*1,Two Sum:***Python\_solution:**Here is a Python solution in O(n) time  
class Solution(object):  
 def twoSum(self, nums, target):  
 if len(nums) <= 1:  
 return False  
 buff\_dict = {}  
 for i in range(len(nums)):  
 if nums[i] in buff\_dict:  
 return [buff\_dict[nums[i]], i]  
 else:  
 buff\_dict[target - nums[i]] = i  
**Best\_solution:**Accepted Java O(n) Solution  
public int[] twoSum(int[] numbers, int target) {  
 int[] result = new int[2];  
 Map<Integer, Integer> map = new HashMap<Integer, Integer>();  
 for (int i = 0; i < numbers.length; i++) {  
 if (map.containsKey(target - numbers[i])) {  
 result[1] = i + 1;  
 result[0] = map.get(target - numbers[i]);  
 return result;  
 }  
 map.put(numbers[i], i + 1);  
 }  
 return result;  
}  
  
*2,Add Two Numbers:***Python\_solution:**Clear python code, straight forward  
class Solution:  
# @return a ListNode  
def addTwoNumbers(self, l1, l2):  
 carry = 0  
 root = n = ListNode(0)  
 while l1 or l2 or carry:  
 v1 = v2 = 0  
 if l1:  
 v1 = l1.val  
 l1 = l1.next  
 if l2:  
 v2 = l2.val  
 l2 = l2.next  
 carry, val = divmod(v1+v2+carry, 10)  
 n.next = ListNode(val)  
 n = n.next  
 return root.next  
**Best\_solution:**Is this Algorithm optimal or what?  
public class Solution {  
 public ListNode addTwoNumbers(ListNode l1, ListNode l2) {  
 ListNode c1 = l1;  
 ListNode c2 = l2;  
 ListNode sentinel = new ListNode(0);  
 ListNode d = sentinel;  
 int sum = 0;  
 while (c1 != null || c2 != null) {  
 sum /= 10;  
 if (c1 != null) {  
 sum += c1.val;  
 c1 = c1.next;  
 }  
 if (c2 != null) {  
 sum += c2.val;  
 c2 = c2.next;  
 }  
 d.next = new ListNode(sum % 10);  
 d = d.next;  
 }  
 if (sum / 10 == 1)  
 d.next = new ListNode(1);  
 return sentinel.next;  
 }  
}  
  
*3,Longest Substring Without Repeating Characters:***Python\_solution:**A Python solution - 85ms - O(n)  
class Solution:  
 # @return an integer  
 def lengthOfLongestSubstring(self, s):  
 start = maxLength = 0  
 usedChar = {}  
   
 for i in range(len(s)):  
 if s[i] in usedChar and start <= usedChar[s[i]]:  
 start = usedChar[s[i]] + 1  
 else:  
 maxLength = max(maxLength, i - start + 1)  
  
 usedChar[s[i]] = i  
  
 return maxLength  
**Best\_solution:**11-line simple Java solution, O(n) with explanation  
 public int lengthOfLongestSubstring(String s) {  
 if (s.length()==0) return 0;  
 HashMap<Character, Integer> map = new HashMap<Character, Integer>();  
 int max=0;  
 for (int i=0, j=0; i<s.length(); ++i){  
 if (map.containsKey(s.charAt(i))){  
 j = Math.max(j,map.get(s.charAt(i))+1);  
 }  
 map.put(s.charAt(i),i);  
 max = Math.max(max,i-j+1);  
 }  
 return max;  
 }  
  
*4,Median of Two Sorted Arrays:***Python\_solution:**Intuitive Python O(log (m+n)) solution, by kth smallest in the two sorted arrays, 252ms  
def findMedianSortedArrays(self, A, B):  
 l = len(A) + len(B)  
 if l % 2 == 1:  
 return self.kth(A, B, l // 2)  
 else:  
 return (self.kth(A, B, l // 2) + self.kth(A, B, l // 2 - 1)) / 2.   
   
def kth(self, a, b, k):  
 if not a:  
 return b[k]  
 if not b:  
 return a[k]  
 ia, ib = len(a) // 2 , len(b) // 2  
 ma, mb = a[ia], b[ib]  
   
 # when k is bigger than the sum of a and b's median indices   
 if ia + ib < k:  
 # if a's median is bigger than b's, b's first half doesn't include k  
 if ma > mb:  
 return self.kth(a, b[ib + 1:], k - ib - 1)  
 else:  
 return self.kth(a[ia + 1:], b, k - ia - 1)  
 # when k is smaller than the sum of a and b's indices  
 else:  
 # if a's median is bigger than b's, a's second half doesn't include k  
 if ma > mb:  
 return self.kth(a[:ia], b, k)  
 else:  
 return self.kth(a, b[:ib], k)  
**Best\_solution:**Share my O(log(min(m,n)) solution with explanation  
dividing a set into two equal length subsets, that one subset is always greater than the other  
  
*5,Longest Palindromic Substring:***Python\_solution:**Python O(n^2) method with some optimization, 88ms.  
class Solution:  
 # @return a string  
 def longestPalindrome(self, s):  
 if len(s)==0:  
 return 0  
 maxLen=1  
 start=0  
 for i in xrange(len(s)):  
 if i-maxLen >=1 and s[i-maxLen-1:i+1]==s[i-maxLen-1:i+1][::-1]:  
 start=i-maxLen-1  
 maxLen+=2  
 continue  
  
 if i-maxLen >=0 and s[i-maxLen:i+1]==s[i-maxLen:i+1][::-1]:  
 start=i-maxLen  
 maxLen+=1  
 return s[start:start+maxLen]  
**Best\_solution:**Very simple clean java solution  
public class Solution {  
private int lo, maxLen;  
  
public String longestPalindrome(String s) {  
 int len = s.length();  
 if (len < 2)  
 return s;  
   
 for (int i = 0; i < len-1; i++) {  
 extendPalindrome(s, i, i); //assume odd length, try to extend Palindrome as possible  
 extendPalindrome(s, i, i+1); //assume even length.  
 }  
 return s.substring(lo, lo + maxLen);  
}  
  
private void extendPalindrome(String s, int j, int k) {  
 while (j >= 0 && k < s.length() && s.charAt(j) == s.charAt(k)) {  
 j--;  
 k++;  
 }  
 if (maxLen < k - j - 1) {  
 lo = j + 1;  
 maxLen = k - j - 1;  
 }  
}}  
  
*6,ZigZag Conversion:***Python\_solution:**Python O(n) Solution in 96ms (99.43%)  
class Solution(object):  
 def convert(self, s, numRows):  
 """  
 :type s: str  
 :type numRows: int  
 :rtype: str  
 """  
 if numRows == 1 or numRows >= len(s):  
 return s  
  
 L = [''] \* numRows  
 index, step = 0, 1  
  
 for x in s:  
 L[index] += x  
 if index == 0:  
 step = 1  
 elif index == numRows -1:  
 step = -1  
 index += step  
  
 return ''.join(L)  
**Best\_solution:**Easy to understand Java solution  
public String convert(String s, int nRows) {  
 char[] c = s.toCharArray();  
 int len = c.length;  
 StringBuffer[] sb = new StringBuffer[nRows];  
 for (int i = 0; i < sb.length; i++) sb[i] = new StringBuffer();  
   
 int i = 0;  
 while (i < len) {  
 for (int idx = 0; idx < nRows && i < len; idx++) // vertically down  
 sb[idx].append(c[i++]);  
 for (int idx = nRows-2; idx >= 1 && i < len; idx--) // obliquely up  
 sb[idx].append(c[i++]);  
 }  
 for (int idx = 1; idx < sb.length; idx++)  
 sb[0].append(sb[idx]);  
 return sb[0].toString();  
}  
  
*7,Reverse Integer:***Python\_solution:**Golfing in Python  
s  
**Best\_solution:**My accepted 15 lines of code for Java  
public int reverse(int x)  
{  
 int result = 0;  
  
 while (x != 0)  
 {  
 int tail = x % 10;  
 int newResult = result \* 10 + tail;  
 if ((newResult - tail) / 10 != result)  
 { return 0; }  
 result = newResult;  
 x = x / 10;  
 }  
  
 return result;  
}  
  
*8,String to Integer (atoi):***Python\_solution:**Python solution based on RegEx  
class Solution:  
 # @return an integer  
 def atoi(self, str):  
 str = str.strip()  
 str = re.findall('(^[\+\-0]\*\d+)\D\*', str)  
  
 try:  
 result = int(''.join(str))  
 MAX\_INT = 2147483647  
 MIN\_INT = -2147483648  
 if result > MAX\_INT > 0:  
 return MAX\_INT  
 elif result < MIN\_INT < 0:  
 return MIN\_INT  
 else:  
 return result  
 except:  
 return 0  
**Best\_solution:**My simple solution  
int atoi(const char \*str) {  
 int sign = 1, base = 0, i = 0;  
 while (str[i] == ' ') { i++; }  
 if (str[i] == '-' || str[i] == '+') {  
 sign = 1 - 2 \* (str[i++] == '-');   
 }  
 while (str[i] >= '0' && str[i] <= '9') {  
 if (base > INT\_MAX / 10 || (base == INT\_MAX / 10 && str[i] - '0' > 7)) {  
 if (sign == 1) return INT\_MAX;  
 else return INT\_MIN;  
 }  
 base = 10 \* base + (str[i++] - '0');  
 }  
 return base \* sign;  
}  
  
*9,Palindrome Number:***Python\_solution:**Python solution based on the algorithm in leetcode blog  
class Solution:  
 # @param x, an integer  
 # @return a boolean  
 def isPalindrome(self, x):  
 if x < 0:  
 return False  
  
 ranger = 1  
 while x / ranger >= 10:  
 ranger \*= 10  
  
 while x:  
 left = x / ranger  
 right = x % 10  
 if left != right:  
 return False  
   
 x = (x % ranger) / 10  
 ranger /= 100  
  
 return True  
**Best\_solution:**9-line accepted Java code, without the need of handling overflow  
public boolean isPalindrome(int x) {  
 if (x<0 || (x!=0 && x%10==0)) return false;  
 int rev = 0;  
 while (x>rev){  
 rev = rev\*10 + x%10;  
 x = x/10;  
 }  
 return (x==rev || x==rev/10);  
}  
  
*10,Regular Expression Matching:***Python\_solution:**My DP approach in Python with comments and unittest  
test\_symbol\_0  
**Best\_solution:**My concise recursive and DP solutions with full explanation in C++  
class Solution {  
public:  
 bool isMatch(string s, string p) {  
 if (p.empty()) return s.empty();  
   
 if ('\*' == p[1])  
 // x\* matches empty string or at least one character: x\* -> xx\*  
 // \*s is to ensure s is non-empty  
 return (isMatch(s, p.substr(2)) || !s.empty() && (s[0] == p[0] || '.' == p[0]) && isMatch(s.substr(1), p));  
 else  
 return !s.empty() && (s[0] == p[0] || '.' == p[0]) && isMatch(s.substr(1), p.substr(1));  
 }  
};  
  
class Solution {  
public:  
 bool isMatch(string s, string p) {  
 /\*\*  
 \* f[i][j]: if s[0..i-1] matches p[0..j-1]  
 \* if p[j - 1] != '\*'  
 \* f[i][j] = f[i - 1][j - 1] && s[i - 1] == p[j - 1]  
 \* if p[j - 1] == '\*', denote p[j - 2] with x  
 \* f[i][j] is true iff any of the following is true  
 \* 1) "x\*" repeats 0 time and matches empty: f[i][j - 2]  
 \* 2) "x\*" repeats >= 1 times and matches "x\*x": s[i - 1] == x && f[i - 1][j]  
 \* '.' matches any single character  
 \*/  
 int m = s.size(), n = p.size();  
 vector<vector<bool>> f(m + 1, vector<bool>(n + 1, false));  
   
 f[0][0] = true;  
 for (int i = 1; i <= m; i++)  
 f[i][0] = false;  
 // p[0.., j - 3, j - 2, j - 1] matches empty iff p[j - 1] is '\*' and p[0..j - 3] matches empty  
 for (int j = 1; j <= n; j++)  
 f[0][j] = j > 1 && '\*' == p[j - 1] && f[0][j - 2];  
   
 for (int i = 1; i <= m; i++)  
 for (int j = 1; j <= n; j++)  
 if (p[j - 1] != '\*')  
 f[i][j] = f[i - 1][j - 1] && (s[i - 1] == p[j - 1] || '.' == p[j - 1]);  
 else  
 // p[0] cannot be '\*' so no need to check "j > 1" here  
 f[i][j] = f[i][j - 2] || (s[i - 1] == p[j - 2] || '.' == p[j - 2]) && f[i - 1][j];  
   
 return f[m][n];  
 }  
};  
  
  
*11,Container With Most Water:***Best\_solution:**My concise recursive and DP solutions with full explanation in C++  
class Solution {  
public:  
 bool isMatch(string s, string p) {  
 if (p.empty()) return s.empty();  
   
 if ('\*' == p[1])  
 // x\* matches empty string or at least one character: x\* -> xx\*  
 // \*s is to ensure s is non-empty  
 return (isMatch(s, p.substr(2)) || !s.empty() && (s[0] == p[0] || '.' == p[0]) && isMatch(s.substr(1), p));  
 else  
 return !s.empty() && (s[0] == p[0] || '.' == p[0]) && isMatch(s.substr(1), p.substr(1));  
 }  
};  
  
class Solution {  
public:  
 bool isMatch(string s, string p) {  
 /\*\*  
 \* f[i][j]: if s[0..i-1] matches p[0..j-1]  
 \* if p[j - 1] != '\*'  
 \* f[i][j] = f[i - 1][j - 1] && s[i - 1] == p[j - 1]  
 \* if p[j - 1] == '\*', denote p[j - 2] with x  
 \* f[i][j] is true iff any of the following is true  
 \* 1) "x\*" repeats 0 time and matches empty: f[i][j - 2]  
 \* 2) "x\*" repeats >= 1 times and matches "x\*x": s[i - 1] == x && f[i - 1][j]  
 \* '.' matches any single character  
 \*/  
 int m = s.size(), n = p.size();  
 vector<vector<bool>> f(m + 1, vector<bool>(n + 1, false));  
   
 f[0][0] = true;  
 for (int i = 1; i <= m; i++)  
 f[i][0] = false;  
 // p[0.., j - 3, j - 2, j - 1] matches empty iff p[j - 1] is '\*' and p[0..j - 3] matches empty  
 for (int j = 1; j <= n; j++)  
 f[0][j] = j > 1 && '\*' == p[j - 1] && f[0][j - 2];  
   
 for (int i = 1; i <= m; i++)  
 for (int j = 1; j <= n; j++)  
 if (p[j - 1] != '\*')  
 f[i][j] = f[i - 1][j - 1] && (s[i - 1] == p[j - 1] || '.' == p[j - 1]);  
 else  
 // p[0] cannot be '\*' so no need to check "j > 1" here  
 f[i][j] = f[i][j - 2] || (s[i - 1] == p[j - 2] || '.' == p[j - 2]) && f[i - 1][j];  
   
 return f[m][n];  
 }  
};  
  
  
*12,Integer to Roman:***Python\_solution:**Share My Python Solution 96ms  
M = ["", "M", "MM", "MMM"];  
C = ["", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"];  
X = ["", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"];  
I = ["", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"];  
return M[num/1000] + C[(num%1000)/100] + X[(num%100)/10] + I[num%10];  
**Best\_solution:**Simple Solution  
public static String intToRoman(int num) {  
 String M[] = {"", "M", "MM", "MMM"};  
 String C[] = {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"};  
 String X[] = {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"};  
 String I[] = {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"};  
 return M[num/1000] + C[(num%1000)/100] + X[(num%100)/10] + I[num%10];  
}  
  
  
*13,Roman to Integer:***Python\_solution:**My Straightforward Python Solution  
class Solution:  
# @param {string} s  
# @return {integer}  
def romanToInt(self, s):  
 roman = {'M': 1000,'D': 500 ,'C': 100,'L': 50,'X': 10,'V': 5,'I': 1}  
 z = 0  
 for i in range(0, len(s) - 1):  
 if roman[s[i]] < roman[s[i+1]]:  
 z -= roman[s[i]]  
 else:  
 z += roman[s[i]]  
 return z + roman[s[-1]]  
  
**Best\_solution:**My solution for this question but I don&#x27;t know is there any easier way?  
public int romanToInt(String s) {  
 int sum=0;  
 if(s.indexOf("IV")!=-1){sum-=2;}  
 if(s.indexOf("IX")!=-1){sum-=2;}  
 if(s.indexOf("XL")!=-1){sum-=20;}  
 if(s.indexOf("XC")!=-1){sum-=20;}  
 if(s.indexOf("CD")!=-1){sum-=200;}  
 if(s.indexOf("CM")!=-1){sum-=200;}  
   
 char c[]=s.toCharArray();  
 int count=0;  
   
 for(;count<=s.length()-1;count++){  
 if(c[count]=='M') sum+=1000;  
 if(c[count]=='D') sum+=500;  
 if(c[count]=='C') sum+=100;  
 if(c[count]=='L') sum+=50;  
 if(c[count]=='X') sum+=10;  
 if(c[count]=='V') sum+=5;  
 if(c[count]=='I') sum+=1;  
   
 }  
   
 return sum;  
   
}  
  
*14,Longest Common Prefix:***Python\_solution:**Simple Python solution  
class Solution:  
 # @return a string  
 def longestCommonPrefix(self, strs):  
 if not strs:  
 return ""  
   
 for i, letter\_group in enumerate(zip(\*strs)):  
 if len(set(letter\_group)) > 1:  
 return strs[0][:i]  
 else:  
 return min(strs)  
**Best\_solution:**Java code with 13 lines  
public String longestCommonPrefix(String[] strs) {  
 if(strs == null || strs.length == 0) return "";  
 String pre = strs[0];  
 int i = 1;  
 while(i < strs.length){  
 while(strs[i].indexOf(pre) != 0)  
 pre = pre.substring(0,pre.length()-1);  
 i++;  
 }  
 return pre;  
}  
  
*15,3Sum:***Python\_solution:**Python easy to understand solution (O(n\*n) time).  
def threeSum(self, nums):  
 res = []  
 nums.sort()  
 for i in xrange(len(nums)-2):  
 if i > 0 and nums[i] == nums[i-1]:  
 continue  
 l, r = i+1, len(nums)-1  
 while l < r:  
 s = nums[i] + nums[l] + nums[r]  
 if s < 0:  
 l +=1   
 elif s > 0:  
 r -= 1  
 else:  
 res.append((nums[i], nums[l], nums[r]))  
 while l < r and nums[l] == nums[l+1]:  
 l += 1  
 while l < r and nums[r] == nums[r-1]:  
 r -= 1  
 l += 1; r -= 1  
 return res  
**Best\_solution:**Concise O(N^2) Java solution  
public List<List<Integer>> threeSum(int[] num) {  
 Arrays.sort(num);  
 List<List<Integer>> res = new LinkedList<>();   
 for (int i = 0; i < num.length-2; i++) {  
 if (i == 0 || (i > 0 && num[i] != num[i-1])) {  
 int lo = i+1, hi = num.length-1, sum = 0 - num[i];  
 while (lo < hi) {  
 if (num[lo] + num[hi] == sum) {  
 res.add(Arrays.asList(num[i], num[lo], num[hi]));  
 while (lo < hi && num[lo] == num[lo+1]) lo++;  
 while (lo < hi && num[hi] == num[hi-1]) hi--;  
 lo++; hi--;  
 } else if (num[lo] + num[hi] < sum) lo++;  
 else hi--;  
 }  
 }  
 }  
 return res;  
}  
  
