(l.size()>0){
                vector.add(l);
            }
        }
    }

    @Override
    public Integer next() {
        int val = vector.get(p).get(q);
        q++;
        if(q == vector.get(p).size()){
            p++;
            q=0;
        }
        return val;
    }

    @Override
    public boolean hasNext() {

        if(p >= vector.size()) return false;
        if(p == (vector.size()-1) && q >= vector.get(p).size())
return false;
        else return true;
    }
}

/**
 * Your Vector2D object will be instantiated and called as such:
 * Vector2D i = new Vector2D(vec2d);
 * while (i.hasNext()) v[f()] = i.next();
 */
```

252. Meeting Rooms

Given an array of meeting time intervals consisting of start and end times [[s1,e1], [s2,e2],...] (si < ei), determine if a person could attend all meetings.

For example,

Given [[0, 30],[5, 10],[15, 20]],

return false.

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public boolean canAttendMeetings(Interval[] intervals) {
        if(intervals == null || intervals.length <= 1) return tr
ue;

        Comparator<Interval> comp = new Comparator<Interval>(){
            @Override
            public int compare(Interval a , Interval b){
                if(a.start == b.start){
                    return a.end - b.end;
                }else{
                    return a.start - b.start;
                }

            }
        };
        Arrays.sort(intervals, comp);
        for(int i=1; i< intervals.length; i++){
            Interval cur = intervals[i];
            Interval prev = intervals[i-1];

            if(cur.start < prev.end) return false;
        }

        return true;
    }
}
```

253. Meeting Rooms II

Given an array of meeting time intervals consisting of start and end times [[s1,e1], [s2,e2],...] (si < ei), find the minimum number of conference rooms required.

For example, Given [[0, 30],[5, 10],[15, 20]], return 2.

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public int minMeetingRooms(Interval[] intervals) {
        if(intervals == null || intervals.length == 0) return 0;
        if(intervals.length == 1) return 1;

        Comparator<Interval> comp = new Comparator<Interval>(){
            @Override
            public int compare(Interval a, Interval b){
                if(a.start == b.start) return a.end - b.end;
                return a.start - b.start;
            }
        };

        Arrays.sort(intervals, comp);
        Comparator<Interval> comp2 = new Comparator<Interval>(){
            @Override
            public int compare(Interval a, Interval b){

                return a.end - b.end;
            }
        };
        PriorityQueue<Interval> queue = new PriorityQueue<>(10,
```

```
comp2);
        int count =1;
        queue.offer(intervals[0]);

        for(int i=1; i< intervals.length; i++){
            Interval top = queue.peek();
            Interval next = intervals[i];
            if(next.start >= top.end){
                queue.poll();
            }else{
                count++;
            }

            queue.offer(next);
        }

        return count;
    }
}
```

> improve 1 : for comp2, you only need to save the ending time queue, improve 2 : you only need to measure the queue size in the end.

255. Verify Preorder Sequence in Binary Search Tree

Given an array of numbers, verify whether it is the correct preorder traversal sequence of a binary search tree.

You may assume each number in the sequence is unique.

Follow up: Could you do it using only constant space complexity?

```
public class Solution {
    public boolean verifyPreorder(int[] preorder) {
        Stack<Integer> stack =new Stack<>();
        int min = Integer.MIN_VALUE;
        for(int val:preorder){
            if( val < min) return false;

            while(!stack.isEmpty() && val > stack.peek()){
                min = stack.pop();
            }
            stack.push(val);
        }
        return true;
    }
}
```

257. Binary Tree Paths

Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:

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

All root-to-leaf paths are:

```
["1->2->5", "1->3"]
```

Solution recursive

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<String> result = new ArrayList<>();
    public List<String> binaryTreePaths(TreeNode root) {
        if(root == null) return result;
        List<TreeNode> path = new ArrayList<>();
        buildPath(root, path);
        return result;
    }
```

```
    private void buildPath(TreeNode root, List<TreeNode> path){

        if(root.left == null && root.right == null){
            path.add(root);
            StringBuilder sb = new StringBuilder();
            for(int i=0; i< path.size()-1; i++){
                TreeNode tn = (TreeNode)path.get(i);
                sb.append(tn.val);
                sb.append("->");
            }
            TreeNode last = (TreeNode)path.get(path.size()-1);
            sb.append(last.val);
            path.remove(root);
            result.add(sb.toString());
        }

        path.add(root);
        if(root.left != null){
            buildPath(root.left, path);
        }
        if(root.right != null){
            buildPath(root.right, path);
        }
        path.remove(root);
    }
}
```

> A clear DFS

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<String> res = new ArrayList<>();
    public List<String> binaryTreePaths(TreeNode root) {

        if(root == null){
            return res;
        }

        buildPath(root, String.valueOf(root.val));
        return res;
    }

    void buildPath(TreeNode root, String list){
        if(root.left == null && root.right == null){
            res.add(list);
            return;
        }
        if(root.left != null){
            buildPath(root.left, list + "->" + String.valueOf(ro
ot.left.val));
        }

        if(root.right != null){
            buildPath(root.right, list + "->" + String.valueOf(r
oot.right.val));
        }
    }
}
```

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<String> res = new ArrayList<>();
    public List<String> binaryTreePaths(TreeNode root) {
        List<Integer> list = new ArrayList<>();

        print(root, list);
        return res;
    }

    void print(TreeNode root, List<Integer> list){
        if(root == null) return;
        list.add(root.val);
        if(root.left == null && root.right == null){

            StringBuilder sb = new StringBuilder();
            sb.append(list.get(0));

            for(int i=1; i<list.size(); i++){
                sb.append("->");
                sb.append(list.get(i));
            }
            res.add(sb.toString());
        }
        print(root.left, list);
        print(root.right, list);
        list.remove(list.size()-1);
    }
}
```

258. Add Digits

Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

For example:

Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.

Follow up: Could you do it without any loop/recursion in O(1) runtime?

Hint:

A naive implementation of the above process is trivial. Could you come up with other methods? What are all the possible results? How do they occur, periodically or randomly? You may find this Wikipedia article useful.

the simpler solution, is to use recursion/loop, each time, add all digits number, until generates a single digit number.

```
public class Solution {
    public int addDigits(int num) {
        while(num > 9){
            int res =0;
            while(num > 0){
                res += num%10;
                num /= 10;
            }
            num = res;
        }

        return num;

    }
}
```

There a a constant solution, the result set can only be in [0...9], 0 can only be generated by 0, so the result set of all positive number is [1...9], each time the number increase by 10, the number that adds up to 9 shift left by 1, say res(18) = 9, res(27) = 9, ... so this question becomes : how many numbers are there between this number and the last res(X) = 9;

so the question becomes X = 9 * ((num-1)/9) and the final result is num - X;

```
public class Solution {
    public int addDigits(int num) {
        return num - 9 * ((num-1)/9);
    }
}
```

259. 3Sum Smaller

Given an array of n integers nums and a target, find the number of index triplets i, j, k with 0 <= i < j < k < n that satisfy the condition nums[i] + nums[j] + nums[k] < target.

For example, given nums = [-2, 0, 1, 3], and target = 2.

Return 2. Because there are two triplets which sums are less than 2:

[-2, 0, 1] [-2, 0, 3] Follow up: Could you solve it in O(n2) runtime?

why sort array is ok?

```
because sort array won't change the fact that a good triplet sti
ll stands
```

Fix first number, then use two pointers, notice that when you find a i(fixed),j,k triplet, which means all combine between k-j is valid triplet.

```
public class Solution {
    public int threeSumSmaller(int[] nums, int target) {
        Arrays.sort(nums);// even though sorting scrambles the i
ndice. however
                          // it doesn't change the number of goo
d triplet.
        int count = 0;
        for(int i=0; i<nums.length-2; i++){
            int j = i+1, k = nums.length-1;
            while(j < k){
                if((nums[i] + nums[j] + nums[k]) >= target) k--;
                else{
                    count += k -j; // all the combine between k-
j are good triplet.
                    j++;

                }
            }
        }

        return count;
    }
}
```

260. Single Number III

Given an array of numbers nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

For example:

Given nums = [1, 2, 1, 3, 2, 5], return [3, 5].

Note : The order of the result is not important. So in the above example, [5, 3] is also correct.

Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

Related Issue : Single Number I Single Number II

```
public class Solution {
    public int[] singleNumber(int[] nums) {
        int xor = 0;
        for(int v : nums){
            xor ^= v;
        }
        int r = xor &(~(xor-1));
        int[] res = new int[2];

        for(int v : nums){
            if((v & r) != 0) res[0] ^= v;
        }
        res[1] = res[0]^xor;

        return res;
    }
}
```

261. Graph Valid Tree

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

For example:

Given n = 5 and edges = [[0, 1], [0, 2], [0, 3], [1, 4]], return true.

Given n = 5 and edges = [[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]], return false.

Note: you can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

This is an undirected map, so in-degree methods is not very suitable. Using bfs, since in a valid tree traversal, each not should be only visited once, if there is a loop, you end up visit an already visited node, this is not a valid tree. using a queue, every time, if node is not visited, mark it as visited, then traverse its neighbors, if is not visited, enque it, if there is any cycle in the graph, your queue ends up containing multiple instance of certain numbers.

```java
public class Solution {
    public boolean validTree(int n, int[][] edges) {
        List<List<Integer>> list = new ArrayList<>();
        for(int i=0; i<n; i++){
            list.add(new ArrayList<>());
        }
        for(int[] edge : edges){
            list.get(edge[0]).add(edge[1]);
            list.get(edge[1]).add(edge[0]);
        }
        boolean[] visited = new boolean[n];
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(0);
        while(!queue.isEmpty()){
            int id = queue.poll();
            if(visited[id]){
                return false;
            }
            visited[id] = true;
            for(int neighbor : list.get(id)){
                if(!visited[neighbor]){
                    queue.offer(neighbor);
                }
            }
        }

        for(boolean b : visited){
            if(!b) return false;
        }
        return true;
    }
}
```

This question can be viewed in different perspective, building a set, initially each node has its own set, with more edges coming, merge two related sets, until the set remain equals to 1. this union-find algorithm.

```
public class Solution {
    public boolean validTree(int n, int[][] edges) {
        int[] id = new int[n];
        for(int i=0; i< n; i++) id[i] = i;

        for(int[] e : edges){
            int u = e[0];
            int v = e[1];
            if(id[u] == id[v]) return false;
            merge(id, u, v);
        }
        for(int i=1; i<n; i++){
            if(id[i-1] != id[i]) return false;
        }

        return true;
    }

    void merge(int[] id, int u, int v){
        int t = id[u];
        for(int i=0; i< id.length; i++){
            if(id[i] == t) id[i] = id[v]; // if id[i] == id[u] M
AY NOT WORK, cause the id[u] val may changed.
        }
    }
}
```

263. Ugly Number

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7.

Note that 1 is typically treated as an ugly number.

```
public class Solution {
    public boolean isUgly(int num) {
        if(num <= 0) return false;
        while(num%2 == 0) num /=2;
        while(num%3 == 0) num /=3;
        while(num%5 == 0) num /=5;

        return num == 1;
    }
}
```

266. Palindrome Permutation

Given a string, determine if a permutation of the string could form a palindrome.

For example

```
"code" -> False, "aab" -> True, "carerac" -> True.
```

```
public class Solution {
    Map<Character, Integer> dict = new HashMap<>();
    public boolean canPermutePalindrome(String s) {
        for(int i=0; i<s.length();i++){
            char ch = s.charAt(i);
            if(dict.containsKey(ch)){
                dict.put(ch, dict.get(ch)+1);
            }else{
                dict.put(ch, 1);
            }
        }
        int oddCount = 0;
        for(Character ch : dict.keySet()){
            int count = dict.get(ch);
            if((count%2) == 1) oddCount++;
        }
        if(oddCount>1) return false;
        else return true;
    }
}
```

> you don't really need to count.

```
public class Solution {
    public boolean canPermutePalindrome(String s) {
        Set<Character> set = new HashSet<>();
        for(char ch : s.toCharArray()){
            if(set.contains(ch)) set.remove(ch);
            else set.add(ch);
        }
        return s.length() %2 == 1 ? set.size() == 1 : set.size()
 == 0;
    }
}
```

268. Missing Number

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

For example,

Given nums = [0, 1, 3] return 2.

Note: Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

- solution 1, n*(n+1)/2 - sum(array), overflow issue.
- hash table.
- bit manipulation.

think of the array as newArray = nums + [0...n], so each number show up twice except the missing one. similar to Single Number

```
public class Solution {
    public int missingNumber(int[] nums) {
        int res = 0;
        for(int i=0; i<nums.length; i++)
            res ^= (i+1) ^ nums[i];
        return res;
    }
}
```

269. Alien Dictionary

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of words from the dictionary, where words are sorted lexicographically by the rules of this new language. Derive the order of letters in this language.

For example, Given the following words in dictionary,

```
[
  "wrt",
  "wrf",
  "er",
  "ett",
  "rftt"
]
```

The correct order is: "wertf".

> There are few points not clear :
>
> - each word itself has no order, ie abc can not deduce the order of a, b ,c
> - if any chars has not found and there is no dependence what then ?

```
public class Solution {
    public String alienOrder(String[] words) {
        if(words == null || words.length == 0) return "";

        Map<Character, List<Character>> map = new HashMap<>();

        Set<Character> letters = new HashSet<>();
        for(String s : words){
            for(char c : s.toCharArray()){
                letters.add(c);
            }
        }
```

```
        int[] indg = new int[26];

        for(int i=1; i< words.length; i++){
            String prev = words[i-1];
            String cur = words[i];
            for(int k=0; k< prev.length() && k < cur.length(); k
++){
                Character c1 = prev.charAt(k) ;
                Character c2 = cur.charAt(k);

                if(c1 != c2){
                    if(!map.containsKey(c1)){
                        List<Character> list = new ArrayList<>()
;

                        map.put(c1, list);
                    }
                    map.get(c1).add(c2);
                    indg[c2-'a']++;
                    break;
                }
            }
        }
        PriorityQueue<Character> pq = new PriorityQueue<>();

        for(Character c : letters){
            if(indg[c-'a'] == 0) pq.offer(c);
        }

        StringBuilder sb = new StringBuilder();
        while(!pq.isEmpty()){
            Character c = pq.poll();
            sb.append(c);
            List<Character> list = map.get(c);
            if(list == null) continue;
            for(Character ch : list){
                if(--indg[ch-'a'] == 0) pq.offer(ch);
            }
        }

        return sb.length() == letters.size() ? sb.toString(): ""
```

```
;


    }
}
```

Solution. Topological sort.

1. find all possible edges, which means, in two adjecent strings, s1, s2, if s1[i] != s2[j], there is an edge from s1[i] to s2[j];
2. in the mean time, build a in-degree map to reflect the fact that how many nodes points to current node, also need to mantain a list of node a current can reach.

```
public class Solution {
    public String alienOrder(String[] words) {
        if(words == null || words.length == 0) return "";

        Map<Character, List<Character>> map = new HashMap<>();

        Set<Character> letters = new HashSet<>();
        for(String s : words){
            for(char c : s.toCharArray()){
                letters.add(c);
            }
        }

        int[] indg = new int[26];

        for(int i=1; i< words.length; i++){
            String prev = words[i-1];
            String cur = words[i];
            for(int k=0; k< prev.length() && k < cur.length(); k
++){
                Character c1 = prev.charAt(k) ;
                Character c2 = cur.charAt(k);

                if(c1 != c2){
```

```
                    if(!map.containsKey(c1)){
                        List<Character> list = new ArrayList<>()
;
                        map.put(c1, list);
                    }
                    map.get(c1).add(c2);
                    indg[c2-'a']++;
                    break;
                }
            }
        }
        PriorityQueue<Character> pq = new PriorityQueue<>();

        for(Character c : letters){
            if(indg[c-'a'] == 0) pq.offer(c);
        }

        StringBuilder sb = new StringBuilder();
        while(!pq.isEmpty()){
            Character c = pq.poll();
            sb.append(c);
            List<Character> list = map.get(c);
            if(list == null) continue;
            for(Character ch : list){
                if(--indg[ch-'a'] == 0) pq.offer(ch);
            }
        }

        return sb.length() == letters.size() ? sb.toString(): ""
;


    }
}
```

> the below solution is not very clear. ```java public class Solution { public String alienOrder(String[] words) { Map> graph = new HashMap<>(); StringBuilder sb = new StringBuilder(); for(int i =0; i< words.length; i++){ String s = words[i]; for(int j=0; j()); } }

```
        if(i >0){
            buildEdge(graph, words[i-1], s);
        }
    }

    Map<Character, Integer> visited = new HashMap<>(); //(-1, no
de visited, 1 node neighbors visited)
    Iterator it = graph.entrySet().iterator();
    while(it.hasNext()){
        Map.Entry pair = (Map.Entry)it.next();
        char ch = (char)pair.getKey();
        if(!tSort(ch, graph, sb, visited)){
            return "";
        }
    }

    return sb.toString();
}
void buildEdge(Map<Character, Set<Character>> graph, String prev
, String cur){
    if(prev == null || cur == null){
        return;
    }
    for(int i=0; i< prev.length() && i< cur.length(); i++){
        char p = prev.charAt(i);
        char q = cur.charAt(i);
        if(p != q){
            if(!graph.get(p).contains(q)){
                graph.get(p).add(q);
            }
            break;
        }
    }
}

boolean tSort(char ch, Map<Character, Set<Character>> graph, Str
ingBuilder sb, Map<Character,Integer> visited){
    if(visited.containsKey(ch)){
        if(visited.get(ch) == -1){
            return false;
```

```
        }else{
            return true;
    }
    }else{
        visited.put(ch, -1);
    }

    Set<Character> neighbors = graph.get(ch);
    for(char n : neighbors){
        if(!tSort(n, graph, sb, visited)){
            return false;
        }
    }
    sb.insert(0, ch);
    visited.put(ch, 1);
    return true;
}
```

}

```
updating in-degrees, failed at ["za","zb","ca","cb"] test cases
why(output is "azc")?

```java
public class Solution {
    public String alienOrder(String[] words) {
        Map<Character, Set<Character>> graph = new HashMap<>();
        Map<Character, Integer> ingrees = new HashMap<>();
        StringBuilder sb = new StringBuilder();
        for(int i =0; i< words.length; i++){
            String s = words[i];
            for(int j=0; j<s.length(); j++){
                if(!graph.containsKey(s.charAt(j))){
                    graph.put(s.charAt(j), new HashSet<Character
>());
                    if(!ingrees.containsKey(s.charAt(j)))
                        ingrees.put(s.charAt(j), 0);
                }
            }
```
```

```
            if(i >0){
                updateIngrees(graph, ingrees, words[i-1], s);
            }else{
                if(words[0].length() > 0 ){
                    ingrees.put(words[0].charAt(0), 0);
                }
            }
        }

        Queue<Character> queue = new LinkedList<>();

        Iterator it = ingrees.entrySet().iterator();
        while(it.hasNext()){
            Map.Entry pair = (Map.Entry)it.next();
            char ch = (char)pair.getKey();
            int val = (int)pair.getValue();
            if(val == 0){
                queue.offer(ch);
            }
        }

        while(!queue.isEmpty()){
            char top = queue.poll();
            sb.append(top);
            Set<Character> neighbors = graph.get(top);
            for(char n : neighbors){
                ingrees.put(n, ingrees.get(n)-1);
                if(ingrees.get(n) == 0){
                  queue.offer(n);
                }
            }
        }

        String res = sb.toString();
        if(res.length() == graph.size()) return res;
        else return "";
    }

    void updateIngrees(Map<Character, Set<Character>> graph, Map
```

```
<Character, Integer> ingrees, String prev, String cur){
        if(prev == null || cur == null) return;
        for(int i=0; i< prev.length() && i< cur.length(); i++){
            char p = prev.charAt(i);
            char q = cur.charAt(i);
            if(p != q){
                //update edges
                if(!graph.containsKey(p)){
                    Set<Character> set = new HashSet<>();
                    set.add(q);
                    graph.put(p, set);
                }else{
                    graph.get(p).add(q);
                }
                // update ingrees;
                if(!ingrees.containsKey(p)){
                    ingrees.put(p, 0);
                }

                if(ingrees.containsKey(q)){
                    ingrees.put(q, ingrees.get(q) +1);// this li
ne is actually failing the code, you can simply increase ingree
on q, cause if you have mutiple <p-q> instances while building,
the q's ingree is not properly calculated.
                }else{
                    ingrees.put(q, 1);
                }
                break;
            }
        }
    }


}
```

270. Closest Binary Search Tree Value

Given a non-empty binary search tree and a target value, find the value in the BST that is closest to the target.

Note: Given target value is a floating point. You are guaranteed to have only one unique value in the BST that is closest to the target.

basic observation: if root.val is greater than target value, the right subtree can't be closer than root/left subtree.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int closestValue(TreeNode root, double target) {
        TreeNode child = root.val < target ? root.right : root.l
eft;
        if(child == null) return root.val;
        int childClose = closestValue(child, target);
        return Math.abs(target-childClose) > Math.abs(target-roo
t.val) ? root.val : childClose;
    }
}
```

271. Encode and Decode Strings

Design an algorithm to encode a list of strings to a string. The encoded string is then sent over the network and is decoded back to the original list of strings.

Machine 1 (sender) has the function:

```
string encode(vector<string> strs) {
  // ... your code
  return encoded_string;
}
Machine 2 (receiver) has the function:
vector<string> decode(string s) {
  //... your code
  return strs;
}

So Machine 1 does:

string encoded_string = encode(strs);
and Machine 2 does:

vector<string> strs2 = decode(encoded_string);
strs2 in Machine 2 should be the same as strs in Machine 1.
```

Implement the encode and decode methods.

```java
public class Codec {

    // Encodes a list of strings to a single string.
    public String encode(List<String> strs) {

        StringBuilder sb = new StringBuilder();
        for(String s : strs){
            sb.append(s.length());
            sb.append("#");
            sb.append(s);
        }

        return sb.toString();

    }

    // Decodes a single string to a list of strings.
    public List<String> decode(String s) {

        List<String> res = new ArrayList<>();
        int k = 0;

        while(k < s.length()){
            int pound = s.indexOf("#", k);
            int size = Integer.parseInt(s.substring(k, pound));
            res.add(s.substring(pound+1, pound+1+size));
            k = pound+1+size;
        }

        return res;

    }
}

// Your Codec object will be instantiated and called as such:
// Codec codec = new Codec();
// codec.decode(codec.encode(strs));
```

272. Closest Binary Search Tree Value II

Given a non-empty binary search tree and a target value, find k values in the BST that are closest to the target.

Note: Given target value is a floating point.

You may assume k is always valid, that is: k ≤ total nodes.

You are guaranteed to have only one unique set of k values in the BST that are closest to the target.

Follow up: Assume that the BST is balanced, could you solve it in less than O(n) runtime (where n = total nodes)?

links : 94 Binary Tree Inorder Traversal

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> closestKValues(TreeNode root, double ta
rget, int k) {
        Queue<Integer> list = new LinkedList<>();

        Stack<TreeNode> stack = new Stack<>();
        TreeNode node = root;
        while(!stack.isEmpty() || node != null){
            while(node != null){
                stack.push(node);
                node = node.left;
            }
```

```
            node = stack.pop();

            if(list.size() < k){
                list.offer(node.val);
            }else{
                if(Math.abs(list.peek() - target) > Math.abs(nod
e.val - target)){
                    list.poll();
                    list.offer(node.val);
                }else{
                    break;
                }
            }

            node = node.right;
        }

        return (List<Integer>) list;
    }
}
```

274. H-Index

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index h if h of his/her N papers have at least h citations each, and the other N − h papers have no more than h citations each."

For example, given citations = [3, 0, 6, 1, 5], which means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively.

Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, his h-index is 3.

Note: If there are several possible values for h, the maximum one is taken as the h-index.

count sort.

```
public class Solution {
    public int hIndex(int[] citations) {

        int[] cc = new int[citations.length+1];

        for(int i=0; i< citations.length; i++){
            if(citations[i]>= citations.length){
                cc[citations.length]++;
            }else{
                cc[citations[i]]++;
            }
        }

        int sum = 0;
        for(int i=cc.length-1; i>=0;i--){
            if(sum + cc[i] >= i){
                return i;
            }else
            {
                sum += cc[i];
            }
        }

        return 0;

    }
}
```

Sort the array in descending order, if i >= citations[i] then H-index is i;

```
public class Solution {
    public int hIndex(int[] citations) {

        Arrays.sort(citations);

        for(int i = citations.length - 1;  i>=0; i--){
            int h = citations.length - i-1;
            if(h >= citations[i]) return h;
        }

        return citations.length;

    }
}
```

sort the array, if citations[i], then there are citations.length - i of papers are > citations[i]; so the local H-index is min(citations[i], citations.length-i);

```
public class Solution {
    public int hIndex(int[] citations) {
        int h=0;
        Arrays.sort(citations);
        for(int i=0; i< citations.length; i++){
            h = Math.max(Math.min(citations[i], citations.length
 - i), h);
        }
        return h;
    }
}
```

275. H-Index II

Follow up for H-Index: What if the citations array is sorted in ascending order? Could you optimize your algorithm?

Hint: Expected runtime complexity is in O(log n) and the input is sorted. Binary search.

```
public class Solution {
    public int hIndex(int[] citations) {
        int left =0;
        int right = citations.length-1;

        while(left <= right){
            int mid = left + (right-left)/2;
            if(citations[mid] == citations.length - mid) return
citations.length - mid;
            if(citations.length - mid < citations[mid]){
                right = mid-1;
            }else{
                left = mid +1;
            }
        }

        return citations.length - left;
    }
}
```

277. Find the Celebrity

Suppose you are at a party with n people (labeled from 0 to n - 1) and among them, there may exist one celebrity. The definition of a celebrity is that all the other n - 1 people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n), your function should minimize the number of calls to knows.

> First approach takes O(n^2), for each pair of person, if A doesn't know B, then A is a potential candidate of the Celebrity, then the question is reduced to half size, continue this approach until the size reduced to 1. Then sweep the array to make sure the Celebrity is known by others, otherwise, he/she is fake 'Celebrity'
>
> The following solution is based on a simple assumption. if know(L,R) return false, then L must be the Celebrity otherwise there is no Celebrity.

```
/* The knows API is defined in the parent class Relation.
      boolean knows(int a, int b); */

public class Solution extends Relation {
    public int findCelebrity(int n) {
        int l=0, r = n-1;
        while(l < r){
            if(knows(l, r)){
                l++;
            }else{
                r--;
            }
        }
        for(int i=0; i< n; i++){
            if(i == l) continue;
            if(knows(l, i) || !knows(i , l)) return -1;
        }
        return l;

    }
}
```

278. First Bad Version

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

> the real question is : given a sorted array, find the first appearance of certain number. the trick here is the return value, should it be left ? right? in this case always left. why ? if middle is not the number, so mid+1 becomes the left. if middle is the number, there are 2 scenarios, one this is the first number. one it is not:
>
> - in the first case, your left, mid and right eventually become the same, but the number is not the required, the next is, the binary search in this case will point left to next value(left = mid +1)
> - in the second case, your new right, mid -1 may be the desired number. so continue the search, you will eventually reach the first case.

```
/* The isBadVersion API is defined in the parent class VersionCo
ntrol.
      boolean isBadVersion(int version); */

public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int l = 1, r=n;
        while(l <= r){
            int m = l + (r-l)/2;
            if(isBadVersion(m)){
                r = m-1;
            }else{
                l = m+1;
            }
        }

        return l;
    }
}
```

279. Perfect Squares

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.

For example, given n = 12, return 3 because 12 = 4 + 4 + 4; given n = 13, return 2 because 13 = 4 + 9.

Solution 1. According to Lagrange's four-square theorem(please wiki/google), any positive number can be represented as 4(at most) square number sum.

1. divide by 4, notice that, 2 and 8, 3 and 12, 4 and 16 has the same number of square factors.
2. if number%8==7, this result in a square factors 2^2 + 1 +1 +1, which is four.
3. if any two numbers can suqare sum to n, return 1 or 2.
4. otherwise result is 3.

```
public class Solution {
    public int numSquares(int n) {
        while(n%4 == 0)  n /= 4;
        if(n%8 == 7) return 4;
        for(int x=0; x*x <=n; x++){
            int y = (int)Math.sqrt(n - x*x);
            if(x*x + y*y == n){
                if(x == 0 || y == 0) return 1;
                else return 2;
            }
        }
        return 3;
    }
}
```

Solution 2. recusive.

```
public class Solution {
    public int numSquares(int n) {
        int res = n, num =2;
        while(num * num <=n){
            int x = n/(num*num), y = n%(num*num);
            res = Math.min(res, x + numSquares(y));
            ++num;
        }

        return res;
    }
}
```

Solution 3. DP. this is a forward dp question, using an array dp[], dp[i] means the number need to square-sum up to i, then, for all the j from 1 to i+j*j <=n; calculate dp[i+j*j] = ?, initially set each dp[i] equals to max.

```
public class Solution {
    public int numSquares(int n) {
        int[] dp = new int[n+1];
        for(int i=1; i<=n; i++){
            dp[i] = Integer.MAX_VALUE;
        }
        dp[0]=0;
        for(int i=0; i<=n; i++){
            for(int j=1; i+ j*j <=n; j++){
                dp[i+j*j] = Math.min(dp[i+j*j], dp[i]+1);
            }
        }

        return dp[n];
    }
}
```

280. Wiggle Sort

Given an unsorted array nums, reorder it in-place such that nums[0] <= nums[1] >= nums[2] <= nums[3]....

For example, given nums = [3, 5, 2, 1, 6, 4], one possible answer is [1, 6, 2, 5, 3, 4].

```
the pattern is number in odd position is peak.
```

First try to solve it without in-place:

1. sort the array in increasing order.
2. create a result array of the same size.
3. keep 2 pointers, one from the beginning, one from the middle(notice odd/even of array).
4. put beginning first, then the middle pointer, into the result array.

Solve it in-place.

```
public class Solution {
    public void wiggleSort(int[] nums) {
        if(nums.length<2)  return;

        Arrays.sort(nums);
        int i=2;
        for( ; i< nums.length-1 ; ) {
            int x = nums[i];
            nums[i] = nums[i+1];
            nums[i+1] = x;
            i +=2;
        }
    }
}
```

```
Is it really necessarily to sort the array initially ?
--NO--
```

noticing that the property of wiggle array is:

```
  if i%2 == 1, nums[i] >= nums[i-1];
  if i%2 == 0, nums[i] <= nums[i-1];
```

if the property is not observed, simple swap the property-break number. the previous relation at i-2 will still stand.

```
public class Solution {
    public void wiggleSort(int[] nums) {
        for(int i=1; i< nums.length;i++){
            if((i&1) == 1 && nums[i] < nums[i-1]
                || (i&1) == 0 && nums[i] > nums[i-1]){
                int t = nums[i];
                nums[i] = nums[i-1];
                nums[i-1] = t;
            }
        }
    }
}
```

281. Zigzag Iterator

Given two 1d vectors, implement an iterator to return their elements alternately.

For example, given two 1d vectors:

```
v1 = [1, 2]
v2 = [3, 4, 5, 6]
```

By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1, 3, 2, 4, 5, 6].

```
public class ZigzagIterator {
   List<List<Integer>> lists = new ArrayList<>();

    int p = 0; // which list
    int q = 0; // current index in current list;

    int[] indices = new int[2];

    public ZigzagIterator(List<Integer> v1, List<Integer> v2)
{
     lists.add(v1);
     lists.add(v2);
     indices[0] = indices[1] = 0;
    }


    public int next() {

        int val = lists.get(p).get(q);

        indices[p] = q+1;

        p = 1-p;

        q = indices[p];
```

```
            return val;
        }

        public boolean hasNext() {
            if(lists.get(p).size() > indices[p]) return true;
            else{
                p = 1-p;
                q = indices[p];
                return lists.get(p).size() > q;
            }
        }
    }

    /**
     * Your ZigzagIterator object will be instantiated and called
as such:
     * ZigzagIterator i = new ZigzagIterator(v1, v2);
     * while (i.hasNext()) v[f()] = i.next();
     */
```

```

:set nu!
```

282. Expression Add Operators

Given a string that contains only digits 0-9 and a target value, return all possibilities to add binary operators (not unary) +, -, or * between the digits so they evaluate to the target value.