  
*16,3Sum Closest:***Python\_solution:**Python O(N^2) solution  
class Solution:  
 # @return an integer  
 def threeSumClosest(self, num, target):  
 num.sort()  
 result = num[0] + num[1] + num[2]  
 for i in range(len(num) - 2):  
 j, k = i+1, len(num) - 1  
 while j < k:  
 sum = num[i] + num[j] + num[k]  
 if sum == target:  
 return sum  
   
 if abs(sum - target) < abs(result - target):  
 result = sum  
   
 if sum < target:  
 j += 1  
 elif sum > target:  
 k -= 1  
   
 return result  
**Best\_solution:**A n^2 Solution, Can we do better ?  
int threeSumClosest(vector<int> &num, int target) {   
 vector<int> v(num.begin(), num.end()); // I didn't wanted to disturb original array.  
 int n = 0;  
 int ans = 0;  
 int sum;  
   
 sort(v.begin(), v.end());  
   
 // If less then 3 elements then return their sum  
 while (v.size() <= 3) {  
 return accumulate(v.begin(), v.end(), 0);  
 }  
   
 n = v.size();  
   
 /\* v[0] v[1] v[2] ... v[i] .... v[j] ... v[k] ... v[n-2] v[n-1]  
 \* v[i] <= v[j] <= v[k] always, because we sorted our array.   
 \* Now, for each number, v[i] : we look for pairs v[j] & v[k] such that   
 \* absolute value of (target - (v[i] + v[j] + v[k]) is minimised.  
 \* if the sum of the triplet is greater then the target it implies  
 \* we need to reduce our sum, so we do K = K - 1, that is we reduce  
 \* our sum by taking a smaller number.  
 \* Simillarly if sum of the triplet is less then the target then we  
 \* increase out sum by taking a larger number, i.e. J = J + 1.  
 \*/  
 ans = v[0] + v[1] + v[2];  
 for (int i = 0; i < n-2; i++) {  
 int j = i + 1;  
 int k = n - 1;  
 while (j < k) {  
 sum = v[i] + v[j] + v[k];  
 if (abs(target - ans) > abs(target - sum)) {  
 ans = sum;  
 if (ans == target) return ans;  
 }  
 (sum > target) ? k-- : j++;  
 }  
 }  
 return ans;  
}  
  
  
*17,Letter Combinations of a Phone Number:***Python\_solution:**One line python solution  
class Solution:  
 # @return a list of strings, [s1, s2]  
 def letterCombinations(self, digits):  
 if '' == digits: return []  
 kvmaps = {  
 '2': 'abc',  
 '3': 'def',  
 '4': 'ghi',  
 '5': 'jkl',  
 '6': 'mno',  
 '7': 'pqrs',  
 '8': 'tuv',  
 '9': 'wxyz'  
 }  
 return reduce(lambda acc, digit: [x + y for x in acc for y in kvmaps[digit]], digits, [''])  
**Best\_solution:**My java solution with FIFO queue  
 public List<String> letterCombinations(String digits) {  
 LinkedList<String> ans = new LinkedList<String>();  
 String[] mapping = new String[] {"0", "1", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};  
 ans.add("");  
 for(int i =0; i<digits.length();i++){  
 int x = Character.getNumericValue(digits.charAt(i));  
 while(ans.peek().length()==i){  
 String t = ans.remove();  
 for(char s : mapping[x].toCharArray())  
 ans.add(t+s);  
 }  
 }  
 return ans;  
}  
  
*18,4Sum:***Python\_solution:**Python 140ms beats 100%, and works for N-sum (N>=2)   
def fourSum(self, nums, target):  
 nums.sort()  
 results = []  
 self.findNsum(nums, target, 4, [], results)  
 return results  
  
def findNsum(self, nums, target, N, result, results):  
 if len(nums) < N or N < 2: return  
  
 # solve 2-sum  
 if N == 2:  
 l,r = 0,len(nums)-1  
 while l < r:  
 if nums[l] + nums[r] == target:  
 results.append(result + [nums[l], nums[r]])  
 l += 1  
 r -= 1  
 while l < r and nums[l] == nums[l - 1]:  
 l += 1  
 while r > l and nums[r] == nums[r + 1]:  
 r -= 1  
 elif nums[l] + nums[r] < target:  
 l += 1  
 else:  
 r -= 1  
 else:  
 for i in range(0, len(nums)-N+1): # careful about range  
 if target < nums[i]\*N or target > nums[-1]\*N: # take advantages of sorted list  
 break  
 if i == 0 or i > 0 and nums[i-1] != nums[i]: # recursively reduce N  
 self.findNsum(nums[i+1:], target-nums[i], N-1, result+[nums[i]], results)  
 return  
  
**Best\_solution:**7ms java code win over 100%   
public List<List<Integer>> fourSum(int[] nums, int target) {  
 ArrayList<List<Integer>> res = new ArrayList<List<Integer>>();  
 int len = nums.length;  
 if (nums == null || len < 4)  
 return res;  
  
 Arrays.sort(nums);  
  
 int max = nums[len - 1];  
 if (4 \* nums[0] > target || 4 \* max < target)  
 return res;  
  
 int i, z;  
 for (i = 0; i < len; i++) {  
 z = nums[i];  
 if (i > 0 && z == nums[i - 1])// avoid duplicate  
 continue;  
 if (z + 3 \* max < target) // z is too small  
 continue;  
 if (4 \* z > target) // z is too large  
 break;  
 if (4 \* z == target) { // z is the boundary  
 if (i + 3 < len && nums[i + 3] == z)  
 res.add(Arrays.asList(z, z, z, z));  
 break;  
 }  
  
 threeSumForFourSum(nums, target - z, i + 1, len - 1, res, z);  
 }  
  
 return res;  
 }  
  
 /\*  
 \* Find all possible distinguished three numbers adding up to the target  
 \* in sorted array nums[] between indices low and high. If there are,  
 \* add all of them into the ArrayList fourSumList, using  
 \* fourSumList.add(Arrays.asList(z1, the three numbers))  
 \*/  
 public void threeSumForFourSum(int[] nums, int target, int low, int high, ArrayList<List<Integer>> fourSumList,  
 int z1) {  
 if (low + 1 >= high)  
 return;  
  
 int max = nums[high];  
 if (3 \* nums[low] > target || 3 \* max < target)  
 return;  
  
 int i, z;  
 for (i = low; i < high - 1; i++) {  
 z = nums[i];  
 if (i > low && z == nums[i - 1]) // avoid duplicate  
 continue;  
 if (z + 2 \* max < target) // z is too small  
 continue;  
  
 if (3 \* z > target) // z is too large  
 break;  
  
 if (3 \* z == target) { // z is the boundary  
 if (i + 1 < high && nums[i + 2] == z)  
 fourSumList.add(Arrays.asList(z1, z, z, z));  
 break;  
 }  
  
 twoSumForFourSum(nums, target - z, i + 1, high, fourSumList, z1, z);  
 }  
  
 }  
  
 /\*  
 \* Find all possible distinguished two numbers adding up to the target  
 \* in sorted array nums[] between indices low and high. If there are,  
 \* add all of them into the ArrayList fourSumList, using  
 \* fourSumList.add(Arrays.asList(z1, z2, the two numbers))  
 \*/  
 public void twoSumForFourSum(int[] nums, int target, int low, int high, ArrayList<List<Integer>> fourSumList,  
 int z1, int z2) {  
  
 if (low >= high)  
 return;  
  
 if (2 \* nums[low] > target || 2 \* nums[high] < target)  
 return;  
  
 int i = low, j = high, sum, x;  
 while (i < j) {  
 sum = nums[i] + nums[j];  
 if (sum == target) {  
 fourSumList.add(Arrays.asList(z1, z2, nums[i], nums[j]));  
  
 x = nums[i];  
 while (++i < j && x == nums[i]) // avoid duplicate  
 ;  
 x = nums[j];  
 while (i < --j && x == nums[j]) // avoid duplicate  
 ;  
 }  
 if (sum < target)  
 i++;  
 if (sum > target)  
 j--;  
 }  
 return;  
 }  
  
*19,Remove Nth Node From End of List:***Python\_solution:**3 short Python solutions   
class Solution:  
 def removeNthFromEnd(self, head, n):  
 def index(node):  
 if not node:  
 return 0  
 i = index(node.next) + 1  
 if i > n:  
 node.next.val = node.val  
 return i  
 index(head)  
 return head.next  
  
**Best\_solution:**Simple Java solution in one pass   
public ListNode removeNthFromEnd(ListNode head, int n) {  
   
 ListNode start = new ListNode(0);  
 ListNode slow = start, fast = start;  
 slow.next = head;  
   
 //Move fast in front so that the gap between slow and fast becomes n  
 for(int i=1; i<=n+1; i++) {  
 fast = fast.next;  
 }  
 //Move fast to the end, maintaining the gap  
 while(fast != null) {  
 slow = slow.next;  
 fast = fast.next;  
 }  
 //Skip the desired node  
 slow.next = slow.next.next;  
 return start.next;  
}  
  
*20,Valid Parentheses:***Python\_solution:**Simple Python solution with stack   
class Solution:  
 # @return a boolean  
 def isValid(self, s):  
 stack = []  
 dict = {"]":"[", "}":"{", ")":"("}  
 for char in s:  
 if char in dict.values():  
 stack.append(char)  
 elif char in dict.keys():  
 if stack == [] or dict[char] != stack.pop():  
 return False  
 else:  
 return False  
 return stack == []  
  
**Best\_solution:**Short java solution   
public boolean isValid(String s) {  
 Stack<Character> stack = new Stack<Character>();  
 for (char c : s.toCharArray()) {  
 if (c == '(')  
 stack.push(')');  
 else if (c == '{')  
 stack.push('}');  
 else if (c == '[')  
 stack.push(']');  
 else if (stack.isEmpty() || stack.pop() != c)  
 return false;  
 }  
 return stack.isEmpty();  
}  
  
*21,Merge Two Sorted Lists:***Python\_solution:**Python solutions (iteratively, recursively, iteratively in-place).   
# iteratively  
def mergeTwoLists1(self, l1, l2):  
 dummy = cur = ListNode(0)  
 while l1 and l2:  
 if l1.val < l2.val:  
 cur.next = l1  
 l1 = l1.next  
 else:  
 cur.next = l2  
 l2 = l2.next  
 cur = cur.next  
 cur.next = l1 or l2  
 return dummy.next  
   
# recursively   
def mergeTwoLists2(self, l1, l2):  
 if not l1 or not l2:  
 return l1 or l2  
 if l1.val < l2.val:  
 l1.next = self.mergeTwoLists(l1.next, l2)  
 return l1  
 else:  
 l2.next = self.mergeTwoLists(l1, l2.next)  
 return l2  
   
# in-place, iteratively   
def mergeTwoLists(self, l1, l2):  
 if None in (l1, l2):  
 return l1 or l2  
 dummy = cur = ListNode(0)  
 dummy.next = l1  
 while l1 and l2:  
 if l1.val < l2.val:  
 l1 = l1.next  
 else:  
 nxt = cur.next  
 cur.next = l2  
 tmp = l2.next  
 l2.next = nxt  
 l2 = tmp  
 cur = cur.next  
 cur.next = l1 or l2  
 return dummy.next  
**Best\_solution:**A recursive solution   
class Solution {  
public:  
 ListNode \*mergeTwoLists(ListNode \*l1, ListNode \*l2) {  
 if(l1 == NULL) return l2;  
 if(l2 == NULL) return l1;  
   
 if(l1->val < l2->val) {  
 l1->next = mergeTwoLists(l1->next, l2);  
 return l1;  
 } else {  
 l2->next = mergeTwoLists(l2->next, l1);  
 return l2;  
 }  
 }  
};  
  
  
*22,Generate Parentheses:***Python\_solution:**4-7 lines Python   
p  
**Best\_solution:**Easy to understand Java backtracking solution   
 public List<String> generateParenthesis(int n) {  
 List<String> list = new ArrayList<String>();  
 backtrack(list, "", 0, 0, n);  
 return list;  
 }  
   
 public void backtrack(List<String> list, String str, int open, int close, int max){  
   
 if(str.length() == max\*2){  
 list.add(str);  
 return;  
 }  
   
 if(open < max)  
 backtrack(list, str+"(", open+1, close, max);  
 if(close < open)  
 backtrack(list, str+")", open, close+1, max);  
 }  
  
  
*23,Merge k Sorted Lists:***Python\_solution:**10-line python solution with priority queue   
from Queue import PriorityQueue  
class Solution(object):  
 def mergeKLists(self, lists):  
 dummy = ListNode(None)  
 curr = dummy  
 q = PriorityQueue()  
 for node in lists:  
 if node: q.put((node.val,node))  
 while q.qsize()>0:  
 curr.next = q.get()[1]  
 curr=curr.next  
 if curr.next: q.put((curr.next.val, curr.next))  
 return dummy.next  
**Best\_solution:**A java solution based on Priority Queue   
public class Solution {  
 public ListNode mergeKLists(List<ListNode> lists) {  
 if (lists==null||lists.size()==0) return null;  
   
 PriorityQueue<ListNode> queue= new PriorityQueue<ListNode>(lists.size(),new Comparator<ListNode>(){  
 @Override  
 public int compare(ListNode o1,ListNode o2){  
 if (o1.val<o2.val)  
 return -1;  
 else if (o1.val==o2.val)  
 return 0;  
 else   
 return 1;  
 }  
 });  
   
 ListNode dummy = new ListNode(0);  
 ListNode tail=dummy;  
   
 for (ListNode node:lists)  
 if (node!=null)  
 queue.add(node);  
   
 while (!queue.isEmpty()){  
 tail.next=queue.poll();  
 tail=tail.next;  
   
 if (tail.next!=null)  
 queue.add(tail.next);  
 }  
 return dummy.next;  
 }  
}  
  
*24,Swap Nodes in Pairs:***Python\_solution:**7-8 lines C++ / Python / Ruby   
pp  
**Best\_solution:**My accepted java code. used recursion.   
public class Solution {  
 public ListNode swapPairs(ListNode head) {  
 if ((head == null)||(head.next == null))  
 return head;  
 ListNode n = head.next;  
 head.next = swapPairs(head.next.next);  
 n.next = head;  
 return n;  
 }  
}  
  
*25,Reverse Nodes in k-Group:***Python\_solution:**Succinct iterative Python, O(n) time O(1) space   
def reverseKGroup(self, head, k):  
 dummy = jump = ListNode(0)  
 dummy.next = l = r = head  
   
 while True:  
 count = 0  
 while r and count < k: # use r to locate the range  
 r = r.next  
 count += 1  
 if count == k: # if size k satisfied, reverse the inner linked list  
 pre, cur = r, l  
 for \_ in range(k):  
 cur.next, cur, pre = pre, cur.next, cur # standard reversing  
 jump.next, jump, l = pre, l, r # connect two k-groups  
 else:  
 return dummy.next  
**Best\_solution:**Short but recursive Java code with comments   
public ListNode reverseKGroup(ListNode head, int k) {  
 ListNode curr = head;  
 int count = 0;  
 while (curr != null && count != k) { // find the k+1 node  
 curr = curr.next;  
 count++;  
 }  
 if (count == k) { // if k+1 node is found  
 curr = reverseKGroup(curr, k); // reverse list with k+1 node as head  
 // head - head-pointer to direct part,   
 // curr - head-pointer to reversed part;  
 while (count-- > 0) { // reverse current k-group:   
 ListNode tmp = head.next; // tmp - next head in direct part  
 head.next = curr; // preappending "direct" head to the reversed list   
 curr = head; // move head of reversed part to a new node  
 head = tmp; // move "direct" head to the next node in direct part  
 }  
 head = curr;  
 }  
 return head;  
}  
  
  
*26,Remove Duplicates from Sorted Array:***Python\_solution:**Simple Python solution - O(n)   
class Solution:  
 # @param a list of integers  
 # @return an integer  
 def removeDuplicates(self, A):  
 if not A:  
 return 0  
  
 newTail = 0  
  
 for i in range(1, len(A)):  
 if A[i] != A[newTail]:  
 newTail += 1  
 A[newTail] = A[i]  
  
 return newTail + 1  
**Best\_solution:**My Solution : Time O(n), Space O(1)   
class Solution {  
 public:  
 int removeDuplicates(int A[], int n) {  
 if(n < 2) return n;  
 int id = 1;  
 for(int i = 1; i < n; ++i)   
 if(A[i] != A[i-1]) A[id++] = A[i];  
 return id;  
 }  
};  
  
*27,Remove Element:***Python\_solution:**Simple Python O(n) two pointer in place solution   
 def removeElement(self, nums, val):  
 start, end = 0, len(nums) - 1  
 while start <= end:  
 if nums[start] == val:  
 nums[start], nums[end], end = nums[end], nums[start], end - 1  
 else:  
 start +=1  
 return start  
**Best\_solution:**My solution for your reference.   
int removeElement(int A[], int n, int elem) {  
 int begin=0;  
 for(int i=0;i<n;i++) if(A[i]!=elem) A[begin++]=A[i];  
 return begin;  
}  
  
*28,Implement strStr():***Python\_solution:**My answer by Python   
class Solution(object):  
def strStr(self, haystack, needle):  
 """  
 :type haystack: str  
 :type needle: str  
 :rtype: int  
 """  
 for i in range(len(haystack) - len(needle)+1):  
 if haystack[i:i+len(needle)] == needle:  
 return i  
 return -1  
**Best\_solution:**Elegant Java solution   
public int strStr(String haystack, String needle) {  
 for (int i = 0; ; i++) {  
 for (int j = 0; ; j++) {  
 if (j == needle.length()) return i;  
 if (i + j == haystack.length()) return -1;  
 if (needle.charAt(j) != haystack.charAt(i + j)) break;  
 }  
 }  
}  
  
*29,Divide Two Integers:***Python\_solution:**Clear python code   
class Solution:  
# @return an integer  
def divide(self, dividend, divisor):  
 positive = (dividend < 0) is (divisor < 0)  
 dividend, divisor = abs(dividend), abs(divisor)  
 res = 0  
 while dividend >= divisor:  
 temp, i = divisor, 1  
 while dividend >= temp:  
 dividend -= temp  
 res += i  
 i <<= 1  
 temp <<= 1  
 if not positive:  
 res = -res  
 return min(max(-2147483648, res), 2147483647)  
**Best\_solution:**Detailed Explained 8ms C++ solution   
15  
  