Examples:

```
"123", 6 -> ["1+2+3", "1*2*3"]
"232", 8 -> ["2*3+2", "2+3*2"]
"105", 5 -> ["1*0+5","10-5"]
"00", 0 -> ["0+0", "0-0", "0*0"]
"3456237490", 9191 -> []
```

> usually this case you need to traverse the string from the beginning, the recusively build the result for the left.

```
public class Solution {
    List<String> res = new ArrayList<>();
    public List<String> addOperators(String num, int target) {
        add(num, target, "", 0, 0);
        return res;
    }

    private void add(String num, int target, String tmp, long cu
rrRes, long prevNum){
        if(currRes == target && num.length()==0){
            String exp = new String(tmp);
            res.add(exp);
            return;
        }

        for(int i=1; i<= num.length();i++){
            String currStr = num.substring(0,i);
            if(currStr.length()>1 && currStr.charAt(0) == '0') r
eturn;
            String next = num.substring(i);
            long currNum = Long.parseLong(currStr);
            if(tmp.length() !=0){
                add(next, target, tmp+"+"+currNum, currRes+currN
um, currNum);
                add(next, target, tmp+"-"+currNum, currRes-currN
um, -currNum);
                add(next, target, tmp+"*"+currNum, (currRes-prev
Num) + prevNum*currNum, prevNum*currNum);
            }else{
                add(next, target, currStr, currNum, currNum);
            }

        }

    }
}
```

283. Move Zeroes

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

Note: You must do this in-place without making a copy of the array. Minimize the total number of operations.

```
public class Solution {
    public void moveZeroes(int[] nums) {
        int i=0;// 0's head
        while(i < nums.length && nums[i] != 0) i++;
        for(int j = i+1; j< nums.length; j++){
            if(nums[j] != 0){
                nums[i++] = nums[j];
                nums[j] = 0;
            }
        }
    }
}
```

```java
public class Solution {
    public void moveZeroes(int[] nums) {
        int pos =0;
        for(int i=0; i< nums.length;i++){
            if(nums[i] != 0){
                nums[pos++] = nums[i];
            }
        }
        for(; pos < nums.length; pos++){
            nums[pos] =0;
        }
    }
}
```

284. Peeking Iterator

Given an Iterator class interface with methods: next() and hasNext(), design and implement a PeekingIterator that support the peek() operation -- it essentially peek() at the element that will be returned by the next call to next().

Here is an example. Assume that the iterator is initialized to the beginning of the list: [1, 2, 3].

Call next() gets you 1, the first element in the list.

Now you call peek() and it returns 2, the next element. Calling next() after that still return 2.

You call next() the final time and it returns 3, the last element. Calling hasNext() after that should return false.

Hint:

Think of "looking ahead". You want to cache the next element.

Is one variable sufficient? Why or why not?

Test your design with call order of peek() before next() vs next() before peek().

For a clean implementation, check out Google's guava library source code

Follow up: How would you extend your design to be generic and work with all types, not just integer?

> Replace Integer with Generics.

```
// Java Iterator interface reference:
// https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.
html
class PeekingIterator implements Iterator<Integer> {
    Integer peeked = null;
    Iterator<Integer> iter;
    public PeekingIterator(Iterator<Integer> iterator) {
        // initialize any member here.
```

```
        iter = iterator;
    }

    // Returns the next element in the iteration without advanci
ng the iterator.
    public Integer peek() {
        if(peeked != null){
            return peeked;
        }else{
            peeked = iter.next();
            return peeked;
        }
    }

    // hasNext() and next() should behave the same as in the Ite
rator interface.
    // Override them if needed.
    @Override
    public Integer next() {
        if(peeked != null){
            Integer res = new Integer(peeked);
            peeked = null;
            return res;
        }else{
            return iter.next();
        }
    }

    @Override
    public boolean hasNext() {
        return peeked != null || iter.hasNext();
    }
}
```

285. Inorder Successor in BST

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

Note: If the given node has no in-order successor in the tree, return null.

> How about Inorder predecessor ?

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode inorderSuccessor(TreeNode root, TreeNode p)
{
        if(root == null) return null;
        TreeNode is;

        if(root == p){
            is = root.right;
            while(is != null && is.left != null) is = is.left;
            return is;
        }else if(root.val < p.val){
            return inorderSuccessor(root.right, p);
        }else{
            is = inorderSuccessor(root.left, p);
        }

        return is == null ? root : is;
    }
}
```

iterative

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode inorderSuccessor(TreeNode root, TreeNode p)
{
        TreeNode res = null;
        TreeNode target = null;
        Stack<TreeNode> stack = new Stack<>();

        while(!stack.isEmpty() || root != null){
            while(root != null){
                stack.push(root);
                root = root.left;
            }

            TreeNode top = stack.pop();
            if(target != null){
                res = top;
                break;
            }
            if(p == top){
                target = top;
            }

            root = top.right;
        }

        return res;
    }
}
```

286. Walls and Gates

You are given a m x n 2D grid initialized with these three possible values.

-1 - A wall or an obstacle. 0 - A gate. INF - Infinity means an empty room. We use the value 231 - 1 = 2147483647 to represent INF as you may assume that the distance to a gate is less than 2147483647. Fill each empty room with the distance to its nearest gate. If it is impossible to reach a gate, it should be filled with INF.

For example, given the 2D grid:

```
INF  -1  0  INF
INF INF INF  -1
INF  -1 INF  -1
  0  -1 INF INF
```

After running your function, the 2D grid should be:

```
  3  -1   0   1
  2   2   1  -1
  1  -1   2  -1
  0  -1   3   4
```

First try dfs.

```
public class Solution {
    static int EMPTY = 2147483647;


    public void wallsAndGates(int[][] rooms) {
        if(rooms == null || rooms.length == 0) return;
        boolean visited[][] = new boolean[rooms.length][rooms[0]
.length];
        for(int i=0; i< rooms.length; i++){
            for(int j=0; j< rooms[0].length; j++){
```

```
                if(rooms[i][j] == 0){
                    dfs(rooms, i, j, visited, 0);

                }
            }
        }
    }

    private void dfs(int[][] rooms, int x, int y, boolean[][] vi
sited, int dis){

        if(x < 0 || x >= rooms.length
            || y < 0 || y >= rooms[0].length || rooms[x][y] == -1
)
                return;
        if(visited[x][y] || dis > rooms[x][y]) return;

        rooms[x][y] = Math.min(rooms[x][y], dis);// since a room
 is already 0,
                                            // in first level, there
 is no set really.
        visited[x][y] = true;//  to avoid loop.
        dfs(rooms, x+1, y, visited, dis+1);
        dfs(rooms, x-1, y, visited, dis+1);
        dfs(rooms, x, y+1, visited, dis+1);
        dfs(rooms, x, y-1, visited, dis+1);
        visited[x][y] =false;

    }
}
```

the visited array is to avoid loop, which can be done by checking dis > rooms[x][y]

> Why dis > rooms[x][y] should return.

```
// NO VISITED ARRAY SOLUTION.
public class Solution {
    static int EMPTY = 2147483647;


    public void wallsAndGates(int[][] rooms) {
        if(rooms == null || rooms.length == 0) return;
        for(int i=0; i< rooms.length; i++){
            for(int j=0; j< rooms[0].length; j++){
                if(rooms[i][j] == 0){
                    dfs(rooms, i, j, 0);

                }
            }
        }
    }

    private void dfs(int[][] rooms, int x, int y, int dis){

        if(x < 0 || x >= rooms.length
            || y < 0 || y >= rooms[0].length || rooms[x][y] == -1
)
                return;
        if( dis > rooms[x][y]) return;
        // don't really need the min methods, if it is 0 ,then o
nly show once.
        rooms[x][y] = Math.min(rooms[x][y], dis);// since a room
 is already 0,
                                        // in first level, there
 is no set really.
        dfs(rooms, x+1, y, dis+1);
        dfs(rooms, x-1, y, dis+1);
        dfs(rooms, x, y+1, dis+1);
        dfs(rooms, x, y-1, dis+1);
    }
}
```

> BFS classic

```
public class Solution {
    public void wallsAndGates(int[][] rooms) {
        if(rooms == null || rooms.length == 0 || rooms[0].length
 ==0) return;

        for(int i =0 ; i< rooms.length; i++){
            for(int j =0; j< rooms[0].length; j++){
                if(rooms[i][j] == 0 ){
                    bfs(rooms, i, j);
                }
            }
        }

    }

    private void bfs(int[][]rooms, int x, int y){
        int[][] neighbor = {{1,0},{-1,0},{0,1},{0,-1}};

        Queue<int[]> queue = new LinkedList<>();
        queue.offer(new int[]{x,y});
        boolean[][] visited = new boolean[rooms.length][rooms[0]
.length];
        int dis = 0;
        while(!queue.isEmpty()){
            int size = queue.size();
            for(int i =0; i< size; i++){
                int[] top = queue.poll();
                rooms[top[0]][top[1]] = Math.min(rooms[top[0]][t
op[1]], dis);
                for(int j=0; j< 4;j++){
                    int k = top[0] + neighbor[j][0];
                    int l = top[1] + neighbor[j][1];
                    if(k>=0 && l>=0 && k< rooms.length && l < ro
oms[0].length && !visited[k][l] && rooms[k][l] > 0){
                        visited[k][l] = true;
                        queue.offer(new int[]{k,l});
                    }
                }
            }
            dis++;
```

```
            }
        }
    }
```

> A better bfs solution.

```
public class Solution {
    public void wallsAndGates(int[][] rooms) {
        if(rooms == null || rooms.length == 0) return;
        int[][] dir = new int[][]{{0,1},{0,-1},{1,0},{-1,0}};
        class Pair{
            int x;
            int y;
            Pair(int _x, int _y){
                x = _x;
                y = _y;
            }
        };
        Queue<Pair> queue = new LinkedList<>();

        for(int i= 0; i< rooms.length; i++){
            for(int j =0; j< rooms[0].length; j++){
                if(rooms[i][j] == 0){
                    queue.offer(new Pair(i,j));
                }
            }
        }

        while(!queue.isEmpty()){
            Pair p = queue.poll();
            for(int[] d : dir){
                int x = p.x + d[0];
                int y = p.y + d[1];
                if(x <0 || x >= rooms.length || y <0 || y>= room
s[0].length
                        || rooms[x][y] <= rooms[p.x][p.y] + 1)
                    continue;
                rooms[x][y] = rooms[p.x][p.y] + 1;
                queue.offer(new Pair(x,y));
            }
        }
    }
}
```

> without using class is actually slower than using it.

```
public class Solution {
    public void wallsAndGates(int[][] rooms) {
        if(rooms == null || rooms.length == 0) return;

        int m = rooms.length;
        int n = rooms[0].length;

        int[][] dir = new int[][]{{0,1},{0,-1},{1,0},{-1,0}};

        Queue<Integer> queue = new LinkedList<>();

        for(int i= 0; i< rooms.length; i++){
            for(int j =0; j< rooms[0].length; j++){
                if(rooms[i][j] == 0){
                    queue.offer(i*n + j);
                }
            }
        }

        while(!queue.isEmpty()){
            int p = queue.poll();
            int px = p / n;
            int py = p % n;
            for(int[] d : dir){
                int x = px + d[0];
                int y = py + d[1];
                if(x <0 || x >= rooms.length || y <0 || y>= room
s[0].length
                        || rooms[x][y] <= rooms[px][py] + 1)
                    continue;
                rooms[x][y] = rooms[px][py] + 1;
                queue.offer(x*n + y);
            }
        }
    }
}
```

287. Find the Duplicate Number

Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Note: You must not modify the array (assume the array is read only). You must use only constant, O(1) extra space. Your runtime complexity should be less than O(n2). There is only one duplicate number in the array, but it could be repeated more than once.

O(nlgn) Solution.

```
public class Solution {
    public int findDuplicate(int[] nums) {
        Arrays.sort(nums);
        int res = -1;
        for(int i=1; i< nums.length; i++){
            if(nums[i] == nums[i-1]){
                res = nums[i];
                break;
            }
        }
        return res;
    }
}
```

O(n) Solution.

```java
public class Solution {
    public int findDuplicate(int[] nums) {
        for(int i=0; i< nums.length; i++){
            if(nums[i] == i+1) continue;
            else if(nums[nums[i]-1] != nums[i]){
                int t =nums[nums[i] -1];
                nums[nums[i]-1] = nums[i];
                nums[i] = t;
                i--;
            }
        }
        int res = 0;
        for(int i=0; i< nums.length; i++){
            if(nums[i] != i+1){
                res = nums[i];
                break;
            }
        }
        return res;
    }
}
```

> similar as find cycle point in a cycle linked list 142
>
> ```java public class Solution { public int findDuplicate(int[] nums) {

```
    int slow = 0;
    int fast = 0;
    do{
        slow = nums[slow];
        fast = nums[nums[fast]];
    }while(slow != fast);

    fast = 0;
    while(fast != slow){
        fast = nums[fast];
        slow = nums[slow];
    }
    return fast;
 }
```

}```

288. Unique Word Abbreviation

An abbreviation of a word follows the form . Below are some examples of word abbreviations:

```
a) it                      --> it    (no abbreviation)

     1
b) d|o|g                   --> d1g

               1    1  1
     1---5----0----5--8
c) i|nternationalizatio|n  --> i18n

               1
     1---5----0
d) l|ocalizatio|n          --> l10n
```

Assume you have a dictionary and given a word, find whether its abbreviation is unique in the dictionary. A word's abbreviation is unique if no other word from the dictionary has the same abbreviation.

```
Example:
Given dictionary = [ "deer", "door", "cake", "card" ]

isUnique("dear") -> false
isUnique("cart") -> true
isUnique("cane") -> false
isUnique("make") -> true
```

The tricky part of this question is the case of duplications in the dictionary. if input dictionary is ["a", "a", "a"], and query is isUnique("a"), this should return true; because "a" is unique. another point require attention is to query the exactly word in the dictionary, this also should return true.

```
public class ValidWordAbbr {
    private Map<String, Set<String>> dict = new HashMap<>();

    public ValidWordAbbr(String[] dictionary) {
        for(String s : dictionary){

            String key = buildKey(s);
            if(dict.containsKey(key)){
                dict.get(key).add(s);
            }else{
                Set<String> t = new HashSet<>();
                t.add(s);
                dict.put(key, t);
            }
        }
    }
    private String buildKey(String s){
        if(s.length() <= 2 ) return s;
        else return s.charAt(0) + Integer.toString(s.length()-2)
 + s.charAt(s.length()-1);
    }
    public boolean isUnique(String word) {
        String key = buildKey(word);
        if(!dict.containsKey(key)){
            return true;
        }else{
            return dict.get(key).contains(word) && dict.get(key)
.size() <=1;
        }
    }
}


// Your ValidWordAbbr object will be instantiated and called as
such:
// ValidWordAbbr vwa = new ValidWordAbbr(dictionary);
// vwa.isUnique("Word");
// vwa.isUnique("anotherWord");
```

290. Word Pattern

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

Examples:

pattern = "abba", str = "dog cat cat dog" should return true.

pattern = "abba", str = "dog cat cat fish" should return false.

pattern = "aaaa", str = "dog cat cat dog" should return false.

pattern = "abba", str = "dog dog dog dog" should return false.

Notes: You may assume pattern contains only lowercase letters, and str contains lowercase letters separated by a single space.

Solution 1.

Keep two hash tables, one map from pattern.charAt[i] to str[i], one map from the other direction. if not exists in either map, update the key-value pair. then compare whether both maps match each other.

```
public class Solution {
    public boolean wordPattern(String pattern, String str) {
        Map<Character, String> m1 = new HashMap<>();
        Map<String, Character> m2  = new HashMap<>();

        String[] word = str.split(" ");

        if(pattern.length() != word.length) return false;
        for(int i=0;i<pattern.length(); i++){
            char ch = pattern.charAt(i);

            if(!m1.containsKey(ch)) m1.put(ch, word[i]);
            if(!m2.containsKey(word[i])) m2.put(word[i], ch);

            if(!Objects.equals(ch, m2.get(word[i])) || !Objects.
equals(word[i], m1.get(ch))){
                return false;
            }
        }

        return true;


    }
}
```

Solution 2.

There is one more smart solution that take advantage of Map.put()'s return value. and more importantly it utilizes a un-generic map from old java which is not safe.

```
public boolean wordPattern(String pattern, String str) {
    String[] words = str.split(" ");
    if (words.length != pattern.length())
        return false;
    Map index = new HashMap();
    for (int i=0; i<words.length; ++i)
        if (!Objects.equals(index.put(pattern.charAt(i), i),
                            index.put(words[i], i)))
            return false;
    return true;
}
```

a bit safer improvement

```
public class Solution {
    public boolean wordPattern(String pattern, String str) {
        Map<Character, Integer> m1 = new HashMap<>();
        Map<String, Integer> m2 = new HashMap<>();

        String[] words = str.split(" ");
        if(words.length != pattern.length()) return false;

        for(int i=0; i<pattern.length(); i++){
            if(!Objects.equals(m1.put(pattern.charAt(i), i),
                        m2.put(words[i], i))) return false;
        }

        return true;
    }
}
```

293. Flip Game

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: + and -, you and your friend take turns to flip two consecutive "++" into "--". The game ends when a person can no longer make a move and therefore the other person will be the winner.

Write a function to compute all possible states of the string after one valid move.

For example, given s = "++++", after one move, it may become one of the following states:

```
[
  "--++",
  "+--+",
  "++--"
]
```

please notice that the input string may be all '+', or all '-', or somewhere in between.

```
public class Solution {
    public List<String> generatePossibleNextMoves(String s) {
        List<String> res = new ArrayList<>();
        if(s.length() < 2) return res;
        for(int i=0; i<s.length()-1;i++){
            if(s.charAt(i) != '+' || s.charAt(i+1) != '+') conti
nue;
            String t = s.substring(0, i) + "--" + s.substring(i+2
);
            res.add(t);
        }

        return res;
    }
}
```

294. Flip Game II

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: + and -, you and your friend take turns to flip two consecutive "++" into "--". The game ends when a person can no longer make a move and therefore the other person will be the winner.

Write a function to determine if the starting player can guarantee a win.

For example, given s = "++++", return true. The starting player can guarantee a win by flipping the middle "++" to become "+--+".

Follow up: Derive your algorithm's runtime complexity.

```
T(N) = T(N-2) + T(N-3) + [T(2) + T(N-4)] + [T(3) + T(N-5)] + ...

        [T(N-5) + T(3)] + [T(N-4) + T(2)] + T(N-3) + T(N-2)
     = 2 * sum(T[i])  (i = 3..N-2)
```

the runtime complexity will be expotential.

```
public class Solution {
    public boolean canWin(String s) {
        if(s == null || s.length() == 0) return false;
        char [] arr = s.toCharArray();
        return canWin_(arr);
    }

    private boolean canWin_(char[] arr){
        for(int i=0; i<arr.length-1;i++){
            if(arr[i] == '+' && arr[i+1] == '+'){
                arr[i] = '-';
                arr[i+1] = '-';

                boolean otherWin = canWin_(arr);
                arr[i] ='+';
                arr[i+1] = '+';
                if(!otherWin) return true; // this need be the l
ast state, cause you need to recovery the scene for other recurs
ive calls to canWin_; if this line happens before the above 2 li
ne, the scene is recovered during the recursive call stack.
            }
        }

        return false;
    }
}
```

295. Find Median from Data Stream

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 value.

Examples: [2,3,4] , the median is 3

[2,3], the median is (2 + 3) / 2 = 2.5

Design a data structure that supports the following two operations:

void addNum(int num) - Add a integer number from the data stream to the data structure. double findMedian() - Return the median of all elements so far. For example:

```
add(1)
add(2)
findMedian() -> 1.5
add(3)
findMedian() -> 2
```

```
class MedianFinder {

    // Adds a number into the data structure.
    PriorityQueue<Integer> minHeap = new PriorityQueue<>();
    PriorityQueue<Integer> maxHeap = new PriorityQueue<>(10, new
 Comparator<Integer>(){
       @Override
       public int compare(Integer x, Integer y){
           return y-x;
       }
    });

    public void addNum(int num) {
        int size = minHeap.size() + maxHeap.size();
        if((size & 1) == 0){
            if(maxHeap.size() > 0 && num < maxHeap.peek()){
```

```
                maxHeap.offer(num);
                num = maxHeap.poll();
            }
            minHeap.offer(num);
        }else{
            if(minHeap.size() > 0 && num > minHeap.peek()){
                minHeap.offer(num);
                num = minHeap.poll();
            }
            maxHeap.offer(num);
        }
    }

    // Returns the median of current data stream
    public double findMedian() {
        int size = minHeap.size() + maxHeap.size();
        //make sure size is not zero.
        if((size & 1) == 0){
            return (minHeap.peek() + maxHeap.peek())/2.0;
        }else{
            return minHeap.peek();
        }

    }
};
```

299. Bulls and Cows

You are playing the following Bulls and Cows game with your friend: You write down a number and ask your friend to guess what the number is. Each time your friend makes a guess, you provide a hint that indicates how many digits in said guess match your secret number exactly in both digit and position (called "bulls") and how many digits match the secret number but locate in the wrong position (called "cows"). Your friend will use successive guesses and hints to eventually derive the secret number.

For example:

Secret number: "1807"

Friend's guess: "7810"

Hint: 1 bull and 3 cows. (The bull is 8, the cows are 0, 1 and 7.) Write a function to return a hint according to the secret number and friend's guess, use A to indicate the bulls and B to indicate the cows. In the above example, your function should return "1A3B".

Please note that both secret number and friend's guess may contain duplicate digits, for example:

Secret number: "1123"

Friend's guess: "0111"

In this case, the 1st 1 in friend's guess is a bull, the 2nd or 3rd 1 is a cow, and your function should return "1A1B". You may assume that the secret number and your friend's guess only contain digits, and their lengths are always equal.

The question:

if both string has same chars at the same position, it counts as a bull,

if a char instance in guess shows in the secret, it counts as a cow, it behaves like offset this char from both strings ,if there is another instance of same char shows both in guess and secret, still counts as a cow.

```
1, use a int[256] as char set.
2, foreach char in string:
    if they are equal, bulls++;
    else increase count in the set for char at secret
3, for each char in guess and secret:
    if they are not equal, and set has char in guess:
       cow++, decrease count in set for char.
return bulls + 'A' + cows + 'B'
```

Here is a nicer solution, worth remembering and learning from this method.

```
public class Solution {
    public String getHint(String secret, String guess) {
        int[] set = new int[256];
        int bulls =0, cows = 0;
        for( int i=0; i< secret.length();i++){
            char p = secret.charAt(i);
            char q = guess.charAt(i);
            if(p == q){
                bulls++;
            }else{
                if(set[p]++ < 0) cows++; // if set[char] < 0, wh
ich means in guess already show once. cows++, offset by ++
                if(set[q]-- > 0) cows++;// if set[char] >0 which
 means in secret already show once, cow++, offset by --;
            }
        }
        return "" + bulls + "A" + cows + "B";
    }
}
```

300. Longest Increasing Subsequence

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example,

```
Given [10, 9, 2, 5, 3, 7, 101, 18],
```

The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4. Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in O(n2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

Don't really know how to solve this for a long time. The equation is:

```
res[i] = Max(res[j]) + 1; if nums[i] > nums[j]

       = Max(1,res[i]); if nums[i] <= nums[j]
```

```java
public class Solution {
    public int lengthOfLIS(int[] nums) {
        if(nums.length == 0) return 0;
        int[] res = new int[nums.length];
        res[0] = 1;
        int max = 1;
        for(int i=1; i< nums.length; i++){
            for(int j=0; j<i; j++){
                if(nums[i] > nums[j]){
                    res[i] = Math.max(res[j]+1, res[i]);
                }else{
                    res[i] = Math.max(1, res[i]);
                }
            }
            max = Math.max(max, res[i]);
        }
        return max;
    }
}
```

The O(nlogn) algorithms keep an result array, all the number is initialized to MAX_VALUE, then for each number in the data array, try to find the first index that result[index] > number[i]; if so, update result[index] = number[i]; continue doing this, first index whose value is not MAX_VALUE is the result max increasing sequence.

```
Given [10, 9, 2, 5, 3, 7, 101, 18],
[M,M,M,M,M,M,M,M];
for 10, index is 0;
[10,...]
for 9; index is still 0;
[9,...]
for 2, [2,...]
for 5, [2,5...]
for 3, index is 1, [2,3...]
for 7, [2,3,7...]
for 101 [2,3,7,101,...]
for 81 [2,3,7,81...(M)]

so the size is 3 +1;
```

```
public class Solution {
    public int lengthOfLIS(int[] nums) {
        if(nums.length <=1) return nums.length;
        int[] res = new int[nums.length];
        for(int i=0; i< nums.length;i++){
            res[i] = Integer.MAX_VALUE;
        }
        for(int i=0; i< nums.length; i++){
            int index =binarySearchIndex(res, nums[i]);
            res[index] = nums[i];
        }
        int count = 0;
        for(int i=res.length-1; i>=0;i--){
            if(res[i] != Integer.MAX_VALUE) count++;
        }
        return count;
    }

    int binarySearchIndex(int[] nums, int val){
        int left=0;
        int right = nums.length-1;
        while(left < right){
            int mid = left + (right-left)/2;
            if(nums[mid] == val){
                return mid;// because our special case in this q
uestion, there will be only one number, and no repeat.
            }else if(nums[mid] > val){
                right = mid;
            }else{
                left = mid+1;
            }
        }
        return left;

    }
}
```

301. Remove Invalid Parentheses

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses (and).

Examples:

```
"()())()" -> ["()()()", "(())()"]
"(a)())()" -> ["(a)()()", "(a())()"]
")(" -> [""]
```

```
public class Solution {
    public List<String> removeInvalidParentheses(String s) {
        //at
        Queue<String > queue = new LinkedList<>();
        Set<String> visited = new HashSet<>();
        List<String> res = new ArrayList<>();

        queue.offer(s);
        // if found in certain level(bfs), traverse all in the q
ueue, don't go deeper.
        boolean found = false;
        while(!queue.isEmpty()){
            String t = queue.poll();
            if(isP(t)){
                res.add(t);
                found = true;
            }
            if(found) continue; // get all the valid ones from q
ueue.

            for(int i=0; i< t.length(); i++){
                if(t.charAt(i) != '(' && t.charAt(i) != ')') con
tinue;
                String nt = t.substring(0, i) + t.substring(i+1)
```

```
;
                if(!visited.contains(nt)){
                    queue.offer(nt);
                    visited.add(nt);
                }
            }
        }
        return res;
    }

    boolean isP(String s){
        int count =0;
        for(char c : s.toCharArray()){
            if(c == '(') count++;
            else if(c == ')' && count-- == 0 ) return false;
        }
        return count == 0;
    }
}
```

> DFS the order of dfs matters, take "(" as example,

```
public class Solution {
    Set<String> res = new HashSet<>();
    public List<String> removeInvalidParentheses(String s) {
        int lM = 0;
        int rM = 0;
        for(char c : s.toCharArray()){
            if(c == '(') lM++;
            if(c == ')'){
                if(lM != 0) lM--;
                else rM++;
            }
        }

        dfs(s, 0, lM, rM, 0, new StringBuilder());

        return new ArrayList<>(res);
    }
```

```
    void dfs(String s, int k, int lM, int rM, int open, StringBu
ilder sb){
        if(k == s.length() && lM == 0 && rM==0 && open == 0){
            res.add(sb.toString());
            return;
        }
        if( k == s.length() || lM < 0 || rM < 0 || open < 0) ret
urn;
        int len = sb.length();

        if(s.charAt(k) == '('){
            dfs(s, k+1, lM-1, rM, open, sb);
            dfs(s, k+1, lM, rM, open+1, sb.append('('));

        }else if(s.charAt(k) == ')'){

            dfs(s, k+1, lM, rM-1, open, sb);
            dfs(s, k+1, lM, rM, open-1, sb.append(')'));
        }else{
            dfs(s, k+1, lM, rM, open, sb.append(s.charAt(k)));
        }

        sb.setLength(len);
    }
}
```

Why the following not working, StringBuilder is shared between all dfs instances, and the following code set the length to be longer than its need in the second line, then the error, you should use a String instance instead of StringBuilder

```
            dfs(s, k+1, lM, rM, open+1, sb.append('('));
            dfs(s, k+1, lM-1, rM, open, sb);
```

305 Number of Islands II

A 2d grid map of m rows and n columns is initially filled with water. We may perform an addLand operation which turns the water at position (row, col) into a land. Given a list of positions to operate, count the number of islands after each addLand operation. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example:

Given m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]. Initially, the 2d grid grid is filled with water. (Assume 0 represents water and 1 represents land).

```
0 0 0
0 0 0
0 0 0
```

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

```
1 0 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

```
1 1 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

```
1 1 0
0 0 1   Number of islands = 2
0 0 0
```

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

```
1 1 0
0 0 1   Number of islands = 3
0 1 0
```

We return the result as an array: `[1, 1, 2, 3]` .

Use Union-Find Set to solve this problem. to use this alogrithm, we need to design a set of APIs we can follow while updating the required data structure, this API + data structure will:

- assign a id for each element(the data structure), in this case, each island/water area initially has -1 as its id.
- API-1 find(map, i...), return the id for certain element in the map.
- If 2 id returned is not the same, then use API-2 union() try to merge elements with these two ids.

- Quick-Find solution, O(kmn)

```
public class Solution {
    //this is a quick_find solution;
    public List<Integer> numIslands2(int m, int n, int[][] posit
ions) {
        int[] lands = new int[m*n];
        ArrayList<Integer> result = new ArrayList<>();
        int count =0;
        int[][] neighbors = {{1,0},{-1,0},{0,1},{0,-1}};
        for(int i=0; i< m*n; i++){
            lands[i] = -1;
        }
        for(int i=0; i<positions.length;i++){

            int pX= positions[i][0];
            int pY = positions[i][1];
            if( lands[pX*n+pY]!= -1) continue;

            count++;
            lands[pX*n+pY] = pX*n+pY;
```

```
            for(int k=0; k<neighbors.length; k++){
                int nX = pX+ neighbors[k][0];
                int nY = pY + neighbors[k][1];

                if(nX >=0 && nX<m && nY >=0 && nY <n && lands[nX
*n+nY]!=-1 && lands[nX*n+nY] != lands[pX*n+pY]){
                    count--;
                    union(lands, lands[nX*n+nY], lands[pX*n+pY])
;
                }
            }

            result.add(count);
        }

        return result;
    }

    private void union(int[] lands, int pId, int qId){
        for(int i=0;i<lands.length; i++){
            if(lands[i] == qId) lands[i] = pId;
        }
    }
}
```

- Quick-Union, O(klgmn)

```
public class Solution {
    //this is a quick_union solution;
    public List<Integer> numIslands2(int m, int n, int[][] posit
ions) {
        int[] lands = new int[m*n];
        ArrayList<Integer> result = new ArrayList<>();
        int count =0;
        int[][] neighbors = {{1,0},{-1,0},{0,1},{0,-1}};
        for(int i=0; i< m*n; i++){
            lands[i] = -1;
        }
        for(int i=0; i<positions.length;i++){
```

```
            int pX= positions[i][0];
            int pY = positions[i][1];
            if(lands[pX*n+pY]!= -1) continue;
            count++;
            lands[pX*n+pY] = pX*n+pY;
            for(int k=0; k<neighbors.length; k++){
                int nX = pX+ neighbors[k][0];
                int nY = pY + neighbors[k][1];

                if(nX >=0 && nX<m && nY >=0 && nY <n && lands[nX
*n+nY]!=-1){
                    int pRoot = find(lands, pX*n+pY);
                    int nRoot = find(lands, nX*n+nY);
                    if(pRoot != nRoot){
                        count--;
                        lands[pRoot] = nRoot;// union happens he
re
                    }

                }
            }

            result.add(count);
        }

        return result;
    }

    private int find(int[] lands, int index){
        while(index != lands[index]) index = lands[index];
        return index;
    }
}
```

the problem with above solution is that when union happens, each set connects randomly, which cause the next find may be a deep traverse along the tree. by using a weighted method this can be reduced.