*30,Substring with Concatenation of All Words:***Python\_solution:**AC Python 80ms solution, dictionary and two pointers   
def \_findSubstring(self, l, r, n, k, t, s, req, ans):  
 curr = {}  
 while r + k <= n:  
 w = s[r:r + k]  
 r += k  
 if w not in req:  
 l = r  
 curr.clear()  
 else:  
 curr[w] = curr[w] + 1 if w in curr else 1  
 while curr[w] > req[w]:  
 curr[s[l:l + k]] -= 1  
 l += k  
 if r - l == t:  
 ans.append(l)  
  
def findSubstring(self, s, words):  
 if not s or not words or not words[0]:  
 return []  
 n = len(s)  
 k = len(words[0])  
 t = len(words) \* k  
 req = {}  
 for w in words:  
 req[w] = req[w] + 1 if w in req else 1  
 ans = []  
 for i in xrange(min(k, n - t + 1)):  
 self.\_findSubstring(i, i, n, k, t, s, req, ans)  
 return ans  
  
  
# 169 / 169 test cases passed.  
# Status: Accepted  
# Runtime: 80 ms  
# 98.60%  
  
**Best\_solution:**An O(N) solution with detailed explanation   
 // travel all the words combinations to maintain a window  
 // there are wl(word len) times travel  
 // each time, n/wl words, mostly 2 times travel for each word  
 // one left side of the window, the other right side of the window  
 // so, time complexity O(wl \* 2 \* N/wl) = O(2N)  
 vector<int> findSubstring(string S, vector<string> &L) {  
 vector<int> ans;  
 int n = S.size(), cnt = L.size();  
 if (n <= 0 || cnt <= 0) return ans;  
   
 // init word occurence  
 unordered\_map<string, int> dict;  
 for (int i = 0; i < cnt; ++i) dict[L[i]]++;  
   
 // travel all sub string combinations  
 int wl = L[0].size();  
 for (int i = 0; i < wl; ++i) {  
 int left = i, count = 0;  
 unordered\_map<string, int> tdict;  
 for (int j = i; j <= n - wl; j += wl) {  
 string str = S.substr(j, wl);  
 // a valid word, accumulate results  
 if (dict.count(str)) {  
 tdict[str]++;  
 if (tdict[str] <= dict[str])   
 count++;  
 else {  
 // a more word, advance the window left side possiablly  
 while (tdict[str] > dict[str]) {  
 string str1 = S.substr(left, wl);  
 tdict[str1]--;  
 if (tdict[str1] < dict[str1]) count--;  
 left += wl;  
 }  
 }  
 // come to a result  
 if (count == cnt) {  
 ans.push\_back(left);  
 // advance one word  
 tdict[S.substr(left, wl)]--;  
 count--;  
 left += wl;  
 }  
 }  
 // not a valid word, reset all vars  
 else {  
 tdict.clear();  
 count = 0;  
 left = j + wl;  
 }  
 }  
 }  
   
 return ans;  
 }  
  
*31,Next Permutation:***Python\_solution:**Easy python solution based on lexicographical permutation algorithm   
class Solution(object):  
 def nextPermutation(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: void Do not return anything, modify nums in-place instead.  
 """  
 # find longest non-increasing suffix  
 right = len(nums)-1  
 while nums[right] <= nums[right-1] and right-1 >=0:  
 right -= 1  
 if right == 0:  
 return self.reverse(nums,0,len(nums)-1)  
 # find pivot  
 pivot = right-1  
 successor = 0  
 # find rightmost succesor  
 for i in range(len(nums)-1,pivot,-1):  
 if nums[i] > nums[pivot]:  
 successor = i  
 break  
 # swap pivot and successor  
 nums[pivot],nums[successor] = nums[successor],nums[pivot]   
 # reverse suffix  
 self.reverse(nums,pivot+1,len(nums)-1)  
   
 def reverse(self,nums,l,r):  
 while l < r:  
 nums[l],nums[r] = nums[r],nums[l]  
 l += 1  
 r -= 1  
  
**Best\_solution:**Share my O(n) time solution   
public void nextPermutation(int[] num) {  
 int n=num.length;  
 if(n<2)  
 return;  
 int index=n-1;   
 while(index>0){  
 if(num[index-1]<num[index])  
 break;  
 index--;  
 }  
 if(index==0){  
 reverseSort(num,0,n-1);  
 return;  
 }  
 else{  
 int val=num[index-1];  
 int j=n-1;  
 while(j>=index){  
 if(num[j]>val)  
 break;  
 j--;  
 }  
 swap(num,j,index-1);  
 reverseSort(num,index,n-1);  
 return;  
 }  
}  
  
public void swap(int[] num, int i, int j){  
 int temp=0;  
 temp=num[i];  
 num[i]=num[j];  
 num[j]=temp;  
}  
  
public void reverseSort(int[] num, int start, int end){   
 if(start>end)  
 return;  
 for(int i=start;i<=(end+start)/2;i++)  
 swap(num,i,start+end-i);  
}  
  
*32,Longest Valid Parentheses:***Python\_solution:**Pure 1D-DP without using stack (python) with detailed explanation   
class Solution(object):  
 def longestValidParentheses(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 # use 1D DP  
 # dp[i] records the longestValidParenthese EXACTLY ENDING at s[i]  
 dp = [0 for x in xrange(len(s))]  
 max\_to\_now = 0  
 for i in xrange(1,len(s)):  
 if s[i] == ')':  
 # case 1: ()()  
 if s[i-1] == '(':  
 # add nearest parentheses pairs + 2  
 dp[i] = dp[i-2] + 2  
 # case 2: (())   
 # i-dp[i-1]-1 is the index of last "(" not paired until this ")"  
 elif i-dp[i-1]-1 >= 0 and s[i-dp[i-1]-1] == '(':  
 if dp[i-1] > 0: # content within current matching pair is valid   
 # add nearest parentheses pairs + 2 + parentheses before last "("  
 dp[i] = dp[i-1] + 2 + dp[i-dp[i-1]-2]  
 else:  
 # otherwise is 0  
 dp[i] = 0  
 max\_to\_now = max(max\_to\_now, dp[i])  
 return max\_to\_now  
**Best\_solution:**My O(n) solution using a stack   
class Solution {  
public:  
 int longestValidParentheses(string s) {  
 int n = s.length(), longest = 0;  
 stack<int> st;  
 for (int i = 0; i < n; i++) {  
 if (s[i] == '(') st.push(i);  
 else {  
 if (!st.empty()) {  
 if (s[st.top()] == '(') st.pop();  
 else st.push(i);  
 }  
 else st.push(i);  
 }  
 }  
 if (st.empty()) longest = n;  
 else {  
 int a = n, b = 0;  
 while (!st.empty()) {  
 b = st.top(); st.pop();  
 longest = max(longest, a-b-1);  
 a = b;  
 }  
 longest = max(longest, a);  
 }  
 return longest;  
 }  
};  
  
  
*33,Search in Rotated Sorted Array:***Python\_solution:**Pretty short C++/Java/Ruby/Python   
def search(nums, target)  
 i = (0...nums.size).bsearch { |i|  
 (nums[0] <= target) ^ (nums[0] > nums[i]) ^ (target > nums[i])  
 }  
 nums[i || 0] == target ? i : -1  
end  
  
**Best\_solution:**Concise O(log N) Binary search solution   
class Solution {  
public:  
 int search(int A[], int n, int target) {  
 int lo=0,hi=n-1;  
 // find the index of the smallest value using binary search.  
 // Loop will terminate since mid < hi, and lo or hi will shrink by at least 1.  
 // Proof by contradiction that mid < hi: if mid==hi, then lo==hi and loop would have been terminated.  
 while(lo<hi){  
 int mid=(lo+hi)/2;  
 if(A[mid]>A[hi]) lo=mid+1;  
 else hi=mid;  
 }  
 // lo==hi is the index of the smallest value and also the number of places rotated.  
 int rot=lo;  
 lo=0;hi=n-1;  
 // The usual binary search and accounting for rotation.  
 while(lo<=hi){  
 int mid=(lo+hi)/2;  
 int realmid=(mid+rot)%n;  
 if(A[realmid]==target)return realmid;  
 if(A[realmid]<target)lo=mid+1;  
 else hi=mid-1;  
 }  
 return -1;  
 }  
};  
  
*34,Search for a Range:***Python\_solution:**Search for the position target-0.5 and target+0.5, a simple python code with a little trick   
class Solution:  
# @param A, a list of integers  
# @param target, an integer to be searched  
# @return a list of length 2, [index1, index2]  
def searchRange(self, arr, target):  
 start = self.binary\_search(arr, target-0.5)  
 if arr[start] != target:  
 return [-1, -1]  
 arr.append(0)  
 end = self.binary\_search(arr, target+0.5)-1  
 return [start, end]  
  
def binary\_search(self, arr, target):  
 start, end = 0, len(arr)-1  
 while start < end:  
 mid = (start+end)//2  
 if target < arr[mid]:  
 end = mid  
 else:  
 start = mid+1  
 return start  
  
**Best\_solution:**Clean iterative solution with two binary searches (with explanation)   
case 1: [5 7] (A[i] = target < A[j])  
case 2: [5 3] (A[i] = target > A[j])  
case 3: [5 5] (A[i] = target = A[j])  
case 4: [3 5] (A[j] = target > A[i])  
case 5: [3 7] (A[i] < target < A[j])  
case 6: [3 4] (A[i] < A[j] < target)  
case 7: [6 7] (target < A[i] < A[j])  
  
  
*35,Search Insert Position:***Python\_solution:**Python beats 98%   
class Solution(object):  
def searchInsert(self, nums, key):  
 if key > nums[len(nums) - 1]:  
 return len(nums)  
  
 if key < nums[0]:  
 return 0  
  
 l, r = 0, len(nums) - 1  
 while l <= r:  
 m = (l + r)/2  
 if nums[m] > key:  
 r = m - 1  
 if r >= 0:  
 if nums[r] < key:  
 return r + 1  
 else:  
 return 0  
  
 elif nums[m] < key:  
 l = m + 1  
 if l < len(nums):  
 if nums[l] > key:  
 return l  
 else:  
 return len(nums)  
 else:  
 return m  
  
**Best\_solution:**My 8 line Java solution   
 public int searchInsert(int[] A, int target) {  
 int low = 0, high = A.length-1;  
 while(low<=high){  
 int mid = (low+high)/2;  
 if(A[mid] == target) return mid;  
 else if(A[mid] > target) high = mid-1;  
 else low = mid+1;  
 }  
 return low;  
 }  
  
*36,Valid Sudoku:***Python\_solution:**1-7 lines Python, 4 solutions   
Counter  
**Best\_solution:**My short solution by C++. O(n2)   
class Solution  
{  
public:  
 bool isValidSudoku(vector<vector<char> > &board)  
 {  
 int used1[9][9] = {0}, used2[9][9] = {0}, used3[9][9] = {0};  
   
 for(int i = 0; i < board.size(); ++ i)  
 for(int j = 0; j < board[i].size(); ++ j)  
 if(board[i][j] != '.')  
 {  
 int num = board[i][j] - '0' - 1, k = i / 3 \* 3 + j / 3;  
 if(used1[i][num] || used2[j][num] || used3[k][num])  
 return false;  
 used1[i][num] = used2[j][num] = used3[k][num] = 1;  
 }  
   
 return true;  
 }  
};  
  
*37,Sudoku Solver:***Best\_solution:**Straight Forward Java Solution Using Backtracking   
public class Solution {  
 public void solveSudoku(char[][] board) {  
 if(board == null || board.length == 0)  
 return;  
 solve(board);  
 }  
   
 public boolean solve(char[][] board){  
 for(int i = 0; i < board.length; i++){  
 for(int j = 0; j < board[0].length; j++){  
 if(board[i][j] == '.'){  
 for(char c = '1'; c <= '9'; c++){//trial. Try 1 through 9  
 if(isValid(board, i, j, c)){  
 board[i][j] = c; //Put c for this cell  
   
 if(solve(board))  
 return true; //If it's the solution return true  
 else  
 board[i][j] = '.'; //Otherwise go back  
 }  
 }  
   
 return false;  
 }  
 }  
 }  
 return true;  
 }  
   
 private boolean isValid(char[][] board, int row, int col, char c){  
 for(int i = 0; i < 9; i++) {  
 if(board[i][col] != '.' && board[i][col] == c) return false; //check row  
 if(board[row][i] != '.' && board[row][i] == c) return false; //check column  
 if(board[3 \* (row / 3) + i / 3][ 3 \* (col / 3) + i % 3] != '.' &&   
board[3 \* (row / 3) + i / 3][3 \* (col / 3) + i % 3] == c) return false; //check 3\*3 block  
 }  
 return true;  
 }  
}  
  
  
*38,Count and Say:***Python\_solution:**4-5 lines Python solutions   
def countAndSay(self, n):  
 s = '1'  
 for \_ in range(n - 1):  
 s = re.sub(r'(.)\1\*', lambda m: str(len(m.group(0))) + m.group(1), s)  
 return s  
  
**Best\_solution:**Please change the misleading description   
None  
  
*39,Combination Sum:***Python\_solution:**Python dfs solution.   
def combinationSum(self, candidates, target):  
 res = []  
 candidates.sort()  
 self.dfs(candidates, target, 0, [], res)  
 return res  
   
def dfs(self, nums, target, index, path, res):  
 if target < 0:  
 return # backtracking  
 if target == 0:  
 res.append(path)  
 return   
 for i in xrange(index, len(nums)):  
 self.dfs(nums, target-nums[i], i, path+[nums[i]], res)  
**Best\_solution:**A general approach to backtracking questions in Java (Subsets, Permutations, Combination Sum, Palindrome Partitioning)   
public List<List<Integer>> subsets(int[] nums) {  
 List<List<Integer>> list = new ArrayList<>();  
 Arrays.sort(nums);  
 backtrack(list, new ArrayList<>(), nums, 0);  
 return list;  
}  
  
private void backtrack(List<List<Integer>> list , List<Integer> tempList, int [] nums, int start){  
 list.add(new ArrayList<>(tempList));  
 for(int i = start; i < nums.length; i++){  
 tempList.add(nums[i]);  
 backtrack(list, tempList, nums, i + 1);  
 tempList.remove(tempList.size() - 1);  
 }  
}  
  
  
*40,Combination Sum II:***Python\_solution:**DP solution in Python   
def combinationSum2(self, candidates, target):  
 candidates.sort()  
 table = [None] + [set() for i in range(target)]  
 for i in candidates:  
 if i > target:  
 break  
 for j in range(target - i, 0, -1):  
 table[i + j] |= {elt + (i,) for elt in table[j]}  
 table[i].add((i,))  
 return map(list, table[target])  
**Best\_solution:**Java solution using dfs, easy understand   
 public List<List<Integer>> combinationSum2(int[] cand, int target) {  
 Arrays.sort(cand);  
 List<List<Integer>> res = new ArrayList<List<Integer>>();  
 List<Integer> path = new ArrayList<Integer>();  
 dfs\_com(cand, 0, target, path, res);  
 return res;  
}  
void dfs\_com(int[] cand, int cur, int target, List<Integer> path, List<List<Integer>> res) {  
 if (target == 0) {  
 res.add(new ArrayList(path));  
 return ;  
 }  
 if (target < 0) return;  
 for (int i = cur; i < cand.length; i++){  
 if (i > cur && cand[i] == cand[i-1]) continue;  
 path.add(path.size(), cand[i]);  
 dfs\_com(cand, i+1, target - cand[i], path, res);  
 path.remove(path.size()-1);  
 }  
}  
  
*41,First Missing Positive:***Python\_solution:**Python O(1) space, O(n) time solution with explanation   
 def firstMissingPositive(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 Basic idea:  
 1. for any array whose length is l, the first missing positive must be in range [1,...,l+1],   
 so we only have to care about those elements in this range and remove the rest.  
 2. we can use the array index as the hash to restore the frequency of each number within   
 the range [1,...,l+1]   
 """  
 nums.append(0)  
 n = len(nums)  
 for i in range(len(nums)): #delete those useless elements  
 if nums[i]<0 or nums[i]>=n:  
 nums[i]=0  
 for i in range(len(nums)): #use the index as the hash to record the frequency of each number  
 nums[nums[i]%n]+=n  
 for i in range(1,len(nums)):  
 if nums[i]/n==0:  
 return i  
 return n  
**Best\_solution:**My short c++ solution, O(1) space, and O(n) time   
class Solution  
{  
public:  
 int firstMissingPositive(int A[], int n)  
 {  
 for(int i = 0; i < n; ++ i)  
 while(A[i] > 0 && A[i] <= n && A[A[i] - 1] != A[i])  
 swap(A[i], A[A[i] - 1]);  
   
 for(int i = 0; i < n; ++ i)  
 if(A[i] != i + 1)  
 return i + 1;  
   
 return n + 1;  
 }  
};  
  
*42,Trapping Rain Water:***Python\_solution:**8-lines C/C++/Java/Python Solution   
l  
**Best\_solution:**Sharing my simple c++ code: O(n) time, O(1) space   
class Solution {  
public:  
 int trap(int A[], int n) {  
 int left=0; int right=n-1;  
 int res=0;  
 int maxleft=0, maxright=0;  
 while(left<=right){  
 if(A[left]<=A[right]){  
 if(A[left]>=maxleft) maxleft=A[left];  
 else res+=maxleft-A[left];  
 left++;  
 }  
 else{  
 if(A[right]>=maxright) maxright= A[right];  
 else res+=maxright-A[right];  
 right--;  
 }  
 }  
 return res;  
 }  
};  
  
*43,Multiply Strings:***Python\_solution:**Simple Python solution, 18 lines   
def multiply(num1, num2):  
 product = [0] \* (len(num1) + len(num2))  
 pos = len(product)-1  
   
 for n1 in reversed(num1):  
 tempPos = pos  
 for n2 in reversed(num2):  
 product[tempPos] += int(n1) \* int(n2)  
 product[tempPos-1] += product[tempPos]/10  
 product[tempPos] %= 10  
 tempPos -= 1  
 pos -= 1  
   
 pt = 0  
 while pt < len(product)-1 and product[pt] == 0:  
 pt += 1  
  
 return ''.join(map(str, product[pt:]))  
**Best\_solution:**Easiest JAVA Solution with Graph Explanation   
 `num1[i] \* num2[j]` will be placed at indices `[i + j`, `i + j + 1]`   
  
  
*44,Wildcard Matching:***Python\_solution:**Python DP solution   
class Solution:  
# @return a boolean  
def isMatch(self, s, p):  
 length = len(s)  
 if len(p) - p.count('\*') > length:  
 return False  
 dp = [True] + [False]\*length  
 for i in p:  
 if i != '\*':  
 for n in reversed(range(length)):  
 dp[n+1] = dp[n] and (i == s[n] or i == '?')  
 else:  
 for n in range(1, length+1):  
 dp[n] = dp[n-1] or dp[n]  
 dp[0] = dp[0] and i == '\*'  
 return dp[-1]  
  
**Best\_solution:**Linear runtime and constant space solution   
 bool isMatch(const char \*s, const char \*p) {  
 const char\* star=NULL;  
 const char\* ss=s;  
 while (\*s){  
 //advancing both pointers when (both characters match) or ('?' found in pattern)  
 //note that \*p will not advance beyond its length   
 if ((\*p=='?')||(\*p==\*s)){s++;p++;continue;}   
  
 // \* found in pattern, track index of \*, only advancing pattern pointer   
 if (\*p=='\*'){star=p++; ss=s;continue;}   
  
 //current characters didn't match, last pattern pointer was \*, current pattern pointer is not \*  
 //only advancing pattern pointer  
 if (star){ p = star+1; s=++ss;continue;}   
  