- Weighted Quick-Union, reduce the worst case. time consumption reduce a

bit.

```
for(int i=0; i< m*n; i++){
   lands[i] = -1;
   sz[i] = 1;
}
...// same as above;
count--;
// make smaller tree points to larget tree.
if(sz[pRoot] < sz[nRoot]){
    lands[pRoot] = nRoot;
    sz[nRoot] += sz[pRoot];
}else{
    lands[nRoot] = pRoot;
    sz[pRoot] += sz[nRoot];
}
```

we can still improve the performance using path compression, simply update the root information in the find() method

```
//method 1, the entire tree is flatten
int old = index;
while(index != lands[index]) index = lands[index];
while(old != index){
     int tmp = lands[old];
     lands[old] = index;
     old = tmp;
}
//method 2.
for(;index != lands[index]; index = lands[index])
   lands[index] = lands[lands[index]];
```

311. Sparse Matrix Multiplication

Given two sparse matrices A and B, return the result of AB.

You may assume that A's column number is equal to B's row number.

Example:

```
A = [
  [ 1, 0, 0],
  [-1, 0, 3]
]

B = [
  [ 7, 0, 0 ],
  [ 0, 0, 0 ],
  [ 0, 0, 1 ]
]


     |  1 0 0 |   | 7 0 0 |   |  7 0 0 |
AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |
                  | 0 0 1 |
```

Even direct multiplying gives better result. this question is expecting you to use hash table.

```
public class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length;
        int n = A[0].length;
        int l = B[0].length;
        int[][] C = new int[m][l];

        for(int i=0; i< m; i++){
            for(int j = 0; j< n; j++){
                if(A[i][j] != 0){
                    for(int k = 0; k<l; k++)
                        C[i][k] += A[i][j]* B[j][k];
                }
            }
        }

        return C;
    }
}
```

> HashTable solution takes more time to solve the test cases, and it will pay off in larger set.

```
public class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length;
        int n = A[0].length;
        int l = B[0].length;
        int[][] C = new int[m][l];
        // i , j, value
        Map<Integer, Map<Integer, Integer>> map = new HashMap<>(
);

        for(int i=0; i< n; i++){
            map.put(i, new HashMap<Integer, Integer>());
            for(int j = 0; j<l; j++){
                if(B[i][j] != 0){
                    map.get(i).put(j, B[i][j]);
                }
            }
        }
        for(int i=0; i< m; i++){
            for(int j =0; j<n; j++){
                if(A[i][j] != 0){
                    for(int k : map.get(j).keySet()){
                        C[i][k] += A[i][j]  * map.get(j).get(k);
                    }
                }
            }
        }
        return C;
    }
}
```

313. Super Ugly Number

Write a program to find the nth super ugly number.

Super ugly numbers are positive numbers whose all prime factors are in the given prime list primes of size k. For example, [1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32] is the sequence of the first 12 super ugly numbers given primes = [2, 7, 13, 19] of size 4.

Note: (1) 1 is a super ugly number for any given primes. (2) The given numbers in primes are in ascending order. (3) $0 < k \leq 100$, $0 < n \leq 106$, $0 < primes[i] < 1000$.

```
public class Solution {
    public int nthSuperUglyNumber(int n, int[] primes) {

        List<Integer> res = new ArrayList<>();
        res.add(1);
        int[] index = new int[primes.length];
        for(int i=0;i<index.length;i++) index[i] = 0;

        while(true){
            if(res.size() == n) return res.get(res.size()-1);
            int tmp = Integer.MAX_VALUE;
            int smallestIndex = -1;
            for(int i=0; i< index.length;i++){
                int t = res.get(index[i])*primes[i];
                if(t < tmp){
                    tmp = t;
                    smallestIndex = i;
                }
            }
            index[smallestIndex]++;
            if(res.get(res.size()-1) != tmp)
                res.add(tmp);

        }

    }
}
```

another method is to use PriorityQueue

```
to be finished
```

314. Binary Tree Vertical Order Traversal

Given a binary tree, return the vertical order traversal of its nodes' values. (ie, from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from left to right.

Examples:

Given binary tree [3,9,20,null,null,15,7],

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

return its vertical order traversal as:

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

Given binary tree [3,9,8,4,0,1,7],

```
     3
    /\
   /  \
   9   8
  /\  /\
 /  \/  \
 4  01   7
```

return its vertical order traversal as:

```
[
  [4],
  [9],
  [3,0,1],
  [8],
  [7]
]
```

Given binary tree [3,9,8,4,0,1,7,null,null,null,2,5] (0's right child is 2 and 1's left child is 5),

```
     3
    /\
   /  \
   9   8
  /\  /\
 /  \/  \
 4  01   7
    /\
   /  \
   5   2
```

return its vertical order traversal as:

```
[
  [4],
  [9,5],
  [3,0,1],
  [8,2],
  [7]
]
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> verticalOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if(root == null) return res;
        Map<Integer, List<Integer>> dict = new TreeMap<>();
        Queue<TreeNode> queue = new LinkedList<>();
        Queue<Integer> code = new LinkedList<>();

        queue.offer(root);
        code.offer(0);
        while(!queue.isEmpty()){
            TreeNode front = queue.poll();
            int val = code.poll();
            if(!dict.containsKey(val)){
                dict.put(val, new ArrayList<>());
            }
            dict.get(val).add(front.val);
            if(front.left != null){
                queue.offer(front.left);
                code.offer(val-1);
            }
```

```
            if(front.right != null){
                queue.offer(front.right);
                code.offer(val+1);
            }
        }
        res.addAll(dict.values());
        return res;

    }
}
```

1. Count of Smaller Numbers After Self

You are given an integer array nums and you have to return a new counts array. The counts array has the property where counts[i] is the number of smaller elements to the right of nums[i].

Example:

```
Given nums = [5, 2, 6, 1]

To the right of 5 there are 2 smaller elements (2 and 1).
To the right of 2 there is only 1 smaller element (1).
To the right of 6 there is 1 smaller element (1).
To the right of 1 there is 0 smaller element.
Return the array [2, 1, 1, 0].
```

Segment Tree

```
class SegmentTreeNode {
    int start, end;
    int num;
    SegmentTreeNode ltree, rtree;
    public SegmentTreeNode(int s, int e) {
        start = s;
        end = e;
    }
}

public class Solution {
    SegmentTreeNode root = null;

    public SegmentTreeNode buildTree(int[] nums, int left, int r
ight) {
        SegmentTreeNode root = new SegmentTreeNode(left, right);
        if (left != right) {
            int mid = left + (right - left)/2;
            root.ltree = buildTree(nums, left, mid);
            root.rtree = buildTree(nums, mid+1, right);
        }
```

```
        return root;
    }

    private void update(SegmentTreeNode root, int i, int val) {
        if (root.start == root.end) {
            root.num += 1;
        } else {
            int mid = root.start + (root.end - root.start)/2;
            if (i <= mid) {
                update(root.ltree, i, val);
            } else {
                update(root.rtree, i, val);
            }
            root.num = root.ltree.num + root.rtree.num;
        }
    }

    private int query(SegmentTreeNode root, int i, int j) {
        if (root.start == i && root.end == j) {
            return root.num;
        } else {
            int mid = root.start + (root.end - root.start)/2;
            if (j <= mid) {
                return query(root.ltree, i, j);
            } else if (i > mid) {
                return query(root.rtree, i, j);
            } else {
                return query(root.ltree, i, root.ltree.end) + qu
ery(root.rtree, root.rtree.start, j);
            }
        }
    }

    public List<Integer> countSmaller(int[] nums) {
        int[] tmp = nums.clone();
        Arrays.sort(tmp);
        for (int i = 0; i < nums.length; i++) {
            nums[i] = Arrays.binarySearch(tmp, nums[i]) + 1;
        }
        int[] bit = new int[nums.length + 1];
```

```
        root = buildTree(bit, 0, bit.length-1);
        Integer[] ans = new Integer[nums.length];
        for (int i = nums.length - 1; i >= 0; i--) {
            ans[i] = query(root, 0, nums[i] - 1);
            update(root, nums[i], 1);
        }
        return Arrays.asList(ans);
    }
}
```

Use binary to insert each node from back of array, tail element is the root node. this is an augmented binary search tree where each node contains how many nodes have less values than current node, then when new node is inserted and the new value is greater than root node, there will be root.count number of nodes, the count is initialized to 1, cause root node itself count when greater value is inserted.

```
public class Solution {
    public List<Integer> countSmaller(int[] nums) {
        List<Integer> res = new ArrayList<>();
        if(nums.length == 0) return res;
        TreeNode root= new TreeNode(nums[nums.length-1]);
        res.add(0);
        for(int i= nums.length-2; i>=0; i-- ){
            int count = insert(root, nums[i]);
            res.add(count);
        }
        Collections.reverse(res);
        return res;
    }

    private int insert(TreeNode root, int val){
        int c = 0;
        while(true){
            if(val <=  root.val){
                root.count++;
                if(root.left == null){
                    root.left = new TreeNode(val);
                    break;
```

```
                }else{
                    root = root.left;
                }

            }else{
                c += root.count;
                if(root.right == null){
                    root.right = new TreeNode(val);
                    break;
                }else{
                    root = root.right;
                }
            }
        }
        return c;
    }

    class TreeNode{
        TreeNode left;
        TreeNode right;
        int val;
        int count = 1;
        public TreeNode(int val){
            this.val = val;
        }
    }
}
```

317. Shortest Distance from All Buildings

You want to build a house on an empty land which reaches all buildings in the shortest amount of distance. You can only move up, down, left and right. You are given a 2D grid of values 0, 1 or 2, where:

Each 0 marks an empty land which you can pass by freely.

Each 1 marks a building which you cannot pass through.

Each 2 marks an obstacle which you cannot pass through.

For example, given three buildings at (0,0), (0,4), (2,2), and an obstacle at (0,2):

```
1 - 0 - 2 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

The point (1,2) is an ideal empty land to build a house, as the total travel distance of 3+3+1=7 is minimal. So return 7.

Note: There will be at least one building. If it is not possible to build such house according to the above rules, return -1.

The question requires BFS to solve, classic, generally need queue.

```java
public class Solution {
    public int shortestDistance(int[][] grid) {

        //This is a bfs solution, bfs needs queue generally.

        int row = grid.length;
        if(row == 0) return -1;
        int col  = grid[0].length;
        if(col == 0) return -1;
```

```
        int buildingNums = 0;

        int[][] dis = new int[row][col]; // distance sum of all
bulding to dis[x][y];
        int[][] num = new int[row][col]; // how many buildings c
an reach num[x][y]

        for(int i=0 ; i< row; i++){
            for(int j = 0; j< col; j++){
                if(grid[i][j] == 1){
                    buildingNums++;
                    bfs(grid, dis, num, i, j);
                }
            }
        }

        int min = Integer.MAX_VALUE;
        for(int i=0; i<row; i++){
            for(int j=0; j<col; j++){
                if(grid[i][j] == 0 && dis[i][j] != 0 && num[i][j
] == buildingNums){
                    min = Math.min(min, dis[i][j]);
                }
            }
        }
        if(min < Integer.MAX_VALUE) return min;
        return -1;
    }

    private void bfs(int[][] grid, int[][] dis, int[][] num, int
 x, int y){
        int row = grid.length;
        int col = grid[0].length;
        int[][] neighbor = {{1,0},{-1,0},{0,1},{0,-1}};
        Queue<int[]> queue = new LinkedList<>();
        queue.offer(new int[]{x, y});

        boolean[][] visited = new boolean[row][col];
        int dist = 0;
        while(!queue.isEmpty()){
```

```
            dist++;
            int size = queue.size();// all size number of node,
their neigbors belongs to next dist, which for distance.
            for(int i=0 ; i<size; i++){
                int[] top = queue.poll();
                for(int j=0; j< 4;j++){
                    int k = top[0] + neighbor[j][0];
                    int l = top[1] + neighbor[j][1];
                    if(k>=0 && k< row && l >= 0 && l < col && gr
id[k][l] == 0 && !visited[k][l]){
                        visited[k][l] = true;
                        dis[k][l] += dist;
                        num[k][l]++;
                        queue.add(new int[]{k,l});
                    }
                }
            }
        }

    }
}
```

319. Bulb Switcher

There are n bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the ith round, you toggle every i bulb. For the nth round, you only toggle the last bulb. Find how many bulbs are on after n rounds.

Example:

Given n = 3.

```
At first, the three bulbs are [off, off, off].
After first round, the three bulbs are [on, on, on].
After second round, the three bulbs are [on, off, on].
After third round, the three bulbs are [on, off, off].
```

So you should return 1, because there is only one bulb is on.

There are three types of numbers.

- prime number: only factor is 1 and itself, when at 1, the bulb is set, when at itself, the bulb is unset, and there is not other way the prime-number-index bulb get accessed again.
- regular number(not perfect square number), since each such number will have factors count in pairs. so the result of such bulb is reset to off.
- perfect square number, it may have several pairs of factors, but it also has a factor which x^2 = n, this x number only access the bulb once, so all the perfect-square number is will keep the bulb on.

```java
public class Solution {
    public int bulbSwitch(int n) {
        return (int)Math.sqrt(n);
    }
}
```

322. Coin Change

You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

```
Example 1:
coins = [1, 2, 5], amount = 11
return 3 (11 = 5 + 5 + 1)
```

```
Example 2:
coins = [2], amount = 3
return -1.
```

Note: You may assume that you have an infinite number of each kind of coin.

This is a forward DP, by knowing DP[i], calculate DP[i + x]. **Note the direction**, also here we need to caculate dp[i] + 1, which may overflow. so the MAX value is **actually** Integer.MAX_VALUE -1, or 0x7ffffffe;

```
public class Solution {
    public int coinChange(int[] coins, int amount) {
        int dp[] = new int[amount+1];
        for(int i= 1; i<= amount; i++){
            dp[i] = 0x7ffffffe;// why not Integer.MAX_VALUE ???
        }

        for(int i= 0; i<= amount; i++){
            for(int c : coins){
                if(i+c <= amount){
                    dp[i + c] = Math.min(dp[i+c], dp[i] + 1);
                    //casue dp[i] may be Integer.MAX_VALUE, and
+ 1 make it overflow;
                }
            }
        }
        return dp[amount] == 0x7ffffffe ? -1 : dp[amount];
    }
}
```

324. Wiggle Sort II

Given an unsorted array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3]....

Example: (1) Given nums = [1, 5, 1, 1, 6, 4], one possible answer is [1, 4, 1, 5, 1, 6].

(2) Given nums = [1, 3, 2, 2, 3, 1], one possible answer is [2, 3, 1, 3, 1, 2].

Note: You may assume all input has valid answer.

> Follow Up: Can you do it in O(n) time and/or in-place with O(1) extra space?

Solution

> first sort the array, then cut the array 2 parts, each round , pick a number from the end of each parts.

```
public class Solution {
    public void wiggleSort(int[] nums) {
        if(nums == null ) return ;

        int[] tmp = Arrays.copyOf(nums, nums.length);
        Arrays.sort(tmp);
        int i=0;
        int j=tmp.length;
        int k = (tmp.length+1)/2;

        for(i=0; i< nums.length; i++){
            nums[i] = (i & 1) == 0 ?  tmp[--k] : tmp[--j];
        }

    }
}
```

> How to do the follow up ??

325. Maximum Size Subarray Sum Equals k

Given an array nums and a target value k, find the maximum length of a subarray that sums to k. If there isn't one, return 0 instead.

Example 1: Given nums = [1, -1, 5, -2, 3], k = 3, return 4. (because the subarray [1, -1, 5, -2] sums to 3 and is the longest)

Example 2: Given nums = [-2, -1, 2, 1], k = 1, return 2. (because the subarray [-1, 2] sums to 1 and is the longest)

```
public class Solution {
    public int maxSubArrayLen(int[] nums, int k) {
        if(nums == null || nums.length == 0) return 0;
        // save sum from 0 to i, and sum is key,
        Map<Integer, Integer> map = new HashMap<>();
        int len = Integer.MIN_VALUE;
        int sum = 0;
        map.put(0, -1);// This is very important, if any number
range sum[i..j] == 0, then without this line,
        //j-i is not count as the max len, this line is used to
offset the [i...j] range.
        for(int i= 0; i< nums.length; i++){
            sum += nums[i];
            if(!map.containsKey(sum)){
                map.put(sum, i);
            }
            if( map.containsKey(sum - k)){
                len = Math.max(len, i - map.get(sum - k));
            }
        }

        return len == Integer.MIN_VALUE ? 0 : len;
    }
}
```

328. Odd Even Linked List

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example: Given 1->2->3->4->5->NULL, return 1->3->5->2->4->NULL.

Note: The relative order inside both the even and odd groups should remain as it was in the input. The first node is considered odd, the second node even and so on ...

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode oddEvenList(ListNode head) {
        ListNode odd = new ListNode(-1);
        ListNode even = new ListNode(-1);
        ListNode o = odd;
        ListNode e = even;
        int count = 0;

        while(head != null){
            count++;
            if((count&1) == 1){
                o.next = head;
                o = o.next;
            }else{
                e.next = head;
                e = e.next;
            }
            head = head.next;
        }
        e.next = null;
        o.next = even.next;

        return odd.next;

    }
}
```

329. Longest Increasing Path in a Matrix

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:

```
nums = [
  [9,9,4],
  [6,6,8],
  [2,1,1]
]
```

Return 4

The longest increasing path is [1, 2, 6, 9].

Example 2:

```
nums = [
  [3,4,5],
  [3,2,6],
  [2,2,1]
]
```

Return 4

The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

DONT think too hard. it is simply a dfs traversal on each node, while traversing, remember the longest distance from each node.

```
public class Solution {
    int[][] neighbor = {{-1,0}, {0, -1}, {1, 0}, {0,1}};

    public int longestIncreasingPath(int[][] matrix) {

        if(matrix.length == 0) return 0;
        int[][] dis = new int[matrix.length][matrix[0].length];
        int res = 0;
        for(int i=0; i< matrix.length; i++){
            for(int j=0; j< matrix[0].length; j++){
                res = Math.max(res, dfs(matrix, i, j, dis));
            }
        }
        return res;
    }

    private int dfs(int[][] matrix, int x, int y, int[][] dis){
        if(dis[x][y] != 0) return dis[x][y];

        for(int i=0; i<4; i++){
            int k = x + neighbor[i][0];
            int l = y + neighbor[i][1];
            if(k >=0 && k<matrix.length && l >=0 && l < matrix[0
].length && matrix[x][y] < matrix[k][l]){
                dis[x][y] = Math.max(dis[x][y], dfs(matrix, k, l
, dis));
            }
        }
        return ++dis[x][y];
    }
}
```

330. Patching Array

Given a sorted positive integer array nums and an integer n, add/patch elements to the array such that any number in range [1, n] inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

Example 1:

nums = [1, 3], n = 6

Return 1.

Combinations of nums are [1], [3], [1,3], which form possible sums of: 1, 3, 4.

Now if we add/patch 2 to nums, the combinations are: [1], [2], [3], [1,3], [2,3], [1,2,3].

Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6].

So we only need 1 patch.

Example 2:

nums = [1, 5, 10], n = 20

Return 2.

The two patches can be [2, 4].

Example 3:

nums = [1, 2, 2], n = 5

Return 0

Consider [1,2,4,11,30]

with 1,2,4, we can represent [1...7], but 8 cannot, so patch 8, after patch 8, the entire array can represent [1...15], can the patched array combine to 16 ? anwser is yes, [1...15] + 11 can represent up to 26, so no need to add 16 or any number

before 27, then just same as patching 8, we need to patch 27, now the entire array can combine to 53, which is > 50, so only need to patch 2 numbers.

```
why the missing has to be long ?

cause missing += nums[i], missing+=missing, may overflow(for int
).
which will cause the while loop never ends
```

```
public class Solution {
    public int minPatches(int[] nums, int n) {
        int count = 0;
        long missing = 1; // before missing number, all numbers
can be combined.
                // has to be long, cause missing+=missing or mis
sing += nums[i] may overflow when missing is int.
        int i = 0;
        while(missing <= n){
            if( i < nums.length && nums[i] <= missing ){
                missing += nums[i];// now I can sum up to missin
g+nums[i] - 1, next missing .
                i++;
            }else{
                missing += missing; // next missing number.
                count++;
            }
        }

        return count;
    }
}
```

333. Largest BST Subtree

Given a binary tree, find the largest subtree which is a Binary Search Tree (BST), where largest means subtree with largest number of nodes in it.

Note: A subtree must include all of its descendants. Here's an example:

```
    10
    / \
   5  15
  / \   \
 1   8   7
```

The Largest BST Subtree in this case is the highlighted one. The return value is the subtree's size, which is 3.

> **Related Issue 98 Validate Binary Search Tree**

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    /*
        1. you need to track each subtree is bst or not.
        2. you need to track the size of subtree if it is a bst.
        3. thus global variable/TreeNode won't keep consistent i
nfo
            regarding 1&2.
        4. you need a wrapper to hold such 2 information. along
with the
            current range of substree.
```

```
    */
    class Node{
        int size;
        int left,right;
        boolean isBst;
        Node(){
            size = 0;
            isBst = true;
            left = Integer.MAX_VALUE;
            right = Integer.MIN_VALUE;
        }
    }
    public int largestBSTSubtree(TreeNode root) {

        Node n = isBST(root);
        return n.size;
    }

    Node isBST(TreeNode root){
        Node node = new Node();
        if(root == null){
            return node;
        }

        Node l = isBST(root.left);
        Node r = isBST(root.right);

        node.left = Math.min(l.left, root.val);
        node.right = Math.max(r.right, root.val);

        if(l.isBst && r.isBst && l.right <= root.val && r.left >
= root.val){
            node.size = l.size + r.size +1;
            node.isBst = true;
        }else{
            node.size = Math.max(l.size, r.size);
            node.isBst = false;
        }
```

```
            return node;
        }
    }
```

334. Increasing Triplet Subsequence

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array.

Formally the function should: Return true if there exists i, j, k such that **arr[i] < arr[j] < arr[k] given 0 ≤ i < j < k ≤ n-1** else return false.

Your algorithm should run in O(n) time complexity and O(1) space complexity.

Examples: Given [1, 2, 3, 4, 5], return true.

Given [5, 4, 3, 2, 1], return false.

Related issue Longest Increasing Subsequence

Use two variables, min, seondMin to narrow the search range, initially, both set to MAX_VALUE, min is the smallest number so far, and secondMin is a number larger than min, but after min's position.

```
if val <= min, min =val;
if min < val <= secondMin, secondMin = val;
else return true; val is the third val in the sequence.
```

```java
public class Solution {
    public boolean increasingTriplet(int[] nums) {
        int min = Integer.MAX_VALUE;
        int secondMin = Integer.MAX_VALUE;

        for(int val : nums){
            if(val <= min){
                min = val;
            }else if(min < val && val <= secondMin){
                secondMin = val;
            }else{
                return true;
            }
        }
        return false;
    }
}
```

1. House Robber III

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example 1:

```
     3
    / \
   2   3
    \   \
     3   1
```

Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.

Example 2:

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

Maximum amount of money the thief can rob = 4 + 5 = 9.

This one has to be recursive, cause you don't really know which is going to be picked during the process. RobValue(Root)=Max of

1. RobValue(Root.left) + RobValue(Root.right)
2. RobValue(Root.left without left child) + RobValue(Root.right without right child) + Root.val;

The trick of this question is to return a pair of value from recursive call, res[2], first value is the max rob value from the current node, and res[1] is the max rob value without rob current node.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int rob(TreeNode root) {
        return rob_(root)[0];
    }

    private int[] rob_(TreeNode root){
        int[] res = {0 /*max of rob current sub tree*/,
                     0 /* max of rob left child and right child*/
};
        if(root != null){
            int[] left = rob_(root.left);
            int[] right = rob_(root.right);
            res[1] = left[0] + right[0];
            res[0] = Math.max(res[1], left[1] + right[1] + root.
val);
        }

        return res;
    }
}
```

339. Nested List Weight Sum

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

Given the list [[1,1],2,[1,1]], return 10. (four 1's at depth 2, one 2 at depth 1)

Example 2:

Given the list [1,[4,[6]]], return 27. (one 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 4*2* + 6*3* = 27)

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * public interface NestedInteger {
 *
 *     // @return true if this NestedInteger holds a single integer, rather than a nested list.
 *     public boolean isInteger();
 *
 *     // @return the single integer that this NestedInteger holds, if it holds a single integer
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // @return the nested list that this NestedInteger holds, if it holds a nested list
 *     // Return null if this NestedInteger holds a single integer
 *     public List<NestedInteger> getList();
 * }
```

```
 */
public class Solution {
    public int depthSum(List<NestedInteger> nestedList) {
        int res = 0;
        for(NestedInteger ni : nestedList){
            res += weight(ni, 1);
        }
        return res;
    }

    private int weight(NestedInteger ni, int w){
        if(ni.isInteger()){
            return ni.getInteger() * w;
        }
        int res =0;
        for(NestedInteger nni : ni.getList()){
            res += weight(nni, w+1);
        }
        return res;
    }
}
```

> a clearer solution.

```
 public class Solution {
    public int depthSum(List<NestedInteger> nestedList) {
        return sum(nestedList, 1);

    }
    private int sum(List<NestedInteger> nl, int depth){
        int sum = 0;
        for(NestedInteger ni : nl){
            if(ni.isInteger()){
                sum += ni.getInteger() * depth;
            }else{
                sum += sum(ni.getList(), depth+1);
            }
        }
        return sum;
    }

}
```

341. Flatten Nested List Iterator

Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

Given the list [[1,1],2,[1,1]],

By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,1,2,1,1].

Example 2:

Given the list [1,[4,[6]]],

By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,4,6].

```
/**
 * // This is the interface that allows for creating nested list
s.
 * // You should not implement it, or speculate about its implem
entation
 * public interface NestedInteger {
 *
 *     // @return true if this NestedInteger holds a single inte
ger, rather than a nested list.
 *     public boolean isInteger();
 *
 *     // @return the single integer that this NestedInteger hol
ds, if it holds a single integer
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // @return the nested list that this NestedInteger holds,
 if it holds a nested list
```

```
 *     // Return null if this NestedInteger holds a single integ
er
 *     public List<NestedInteger> getList();
 * }
 */
public class NestedIterator implements Iterator<Integer> {

    private List<Integer> mList = new ArrayList<>();
    private int index = 0;
    public NestedIterator(List<NestedInteger> nestedList) {
        for(int i=0;i<nestedList.size(); i++){
            flatten(nestedList.get(i));
            index = 0;
        }

    }

    private void flatten(NestedInteger ni){
        if(ni.isInteger()){
            mList.add(ni.getInteger());
        }else{
            for(NestedInteger i : ni.getList()){
                flatten(i);
            }
        }
    }

    @Override
    public Integer next() {
        return mList.get(index++);
    }

    @Override
    public boolean hasNext() {
        return index < mList.size();
    }
}

/**
 * Your NestedIterator object will be instantiated and called as
```

```
 such:
 * NestedIterator i = new NestedIterator(nestedList);
 * while (i.hasNext()) v[f()] = i.next();
 */
```

> You can also use a stack:
>
> - reversely push each NestedInteger into a stack.
> - hasNext will check the top if is value then return true, else, poll it push all its content into the stack again, reversely. from size()-1;

```
/**
 * // This is the interface that allows for creating nested list
s.
 * // You should not implement it, or speculate about its implem
entation
 * public interface NestedInteger {
 *
 *     // @return true if this NestedInteger holds a single inte
ger, rather than a nested list.
 *     public boolean isInteger();
 *
 *     // @return the single integer that this NestedInteger hol
ds, if it holds a single integer
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // @return the nested list that this NestedInteger holds,
 if it holds a nested list
 *     // Return null if this NestedInteger holds a single integ
er
 *     public List<NestedInteger> getList();
 * }
 */
public class NestedIterator implements Iterator<Integer> {

    Stack<NestedInteger> stack = new Stack<>();
    public NestedIterator(List<NestedInteger> nestedList) {
        for(int i=nestedList.size()-1; i>=0; i--){
```

```
                stack.push(nestedList.get(i));
            }
        }

        @Override
        public Integer next() {
            return stack.pop().getInteger();
        }

        @Override
        public boolean hasNext() {
            while(!stack.isEmpty()){
                NestedInteger ni = stack.peek();
                if(ni.isInteger()){
                    return true;
                }else{
                    stack.pop();
                    List<NestedInteger> lni = ni.getList();
                    for(int i = lni.size() -1; i>= 0; i--){
                        stack.push(lni.get(i));
                    }
                }
            }

            return false;
        }
    }

    /**
     * Your NestedIterator object will be instantiated and called as
     such:
     * NestedIterator i = new NestedIterator(nestedList);
     * while (i.hasNext()) v[f()] = i.next();
     */
```

342. Power of Four

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example: Given num = 16, return true. Given num = 5, return false.

```
public class Solution {
    public boolean isPowerOfFour(int num) {

        return num != 0 && (num & (num-1)) == 0 && (num & 0xAAAA
AAAA) == 0;

    }
}
```

343. Integer Break

Given a positive integer n, break it into the sum of at least two positive integers and maximize the product of those integers. Return the maximum product you can get.

For example, given n = 2, return 1 (2 = 1 + 1); given n = 10, return 36 (10 = 3 + 3 + 4).

Note: you may assume that n is not less than 2.

DP.

```
product[i] = max(product[k] * (i-k), k * (i-k); k is [1...i)
```

```
public class Solution {
    public int integerBreak(int n) {
        if(n== 0) return 0;
        int[] product = new int[n+1];
        product[1]= 1;
        for(int i=1; i<=n;i++){
            for(int k=1; k<i;k++){  // <i?
                product[i] = Math.max(product[i], Math.max( k *
(i-k), product[k] * (i-k)));
            }
        }
        return product[n];
    }
}
```

344. Reverse String

Write a function that takes a string as input and returns the string reversed.

Example: Given s = "hello", return "olleh".

```
public class Solution {
    public String reverseString(String s) {
        char[] chars = s.toCharArray();
        int l =0, r = chars.length-1;
        while(l < r){
            char tmp = chars[l];
            chars[l] = chars[r];
            chars[r] = tmp;
            l++;
            r--;
        }

        return new String(chars);

    }
}
```

345. Reverse Vowels of a String

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1:

Given s = "hello", return "holle".

Example 2:

Given s = "leetcode", return "leotcede".

NOTICE THAT left cannot go beyond right, otherwise you end up swap already swapped pairs.

```
public class Solution {
    public String reverseVowels(String s) {
        char[] d = s.toCharArray();
        String dict = "aeiouAEIOU";

        int left = 0;
        int right = d.length-1;
        while(left < right){
            while(left < d.length && dict.indexOf(d[left]) == -1
) left++;
            if(left >= d.length || left >= right) break;
            while(right >= 0 && dict.indexOf(d[right]) == -1) ri
ght--;
            if(right < 0 || right <= left) break;

            char tmp = d[left];
            d[left] = d[right];
            d[right] = tmp;
            left++;
            right--;
        }

        return new String(d);
    }
}
```

346. Moving Average from Data Stream

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

For example,

```
MovingAverage m = new MovingAverage(3);
m.next(1) = 1
m.next(10) = (1 + 10) / 2
m.next(3) = (1 + 10 + 3) / 3
m.next(5) = (10 + 3 + 5) / 3
```

```
public class MovingAverage {

    /** Initialize your data structure here. */
    Queue<Integer> data = new LinkedList<>();
    int size;
    int sum;
    public MovingAverage(int size) {
        this.size = size;
        this.sum = 0;
    }

    public double next(int val) {
        if(data.size() >= size){
            data.offer(val);
            int head = data.poll();
            sum = sum - head + val;

            return sum/size;
        }else{
            sum += val;
            data.offer(val);
            return sum / data.size();
        }
    }
}

/**
 * Your MovingAverage object will be instantiated and called as
such:
 * MovingAverage obj = new MovingAverage(size);
 * double param_1 = obj.next(val);
 */
```

347. Top K Frequent Elements

Given a non-empty array of integers, return the k most frequent elements.

For example, Given [1,1,1,2,2,3] and k = 2, return [1,2].

Note: You may assume k is always valid, 1 ≤ k ≤ number of unique elements.

Your algorithm's time complexity must be better than O(n log n), where n is the array's size.

There is one case, if two value has same count.