 //current pattern pointer is not star, last patter pointer was not \*  
 //characters do not match  
 return false;  
 }  
  
 //check for remaining characters in pattern  
 while (\*p=='\*'){p++;}  
  
 return !\*p;   
 }  
  
  
*45,Jump Game II:***Python\_solution:**10-lines C++ (16ms) / Python BFS Solutions with Explanations   
nums = [2, 3, 1, 1, 4]  
**Best\_solution:**O(n), BFS solution   
 int jump(int A[], int n) {  
 if(n<2)return 0;  
 int level=0,currentMax=0,i=0,nextMax=0;  
  
 while(currentMax-i+1>0){ //nodes count of current level>0  
 level++;  
 for(;i<=currentMax;i++){ //traverse current level , and update the max reach of next level  
 nextMax=max(nextMax,A[i]+i);  
 if(nextMax>=n-1)return level; // if last element is in level+1, then the min jump=level   
 }  
 currentMax=nextMax;  
 }  
 return 0;  
 }  
  
*46,Permutations:***Python\_solution:**My AC simple iterative java/python solution   
List<List<Integer>>  
**Best\_solution:**A general approach to backtracking questions in Java (Subsets, Permutations, Combination Sum, Palindrome Partioning)   
public List<List<Integer>> subsets(int[] nums) {  
 List<List<Integer>> list = new ArrayList<>();  
 Arrays.sort(nums);  
 backtrack(list, new ArrayList<>(), nums, 0);  
 return list;  
}  
  
private void backtrack(List<List<Integer>> list , List<Integer> tempList, int [] nums, int start){  
 list.add(new ArrayList<>(tempList));  
 for(int i = start; i < nums.length; i++){  
 tempList.add(nums[i]);  
 backtrack(list, tempList, nums, i + 1);  
 tempList.remove(tempList.size() - 1);  
 }  
}  
  
  
*47,Permutations II:***Python\_solution:**9-line python solution with 1 line to handle duplication, beat 99% of others :-)   
def permuteUnique(self, nums):  
 ans = [[]]  
 for n in nums:  
 new\_ans = []  
 for l in ans:  
 for i in xrange(len(l)+1):  
 new\_ans.append(l[:i]+[n]+l[i:])  
 if i<len(l) and l[i]==n: break #handles duplication  
 ans = new\_ans  
 return ans  
**Best\_solution:**A simple C++ solution in only 20 lines   
class Solution {  
public:  
 void recursion(vector<int> num, int i, int j, vector<vector<int> > &res) {  
 if (i == j-1) {  
 res.push\_back(num);  
 return;  
 }  
 for (int k = i; k < j; k++) {  
 if (i != k && num[i] == num[k]) continue;  
 swap(num[i], num[k]);  
 recursion(num, i+1, j, res);  
 }  
 }  
 vector<vector<int> > permuteUnique(vector<int> &num) {  
 sort(num.begin(), num.end());  
 vector<vector<int> >res;  
 recursion(num, 0, num.size(), res);  
 return res;  
 }  
};  
  
*48,Rotate Image:***Best\_solution:**A common method to rotate the image   
/\*  
 \* clockwise rotate  
 \* first reverse up to down, then swap the symmetry   
 \* 1 2 3 7 8 9 7 4 1  
 \* 4 5 6 => 4 5 6 => 8 5 2  
 \* 7 8 9 1 2 3 9 6 3  
\*/  
void rotate(vector<vector<int> > &matrix) {  
 reverse(matrix.begin(), matrix.end());  
 for (int i = 0; i < matrix.size(); ++i) {  
 for (int j = i + 1; j < matrix[i].size(); ++j)  
 swap(matrix[i][j], matrix[j][i]);  
 }  
}  
  
/\*  
 \* anticlockwise rotate  
 \* first reverse left to right, then swap the symmetry  
 \* 1 2 3 3 2 1 3 6 9  
 \* 4 5 6 => 6 5 4 => 2 5 8  
 \* 7 8 9 9 8 7 1 4 7  
\*/  
void anti\_rotate(vector<vector<int> > &matrix) {  
 for (auto vi : matrix) reverse(vi.begin(), vi.end());  
 for (int i = 0; i < matrix.size(); ++i) {  
 for (int j = i + 1; j < matrix[i].size(); ++j)  
 swap(matrix[i][j], matrix[j][i]);  
 }  
}  
  
*49,Group Anagrams:***Python\_solution:**2-line Python solution, AC with 350ms (some useful Python tricks)   
 def anagrams(self, strs):  
 count = collections.Counter([tuple(sorted(s)) for s in strs])  
 return filter(lambda x: count[tuple(sorted(x))]>1, strs)  
  
**Best\_solution:**Share my short JAVA solution   
public class Solution {  
 public List<List<String>> groupAnagrams(String[] strs) {  
 if (strs == null || strs.length == 0) return new ArrayList<List<String>>();  
 Map<String, List<String>> map = new HashMap<String, List<String>>();  
 for (String s : strs) {  
 char[] ca = s.toCharArray();  
 Arrays.sort(ca);  
 String keyStr = String.valueOf(ca);  
 if (!map.containsKey(keyStr)) map.put(keyStr, new ArrayList<String>());  
 map.get(keyStr).add(s);  
 }  
 return new ArrayList<List<String>>(map.values());  
 }  
}  
  
*50,Pow(x, n):***Python\_solution:**Shortest Python - Guaranteed   
pow  
**Best\_solution:**Short and easy to understand solution   
public class Solution {  
 public double pow(double x, int n) {  
 if(n == 0)  
 return 1;  
 if(n<0){  
 n = -n;  
 x = 1/x;  
 }  
 return (n%2 == 0) ? pow(x\*x, n/2) : x\*pow(x\*x, n/2);  
 }  
}  
  
*51,N-Queens:***Python\_solution:**Fast, short, and easy-to-understand python solution, 11 lines, 76ms   
DFS  
**Best\_solution:**Accepted 4ms c++ solution use backtracking and bitmask, easy understand.   
column  
  
*52,N-Queens II:***Python\_solution:**11-line Python solution, easy to understand   
def totalNQueens(self, n):  
 def dfs(board, row):  
 if row == n: return 1  
 count = 0  
 for x in set\_n - set(board):  
 # check diagonal conflict  
 if all(row - i != abs(x - y) for i, y in enumerate(board[:row])):  
 board[row] = x  
 count += dfs(board, row + 1)  
 board[row] = '.'  
 return count  
  
 set\_n = {i for i in xrange(n)}  
 return dfs(['.'] \* n, 0)  
**Best\_solution:**Accepted Java Solution   
/\*\*  
 \* don't need to actually place the queen,  
 \* instead, for each row, try to place without violation on  
 \* col/ diagonal1/ diagnol2.  
 \* trick: to detect whether 2 positions sit on the same diagnol:  
 \* if delta(col, row) equals, same diagnol1;  
 \* if sum(col, row) equals, same diagnal2.  
 \*/  
private final Set<Integer> occupiedCols = new HashSet<Integer>();  
private final Set<Integer> occupiedDiag1s = new HashSet<Integer>();  
private final Set<Integer> occupiedDiag2s = new HashSet<Integer>();  
public int totalNQueens(int n) {  
 return totalNQueensHelper(0, 0, n);  
}  
  
private int totalNQueensHelper(int row, int count, int n) {  
 for (int col = 0; col < n; col++) {  
 if (occupiedCols.contains(col))  
 continue;  
 int diag1 = row - col;  
 if (occupiedDiag1s.contains(diag1))  
 continue;  
 int diag2 = row + col;  
 if (occupiedDiag2s.contains(diag2))  
 continue;  
 // we can now place a queen here  
 if (row == n-1)  
 count++;  
 else {  
 occupiedCols.add(col);  
 occupiedDiag1s.add(diag1);  
 occupiedDiag2s.add(diag2);  
 count = totalNQueensHelper(row+1, count, n);  
 // recover  
 occupiedCols.remove(col);  
 occupiedDiag1s.remove(diag1);  
 occupiedDiag2s.remove(diag2);  
 }  
 }  
   
 return count;  
}  
  
*53,Maximum Subarray:***Python\_solution:**A Python solution   
class Solution:  
 # @param A, a list of integers  
 # @return an integer  
 # 6:57  
 def maxSubArray(self, A):  
 if not A:  
 return 0  
  
 curSum = maxSum = A[0]  
 for num in A[1:]:  
 curSum = max(num, curSum + num)  
 maxSum = max(maxSum, curSum)  
  
 return maxSum  
**Best\_solution:**DP solution & some thoughts   
maxSubArray(int A[], int i, int j)  
  
*54,Spiral Matrix:***Python\_solution:**1-liner in Python   
def spiralOrder(self, matrix):  
 return matrix and list(matrix.pop(0)) + self.spiralOrder(zip(\*matrix)[::-1])  
**Best\_solution:**Super Simple and Easy to Understand Solution   
public class Solution {  
 public List<Integer> spiralOrder(int[][] matrix) {  
   
 List<Integer> res = new ArrayList<Integer>();  
   
 if (matrix.length == 0) {  
 return res;  
 }  
   
 int rowBegin = 0;  
 int rowEnd = matrix.length-1;  
 int colBegin = 0;  
 int colEnd = matrix[0].length - 1;  
   
 while (rowBegin <= rowEnd && colBegin <= colEnd) {  
 // Traverse Right  
 for (int j = colBegin; j <= colEnd; j ++) {  
 res.add(matrix[rowBegin][j]);  
 }  
 rowBegin++;  
   
 // Traverse Down  
 for (int j = rowBegin; j <= rowEnd; j ++) {  
 res.add(matrix[j][colEnd]);  
 }  
 colEnd--;  
   
 if (rowBegin <= rowEnd) {  
 // Traverse Left  
 for (int j = colEnd; j >= colBegin; j --) {  
 res.add(matrix[rowEnd][j]);  
 }  
 }  
 rowEnd--;  
   
 if (colBegin <= colEnd) {  
 // Traver Up  
 for (int j = rowEnd; j >= rowBegin; j --) {  
 res.add(matrix[j][colBegin]);  
 }  
 }  
 colBegin ++;  
 }  
   
 return res;  
 }  
}  
  
*55,Jump Game:***Best\_solution:**Linear and simple solution in C++   
bool canJump(int A[], int n) {  
 int i = 0;  
 for (int reach = 0; i < n && i <= reach; ++i)  
 reach = max(i + A[i], reach);  
 return i == n;  
}  
  
*56,Merge Intervals:***Python\_solution:**7 lines, easy, Python   
def merge(self, intervals):  
 out = []  
 for i in sorted(intervals, key=lambda i: i.start):  
 if out and i.start <= out[-1].end:  
 out[-1].end = max(out[-1].end, i.end)  
 else:  
 out += i,  
 return out  
**Best\_solution:**A simple Java solution   
public List<Interval> merge(List<Interval> intervals) {  
 if (intervals.size() <= 1)  
 return intervals;  
   
 // Sort by ascending starting point using an anonymous Comparator  
 intervals.sort((i1, i2) -> Integer.compare(i1.start, i2.start));  
   
 List<Interval> result = new LinkedList<Interval>();  
 int start = intervals.get(0).start;  
 int end = intervals.get(0).end;  
   
 for (Interval interval : intervals) {  
 if (interval.start <= end) // Overlapping intervals, move the end if needed  
 end = Math.max(end, interval.end);  
 else { // Disjoint intervals, add the previous one and reset bounds  
 result.add(new Interval(start, end));  
 start = interval.start;  
 end = interval.end;  
 }  
 }  
   
 // Add the last interval  
 result.add(new Interval(start, end));  
 return result;  
}  
  
  
*57,Insert Interval:***Python\_solution:**O(n) Python solution   
class Solution:  
 # @param intervals, a list of Intervals  
 # @param newInterval, a Interval  
 # @return a list of Interval  
 def insert(self, intervals, newInterval):  
 start = newInterval.start  
 end = newInterval.end  
 result = []  
 i = 0  
 while i < len(intervals):  
 if start <= intervals[i].end:  
 if end < intervals[i].start:  
 break  
 start = min(start, intervals[i].start)  
 end = max(end, intervals[i].end)  
 else:  
 result.append(intervals[i])  
 i += 1  
 result.append(Interval(start, end))  
 result += intervals[i:]  
 return result  
**Best\_solution:**Short and straight-forward Java solution   
public List<Interval> insert(List<Interval> intervals, Interval newInterval) {  
 List<Interval> result = new LinkedList<>();  
 int i = 0;  
 // add all the intervals ending before newInterval starts  
 while (i < intervals.size() && intervals.get(i).end < newInterval.start)  
 result.add(intervals.get(i++));  
 // merge all overlapping intervals to one considering newInterval  
 while (i < intervals.size() && intervals.get(i).start <= newInterval.end) {  
 newInterval = new Interval( // we could mutate newInterval here also  
 Math.min(newInterval.start, intervals.get(i).start),  
 Math.max(newInterval.end, intervals.get(i).end));  
 i++;  
 }  
 result.add(newInterval); // add the union of intervals we got  
 // add all the rest  
 while (i < intervals.size()) result.add(intervals.get(i++));   
 return result;  
}  
  
  
*58,Length of Last Word:***Python\_solution:**One line Python solution   
def lengthOfLastWord(self, s):  
 return len(s.rstrip(' ').split(' ')[-1])  
  
**Best\_solution:**7-lines 4ms C++ Solution   
s  
  
*59,Spiral Matrix II:***Python\_solution:**4-9 lines Python solutions   
 || => |9| => |8| |6 7| |4 5| |1 2 3|  
 |9| => |9 8| => |9 6| => |8 9 4|  
 |8 7| |7 6 5|  
  
**Best\_solution:**4-9 lines Python solutions   
 || => |9| => |8| |6 7| |4 5| |1 2 3|  
 |9| => |9 8| => |9 6| => |8 9 4|  
 |8 7| |7 6 5|  
  
  
*60,Permutation Sequence:***Python\_solution:**Share my Python solution with detailed explanation   
import math  
class Solution:  
 # @param {integer} n  
 # @param {integer} k  
 # @return {string}  
 def getPermutation(self, n, k):  
 numbers = range(1, n+1)  
 permutation = ''  
 k -= 1  
 while n > 0:  
 n -= 1  
 # get the index of current digit  
 index, k = divmod(k, math.factorial(n))  
 permutation += str(numbers[index])  
 # remove handled number  
 numbers.remove(numbers[index])  
  
 return permutation  
**Best\_solution:**"Explain-like-I'm-five" Java Solution in O(n)   
public class Solution {  
public String getPermutation(int n, int k) {  
 int pos = 0;  
 List<Integer> numbers = new ArrayList<>();  
 int[] factorial = new int[n+1];  
 StringBuilder sb = new StringBuilder();  
   
 // create an array of factorial lookup  
 int sum = 1;  
 factorial[0] = 1;  
 for(int i=1; i<=n; i++){  
 sum \*= i;  
 factorial[i] = sum;  
 }  
 // factorial[] = {1, 1, 2, 6, 24, ... n!}  
   
 // create a list of numbers to get indices  
 for(int i=1; i<=n; i++){  
 numbers.add(i);  
 }  
 // numbers = {1, 2, 3, 4}  
   
 k--;  
   
 for(int i = 1; i <= n; i++){  
 int index = k/factorial[n-i];  
 sb.append(String.valueOf(numbers.get(index)));  
 numbers.remove(index);  
 k-=index\*factorial[n-i];  
 }  
   
 return String.valueOf(sb);  
}  
  
  
*61,Rotate List:***Python\_solution:**97.63% Python Solution   
class Solution(object):  
def rotateRight(self, head, k):  
 """  
 :type head: ListNode  
 :type k: int  
 :rtype: ListNode  
 """  
 if not head:  
 return None  
   
 if head.next == None:  
 return head  
   
 pointer = head  
 length = 1  
   
 while pointer.next:  
 pointer = pointer.next  
 length += 1  
   
 rotateTimes = k%length  
   
 if k == 0 or rotateTimes == 0:  
 return head  
   
 fastPointer = head  
 slowPointer = head  
   
 for a in range (rotateTimes):  
 fastPointer = fastPointer.next  
   
   
 while fastPointer.next:  
 slowPointer = slowPointer.next  
 fastPointer = fastPointer.next  
   
 temp = slowPointer.next  
   
 slowPointer.next = None  
 fastPointer.next = head  
 head = temp  
   
 return head  
**Best\_solution:**My clean C++ code, quite standard (find tail and reconnect the list)   
class Solution {  
public:  
 ListNode\* rotateRight(ListNode\* head, int k) {  
 if(!head) return head;  
   
 int len=1; // number of nodes  
 ListNode \*newH, \*tail;  
 newH=tail=head;  
   
 while(tail->next) // get the number of nodes in the list  
 {  
 tail = tail->next;  
 len++;  
 }  
 tail->next = head; // circle the link  
  
 if(k %= len)   
 {  
 for(auto i=0; i<len-k; i++) tail = tail->next; // the tail node is the (len-k)-th node (1st node is head)  
 }  
 newH = tail->next;   
 tail->next = NULL;  
 return newH;  
 }  
};  
  
*62,Unique Paths:***Python\_solution:**1 Line Math Solution (Python)   
class Solution(object):  
 def uniquePaths(self, m, n):  
 """  
 :type m: int  
 :type n: int  
 :rtype: int  
 """  
 return math.factorial(m+n-2)/math.factorial(m-1)/math.factorial(n-1)  
**Best\_solution:**0ms, 5-lines DP Solution in C++ with Explanations   
(i, j)  
  
*63,Unique Paths II:***Python\_solution:**Accepted simple Python in-place solution   
class Solution:  
 # @param obstacleGrid, a list of lists of integers  
 # @return an integer  
 def uniquePathsWithObstacles(self, obstacleGrid):  
 m = len(obstacleGrid)  
 n = len(obstacleGrid[0])  
 obstacleGrid[0][0] = 1 - obstacleGrid[0][0]  
   
 for i in range(1, n):  
 if not obstacleGrid[0][i]:  
 obstacleGrid[0][i] = obstacleGrid[0][i-1]  
 else:  
 obstacleGrid[0][i] = 0  
   
 for i in range(1, m):  
 if not obstacleGrid[i][0]:  
 obstacleGrid[i][0] = obstacleGrid[i-1][0]  
 else:  
 obstacleGrid[i][0] = 0  
   
 for i in range(1, m):  
 for j in range(1, n):  
 if not obstacleGrid[i][j]:  
 obstacleGrid[i][j] = obstacleGrid[i][j-1]+obstacleGrid[i-1][j]  
 else:  
 obstacleGrid[i][j] = 0  
   
 return obstacleGrid[-1][-1]  
**Best\_solution:**Short JAVA solution   
public int uniquePathsWithObstacles(int[][] obstacleGrid) {  
 int width = obstacleGrid[0].length;  
 int[] dp = new int[width];  
 dp[0] = 1;  
 for (int[] row : obstacleGrid) {  
 for (int j = 0; j < width; j++) {  
 if (row[j] == 1)  
 dp[j] = 0;  
 else if (j > 0)  
 dp[j] += dp[j - 1];  
 }  
 }  
 return dp[width - 1];  
}  
  