```
public class Solution {
    public List<Integer> topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();
        for(int val : nums){
            if(map.containsKey(val)){
                map.put(val, map.get(val) + 1);
            }else{
                map.put(val, 1);
            }
        }

        class Pair{
            int key;
            int count;
            Pair(int k, int c){
                key = k;
                count = c;
            }
        }

        Comparator<Pair> comp = new Comparator<Pair>(){
            @Override
            public int compare(Pair p1, Pair p2){
                if(p1.count != p2.count) return p1.count - p2.co
unt;
                else return p1.key - p2.key;
```

```
            }
        };

        PriorityQueue<Pair> queue = new PriorityQueue<>(k, comp)
;

        for(int val : nums){
            if(!map.containsKey(val)) continue;
            int count = map.get(val);
            Pair p = new Pair(val, count);
            if(queue.size() < k){
                queue.offer(p);
            }else{
                Pair top = queue.peek();
                if(p.count > top.count){
                    queue.poll();
                    queue.offer(p);
                }
            }
            map.remove(val);
        }
        List<Integer> res = new ArrayList<>();
        for(Pair p : queue){
            res.add(p.key);
        }
        return res;

    }
}
```

349. Intersection of Two Arrays

Given two arrays, write a function to compute their intersection.

Example: Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2].

Note:

Each element in the result must be unique.

The result can be in any order.

```
//TODO: you don't really need the set after sort arrays.
public class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        int i=0, j=0;
        Set<Integer> set  = new HashSet<>();
        Arrays.sort(nums1);
        Arrays.sort(nums2);
        while( i < nums1.length && j < nums2.length){
            if(nums1[i] > nums2[j]){
                j++;
            }else if(nums1[i] < nums2[j]){
                i++;
            }else{
                set.add(nums1[i]);
                i++;j++;
            }
        }

        int[] res = new int[set.size()];
        int k =0;
        for(Integer v : set){
            res[k++] = v;
        }

        return res;
    }
}
```

> You can also use set

```java
public class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        Set<Integer> set = new HashSet<>();
        Set<Integer> res = new HashSet<>();

        for(int v : nums1){
            set.add(v);
        }
        for(int v : nums2){
            if(set.contains(v)) res.add(v);
        }

        int[] r = new int[res.size()];
        int k = 0;
        for(int i : res){
            r[k++] = i;
        }

        return r;
    }
}
```

350. Intersection of Two Arrays II

Given two arrays, write a function to compute their intersection.

Example: Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2, 2].

Note: Each element in the result should appear as many times as it shows in both arrays. The result can be in any order.

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?

> Use two pointers

- What if nums1's size is small compared to nums2's size? Which algorithm is better?

> Use Hashmap.

- What if elements of nums2 are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

```
public class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        int i=0,j=0;
        Arrays.sort(nums1);
        Arrays.sort(nums2);

        int[] res = new int[nums1.length > nums2.length ? nums2.
length : nums1.length];
        int k = 0;

        while(i < nums1.length && j < nums2.length){
            if(nums1[i] == nums2[j]){
                res[k++] = nums1[i++];
                j++;
            }else{
               int t = nums1[i] > nums2[j] ? j++ : i++;
            }
        }

        return Arrays.copyOfRange(res, 0, k);
    }
}
```

352. Data Stream as Disjoint Intervals

Given a data stream input of non-negative integers a1, a2, ..., an, ..., summarize the numbers seen so far as a list of disjoint intervals.

For example, suppose the integers from the data stream are 1, 3, 7, 2, 6, ..., then the summary will be:

```
[1, 1]
[1, 1], [3, 3]
[1, 1], [3, 3], [7, 7]
[1, 3], [7, 7]
[1, 3], [6, 7]
```

Follow up: What if there are lots of merges and the number of disjoint intervals are small compared to the data stream's size?

> Time limit exceeded. since it is O(n) complexity. since fast set is O(1), fast get is O(n).
>
> ***TreeSet***

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class SummaryRanges {
    Set<Integer> set = new TreeSet<>();
    /** Initialize your data structure here. */
    public SummaryRanges() {

    }
```

```
    //whether fast add or fast get
    public void addNum(int val) {
        set.add(val);
    }

    public List<Interval> getIntervals() {
        List<Interval> list = new ArrayList<>();
        Set<Integer> checked =new HashSet<>();
        for(Integer i : set){
            if(checked.contains(i)) continue;
            Interval interval = new Interval(i,i);
            checked.add(i);
            int l = i-1;
            while(set.contains(l)){
                interval.start--;
                checked.add(l--);
            }
            int r = i+1;
            while(set.contains(r) ) {
                interval.end++;
                checked.add(r++);
            }
            list.add(interval);
        }
        return list;
    }
}

/**
 * Your SummaryRanges object will be instantiated and called as
such:
 * SummaryRanges obj = new SummaryRanges();
 * obj.addNum(val);
 * List<Interval> param_2 = obj.getIntervals();
 */
```

> use fast get, O(lgN) set, use the property of treeset.

```
/**
```

```
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class SummaryRanges {
    TreeSet<Interval> set;
    /** Initialize your data structure here. */
    public SummaryRanges() {
        set = new TreeSet<Interval>(new Comparator<Interval>(){
                            @Override
                            public int compare(Interval i1, Inte
rval i2){
                                return i1.start - i2.start;
                            }
        });
    }
    //whether fast add or fast get
    public void addNum(int val) {
        Interval v = new Interval(val, val);

        Interval f = set.floor(v);
        if(f != null){
            if(f.end >= val) return;
            if(f.end + 1 == val){
                v.start = f.start;
                set.remove(f);
            }
        }

        Interval h = set.higher(v);

        if(h!= null && h.start == val +1){
            v.end = h.end;
            set.remove(h);
        }
```

```
            set.add(v);
        }

        public List<Interval> getIntervals() {
            List<Interval> list = new ArrayList<Interval>();
            list.addAll(set);
            return list;
        }
}

/**
 * Your SummaryRanges object will be instantiated and called as
such:
 * SummaryRanges obj = new SummaryRanges();
 * obj.addNum(val);
 * List<Interval> param_2 = obj.getIntervals();
 */
```

356. Line Reflection

Given n points on a 2D plane, find if there is such a line parallel to y-axis that reflect the given points.

Example 1: Given points = [[1,1],[-1,1]], return true.

Example 2: Given points = [[1,1],[-1,-1]], return false.

Follow up: Could you do better than O(n2)?

```
public class Solution {
    public boolean isReflected(int[][] points) {
        if(points == null || points.length <=1 ) return true;

        int left =points[0][0];
        int right = left;

        Map<Integer, Set<Integer>> map = new HashMap<>();
        for(int[] p : points){
            left = Math.min(p[0], left);
            right = Math.max(p[0], right);
            if(!map.containsKey(p[1])){
                Set<Integer> set = new HashSet<>();
                set.add(p[0]);
                map.put(p[1], set);
            }else{
                map.get(p[1]).add(p[0]);
            }
        }
        for(int[] p : points){
            Set<Integer> s = map.get(p[1]);
            if(!s.contains(left + right - p[0])) return false;
        }
        return true;

    }
}
```

357. Count Numbers with Unique Digits

Given a non-negative integer n, count all numbers with unique digits, x, where 0 ≤ x < 10n.

Example:

Given n = 2, return 91. (The answer should be the total numbers in the range of 0 ≤ x < 100, excluding `[11,22,33,44,55,66,77,88,99]`)

Hint:

A direct way is to use the backtracking approach.

Backtracking should contains three states which are (the current number, number of steps to get that number and a bitmask which represent which number is marked as visited so far in the current number). Start with state (0,0,0) and count all valid number till we reach number of steps equals to 10n.

This problem can also be solved using a dynamic programming approach and some knowledge of combinatorics.

Let f(k) = count of numbers with unique digits with length equals k.

f(1) = 10, ..., f(k) = 9 *9* 8 * ... (9 - k + 2) [The first factor is 9 because a number cannot start with 0].

The Unique digits here means no-duplicates digits for this number.

dp, if n:

- 0, result is 1, f(0)
- 1, 10, which is 9 + f(0);
- 2, 91, which is 9*9 + f(1), first number gets 9 choice(no zero), next is still 9(with zero),
- 3, `9*9*8` + f(2), each bit will get a choice of 9 - (i-2), i is current bit, -2 means excludes first two bits.

```
public class Solution {
    public int countNumbersWithUniqueDigits(int n) {
        int res = 1;
        for(int i = 1; i<=n;i++){
            int tmp = 9;
            // j>0 here is redundant, the number with no duplica
tes should < 10-bit, otherwise there must be a duplicates.
            for(int j =9; j>0 && j>= 9-i+2; j--)
                tmp *= j;

            res += tmp;
        }

        return res;
    }
}
```

358. Rearrange String k Distance Apart

Given a non-empty string str and an integer k, rearrange the string such that the same characters are at least distance k from each other.

All input strings are given in lowercase letters. If it is not possible to rearrange the string, return an empty string "".

Example 1: str = "aabbcc", k = 3

Result: "abcabc"

The same letters are at least distance 3 from each other. Example 2: str = "aaabc", k = 3

Answer: ""

It is not possible to rearrange the string. Example 3: str = "aaadbbcc", k = 2

Answer: "abacabcd"

Another possible answer is: "abcabcda"

The same letters are at least distance 2 from each other.

> This question is tricky, not only need you process the string tally the number, then you need to use heap to pull char out, and in a defined sequence.

```
public class Solution {
    public String rearrangeString(String str, int k) {

        if(k == 0) return str;

        int len = str.length();
        Map<Character, Integer> counts = new HashMap<>();
        for(int i=0; i< len; i++){
            char ch = str.charAt(i);
            int n =1;
            if(counts.containsKey(ch)){
                n = counts.get(ch)+1;
```

```
            }
            counts.put(ch, n);
        }

        PriorityQueue<Pair> pq = new PriorityQueue<>(10, new Com
parator<Pair>(){
            @Override
            public int compare(Pair p1, Pair p2){
                if(p1.cnt != p2.cnt) return p2.cnt - p1.cnt;
                else return  p2.ch - p1.ch; // to ensure the ord
er of the chars with same count, they should show up in same ord
er.
            }
        });

        for(Map.Entry<Character, Integer> entry : counts.entrySe
t()){
            pq.offer(new Pair(entry.getKey(), entry.getValue()));
// pick the most show-up char first.
        }

        StringBuilder sb = new StringBuilder();
        while(!pq.isEmpty()){
            List<Pair> tmp = new ArrayList<>();// this is avoid
you pick up same char in the same k-segment.
            int d = Math.min(k, len);
            for(int i=0; i< d; i++){
                if(pq.isEmpty()) return "";
                Pair p = pq.poll();
                sb.append(p.ch);
                if(--p.cnt > 0) tmp.add(p);
                len--;
            }
            for(Pair p : tmp) pq.offer(p);

        }

        return sb.toString();

    }
```

```
    class Pair{
        char ch;
        int cnt;
        Pair(char c, int t){
            ch = c;
            cnt = t;
        }
    };
}
```

359. Logger Rate Limiter

Design a logger system that receive stream of messages along with its timestamps, each message should be printed if and only if it is not printed in the last 10 seconds.

Given a message and a timestamp (in seconds granularity), return true if the message should be printed in the given timestamp, otherwise returns false.

It is possible that several messages arrive roughly at the same time.

Example:

```
Logger logger = new Logger();

// logging string "foo" at timestamp 1
logger.shouldPrintMessage(1, "foo"); returns true;

// logging string "bar" at timestamp 2
logger.shouldPrintMessage(2,"bar"); returns true;

// logging string "foo" at timestamp 3
logger.shouldPrintMessage(3,"foo"); returns false;

// logging string "bar" at timestamp 8
logger.shouldPrintMessage(8,"bar"); returns false;

// logging string "foo" at timestamp 10
logger.shouldPrintMessage(10,"foo"); returns false;

// logging string "foo" at timestamp 11
logger.shouldPrintMessage(11,"foo"); returns true;
```

```
public class Logger {
    Map<String, Integer> map = new HashMap<>(); // msg : lst pri
nt timestamp
    int limiter = 10;
    /** Initialize your data structure here. */
    public Logger() {

    }

    /** Returns true if the message should be printed in the giv
en timestamp, otherwise returns false.
        If this method returns false, the message will not be pr
inted.
        The timestamp is in seconds granularity. */
    public boolean shouldPrintMessage(int timestamp, String mess
age) {
        if(!map.containsKey(message)){
            map.put(message, timestamp);
            return true;
        }else{
            if(timestamp - map.get(message) >= limiter){
                map.put(message, timestamp);
                return true;
            }
        }

        return false;
    }
}

/**
 * Your Logger object will be instantiated and called as such:
 * Logger obj = new Logger();
 * boolean param_1 = obj.shouldPrintMessage(timestamp,message);
 */
```

360. Sort Transformed Array

Given a sorted array of integers nums and integer values a, b and c. Apply a function of the form f(x) = ax2 + bx + c to each element x in the array.

The returned array must be in sorted order.

Expected time complexity: O(n)

Example:

```
nums = [-4, -2, 2, 4], a = 1, b = 3, c = 5,

Result: [3, 9, 15, 33]

nums = [-4, -2, 2, 4], a = -1, b = 3, c = 5
```

Result: [-23, -5, 1, 7]

```
public class Solution {
    public int[] sortTransformedArray(int[] nums, int a, int b,
int c) {
        if(nums == null || nums.length == 0) return nums;
        if(nums.length == 1){
            nums[0] = eval(nums[0],a,b,c);
            return nums;
        }

        int l = 0;
        int r = nums.length -1;
        int[] res = new int[nums.length];
        int k = 0;
        while(l <= r){ // need to equal to get the final number.
            int v1 = eval(nums[l], a, b, c);
            int v2 = eval(nums[r], a, b, c);

            if(a > 0){
                res[k++] = v1 > v2 ? v1 : v2;
```

```
                if(v1 > v2)  l++;
                else r--;
            }else{
                res[k++] = v1 > v2 ? v2 : v1;
                if(v1 > v2 ) r--;
                else l++;
            }
        }

        if(a > 0){
            int left = 0;
            int right = res.length -1;
            while(left < right){
                int tmp = res[left];
                res[left] = res[right];
                res[right] = tmp;
                left++;
                right--;
            }
        }

        return res;
    }

    private int eval(int n, int a, int b, int c){
        return a*n*n + b*n + c;
    }
}
```

```
public class Solution {
    public int[] sortTransformedArray(int[] nums, int a, int b,
int c) {
        if(nums == null || nums.length <=1) return nums;

        int[] res = new int[nums.length];
        if(a > 0){
            int k = res.length-1;
            int l = 0, r = k;
            while(k >=0){
```

```
                int tl = getT(nums[l],a,b,c);
                int tr = getT(nums[r],a,b,c);

                if(tl > tr){
                    res[k] = tl;
                    l++;
                }else{
                    res[k] = tr;
                    r--;
                }
                k--;
            }
        }else if(a < 0){
            int k = 0;
            int l = 0, r = res.length-1;
            while(k < nums.length){
                int tl = getT(nums[l],a,b,c);
                int tr = getT(nums[r],a,b,c);

                if(tl < tr){
                    res[k] = tl;
                    l++;
                }else{
                    res[k] = tr;
                    r--;
                }
                k++;
            }
        }else{
            for(int i= 0; i< res.length;i++){
                res[i] = getT(nums[i], 0,b,c);
            }
            if(b<0){
                int l =0, r = res.length-1;
                while(l < r){
                    int tmp = res[l];
                    res[l] = res[r];
                    res[r] = tmp;
                    l++;
                    r--;
```

```
                }
            }
        }

        return res;
    }

    int getT(int x, int a, int b, int c){
        return a*x*x + b*x + c;
    }
}
```

362. Design Hit Counter

Design a hit counter which counts the number of hits received in the past 5 minutes.

Each function accepts a timestamp parameter (in seconds granularity) and you may assume that calls are being made to the system in chronological order (ie, the timestamp is monotonically increasing). You may assume that the earliest timestamp starts at 1.

It is possible that several hits arrive roughly at the same time.