*64,Minimum Path Sum:***Python\_solution:**Simple python dp 70ms   
def minPathSum(self, grid):  
 m = len(grid)  
 n = len(grid[0])  
 for i in range(1, n):  
 grid[0][i] += grid[0][i-1]  
 for i in range(1, m):  
 grid[i][0] += grid[i-1][0]  
 for i in range(1, m):  
 for j in range(1, n):  
 grid[i][j] += min(grid[i-1][j], grid[i][j-1])  
 return grid[-1][-1]  
**Best\_solution:**10-lines 28ms O(n)-space DP solution in C++ with Explanations   
(i, j)  
  
*65,Valid Number:***Python\_solution:**A simple solution in Python based on DFA   
class Solution(object):  
 def isNumber(self, s):  
 """  
 :type s: str  
 :rtype: bool  
 """  
 #define a DFA  
 state = [{},   
 {'blank': 1, 'sign': 2, 'digit':3, '.':4},   
 {'digit':3, '.':4},  
 {'digit':3, '.':5, 'e':6, 'blank':9},  
 {'digit':5},  
 {'digit':5, 'e':6, 'blank':9},  
 {'sign':7, 'digit':8},  
 {'digit':8},  
 {'digit':8, 'blank':9},  
 {'blank':9}]  
 currentState = 1  
 for c in s:  
 if c >= '0' and c <= '9':  
 c = 'digit'  
 if c == ' ':  
 c = 'blank'  
 if c in ['+', '-']:  
 c = 'sign'  
 if c not in state[currentState].keys():  
 return False  
 currentState = state[currentState][c]  
 if currentState not in [3,5,8,9]:  
 return False  
 return True  
  
**Best\_solution:**The worst problem i have ever met in this oj   
None  
  
*66,Plus One:***Python\_solution:**Simple Python solution with explanation (Plus One)   
def plusOne(digits):  
 num = 0  
 for i in range(len(digits)):  
 num += digits[i] \* pow(10, (len(digits)-1-i))  
 return [int(i) for i in str(num+1)]  
  
**Best\_solution:**My Simple Java Solution   
public int[] plusOne(int[] digits) {  
   
 int n = digits.length;  
 for(int i=n-1; i>=0; i--) {  
 if(digits[i] < 9) {  
 digits[i]++;  
 return digits;  
 }  
   
 digits[i] = 0;  
 }  
   
 int[] newNumber = new int [n+1];  
 newNumber[0] = 1;  
   
 return newNumber;  
}  
  
*67,Add Binary:***Python\_solution:**An accepted concise Python recursive solution 10 lines   
#add two binary from back to front, I think it is very self explained, when 1+1 we need a carry.  
 class Solution:  
 def addBinary(self, a, b):  
 if len(a)==0: return b  
 if len(b)==0: return a  
 if a[-1] == '1' and b[-1] == '1':  
 return self.addBinary(self.addBinary(a[0:-1],b[0:-1]),'1')+'0'  
 if a[-1] == '0' and b[-1] == '0':  
 return self.addBinary(a[0:-1],b[0:-1])+'0'  
 else:  
 return self.addBinary(a[0:-1],b[0:-1])+'1'  
**Best\_solution:**Short code by c++   
class Solution  
{  
public:  
 string addBinary(string a, string b)  
 {  
 string s = "";  
   
 int c = 0, i = a.size() - 1, j = b.size() - 1;  
 while(i >= 0 || j >= 0 || c == 1)  
 {  
 c += i >= 0 ? a[i --] - '0' : 0;  
 c += j >= 0 ? b[j --] - '0' : 0;  
 s = char(c % 2 + '0') + s;  
 c /= 2;  
 }  
   
 return s;  
 }  
};  
  
*68,Text Justification:***Python\_solution:**Concise python solution, 10 lines.   
def fullJustify(self, words, maxWidth):  
 res, cur, num\_of\_letters = [], [], 0  
 for w in words:  
 if num\_of\_letters + len(w) + len(cur) > maxWidth:  
 for i in range(maxWidth - num\_of\_letters):  
 cur[i%(len(cur)-1 or 1)] += ' '  
 res.append(''.join(cur))  
 cur, num\_of\_letters = [], 0  
 cur += [w]  
 num\_of\_letters += len(w)  
 return res + [' '.join(cur).ljust(maxWidth)]  
  
**Best\_solution:**Share my concise c++ solution - less than 20 lines   
vector<string> fullJustify(vector<string> &words, int L) {  
 vector<string> res;  
 for(int i = 0, k, l; i < words.size(); i += k) {  
 for(k = l = 0; i + k < words.size() and l + words[i+k].size() <= L - k; k++) {  
 l += words[i+k].size();  
 }  
 string tmp = words[i];  
 for(int j = 0; j < k - 1; j++) {  
 if(i + k >= words.size()) tmp += " ";  
 else tmp += string((L - l) / (k - 1) + (j < (L - l) % (k - 1)), ' ');  
 tmp += words[i+j+1];  
 }  
 tmp += string(L - tmp.size(), ' ');  
 res.push\_back(tmp);  
 }  
 return res;  
}  
  
  
*69,Sqrt(x):***Python\_solution:**Python binary search solution (O(lgn)).   
# Binary search   
def mySqrt(self, x):  
 l, r = 0, x  
 while l <= r:  
 mid = l + (r-l)//2  
 if mid \* mid <= x < (mid+1)\*(mid+1):  
 return mid  
 elif x < mid \* mid:  
 r = mid  
 else:  
 l = mid + 1  
**Best\_solution:**A Binary Search Solution   
public int sqrt(int x) {  
 if (x == 0)  
 return 0;  
 int left = 1, right = Integer.MAX\_VALUE;  
 while (true) {  
 int mid = left + (right - left)/2;  
 if (mid > x/mid) {  
 right = mid - 1;  
 } else {  
 if (mid + 1 > x/(mid + 1))  
 return mid;  
 left = mid + 1;  
 }  
 }  
}  
  
*70,Climbing Stairs:***Python\_solution:**Python different solutions (bottom up, top down).   
# Top down - TLE  
def climbStairs1(self, n):  
 if n == 1:  
 return 1  
 if n == 2:  
 return 2  
 return self.climbStairs(n-1)+self.climbStairs(n-2)  
   
# Bottom up, O(n) space  
def climbStairs2(self, n):  
 if n == 1:  
 return 1  
 res = [0 for i in xrange(n)]  
 res[0], res[1] = 1, 2  
 for i in xrange(2, n):  
 res[i] = res[i-1] + res[i-2]  
 return res[-1]  
  
# Bottom up, constant space  
def climbStairs3(self, n):  
 if n == 1:  
 return 1  
 a, b = 1, 2  
 for i in xrange(2, n):  
 tmp = b  
 b = a+b  
 a = tmp  
 return b  
   
# Top down + memorization (list)  
def climbStairs4(self, n):  
 if n == 1:  
 return 1  
 dic = [-1 for i in xrange(n)]  
 dic[0], dic[1] = 1, 2  
 return self.helper(n-1, dic)  
   
def helper(self, n, dic):  
 if dic[n] < 0:  
 dic[n] = self.helper(n-1, dic)+self.helper(n-2, dic)  
 return dic[n]  
   
# Top down + memorization (dictionary)   
def \_\_init\_\_(self):  
 self.dic = {1:1, 2:2}  
   
def climbStairs(self, n):  
 if n not in self.dic:  
 self.dic[n] = self.climbStairs(n-1) + self.climbStairs(n-2)  
 return self.dic[n]  
**Best\_solution:**Basically it's a fibonacci.   
[n-1]  
  
*71,Simplify Path:***Python\_solution:**9 lines of Python code   
class Solution(object):  
 def simplifyPath(self, path):  
 places = [p for p in path.split("/") if p!="." and p!=""]  
 stack = []  
 for p in places:  
 if p == "..":  
 if len(stack) > 0:  
 stack.pop()  
 else:  
 stack.append(p)  
 return "/" + "/".join(stack)  
**Best\_solution:**C++ 10-lines solution   
string simplifyPath(string path) {  
 string res, tmp;  
 vector<string> stk;  
 stringstream ss(path);  
 while(getline(ss,tmp,'/')) {  
 if (tmp == "" or tmp == ".") continue;  
 if (tmp == ".." and !stk.empty()) stk.pop\_back();  
 else if (tmp != "..") stk.push\_back(tmp);  
 }  
 for(auto str : stk) res += "/"+str;  
 return res.empty() ? "/" : res;  
}  
  
*72,Edit Distance:***Python\_solution:**Python solutions (O(m\*n), O(n) space).   
# O(m\*n) space  
def minDistance1(self, word1, word2):  
 l1, l2 = len(word1)+1, len(word2)+1  
 dp = [[0 for \_ in xrange(l2)] for \_ in xrange(l1)]  
 for i in xrange(l1):  
 dp[i][0] = i  
 for j in xrange(l2):  
 dp[0][j] = j  
 for i in xrange(1, l1):  
 for j in xrange(1, l2):  
 dp[i][j] = min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1]+(word1[i-1]!=word2[j-1]))  
 return dp[-1][-1]  
   
  
# O(n) space with rolling array   
def minDistance(self, word1, word2):  
 l1, l2 = len(word1)+1, len(word2)+1  
 pre = [0 for \_ in xrange(l2)]  
 for j in xrange(l2):  
 pre[j] = j  
 for i in xrange(1, l1):  
 cur = [i]\*l2  
 for j in xrange(1, l2):  
 cur[j] = min(cur[j-1]+1, pre[j]+1, pre[j-1]+(word1[i-1]!=word2[j-1]))  
 pre = cur[:]  
 return pre[-1]  
**Best\_solution:**20ms Detailed Explained C++ Solutions (O(n) Space)   
dp[i][j]  
  
*73,Set Matrix Zeroes:***Python\_solution:**O(1) space solution in Python   
class Solution:  
# @param {integer[][]} matrix  
# @return {void} Do not return anything, modify matrix in-place instead.  
def setZeroes(self, matrix):  
 m = len(matrix)  
 if m == 0:  
 return  
 n = len(matrix[0])  
   
 row\_zero = False  
 for i in range(m):  
 if matrix[i][0] == 0:  
 row\_zero = True  
 col\_zero = False  
 for j in range(n):  
 if matrix[0][j] == 0:  
 col\_zero = True  
   
 for i in range(1, m):  
 for j in range(1, n):  
 if matrix[i][j] == 0:  
 matrix[i][0] = 0  
 matrix[0][j] = 0  
   
 for i in range(1, m):  
 if matrix[i][0] == 0:  
 for j in range(1, n):  
 matrix[i][j] = 0  
   
 for j in range(1, n):  
 if matrix[0][j] == 0:  
 for i in range(1, m):  
 matrix[i][j] = 0  
   
 if col\_zero:  
 for j in range(n):  
 matrix[0][j] = 0  
 if row\_zero:  
 for i in range(m):  
 matrix[i][0] = 0  
**Best\_solution:**Any shorter O(1) space solution?   
void setZeroes(vector<vector<int> > &matrix) {  
 int col0 = 1, rows = matrix.size(), cols = matrix[0].size();  
  
 for (int i = 0; i < rows; i++) {  
 if (matrix[i][0] == 0) col0 = 0;  
 for (int j = 1; j < cols; j++)  
 if (matrix[i][j] == 0)  
 matrix[i][0] = matrix[0][j] = 0;  
 }  
  
 for (int i = rows - 1; i >= 0; i--) {  
 for (int j = cols - 1; j >= 1; j--)  
 if (matrix[i][0] == 0 || matrix[0][j] == 0)  
 matrix[i][j] = 0;  
 if (col0 == 0) matrix[i][0] = 0;  
 }  
}  
  
*74,Search a 2D Matrix:***Python\_solution:**A Python binary search solution - O(logn)   
class Solution:  
 # @param matrix, a list of lists of integers  
 # @param target, an integer  
 # @return a boolean  
 # 8:21  
 def searchMatrix(self, matrix, target):  
 if not matrix or target is None:  
 return False  
  
 rows, cols = len(matrix), len(matrix[0])  
 low, high = 0, rows \* cols - 1  
   
 while low <= high:  
 mid = (low + high) / 2  
 num = matrix[mid / cols][mid % cols]  
  
 if num == target:  
 return True  
 elif num < target:  
 low = mid + 1  
 else:  
 high = mid - 1  
   
 return False  
**Best\_solution:**Don't treat it as a 2D matrix, just treat it as a sorted list   
class Solution {  
public:  
 bool searchMatrix(vector<vector<int> > &matrix, int target) {  
 int n = matrix.size();  
 int m = matrix[0].size();  
 int l = 0, r = m \* n - 1;  
 while (l != r){  
 int mid = (l + r - 1) >> 1;  
 if (matrix[mid / m][mid % m] < target)  
 l = mid + 1;  
 else   
 r = mid;  
 }  
 return matrix[r / m][r % m] == target;  
 }  
};  
  
*75,Sort Colors:***Python\_solution:**AC Python in place one pass solution O(n) time O(1) space, no swap no count   
def sortColors(self, nums):  
 i = j = 0  
 for k in xrange(len(nums)):  
 v = nums[k]  
 nums[k] = 2  
 if v < 2:  
 nums[j] = 1  
 j += 1  
 if v == 0:  
 nums[i] = 0  
 i += 1  
  
# 86 / 86 test cases passed.  
# Status: Accepted  
# Runtime: 44 ms  
# 84.03%  
  
**Best\_solution:**Share my at most two-pass constant space 10-line solution   
 class Solution {  
 public:  
 void sortColors(int A[], int n) {  
 int second=n-1, zero=0;  
 for (int i=0; i<=second; i++) {  
 while (A[i]==2 && i<second) swap(A[i], A[second--]);  
 while (A[i]==0 && i>zero) swap(A[i], A[zero++]);  
 }  
 }  
 };  
  
*76,Minimum Window Substring:***Python\_solution:**12 lines Python   
s[i:j]  
**Best\_solution:**Here is a 10-line template that can solve most 'substring' problems   
string minWindow(string s, string t) {  
 vector<int> map(128,0);  
 for(auto c: t) map[c]++;  
 int counter=t.size(), begin=0, end=0, d=INT\_MAX, head=0;  
 while(end<s.size()){  
 if(map[s[end++]]-->0) counter--; //in t  
 while(counter==0){ //valid  
 if(end-begin<d) d=end-(head=begin);  
 if(map[s[begin++]]++==0) counter++; //make it invalid  
 }   
 }  
 return d==INT\_MAX? "":s.substr(head, d);  
 }  
  
  
*77,Combinations:***Python\_solution:**AC Python backtracking iterative solution 60 ms   
def combine(self, n, k):  
 ans = []  
 stack = []  
 x = 1  
 while True:  
 l = len(stack)  
 if l == k:  
 ans.append(stack[:])  
 if l == k or x > n - k + l + 1:  
 if not stack:  
 return ans  
 x = stack.pop() + 1  
 else:  
 stack.append(x)  
 x += 1  
  
# 26 / 26 test cases passed.  
# Status: Accepted  
# Runtime: 60 ms  
# 98.51%  
  
**Best\_solution:**Backtracking Solution Java   
 public static List<List<Integer>> combine(int n, int k) {  
 List<List<Integer>> combs = new ArrayList<List<Integer>>();  
 combine(combs, new ArrayList<Integer>(), 1, n, k);  
 return combs;  
 }  
 public static void combine(List<List<Integer>> combs, List<Integer> comb, int start, int n, int k) {  
 if(k==0) {  
 combs.add(new ArrayList<Integer>(comb));  
 return;  
 }  
 for(int i=start;i<=n;i++) {  
 comb.add(i);  
 combine(combs, comb, i+1, n, k-1);  
 comb.remove(comb.size()-1);  
 }  
 }  
  
*78,Subsets:***Python\_solution:**Python easy to understand solutions (DFS recursively, Bit Manipulation, Iteratively).   
# DFS recursively   
def subsets1(self, nums):  
 res = []  
 self.dfs(sorted(nums), 0, [], res)  
 return res  
   
def dfs(self, nums, index, path, res):  
 res.append(path)  
 for i in xrange(index, len(nums)):  
 self.dfs(nums, i+1, path+[nums[i]], res)  
   
# Bit Manipulation   
def subsets2(self, nums):  
 res = []  
 nums.sort()  
 for i in xrange(1<<len(nums)):  
 tmp = []  
 for j in xrange(len(nums)):  
 if i & 1 << j: # if i >> j & 1:  
 tmp.append(nums[j])  
 res.append(tmp)  
 return res  
   
# Iteratively  
def subsets(self, nums):  
 res = [[]]  
 for num in sorted(nums):  
 res += [item+[num] for item in res]  
 return res  
**Best\_solution:**A general approach to backtracking questions in Java (Subsets, Permutations, Combination Sum, Palindrome Partitioning)   
public List<List<Integer>> subsets(int[] nums) {  
 List<List<Integer>> list = new ArrayList<>();  
 Arrays.sort(nums);  
 backtrack(list, new ArrayList<>(), nums, 0);  
 return list;  
}  
  
private void backtrack(List<List<Integer>> list , List<Integer> tempList, int [] nums, int start){  
 list.add(new ArrayList<>(tempList));  
 for(int i = start; i < nums.length; i++){  
 tempList.add(nums[i]);  
 backtrack(list, tempList, nums, i + 1);  
 tempList.remove(tempList.size() - 1);  
 }  
}  
  
  
*79,Word Search:***Python\_solution:**Python dfs solution with comments.   
def exist(self, board, word):  
 if not board:  
 return False  
 for i in xrange(len(board)):  
 for j in xrange(len(board[0])):  
 if self.dfs(board, i, j, word):  
 return True  
 return False  
  
# check whether can find word, start at (i,j) position   
def dfs(self, board, i, j, word):  
 if len(word) == 0: # all the characters are checked  
 return True  
 if i<0 or i>=len(board) or j<0 or j>=len(board[0]) or word[0]!=board[i][j]:  
 return False  
 tmp = board[i][j] # first character is found, check the remaining part  
 board[i][j] = "#" # avoid visit agian   
 # check whether can find "word" along one direction  
 res = self.dfs(board, i+1, j, word[1:]) or self.dfs(board, i-1, j, word[1:]) \  
 or self.dfs(board, i, j+1, word[1:]) or self.dfs(board, i, j-1, word[1:])  
 board[i][j] = tmp  
 return res  
**Best\_solution:**Accepted very short Java solution. No additional space.   
public boolean exist(char[][] board, String word) {  
 char[] w = word.toCharArray();  
 for (int y=0; y<board.length; y++) {  
 for (int x=0; x<board[y].length; x++) {  
 if (exist(board, y, x, w, 0)) return true;  
 }  
 }  
 return false;  
}  
  
private boolean exist(char[][] board, int y, int x, char[] word, int i) {  
 if (i == word.length) return true;  
 if (y<0 || x<0 || y == board.length || x == board[y].length) return false;  
 if (board[y][x] != word[i]) return false;  
 board[y][x] ^= 256;  
 boolean exist = exist(board, y, x+1, word, i+1)  
 || exist(board, y, x-1, word, i+1)  
 || exist(board, y+1, x, word, i+1)  
 || exist(board, y-1, x, word, i+1);  
 board[y][x] ^= 256;  
 return exist;  
}  
  
*80,Remove Duplicates from Sorted Array II:***Python\_solution:**3-6 easy lines, C++, Java, Python, Ruby   
int removeDuplicates(vector<int>& nums) {  
 int i = 0;  
 for (int n : nums)  
 if (i < 2 || n > nums[i-2])  
 nums[i++] = n;  
 return i;  
}  
  
**Best\_solution:**3-6 easy lines, C++, Java, Python, Ruby   
int removeDuplicates(vector<int>& nums) {  
 int i = 0;  
 for (int n : nums)  
 if (i < 2 || n > nums[i-2])  
 nums[i++] = n;  
 return i;  
}  
  
  
*81,Search in Rotated Sorted Array II:***Python\_solution:**Python easy to understand solution (with comments).   
def search(self, nums, target):  
 l, r = 0, len(nums)-1  
 while l <= r:  
 mid = l + (r-l)//2  
 if nums[mid] == target:  
 return True  
 while l < mid and nums[l] == nums[mid]: # tricky part  
 l += 1  
 # the first half is ordered  
 if nums[l] <= nums[mid]:  
 # target is in the first half  
 if nums[l] <= target < nums[mid]:  
 r = mid - 1  
 else:  
 l = mid + 1  
 # the second half is ordered  
 else:  
 # target is in the second half  
 if nums[mid] < target <= nums[r]:  
 l = mid + 1  
 else:  
 r = mid - 1  
 return False  
**Best\_solution:**C++ concise log(n) solution   
class Solution {  
public:  
 bool search(int A[], int n, int target) {  
 int lo =0, hi = n-1;  
 int mid = 0;  
 while(lo<hi){  
 mid=(lo+hi)/2;  
 if(A[mid]==target) return true;  
 if(A[mid]>A[hi]){  
 if(A[mid]>target && A[lo] <= target) hi = mid;  
 else lo = mid + 1;  
 }else if(A[mid] < A[hi]){  
 if(A[mid]<target && A[hi] >= target) lo = mid + 1;  
 else hi = mid;  
 }else{  
 hi--;  
 }  
   
 }  
 return A[lo] == target ? true : false;  
 }  
};  
  
*82,Remove Duplicates from Sorted List II:***Python\_solution:**Python in-place solution with dummy head node.   
def deleteDuplicates(self, head):  
 dummy = pre = ListNode(0)  
 dummy.next = head  
 while head and head.next:  
 if head.val == head.next.val:  
 while head and head.next and head.val == head.next.val:  
 head = head.next  
 head = head.next  
 pre.next = head  
 else:  
 pre = pre.next  
 head = head.next  
 return dummy.next  
**Best\_solution:**My accepted Java code   
public ListNode deleteDuplicates(ListNode head) {  
 if(head==null) return null;  
 ListNode FakeHead=new ListNode(0);  
 FakeHead.next=head;  
 ListNode pre=FakeHead;  
 ListNode cur=head;  
 while(cur!=null){  
 while(cur.next!=null&&cur.val==cur.next.val){  
 cur=cur.next;  
 }  
 if(pre.next==cur){  
 pre=pre.next;  
 }  
 else{  
 pre.next=cur.next;  
 }  
 cur=cur.next;  
 }  
 return FakeHead.next;  
 }  
  
*83,Remove Duplicates from Sorted List:***Python\_solution:**Simple iterative Python 6 lines, 60 ms   
def deleteDuplicates(self, head):  
 cur = head  
 while cur:  
 while cur.next and cur.next.val == cur.val:  
 cur.next = cur.next.next # skip duplicated node  
 cur = cur.next # not duplicate of current node, move to next node  
 return head  
**Best\_solution:**3 Line JAVA recursive solution   
public ListNode deleteDuplicates(ListNode head) {  
 if(head == null || head.next == null)return head;  
 head.next = deleteDuplicates(head.next);  
 return head.val == head.next.val ? head.next : head;  
}  
  
  
*84,Largest Rectangle in Histogram:***Python\_solution:**AC Python clean solution using stack 76ms   
def largestRectangleArea(self, height):  
 height.append(0)  
 stack = [-1]  
 ans = 0  
 for i in xrange(len(height)):  
 while height[i] < height[stack[-1]]:  
 h = height[stack.pop()]  
 w = i - stack[-1] - 1  
 ans = max(ans, h \* w)  
 stack.append(i)  
 height.pop()  
 return ans  
  
  
  
# 94 / 94 test cases passed.  
# Status: Accepted  
# Runtime: 76 ms  
# 97.34%  
  
**Best\_solution:**My concise C++ solution, AC 90 ms   
 class Solution {  
 public:  
 int largestRectangleArea(vector<int> &height) {  
   
 int ret = 0;  
 height.push\_back(0);  
 vector<int> index;  
   
 for(int i = 0; i < height.size(); i++)  
 {  
 while(index.size() > 0 && height[index.back()] >= height[i])  
 {  
 int h = height[index.back()];  
 index.pop\_back();  
   
 int sidx = index.size() > 0 ? index.back() : -1;  
 if(h \* (i-sidx-1) > ret)  
 ret = h \* (i-sidx-1);  
 }  
 index.push\_back(i);  
 }  
   
 return ret;  
 }  
 };  
  
*85,Maximal Rectangle:***Python\_solution:**AC Python DP solutioin 120ms based on largest rectangle in histogram   
def maximalRectangle(self, matrix):  
 if not matrix or not matrix[0]:  
 return 0  
 n = len(matrix[0])  
 height = [0] \* (n + 1)  
 ans = 0  
 for row in matrix:  
 for i in xrange(n):  
 height[i] = height[i] + 1 if row[i] == '1' else 0  
 stack = [-1]  
 for i in xrange(n + 1):  
 while height[i] < height[stack[-1]]:  
 h = height[stack.pop()]  
 w = i - 1 - stack[-1]  
 ans = max(ans, h \* w)  
 stack.append(i)  
 return ans  
  
# 65 / 65 test cases passed.  
# Status: Accepted  
# Runtime: 120 ms  
# 100%  
  
**Best\_solution:**Share my DP solution   
class Solution {public:  
int maximalRectangle(vector<vector<char> > &matrix) {  
 if(matrix.empty()) return 0;  
 const int m = matrix.size();  
 const int n = matrix[0].size();  
 int left[n], right[n], height[n];  
 fill\_n(left,n,0); fill\_n(right,n,n); fill\_n(height,n,0);  
 int maxA = 0;  
 for(int i=0; i<m; i++) {  
 int cur\_left=0, cur\_right=n;   
 for(int j=0; j<n; j++) { // compute height (can do this from either side)  
 if(matrix[i][j]=='1') height[j]++;   
 else height[j]=0;  
 }  
 for(int j=0; j<n; j++) { // compute left (from left to right)  
 if(matrix[i][j]=='1') left[j]=max(left[j],cur\_left);  
 else {left[j]=0; cur\_left=j+1;}  
 }  
 // compute right (from right to left)  
 for(int j=n-1; j>=0; j--) {  
 if(matrix[i][j]=='1') right[j]=min(right[j],cur\_right);  
 else {right[j]=n; cur\_right=j;}   
 }  
 // compute the area of rectangle (can do this from either side)  
 for(int j=0; j<n; j++)  
 maxA = max(maxA,(right[j]-left[j])\*height[j]);  
 }  
 return maxA;  
}  
  
  
*86,Partition List:***Python\_solution:**Python concise solution with dummy nodes.   
def partition(self, head, x):  
 h1 = l1 = ListNode(0)  
 h2 = l2 = ListNode(0)  
 while head:  
 if head.val < x:  
 l1.next = head  
 l1 = l1.next  
 else:  
 l2.next = head  
 l2 = l2.next  
 head = head.next  
 l2.next = None  
 l1.next = h2.next  
 return h1.next  
**Best\_solution:**Very concise one pass solution   
ListNode \*partition(ListNode \*head, int x) {  
 ListNode node1(0), node2(0);  
 ListNode \*p1 = &node1, \*p2 = &node2;  
 while (head) {  
 if (head->val < x)  
 p1 = p1->next = head;  
 else  
 p2 = p2->next = head;  
 head = head->next;  
 }  
 p2->next = NULL;  
 p1->next = node2.next;  
 return node1.next;  
}  
  
*87,Scramble String:***Python\_solution:**Python recursive solution   
class Solution:  
# @return a boolean  
def isScramble(self, s1, s2):  
 n, m = len(s1), len(s2)  
 if n != m or sorted(s1) != sorted(s2):  
 return False  
 if n < 4 or s1 == s2:  
 return True  
 f = self.isScramble  
 for i in range(1, n):  
 if f(s1[:i], s2[:i]) and f(s1[i:], s2[i:]) or \  
 f(s1[:i], s2[-i:]) and f(s1[i:], s2[:-i]):  
 return True  
 return False  
**Best\_solution:**Share my 4ms c++ recursive solution   
class Solution {  
public:  
 bool isScramble(string s1, string s2) {  
 if(s1==s2)  
 return true;  
   
 int len = s1.length();  
 int count[26] = {0};  
 for(int i=0; i<len; i++)  
 {  
 count[s1[i]-'a']++;  
 count[s2[i]-'a']--;  
 }  
   
 for(int i=0; i<26; i++)  
 {  
 if(count[i]!=0)  
 return false;  
 }  
   
 for(int i=1; i<=len-1; i++)  
 {  
 if( isScramble(s1.substr(0,i), s2.substr(0,i)) && isScramble(s1.substr(i), s2.substr(i)))  
 return true;  
 if( isScramble(s1.substr(0,i), s2.substr(len-i)) && isScramble(s1.substr(i), s2.substr(0,len-i)))  
 return true;  
 }  
 return false;  
 }  
};  
  
*88,Merge Sorted Array:***Python\_solution:**Beautiful Python Solution   
def merge(self, nums1, m, nums2, n):  
 while m > 0 and n > 0:  
 if nums1[m-1] >= nums2[n-1]:  
 nums1[m+n-1] = nums1[m-1]  
 m -= 1  
 else:  
 nums1[m+n-1] = nums2[n-1]  
 n -= 1  
 if n > 0:  
 nums1[:n] = nums2[:n]  
**Best\_solution:**This is my AC code, may help you   
class Solution {  
public:  
 void merge(int A[], int m, int B[], int n) {  
 int i=m-1;  
 int j=n-1;  
 int k = m+n-1;  
 while(i >=0 && j>=0)  
 {  
 if(A[i] > B[j])  
 A[k--] = A[i--];  
 else  
 A[k--] = B[j--];  
 }  
 while(j>=0)  
 A[k--] = B[j--];  
 }  
};  
  
*89,Gray Code:***Python\_solution:**One-liner Python solution (with demo in comments)   
class Solution:  
 # @return a list of integers  
 '''  
 from up to down, then left to right  
   
 0 1 11 110  
 10 111  
 101  
 100  
   
 start: [0]  
 i = 0: [0, 1]  
 i = 1: [0, 1, 3, 2]  
 i = 2: [0, 1, 3, 2, 6, 7, 5, 4]  
 '''  
 def grayCode(self, n):  
 results = [0]  
 for i in range(n):  
 results += [x + pow(2, i) for x in reversed(results)]  
 return results  
**Best\_solution:**An accepted three line solution in JAVA   
public List<Integer> grayCode(int n) {  
 List<Integer> result = new LinkedList<>();  
 for (int i = 0; i < 1<<n; i++) result.add(i ^ i>>1);  
 return result;  
}  
  
  
*90,Subsets II:***Python\_solution:**Simple python solution without extra space.   
class Solution:  
 # @param num, a list of integer  
 # @return a list of lists of integer  
 def subsetsWithDup(self, S):  
 res = [[]]  
 S.sort()  
 for i in range(len(S)):  
 if i == 0 or S[i] != S[i - 1]:  
 l = len(res)  
 for j in range(len(res) - l, len(res)):  
 res.append(res[j] + [S[i]])  
 return res  
  
**Best\_solution:**C++ solution and explanation   
 class Solution {  
public:  
 vector<vector<int> > subsetsWithDup(vector<int> &S) {  
 vector<vector<int> > totalset = {{}};  
 sort(S.begin(),S.end());  
 for(int i=0; i<S.size();){  
 int count = 0; // num of elements are the same  
 while(count + i<S.size() && S[count+i]==S[i]) count++;  
 int previousN = totalset.size();  
 for(int k=0; k<previousN; k++){  
 vector<int> instance = totalset[k];  
 for(int j=0; j<count; j++){  
 instance.push\_back(S[i]);  
 totalset.push\_back(instance);  
 }  
 }  
 i += count;  
 }  
 return totalset;  
 }  
};  
  
*91,Decode Ways:***Python\_solution:**Accpeted Python DP solution   
class Solution:  
 # @param s, a string  
 # @return an integer  
 def numDecodings(self, s):  
 #dp[i] = dp[i-1] if s[i] != "0"  
 # +dp[i-2] if "09" < s[i-1:i+1] < "27"  
 if s == "": return 0  
 dp = [0 for x in range(len(s)+1)]  
 dp[0] = 1  
 for i in range(1, len(s)+1):  
 if s[i-1] != "0":  
 dp[i] += dp[i-1]  
 if i != 1 and s[i-2:i] < "27" and s[i-2:i] > "09": #"01"ways = 0  
 dp[i] += dp[i-2]  
 return dp[len(s)]  
**Best\_solution:**DP Solution (Java) for reference   
public class Solution {  
 public int numDecodings(String s) {  
 int n = s.length();  
 if (n == 0) return 0;  
   
 int[] memo = new int[n+1];  
 memo[n] = 1;  
 memo[n-1] = s.charAt(n-1) != '0' ? 1 : 0;  
   
 for (int i = n - 2; i >= 0; i--)  
 if (s.charAt(i) == '0') continue;  
 else memo[i] = (Integer.parseInt(s.substring(i,i+2))<=26) ? memo[i+1]+memo[i+2] : memo[i+1];  
   
 return memo[0];  
 }  
}  
  
*92,Reverse Linked List II:***Python\_solution:**Python one pass iterative solution   
class Solution:  
 # @param head, a ListNode  
 # @param m, an integer  
 # @param n, an integer  
 # @return a ListNode  
 def reverseBetween(self, head, m, n):  
 if m == n:  
 return head  
  
 dummyNode = ListNode(0)  
 dummyNode.next = head  
 pre = dummyNode  
  
 for i in range(m - 1):  
 pre = pre.next  
   
 # reverse the [m, n] nodes  
 reverse = None  
 cur = pre.next  
 for i in range(n - m + 1):  
 next = cur.next  
 cur.next = reverse  
 reverse = cur  
 cur = next  
  
 pre.next.next = cur  
 pre.next = reverse  
  
 return dummyNode.next  
**Best\_solution:**Simple Java solution with clear explanation   
public ListNode reverseBetween(ListNode head, int m, int n) {  
 if(head == null) return null;  
 ListNode dummy = new ListNode(0); // create a dummy node to mark the head of this list  
 dummy.next = head;  
 ListNode pre = dummy; // make a pointer pre as a marker for the node before reversing  
 for(int i = 0; i<m-1; i++) pre = pre.next;  
   
 ListNode start = pre.next; // a pointer to the beginning of a sub-list that will be reversed  
 ListNode then = start.next; // a pointer to a node that will be reversed  
   
 // 1 - 2 -3 - 4 - 5 ; m=2; n =4 ---> pre = 1, start = 2, then = 3  
 // dummy-> 1 -> 2 -> 3 -> 4 -> 5  
   
 for(int i=0; i<n-m; i++)  
 {  
 start.next = then.next;  
 then.next = pre.next;  
 pre.next = then;  
 then = start.next;  
 }  
   
 // first reversing : dummy->1 - 3 - 2 - 4 - 5; pre = 1, start = 2, then = 4  
 // second reversing: dummy->1 - 4 - 3 - 2 - 5; pre = 1, start = 2, then = 5 (finish)  
   
 return dummy.next;  
   
}  
  
*93,Restore IP Addresses:***Python\_solution:**Python easy to understand solution with comments (backtracking).   
def restoreIpAddresses(self, s):  
 res = []  
 self.dfs(s, 0, "", res)  
 return res  
   
def dfs(self, s, index, path, res):  
 if index == 4:  
 if not s:  
 res.append(path[:-1])  
 return # backtracking  
 for i in xrange(1, 4):  
 # the digits we choose should no more than the length of s  
 if i <= len(s):  
 #choose one digit  
 if i == 1:   
 self.dfs(s[i:], index+1, path+s[:i]+".", res)  
 #choose two digits, the first one should not be "0"  
 elif i == 2 and s[0] != "0":   
 self.dfs(s[i:], index+1, path+s[:i]+".", res)  
 #choose three digits, the first one should not be "0", and should less than 256  
 elif i == 3 and s[0] != "0" and int(s[:3]) <= 255:  
 self.dfs(s[i:], index+1, path+s[:i]+".", res)  
**Best\_solution:**My code in Java   
public class Solution {  
 public List<String> restoreIpAddresses(String s) {  
 List<String> res = new ArrayList<String>();  
 int len = s.length();  
 for(int i = 1; i<4 && i<len-2; i++){  
 for(int j = i+1; j<i+4 && j<len-1; j++){  
 for(int k = j+1; k<j+4 && k<len; k++){  
 String s1 = s.substring(0,i), s2 = s.substring(i,j), s3 = s.substring(j,k), s4 = s.substring(k,len);  
 if(isValid(s1) && isValid(s2) && isValid(s3) && isValid(s4)){  
 res.add(s1+"."+s2+"."+s3+"."+s4);  
 }  
 }  
 }  
 }  
 return res;  
 }  
 public boolean isValid(String s){  
 if(s.length()>3 || s.length()==0 || (s.charAt(0)=='0' && s.length()>1) || Integer.parseInt(s)>255)  
 return false;  
 return true;  
 }  
}  
  
  
*94,Binary Tree Inorder Traversal:***Python\_solution:**Python recursive and iterative solutions.   
# recursively  
def inorderTraversal1(self, root):  
 res = []  
 self.helper(root, res)  
 return res  
   
def helper(self, root, res):  
 if root:  
 self.helper(root.left, res)  
 res.append(root.val)  
 self.helper(root.right, res)  
   
# iteratively   
def inorderTraversal(self, root):  
 res, stack = [], []  
 while True:  
 while root:  
 stack.append(root)  
 root = root.left  
 if not stack:  
 return res  
 node = stack.pop()  
 res.append(node.val)  
 root = node.right  
**Best\_solution:**Iterative solution in Java - simple and readable   
public List<Integer> inorderTraversal(TreeNode root) {  
 List<Integer> list = new ArrayList<Integer>();  
  
 Stack<TreeNode> stack = new Stack<TreeNode>();  
 TreeNode cur = root;  
  
 while(cur!=null || !stack.empty()){  
 while(cur!=null){  
 stack.add(cur);  
 cur = cur.left;  
 }  
 cur = stack.pop();  
 list.add(cur.val);  
 cur = cur.right;  
 }  
  
 return list;  
}  
  
*95,Unique Binary Search Trees II:***Python\_solution:**Recursive python solution   
class Solution(object):  
 def generateTrees(self, n):  
 """  
 :type n: int  
 :rtype: List[TreeNode]  
 """  
 if n == 0:  
 return [[]]  
 return self.dfs(1, n+1)  
   
 def dfs(self, start, end):  
 if start == end:  
 return None  
 result = []  
 for i in xrange(start, end):  
 for l in self.dfs(start, i) or [None]:  
 for r in self.dfs(i+1, end) or [None]:  
 node = TreeNode(i)  
 node.left, node.right = l, r  
 result.append(node)  
 return result  
  