Example:

```
HitCounter counter = new HitCounter();

// hit at timestamp 1.
counter.hit(1);

// hit at timestamp 2.
counter.hit(2);

// hit at timestamp 3.
counter.hit(3);

// get hits at timestamp 4, should return 3.
counter.getHits(4);

// hit at timestamp 300.
counter.hit(300);

// get hits at timestamp 300, should return 4.
counter.getHits(300);

// get hits at timestamp 301, should return 3.
counter.getHits(301);
```

Follow up: What if the number of hits per second could be very large? Does your design scale?

use a queue to cache the hit, if the head of queue is of the 5min limit. pop it.

```
public class HitCounter {
    Queue<Integer> window = new LinkedList<>();
    /** Initialize your data structure here. */
    public HitCounter() {

    }

    /** Record a hit.
        @param timestamp - The current timestamp (in seconds gra
nularity). */
    public void hit(int timestamp) {
        window.offer(timestamp);
    }

    /** Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds gra
nularity). */
    public int getHits(int timestamp) {
        while(!window.isEmpty() && timestamp - window.peek() >=
300){
            window.poll();
        }
        return window.size();
    }
}

/**
 * Your HitCounter object will be instantiated and called as suc
h:
 * HitCounter obj = new HitCounter();
 * obj.hit(timestamp);
 * int param_2 = obj.getHits(timestamp);
 */
```

364. Nested List Weight Sum II

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Different from the previous question where weight is increasing from root to leaf, now the weight is defined from bottom up. i.e., the leaf level integers have weight 1, and the root level integers have the largest weight.

Example 1: Given the list [[1,1],2,[1,1]], return 8. (four 1's at depth 1, one 2 at depth 2)

Example 2: Given the list [1,[4,[6]]], return 17. (one 1 at depth 3, one 4 at depth 2, and one 6 at depth 1; 1*3* + *4*2 + 6*1 = 17)

```
/**
 * // This is the interface that allows for creating nested list
s.
 * // You should not implement it, or speculate about its implem
entation
 * public interface NestedInteger {
 *
 *     // @return true if this NestedInteger holds a single inte
ger, rather than a nested list.
 *     public boolean isInteger();
 *
 *     // @return the single integer that this NestedInteger hol
ds, if it holds a single integer
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // @return the nested list that this NestedInteger holds,
 if it holds a nested list
 *     // Return null if this NestedInteger holds a single integ
er
```

```
 *     public List<NestedInteger> getList();
 * }
 */
public class Solution {
    public int depthSumInverse(List<NestedInteger> nestedList) {
        List<Integer> res = new ArrayList<>();
        sum(nestedList, res, 0);
        int total = 0;
        int w = res.size();
        for(Integer i : res){
            total += i * w--;
        }
        return total;
    }

    private void sum(List<NestedInteger> nestedList, List<Intege
r> list, int depth){

        while(depth >= list.size()){
            list.add(0);
        }

        for(NestedInteger ni : nestedList){
            if(ni.isInteger()){
                list.set(depth, list.get(depth) + ni.getInteger(
));
            }else{
                sum(ni.getList(), list, depth+1);
            }
        }
    }
}
```

366 Find Leaves of Binary Tree

Given a binary tree, collect a tree's nodes as if you were doing this: Collect and remove all leaves, repeat until the tree is empty.

Example: Given binary tree

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

Returns [4, 5, 3], [2], [1].

Explanation:

1. Removing the leaves [4, 5, 3] would result in this tree:

   ```
        1
       /
      2
   ```

2. Now removing the leaf [2] would result in this tree:

   ```
        1
   ```

1. Now removing the leaf [1] would result in the empty tree:

```
        []
```

Returns [4, 5, 3], [2], [1].

> DFS

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {

    public List<List<Integer>> findLeaves(TreeNode root) {
        if(root== null) return new ArrayList<List<Integer>>();

        List<List<Integer>> l = findLeaves(root.left);
        List<List<Integer>> r = findLeaves(root.right);

        List<Integer> list = new ArrayList<>();
        list.add(root.val);


        if(l == null && r == null){
            List<List<Integer>> res = new ArrayList<>();
            res.add(list);
            return res;
        }else if(l == null || r == null){
            List<List<Integer>> res = l == null ? r : l;
            res.add(list);
            return res;
        }else{
            List<List<Integer>> lli = l.size() > r.size() ? merg
e(l,r): merge(r,l);
            lli.add(list);
            return lli;
        }

    }

    private List<List<Integer>> merge(List<List<Integer>> large,
```

```
 List<List<Integer>> small){
        for(int i=0; i< small.size(); i++){
            large.get(i).addAll(small.get(i));
        }
        return large;
    }
}
```

> AND the concept of ***Tree Depth***, which the root node has most deepest depth. and left has depth of 0.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<List<Integer>> res = new ArrayList<>();
    public List<List<Integer>> findLeaves(TreeNode root) {

        find(root);
        return res;
    }

    private int find(TreeNode root){
        if(root == null) return -1;

        int depth = Math.max(find(root.left), find(root.right)) +
1;
        if(depth >= res.size()){
            res.add(new ArrayList<>());
        }

        res.get(depth).add(root.val);
        return depth;
    }
}
```

367. Valid Perfect Square

Given a positive integer num, write a function which returns True if num is a perfect square else False.

Note: Do not use any built-in library function such as sqrt.

Example 1:

```
Input: 16
Returns: True
Example 2:

Input: 14
Returns: False
```

Binary search

```
public class Solution {
    public boolean isPerfectSquare(int num) {
        int left = 0;
        int right = num;
        while(left <= right){
            int mid = left + (right-left)/2;
            if(mid == 0 || mid < num/mid){
                left = mid +1;
            }else if(mid == num/mid && num%mid == 0){
                return true;
            }else{
                right = mid-1;
            }
        }

        return false;
    }
}
```

```java
public class Solution {
    public boolean isPerfectSquare(int num) {
        int l = 1;
        int r = num;
        while(l <= r){
            int mid = l + (r-l)/2;
            if( mid < num/mid){
                l = mid +1;
            }else if( mid == num/mid && num%mid == 0){
                return true;
            }else{
                r = mid -1;
            }
        }

        return false;
    }
}
```

369. Plus One Linked List

Given a non-negative number represented as a singly linked list of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

Example: Input: 1->2->3

Output: 1->2->4

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode plusOne(ListNode head) {
        int c = po(head);
        if(c != 0){
            ListNode newHead = new ListNode(c);
            newHead.next = head;
            head = newHead;
        }

        return head;
    }

    int po(ListNode head){
        if(head == null) return 1;
        int c = po(head.next);

        int tmp = (head.val + c)%10;
        c = (head.val+c)/10;
        head.val = tmp;

        return c;
    }
}
```

370. Range Addition

Assume you have an array of length n initialized with all 0's and are given k update operations.

Each operation is represented as a triplet: [startIndex, endIndex, inc] which increments each element of subarray A[startIndex ... endIndex] (startIndex and endIndex inclusive) with inc.

Return the modified array after all k operations were executed.

Example:

Given:

```
length = 5,
updates = [
    [1,  3,  2],
    [2,  4,  3],
    [0,  2, -2]
]
```

Output:

```
[-2, 0, 3, 5, 3]
```

Explanation:

Initial state: [0, 0, 0, 0, 0]

After applying operation [1, 3, 2]: [0, 2, 2, 2, 0]

After applying operation [2, 4, 3]: [0, 2, 5, 5, 3]

After applying operation [0, 2, -2]: [-2, 0, 3, 5, 3] Show Hint

```java
public class Solution {
    public int[] getModifiedArray(int length, int[][] updates) {

        int[] res = new int[length+1];
        for(int i=0; i< updates.length; i++){
            int[] inc = updates[i];
            res[inc[0]] += inc[2];
            res[inc[1] +1] -= inc[2];
        }

        for(int i=1; i<res.length; i++){
            res[i] += res[i-1];
        }

        return Arrays.copyOfRange(res, 0, res.length-1);
    }
}
```

371. Sum of Two Integers

Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -.

Example: Given a = 1 and b = 2, return 3.

```
public class Solution {
    public int getSum(int a, int b) {
        while( b != 0){
            int t = a^b;
            b = (a&b) << 1;
            a = t;
        }

        return a;

    }
}
```

372. Super Pow

Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

```
Example1:

a = 2
b = [3]
```

Result: 8

```
Example2:

a = 2
b = [1,0]
```

Result: 1024

Two points:

- x mod y = (m*n) mod y = (m mod y)* (n mod y)
- x^325 = x^300 *x^20* x^5 = (x^(100))^3 *(x^(10))^2* x^5. at each bit of 325, the base is actually x^10 of previous base.

```
public class Solution {
    int mod = 1337;
    public int superPow(int a, int[] b) {
        if(b == null || b.length == 0) return 1;
        int res = 1;
        for( int i = b.length-1; i >=0;i--){
            res = (res * quickPow(a, b[i])) % mod;
            a = quickPow(a, 10);
        }

        return res;
    }
    private int quickPow(int a, int b){
        int res = 1;
        a = a % mod;

        while(b > 0){
            if((b&1) != 0) res = (res * a) % mod;
            a = (a*a)%mod;
            b >>= 1;
        }
        return res;
    }
}
```

373. Find K Pairs with Smallest Sums

You are given two integer arrays nums1 and nums2 sorted in ascending order and an integer k.

Define a pair (u,v) which consists of one element from the first array and one element from the second array.

Find the k pairs (u1,v1),(u2,v2) ...(uk,vk) with the smallest sums.

```
Example 1:
Given nums1 = [1,7,11], nums2 = [2,4,6],  k = 3
Return: [1,2],[1,4],[1,6]
The first 3 pairs are returned from the sequence:
[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
```

```
Example 2:
Given nums1 = [1,1,2], nums2 = [1,2,3],  k = 2
Return: [1,1],[1,1]
The first 2 pairs are returned from the sequence:
[1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
```

```
Example 3:
Given nums1 = [1,2], nums2 = [3],  k = 3
Return: [1,3],[2,3]
All possible pairs are returned from the sequence:
[1,3],[2,3]
```

Solution samiliar to Meeting Room II, using priority queue to save next result.

```
public class Solution {
    public List<int[]> kSmallestPairs(int[] nums1, int[] nums2,
int k) {
        List<int[]> res = new ArrayList<>();
        if(nums1 == null || nums1.length == 0 ||
```

```
                nums2 == null || nums2.length == 0) return res;

        class Pair{
            int x;
            int y;
            Pair(int x, int y){
                this.x = x;
                this.y =y;
            }
        }

        Comparator<Pair> comp = new Comparator<Pair>(){
            @Override
            public int compare(Pair p1, Pair p2){
                return nums1[p1.x] + nums2[p1.y]
                        - nums1[p2.x] - nums2[p2.y];
            }
        };

        PriorityQueue<Pair> queue = new PriorityQueue<Pair>(k, c
omp);

        boolean[][] visited = new boolean[nums1.length][nums2.le
ngth];

        queue.offer(new Pair(0, 0));
        visited[0][0] = true;

        int[][] close = new int[][]{{0,1},{1,0}};
        while(k > 0 && !queue.isEmpty()){
            k--;
            Pair p = queue.poll();
            res.add(new int[]{nums1[p.x], nums2[p.y]});
            for(int i=0; i< 2;i++){
                int tx = p.x + close[i][0];
                int ty = p.y + close[i][1];
                if(tx < nums1.length && ty <nums2.length && !vis
ited[tx][ty]){
                    queue.offer(new Pair(tx, ty));
                    visited[tx][ty] = true;
```

```
                }

            }
        }

        return res;
    }
}
```

374. Guess Number Higher or Lower

We are playing the Guess Game. The game is as follows:

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API guess(int num) which returns 3 possible results (-1, 1, or 0):

```
-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!
```

Example:

```
n = 10, I pick 6.

Return 6.
```

```
/* The guess API is defined in the parent class GuessGame.
   @param num, your guess
   @return -1 if my number is lower, 1 if my number is higher, o
therwise return 0
      int guess(int num); */

public class Solution extends GuessGame {
    public int guessNumber(int n) {
        int left = 1;
        int right = n;
        while(left < right){
            int mid = left + (right -left)/2;
            int res = guess(mid);
            if(res == 0) return mid;
            else if( res > 0){
                left = mid+1;
            }else{
                right = mid-1;
            }
        }

        return left;
    }
}
```

378. Kth Smallest Element in a Sorted Matrix

Given a n x n matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

Note that it is the kth smallest element in the sorted order, not the kth distinct element.

Example:

```
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,

return 13.
```

```
public class Solution {
    public int kthSmallest(int[][] matrix, int k) {

        class Pair{
            int x;
            int y;
            Pair(int a, int b){
                x = a;
                y = b;
            }
        };

        Comparator<Pair> comp = new Comparator<Pair>(){
            @Override
            public int compare(Pair p1, Pair p2){
                return matrix[p1.x][p1.y] - matrix[p2.x][p2.y];
            }
        };

        PriorityQueue<Pair> pq = new PriorityQueue<>(matrix[0].l
ength,comp);

        for(int i=0; i< matrix[0].length; i++){
            pq.add(new Pair(0, i));
        }
        Pair p = null;
        while(k > 1){
            p = pq.poll();
            if(p.x + 1 < matrix.length)
                pq.add(new Pair(p.x+1, p.y));
            k--;
        }

        p = pq.poll();
        return matrix[p.x][p.y];

    }
}
```

380. Insert Delete GetRandom O(1)

Design a data structure that supports all following operations in average O(1) time.

insert(val): Inserts an item val to the set if not already present. remove(val): Removes an item val from the set if present. getRandom: Returns a random element from current set of elements. Each element must have the same probability of being returned. Example:

```
// Init an empty set.
RandomizedSet randomSet = new RandomizedSet();

// Inserts 1 to the set. Returns true as 1 was inserted successf
ully.
randomSet.insert(1);

// Returns false as 2 does not exist in the set.
randomSet.remove(2);

// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);

// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();

// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);

// 2 was already in the set, so return false.
randomSet.insert(2);

// Since 1 is the only number in the set, getRandom always retur
n 1.
randomSet.getRandom();
```

```
public class RandomizedSet {

    List<Integer> list = new ArrayList<>();
    Map<Integer, Integer> map = new HashMap<>(); // value : inde
x in array;
    /** Initialize your data structure here. */
    public RandomizedSet() {

    }

    /** Inserts a value to the set. Returns true if the set did
not already contain the specified element. */
    public boolean insert(int val) {
        if(map.containsKey(val)) return false;
        list.add(val);
        map.put(val, list.size()-1);
        return true;
    }

    /** Removes a value from the set. Returns true if the set co
ntained the specified element. */
    public boolean remove(int val) {
        if(!map.containsKey(val)) return false;
        int index = map.get(val);
        int last = list.get(list.size()-1);
        list.set(index, last);
        list.remove(list.size()-1);

        map.put(last, index);
        map.remove(val);
        return true;


    }

    /** Get a random element from the set. */
    public int getRandom() {
        java.util.Random r = new java.util.Random();
        return list.get(r.nextInt(list.size()));
```

```
    }
}

/**
 * Your RandomizedSet object will be instantiated and called as
such:
 * RandomizedSet obj = new RandomizedSet();
 * boolean param_1 = obj.insert(val);
 * boolean param_2 = obj.remove(val);
 * int param_3 = obj.getRandom();
 */
```

398. Random Pick Index

Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

Note: The array size can be very large. Solution that uses too much extra space will not pass the judge.

Example:

```
int[] nums = new int[] {1,2,3,3,3};
Solution solution = new Solution(nums);

// pick(3) should return either index 2, 3, or 4 randomly. Each
index should have equal probability of returning.
solution.pick(3);

// pick(1) should return 0. Since in the array only nums[0] is e
qual to 1.
solution.pick(1);
```

`offline algorithm`

```
public class Solution {
    int[] data;
    Random r = new Random();
    public Solution(int[] nums) {
        data = nums;
    }

    public int pick(int target) {
        int count = 0; int res = -1;
        ArrayList<Integer> al = new ArrayList<>();
        for(int i=0; i< data.length; i++){
            if(data[i] != target) continue;
            count++;
            al.add(i);
        }
        int k = r.nextInt(count);
        return al.get(k);
    }
}
```

`online`

```java
public class Solution {
    int[] data;
    Random r = new Random();
    public Solution(int[] nums) {
        data = nums;
    }

    public int pick(int target) {
        int count = 0; int res = -1;
        for(int i=0; i< data.length; i++){
            if(data[i] != target) continue;
            count++;
            if(r.nextInt(count) == 0) res = i;
        }

        return res;
    }
}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(nums);
 * int param_1 = obj.pick(target);
 */
```