**Best\_solution:**A simple recursive solution   
public class Solution {  
 public List<TreeNode> generateTrees(int n) {  
   
 return genTrees(1,n);  
 }  
   
 public List<TreeNode> genTrees (int start, int end)  
 {  
  
 List<TreeNode> list = new ArrayList<TreeNode>();  
  
 if(start>end)  
 {  
 list.add(null);  
 return list;  
 }  
   
 if(start == end){  
 list.add(new TreeNode(start));  
 return list;  
 }  
   
 List<TreeNode> left,right;  
 for(int i=start;i<=end;i++)  
 {  
   
 left = genTrees(start, i-1);  
 right = genTrees(i+1,end);  
   
 for(TreeNode lnode: left)  
 {  
 for(TreeNode rnode: right)  
 {  
 TreeNode root = new TreeNode(i);  
 root.left = lnode;  
 root.right = rnode;  
 list.add(root);  
 }  
 }  
   
 }  
   
 return list;  
 }  
}  
  
*96,Unique Binary Search Trees:***Best\_solution:**DP Solution in 6 lines with explanation. F(i, n) = G(i-1) \* G(n-i)   
G(n)  
  
*97,Interleaving String:***Python\_solution:**Python DP solutions (O(m\*n), O(n) space), BFS, DFS.   
# O(m\*n) space  
def isInterleave1(self, s1, s2, s3):  
 r, c, l= len(s1), len(s2), len(s3)  
 if r+c != l:  
 return False  
 dp = [[True for \_ in xrange(c+1)] for \_ in xrange(r+1)]  
 for i in xrange(1, r+1):  
 dp[i][0] = dp[i-1][0] and s1[i-1] == s3[i-1]  
 for j in xrange(1, c+1):  
 dp[0][j] = dp[0][j-1] and s2[j-1] == s3[j-1]  
 for i in xrange(1, r+1):  
 for j in xrange(1, c+1):  
 dp[i][j] = (dp[i-1][j] and s1[i-1] == s3[i-1+j]) or \  
 (dp[i][j-1] and s2[j-1] == s3[i-1+j])  
 return dp[-1][-1]  
  
# O(2\*n) space  
def isInterleave2(self, s1, s2, s3):  
 l1, l2, l3 = len(s1)+1, len(s2)+1, len(s3)+1  
 if l1+l2 != l3+1:  
 return False  
 pre = [True for \_ in xrange(l2)]  
 for j in xrange(1, l2):  
 pre[j] = pre[j-1] and s2[j-1] == s3[j-1]  
 for i in xrange(1, l1):  
 cur = [pre[0] and s1[i-1] == s3[i-1]] \* l2  
 for j in xrange(1, l2):  
 cur[j] = (cur[j-1] and s2[j-1] == s3[i+j-1]) or \  
 (pre[j] and s1[i-1] == s3[i+j-1])  
 pre = cur[:]  
 return pre[-1]  
  
# O(n) space  
def isInterleave3(self, s1, s2, s3):  
 r, c, l= len(s1), len(s2), len(s3)  
 if r+c != l:  
 return False  
 dp = [True for \_ in xrange(c+1)]   
 for j in xrange(1, c+1):  
 dp[j] = dp[j-1] and s2[j-1] == s3[j-1]  
 for i in xrange(1, r+1):  
 dp[0] = (dp[0] and s1[i-1] == s3[i-1])  
 for j in xrange(1, c+1):  
 dp[j] = (dp[j] and s1[i-1] == s3[i-1+j]) or (dp[j-1] and s2[j-1] == s3[i-1+j])  
 return dp[-1]  
   
# DFS   
def isInterleave4(self, s1, s2, s3):  
 r, c, l= len(s1), len(s2), len(s3)  
 if r+c != l:  
 return False  
 stack, visited = [(0, 0)], set((0, 0))  
 while stack:  
 x, y = stack.pop()  
 if x+y == l:  
 return True  
 if x+1 <= r and s1[x] == s3[x+y] and (x+1, y) not in visited:  
 stack.append((x+1, y)); visited.add((x+1, y))  
 if y+1 <= c and s2[y] == s3[x+y] and (x, y+1) not in visited:  
 stack.append((x, y+1)); visited.add((x, y+1))  
 return False  
   
# BFS   
def isInterleave(self, s1, s2, s3):  
 r, c, l= len(s1), len(s2), len(s3)  
 if r+c != l:  
 return False  
 queue, visited = [(0, 0)], set((0, 0))  
 while queue:  
 x, y = queue.pop(0)  
 if x+y == l:  
 return True  
 if x+1 <= r and s1[x] == s3[x+y] and (x+1, y) not in visited:  
 queue.append((x+1, y)); visited.add((x+1, y))  
 if y+1 <= c and s2[y] == s3[x+y] and (x, y+1) not in visited:  
 queue.append((x, y+1)); visited.add((x, y+1))  
 return False  
**Best\_solution:**My DP solution in C++   
 bool isInterleave(string s1, string s2, string s3) {  
   
 if(s3.length() != s1.length() + s2.length())  
 return false;  
   
 bool table[s1.length()+1][s2.length()+1];  
   
 for(int i=0; i<s1.length()+1; i++)  
 for(int j=0; j< s2.length()+1; j++){  
 if(i==0 && j==0)  
 table[i][j] = true;  
 else if(i == 0)  
 table[i][j] = ( table[i][j-1] && s2[j-1] == s3[i+j-1]);  
 else if(j == 0)  
 table[i][j] = ( table[i-1][j] && s1[i-1] == s3[i+j-1]);  
 else  
 table[i][j] = (table[i-1][j] && s1[i-1] == s3[i+j-1] ) || (table[i][j-1] && s2[j-1] == s3[i+j-1] );  
 }  
   
 return table[s1.length()][s2.length()];  
}  
  
  
*98,Validate Binary Search Tree:***Python\_solution:**Python version based on inorder traversal   
# Definition for a binary tree node  
# class TreeNode:  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.left = None  
# self.right = None  
  
class Solution:  
 # @param root, a tree node  
 # @return a boolean  
 # 7:38  
 def isValidBST(self, root):  
 output = []  
 self.inOrder(root, output)  
   
 for i in range(1, len(output)):  
 if output[i-1] >= output[i]:  
 return False  
  
 return True  
  
 def inOrder(self, root, output):  
 if root is None:  
 return  
   
 self.inOrder(root.left, output)  
 output.append(root.val)  
 self.inOrder(root.right, output)  
**Best\_solution:**C++ in-order traversal, and please do not rely on buggy INT\_MAX, INT\_MIN solutions any more   
class Solution {  
public:  
 bool isValidBST(TreeNode\* root) {  
 TreeNode\* prev = NULL;  
 return validate(root, prev);  
 }  
 bool validate(TreeNode\* node, TreeNode\* &prev) {  
 if (node == NULL) return true;  
 if (!validate(node->left, prev)) return false;  
 if (prev != NULL && prev->val >= node->val) return false;  
 prev = node;  
 return validate(node->right, prev);  
 }  
};  
  
  
*99,Recover Binary Search Tree:***Python\_solution:**Tree Deserializer and Visualizer for Python   
deserialize('[1,2,3,null,null,4,null,null,5]')  
**Best\_solution:**No Fancy Algorithm, just Simple and Powerful In-Order Traversal   
private void traverse (TreeNode root) {  
 if (root == null)  
 return;  
 traverse(root.left);  
 // Do some business  
 traverse(root.right);  
}  
  
  
*100,Same Tree:***Python\_solution:**Shortest+simplest Python   
def isSameTree(self, p, q):  
 if p and q:  
 return p.val == q.val and self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)  
 return p is q  
  
**Best\_solution:**Five line Java solution with recursion   
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 isSameTree(p.left, q.left) && isSameTree(p.right, q.right);  
 return false;  
}  
  
*101,Symmetric Tree:***Python\_solution:**Recursively and iteratively solution in Python   
class Solution:  
 def isSymmetric(self, root):  
 if root is None:  
 return True  
 else:  
 return self.isMirror(root.left, root.right)  
  
 def isMirror(self, left, right):  
 if left is None and right is None:  
 return True  
 if left is None or right is None:  
 return False  
  
 if left.val == right.val:  
 outPair = self.isMirror(left.left, right.right)  
 inPiar = self.isMirror(left.right, right.left)  
 return outPair and inPiar  
 else:  
 return False  
  
**Best\_solution:**Recursive and non-recursive solutions in Java   
public boolean isSymmetric(TreeNode root) {  
 return root==null || isSymmetricHelp(root.left, root.right);  
}  
  
private boolean isSymmetricHelp(TreeNode left, TreeNode right){  
 if(left==null || right==null)  
 return left==right;  
 if(left.val!=right.val)  
 return false;  
 return isSymmetricHelp(left.left, right.right) && isSymmetricHelp(left.right, right.left);  
}  
  
  
*102,Binary Tree Level Order Traversal:***Python\_solution:**5-6 lines fast python solution (48 ms)   
level  
**Best\_solution:**Java solution with a queue used   
public class Solution {  
 public List<List<Integer>> levelOrder(TreeNode root) {  
 Queue<TreeNode> queue = new LinkedList<TreeNode>();  
 List<List<Integer>> wrapList = new LinkedList<List<Integer>>();  
   
 if(root == null) return wrapList;  
   
 queue.offer(root);  
 while(!queue.isEmpty()){  
 int levelNum = queue.size();  
 List<Integer> subList = new LinkedList<Integer>();  
 for(int i=0; i<levelNum; i++) {  
 if(queue.peek().left != null) queue.offer(queue.peek().left);  
 if(queue.peek().right != null) queue.offer(queue.peek().right);  
 subList.add(queue.poll().val);  
 }  
 wrapList.add(subList);  
 }  
 return wrapList;  
 }  
}  
  
*103,Binary Tree Zigzag Level Order Traversal:***Python\_solution:**Python simple BFS   
class Solution(object):  
def zigzagLevelOrder(self, root):  
 """  
 :type root: TreeNode  
 :rtype: List[List[int]]  
 """  
 if not root: return []  
 res, temp, stack, flag=[], [], [root], 1  
 while stack:  
 for i in xrange(len(stack)):  
 node=stack.pop(0)  
 temp+=[node.val]  
 if node.left: stack+=[node.left]  
 if node.right: stack+=[node.right]  
 res+=[temp[::flag]]  
 temp=[]  
 flag\*=-1  
 return res  
**Best\_solution:**My accepted JAVA solution   
public class Solution {  
 public List<List<Integer>> zigzagLevelOrder(TreeNode root)   
 {  
 List<List<Integer>> sol = new ArrayList<>();  
 travel(root, sol, 0);  
 return sol;  
 }  
   
 private void travel(TreeNode curr, List<List<Integer>> sol, int level)  
 {  
 if(curr == null) return;  
   
 if(sol.size() <= level)  
 {  
 List<Integer> newLevel = new LinkedList<>();  
 sol.add(newLevel);  
 }  
   
 List<Integer> collection = sol.get(level);  
 if(level % 2 == 0) collection.add(curr.val);  
 else collection.add(0, curr.val);  
   
 travel(curr.left, sol, level + 1);  
 travel(curr.right, sol, level + 1);  
 }  
}  
  
  
*104,Maximum Depth of Binary Tree:***Python\_solution:**1 line Ruby and Python   
def max\_depth(root)  
 root ? 1 + [max\_depth(root.left), max\_depth(root.right)].max : 0  
end  
  
**Best\_solution:**Can leetcode share top performing solution(s) of problems for each supported language ?   
int maxDepth(TreeNode\* root) {  
 if (root == NULL)  
 return 0;  
 stack<TreeNode\*> myStack;  
 stack<int> depthStack;  
 if (root != NULL) myStack.push(root);  
 int maxDepth = 0;  
 depthStack.push(0);  
 while (!myStack.empty()) {  
 TreeNode \*p = myStack.top();  
 int d = depthStack.top();  
 if (d > maxDepth) maxDepth = d;  
 myStack.pop();  
 depthStack.pop();  
 if (p->left != NULL) {myStack.push(p->left); depthStack.push(d + 1);}  
 if (p->right != NULL) {myStack.push(p->right); depthStack.push(d + 1);}  
 }  
 return maxDepth + 1;  
}  
  
*105,Construct Binary Tree from Preorder and Inorder Traversal:***Python\_solution:**Python short recursive solution.   
def buildTree(self, preorder, inorder):  
 if inorder:  
 ind = inorder.index(preorder.pop(0))  
 root = TreeNode(inorder[ind])  
 root.left = self.buildTree(preorder, inorder[0:ind])  
 root.right = self.buildTree(preorder, inorder[ind+1:])  
 return root  
**Best\_solution:**My Accepted Java Solution   
public TreeNode buildTree(int[] preorder, int[] inorder) {  
 return helper(0, 0, inorder.length - 1, preorder, inorder);  
}  
  
public TreeNode helper(int preStart, int inStart, int inEnd, int[] preorder, int[] inorder) {  
 if (preStart > preorder.length - 1 || inStart > inEnd) {  
 return null;  
 }  
 TreeNode root = new TreeNode(preorder[preStart]);  
 int inIndex = 0; // Index of current root in inorder  
 for (int i = inStart; i <= inEnd; i++) {  
 if (inorder[i] == root.val) {  
 inIndex = i;  
 }  
 }  
 root.left = helper(preStart + 1, inStart, inIndex - 1, preorder, inorder);  
 root.right = helper(preStart + inIndex - inStart + 1, inIndex + 1, inEnd, preorder, inorder);  
 return root;  
}  
  
  
*106,Construct Binary Tree from Inorder and Postorder Traversal:***Python\_solution:**A Python recursive solution   
# Definition for a binary tree node  
# class TreeNode:  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.left = None  
# self.right = None  
  
class Solution:  
 # @param inorder, a list of integers  
 # @param postorder, a list of integers  
 # @return a tree node  
 # 12:00  
 def buildTree(self, inorder, postorder):  
 if not inorder or not postorder:  
 return None  
   
 root = TreeNode(postorder.pop())  
 inorderIndex = inorder.index(root.val)  
  
 root.right = self.buildTree(inorder[inorderIndex+1:], postorder)  
 root.left = self.buildTree(inorder[:inorderIndex], postorder)  
  
 return root  
**Best\_solution:**My recursive Java code with O(n) time and O(n) space   
public TreeNode buildTreePostIn(int[] inorder, int[] postorder) {  
 if (inorder == null || postorder == null || inorder.length != postorder.length)  
 return null;  
 HashMap<Integer, Integer> hm = new HashMap<Integer,Integer>();  
 for (int i=0;i<inorder.length;++i)  
 hm.put(inorder[i], i);  
 return buildTreePostIn(inorder, 0, inorder.length-1, postorder, 0,   
 postorder.length-1,hm);  
}  
  
private TreeNode buildTreePostIn(int[] inorder, int is, int ie, int[] postorder, int ps, int pe,   
 HashMap<Integer,Integer> hm){  
 if (ps>pe || is>ie) return null;  
 TreeNode root = new TreeNode(postorder[pe]);  
 int ri = hm.get(postorder[pe]);  
 TreeNode leftchild = buildTreePostIn(inorder, is, ri-1, postorder, ps, ps+ri-is-1, hm);  
 TreeNode rightchild = buildTreePostIn(inorder,ri+1, ie, postorder, ps+ri-is, pe-1, hm);  
 root.left = leftchild;  
 root.right = rightchild;  
 return root;  
}  
  
*107,Binary Tree Level Order Traversal II:***Python\_solution:**Python solutions (dfs recursively, dfs+stack, bfs+queue).   
# dfs recursively  
def levelOrderBottom1(self, root):  
 res = []  
 self.dfs(root, 0, res)  
 return res  
  
def dfs(self, root, level, res):  
 if root:  
 if len(res) < level + 1:  
 res.insert(0, [])  
 res[-(level+1)].append(root.val)  
 self.dfs(root.left, level+1, res)  
 self.dfs(root.right, level+1, res)  
   
# dfs + stack  
def levelOrderBottom2(self, root):  
 stack = [(root, 0)]  
 res = []  
 while stack:  
 node, level = stack.pop()  
 if node:  
 if len(res) < level+1:  
 res.insert(0, [])  
 res[-(level+1)].append(node.val)  
 stack.append((node.right, level+1))  
 stack.append((node.left, level+1))  
 return res  
   
# bfs + queue   
def levelOrderBottom(self, root):  
 queue, res = collections.deque([(root, 0)]), []  
 while queue:  
 node, level = queue.popleft()  
 if node:  
 if len(res) < level+1:  
 res.insert(0, [])  
 res[-(level+1)].append(node.val)  
 queue.append((node.left, level+1))  
 queue.append((node.right, level+1))  
 return res  
**Best\_solution:**My DFS and BFS java solution   
public class Solution {  
 public List<List<Integer>> levelOrderBottom(TreeNode root) {  
 Queue<TreeNode> queue = new LinkedList<TreeNode>();  
 List<List<Integer>> wrapList = new LinkedList<List<Integer>>();  
   
 if(root == null) return wrapList;  
   
 queue.offer(root);  
 while(!queue.isEmpty()){  
 int levelNum = queue.size();  
 List<Integer> subList = new LinkedList<Integer>();  
 for(int i=0; i<levelNum; i++) {  
 if(queue.peek().left != null) queue.offer(queue.peek().left);  
 if(queue.peek().right != null) queue.offer(queue.peek().right);  
 subList.add(queue.poll().val);  
 }  
 wrapList.add(0, subList);  
 }  
 return wrapList;  
 }  
}  
  
  
*108,Convert Sorted Array to Binary Search Tree:***Python\_solution:**An easy Python solution   
# Definition for a binary tree node  
# class TreeNode:  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.left = None  
# self.right = None  
  
class Solution:  
 # @param num, a list of integers  
 # @return a tree node  
 # 12:37  
 def sortedArrayToBST(self, num):  
 if not num:  
 return None  
  
 mid = len(num) // 2  
  
 root = TreeNode(num[mid])  
 root.left = self.sortedArrayToBST(num[:mid])  
 root.right = self.sortedArrayToBST(num[mid+1:])  
  
 return root  
**Best\_solution:**My Accepted Java Solution   
public TreeNode sortedArrayToBST(int[] num) {  
 if (num.length == 0) {  
 return null;  
 }  
 TreeNode head = helper(num, 0, num.length - 1);  
 return head;  
}  
  
public TreeNode helper(int[] num, int low, int high) {  
 if (low > high) { // Done  
 return null;  
 }  
 int mid = (low + high) / 2;  
 TreeNode node = new TreeNode(num[mid]);  
 node.left = helper(num, low, mid - 1);  
 node.right = helper(num, mid + 1, high);  
 return node;  
}  
  
*109,Convert Sorted List to Binary Search Tree:***Python\_solution:**Python recursive solution with detailed comments (operate linked-list directly).   
# recursively  
def sortedListToBST(self, head):  
 if not head:  
 return   
 if not head.next:  
 return TreeNode(head.val)  
 # here we get the middle point,  
 # even case, like '1234', slow points to '2',  
 # '3' is root, '12' belongs to left, '4' is right  
 # odd case, like '12345', slow points to '2', '12'  
 # belongs to left, '3' is root, '45' belongs to right  
 slow, fast = head, head.next.next  
 while fast and fast.next:  
 fast = fast.next.next  
 slow = slow.next  
 # tmp points to root  
 tmp = slow.next  
 # cut down the left child  
 slow.next = None  
 root = TreeNode(tmp.val)  
 root.left = self.sortedListToBST(head)  
 root.right = self.sortedListToBST(tmp.next)  
 return root  
**Best\_solution:**Share my JAVA solution, 1ms, very short and concise.   
public class Solution {  
public TreeNode sortedListToBST(ListNode head) {  
 if(head==null) return null;  
 return toBST(head,null);  
}  
public TreeNode toBST(ListNode head, ListNode tail){  
 ListNode slow = head;  
 ListNode fast = head;  
 if(head==tail) return null;  
   
 while(fast!=tail&&fast.next!=tail){  
 fast = fast.next.next;  
 slow = slow.next;  
 }  
 TreeNode thead = new TreeNode(slow.val);  
 thead.left = toBST(head,slow);  
 thead.right = toBST(slow.next,tail);  
 return thead;  
}  
  
  
*110,Balanced Binary Tree:***Python\_solution:**A simple Python recursive solution - 172ms   
# Definition for a binary tree node.  
# class TreeNode:  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.left = None  
# self.right = None  
  
class Solution:  
 # @param {TreeNode} root  
 # @return {boolean}  
 def isBalanced(self, root):  
 if not root:  
 return True  
  
 return abs(self.getHeight(root.left) - self.getHeight(root.right)) < 2 and self.isBalanced(root.left) and self.isBalanced(root.right)  
  
 def getHeight(self, root):  
 if not root:  
 return 0  
  
 return 1 + max(self.getHeight(root.left), self.getHeight(root.right))  
**Best\_solution:**The bottom up O(N) solution would be better   
class solution {  
public:  
 int depth (TreeNode \*root) {  
 if (root == NULL) return 0;  
 return max (depth(root -> left), depth (root -> right)) + 1;  
 }  
  
 bool isBalanced (TreeNode \*root) {  
 if (root == NULL) return true;  
   
 int left=depth(root->left);  
 int right=depth(root->right);  
   
 return abs(left - right) <= 1 && isBalanced(root->left) && isBalanced(root->right);  
 }  
};  
  
  
*111,Minimum Depth of Binary Tree:***Python\_solution:**My solution in python   
class Solution:  
 # @param root, a tree node  
 # @return an integer   
 def minDepth(self, root):  
 if root == None:  
 return 0  
 if root.left==None or root.right==None:  
 return self.minDepth(root.left)+self.minDepth(root.right)+1  
 return min(self.minDepth(root.right),self.minDepth(root.left))+1  
**Best\_solution:**My 4 Line java solution   
public class Solution {  
 public int minDepth(TreeNode root) {  
 if(root == null) return 0;  
 int left = minDepth(root.left);  
 int right = minDepth(root.right);  
 return (left == 0 || right == 0) ? left + right + 1: Math.min(left,right) + 1;  
   
 }  
}  
  
*112,Path Sum:***Python\_solution:**Short Python recursive solution - O(n)   
# Definition for a binary tree node  
# class TreeNode:  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.left = None  
# self.right = None  
  
class Solution:  
 # @param root, a tree node  
 # @param sum, an integer  
 # @return a boolean  
 # 1:27  
 def hasPathSum(self, root, sum):  
 if not root:  
 return False  
  
 if not root.left and not root.right and root.val == sum:  
 return True  
   
 sum -= root.val  
  
 return self.hasPathSum(root.left, sum) or self.hasPathSum(root.right, sum)  
**Best\_solution:**[Accepted]My recursive solution in Java   
public class Solution {  
 public boolean hasPathSum(TreeNode root, int sum) {  
 if(root == null) return false;  
   
 if(root.left == null && root.right == null && sum - root.val == 0) return true;  
   
 return hasPathSum(root.left, sum - root.val) || hasPathSum(root.right, sum - root.val);  
 }  
}  
  
*113,Path Sum II:***Python\_solution:**Short python solution   
class Solution:  
  
def pathSum(self, root, sum):  
 if not root: return []  
 if root.left == None and root.right == None:  
 if sum == root.val:   
 return [[root.val]]  
 else:   
 return []  
 a = self.pathSum(root.left, sum - root.val) + \  
 self.pathSum(root.right, sum - root.val)  
 return [[root.val] + i for i in a]  
**Best\_solution:**DFS with one LinkedList , accepted java solution   
public List<List<Integer>> pathSum(TreeNode root, int sum){  
 List<List<Integer>> result = new LinkedList<List<Integer>>();  
 List<Integer> currentResult = new LinkedList<Integer>();  
 pathSum(root,sum,currentResult,result);  
 return result;  
}  
  
public void pathSum(TreeNode root, int sum, List<Integer> currentResult,  
 List<List<Integer>> result) {  
  
 if (root == null)  
 return;  
 currentResult.add(new Integer(root.val));  
 if (root.left == null && root.right == null && sum == root.val) {  
 result.add(new LinkedList(currentResult));  
 currentResult.remove(currentResult.size() - 1);//don't forget to remove the last integer  
 return;  
 } else {  
 pathSum(root.left, sum - root.val, currentResult, result);  
 pathSum(root.right, sum - root.val, currentResult, result);  
 }  
 currentResult.remove(currentResult.size() - 1);  
}  
  
*114,Flatten Binary Tree to Linked List:***Python\_solution:**An inorder python solution   
class Solution:  
# @param root, a tree node  
# @return nothing, do it in place  
prev = None  
def flatten(self, root):  
 if not root:  
 return  
 self.prev = root  
 self.flatten(root.left)  
  
 temp = root.right  
 root.right, root.left = root.left, None  
 self.prev.right = temp  
  
 self.flatten(temp)  
  
  
  
 \*  
 /  
 n  
 / \  
 left right  
 \   
 \*  
 \*  
 \  
 p  
  
**Best\_solution:**My short post order traversal Java solution for share   
private TreeNode prev = null;  
  
public void flatten(TreeNode root) {  
 if (root == null)  
 return;  
 flatten(root.right);  
 flatten(root.left);  
 root.right = prev;  
 root.left = null;  
 prev = root;  
}  
  
*115,Distinct Subsequences:***Best\_solution:**Easy to understand DP in Java   
mem  
  
*116,Populating Next Right Pointers in Each Node:***Best\_solution:**A simple accepted solution   
void connect(TreeLinkNode \*root) {  
 if (root == NULL) return;  
 TreeLinkNode \*pre = root;  
 TreeLinkNode \*cur = NULL;  
 while(pre->left) {  
 cur = pre;  
 while(cur) {  
 cur->left->next = cur->right;  
 if(cur->next) cur->right->next = cur->next->left;  
 cur = cur->next;  
 }  
 pre = pre->left;  
 }  
}  
  
  
*117,Populating Next Right Pointers in Each Node II:***Python\_solution:**AC Python O(1) space solution 12 lines and easy to understand   
def connect(self, node):  
 tail = dummy = TreeLinkNode(0)  
 while node:  
 tail.next = node.left  
 if tail.next:  
 tail = tail.next  
 tail.next = node.right  
 if tail.next:  
 tail = tail.next  
 node = node.next  
 if not node:  
 tail = dummy  
 node = dummy.next  
  
  
# 61 / 61 test cases passed.  
# Status: Accepted  
# Runtime: 100 ms  
# 95.26%  
**Best\_solution:**O(1) space O(n) complexity Iterative Solution   
public class Solution {  
   
 //based on level order traversal  
 public void connect(TreeLinkNode root) {  
  
 TreeLinkNode head = null; //head of the next level  
 TreeLinkNode prev = null; //the leading node on the next level  
 TreeLinkNode cur = root; //current node of current level  
  
 while (cur != null) {  
   
 while (cur != null) { //iterate on the current level  
 //left child  
 if (cur.left != null) {  
 if (prev != null) {  
 prev.next = cur.left;  
 } else {  
 head = cur.left;  
 }  
 prev = cur.left;  
 }  
 //right child  
 if (cur.right != null) {  
 if (prev != null) {  
 prev.next = cur.right;  
 } else {  
 head = cur.right;  
 }  
 prev = cur.right;  
 }  
 //move to next node  
 cur = cur.next;  
 }  
   
 //move to next level  
 cur = head;  
 head = null;  
 prev = null;  
 }  
   
 }  
}  
  
*118,Pascal's Triangle:***Python\_solution:**Python 4 lines short solution using map.   
def generate(self, numRows):  
 res = [[1]]  
 for i in range(1, numRows):  
 res += [map(lambda x, y: x+y, res[-1] + [0], [0] + res[-1])]  
 return res[:numRows]  
  
**Best\_solution:**My concise solution in Java   
public class Solution {  
public List<List<Integer>> generate(int numRows)  
{  
 List<List<Integer>> allrows = new ArrayList<List<Integer>>();  
 ArrayList<Integer> row = new ArrayList<Integer>();  
 for(int i=0;i<numRows;i++)  
 {  
 row.add(0, 1);  
 for(int j=1;j<row.size()-1;j++)  
 row.set(j, row.get(j)+row.get(j+1));  
 allrows.add(new ArrayList<Integer>(row));  
 }  
 return allrows;  
   
}  
  
  
*119,Pascal's Triangle II:***Python\_solution:**Very simple Python solution   
class Solution(object):  
 def getRow(self, rowIndex):  
 """  
 :type rowIndex: int  
 :rtype: List[int]  
 """  
 row = [1]  
 for \_ in range(rowIndex):  
 row = [x + y for x, y in zip([0]+row, row+[0])]  
 return row  
  
**Best\_solution:**Here is my brief O(k) solution   
class Solution {  
public:  
 vector<int> getRow(int rowIndex) {  
 vector<int> A(rowIndex+1, 0);  
 A[0] = 1;  
 for(int i=1; i<rowIndex+1; i++)  
 for(int j=i; j>=1; j--)  
 A[j] += A[j-1];  
 return A;  
 }  
};  
  
*120,Triangle:***Python\_solution:**Python easy to understand solutions (top-down, bottom-up).   
# O(n\*n/2) space, top-down   
def minimumTotal1(self, triangle):  
 if not triangle:  
 return   
 res = [[0 for i in xrange(len(row))] for row in triangle]  
 res[0][0] = triangle[0][0]  
 for i in xrange(1, len(triangle)):  
 for j in xrange(len(triangle[i])):  
 if j == 0:  
 res[i][j] = res[i-1][j] + triangle[i][j]  
 elif j == len(triangle[i])-1:  
 res[i][j] = res[i-1][j-1] + triangle[i][j]  
 else:  
 res[i][j] = min(res[i-1][j-1], res[i-1][j]) + triangle[i][j]  
 return min(res[-1])  
   
# Modify the original triangle, top-down  
def minimumTotal2(self, triangle):  
 if not triangle:  
 return   
 for i in xrange(1, len(triangle)):  
 for j in xrange(len(triangle[i])):  
 if j == 0:  
 triangle[i][j] += triangle[i-1][j]  
 elif j == len(triangle[i])-1:  
 triangle[i][j] += triangle[i-1][j-1]  
 else:  
 triangle[i][j] += min(triangle[i-1][j-1], triangle[i-1][j])  
 return min(triangle[-1])  
   
# Modify the original triangle, bottom-up  
def minimumTotal3(self, triangle):  
 if not triangle:  
 return   
 for i in xrange(len(triangle)-2, -1, -1):  
 for j in xrange(len(triangle[i])):  
 triangle[i][j] += min(triangle[i+1][j], triangle[i+1][j+1])  
 return triangle[0][0]  
  
# bottom-up, O(n) space  
def minimumTotal(self, triangle):  
 if not triangle:  
 return   
 res = triangle[-1]  
 for i in xrange(len(triangle)-2, -1, -1):  
 for j in xrange(len(triangle[i])):  
 res[j] = min(res[j], res[j+1]) + triangle[i][j]  
 return res[0]  
**Best\_solution:**DP Solution for Triangle   
minpath[k][i] = min( minpath[k+1][i], minpath[k+1][i+1]) + triangle[k][i];  
  
  
*121,Best Time to Buy and Sell Stock:***Python\_solution:**Easy O(n) Python solution   
def maxProfit(prices):  
 max\_profit, min\_price = 0, float('inf')  
 for price in prices:  
 min\_price = min(min\_price, price)  
 profit = price - min\_price  
 max\_profit = max(max\_profit, profit)  
 return max\_profit  
**Best\_solution:**Kadane's Algorithm - Since no one has mentioned about this so far :) (In case if interviewer twists the input)   
Kadane's Algorithm  
  
*122,Best Time to Buy and Sell Stock II:***Python\_solution:**Clear 1-line Python Solution   
class Solution(object):  
 def maxProfit(self, prices):  
 return sum(max(prices[i + 1] - prices[i], 0) for i in range(len(prices) - 1))  
**Best\_solution:**Is this question a joke?   
public class Solution {  
public int maxProfit(int[] prices) {  
 int total = 0;  
 for (int i=0; i< prices.length-1; i++) {  
 if (prices[i+1]>prices[i]) total += prices[i+1]-prices[i];  
 }  
   
 return total;  
}  
  
  
*123,Best Time to Buy and Sell Stock III:***Best\_solution:**Is it Best Solution with O(n), O(1).   
public class Solution {  
 public int maxProfit(int[] prices) {  
 int hold1 = Integer.MIN\_VALUE, hold2 = Integer.MIN\_VALUE;  
 int release1 = 0, release2 = 0;  
 for(int i:prices){ // Assume we only have 0 money at first  
 release2 = Math.max(release2, hold2+i); // The maximum if we've just sold 2nd stock so far.  
 hold2 = Math.max(hold2, release1-i); // The maximum if we've just buy 2nd stock so far.  
 release1 = Math.max(release1, hold1+i); // The maximum if we've just sold 1nd stock so far.  
 hold1 = Math.max(hold1, -i); // The maximum if we've just buy 1st stock so far.   
 }  
 return release2; ///Since release1 is initiated as 0, so release2 will always higher than release1.  
 }  
}  
  
*124,Binary Tree Maximum Path Sum:***Python\_solution:**12 lines of Python code, fast and easy to understand   
class Solution(object):  
 def maxPathSum(self, root):  
 def dfs(node): # returns: max one side path sum, max path sum  
 l = r = 0  
 ls = rs = None  
 if node.left:  
 l, ls = dfs(node.left)  
 l = max(l, 0)  
 if node.right:  
 r, rs = dfs(node.right)  
 r = max(r, 0)  
 return node.val + max(l, r), max(node.val + l + r, ls, rs)  
 if root:  
 return dfs(root)[1]  
 return 0  
**Best\_solution:**Accepted short solution in Java   
maxPathDown(TreeNode node)  
  
*125,Valid Palindrome:***Python\_solution:**Python in-place two-pointer solution.   
def isPalindrome(self, s):  
 l, r = 0, len(s)-1  
 while l < r:  
 while l < r and not s[l].isalnum():  
 l += 1  
 while l <r and not s[r].isalnum():  
 r -= 1  
 if s[l].lower() != s[r].lower():  
 return False  
 l +=1; r -= 1  
 return True  
**Best\_solution:**Accepted pretty Java solution(271ms)   
public class Solution {  
 public boolean isPalindrome(String s) {  
 if (s.isEmpty()) {  
 return true;  
 }  
 int head = 0, tail = s.length() - 1;  
 char cHead, cTail;  
 while(head <= tail) {  
 cHead = s.charAt(head);  
 cTail = s.charAt(tail);  
 if (!Character.isLetterOrDigit(cHead)) {  
 head++;  
 } else if(!Character.isLetterOrDigit(cTail)) {  
 tail--;  
 } else {  
 if (Character.toLowerCase(cHead) != Character.toLowerCase(cTail)) {  
 return false;  
 }  
 head++;  
 tail--;  
 }  
 }  
   
 return true;  
 }  
}  
  
*126,Word Ladder II:***Python\_solution:**Use defaultdict for traceback and easy writing, 20 lines python code   
class Solution:  
# @param start, a string  
# @param end, a string  
# @param dict, a set of string  
# @return a list of lists of string  
def findLadders(self, start, end, dic):  
 dic.add(end)  
 level = {start}  
 parents = collections.defaultdict(set)  
 while level and end not in parents:  
 next\_level = collections.defaultdict(set)  
 for node in level:  
 for char in string.ascii\_lowercase:  
 for i in range(len(start)):  
 n = node[:i]+char+node[i+1:]  
 if n in dic and n not in parents:  
 next\_level[n].add(node)  
 level = next\_level  
 parents.update(next\_level)  
 res = [[end]]  
 while res and res[0][0] != start:  
 res = [[p]+r for r in res for p in parents[r[0]]]  
 return res  
  
**Best\_solution:**Share two similar Java solution that Accpted by OJ.   
public class Solution {  
 Map<String,List<String>> map;  
 List<List<String>> results;  
 public List<List<String>> findLadders(String start, String end, Set<String> dict) {   
 results= new ArrayList<List<String>>();  
 if (dict.size() == 0)  
 return results;  
   
 int min=Integer.MAX\_VALUE;  
   
 Queue<String> queue= new ArrayDeque<String>();  
 queue.add(start);  
   
 map = new HashMap<String,List<String>>();  
   
 Map<String,Integer> ladder = new HashMap<String,Integer>();  
 for (String string:dict)  
 ladder.put(string, Integer.MAX\_VALUE);  
 ladder.put(start, 0);  
   
 dict.add(end);  
 //BFS: Dijisktra search  
 while (!queue.isEmpty()) {  
   
 String word = queue.poll();  
   
 int step = ladder.get(word)+1;//'step' indicates how many steps are needed to travel to one word.   
   
 if (step>min) break;  
   
 for (int i = 0; i < word.length(); i++){  
 StringBuilder builder = new StringBuilder(word);   
 for (char ch='a'; ch <= 'z'; ch++){  
 builder.setCharAt(i,ch);  
 String new\_word=builder.toString();   
 if (ladder.containsKey(new\_word)) {  
   
 if (step>ladder.get(new\_word))//Check if it is the shortest path to one word.  
 continue;  
 else if (step<ladder.get(new\_word)){  
 queue.add(new\_word);  
 ladder.put(new\_word, step);  
 }else;// It is a KEY line. If one word already appeared in one ladder,  
 // Do not insert the same word inside the queue twice. Otherwise it gets TLE.  
   
 if (map.containsKey(new\_word)) //Build adjacent Graph  
 map.get(new\_word).add(word);  
 else{  
 List<String> list= new LinkedList<String>();  
 list.add(word);  
 map.put(new\_word,list);  
 //It is possible to write three lines in one:  
 //map.put(new\_word,new LinkedList<String>(Arrays.asList(new String[]{word})));  
 //Which one is better?  
 }  
   
 if (new\_word.equals(end))  
 min=step;  
  
 }//End if dict contains new\_word  
 }//End:Iteration from 'a' to 'z'  
 }//End:Iteration from the first to the last  
 }//End While  
  
 //BackTracking  
 LinkedList<String> result = new LinkedList<String>();  
 backTrace(end,start,result);  
  
 return results;   
 }  
 private void backTrace(String word,String start,List<String> list){  
 if (word.equals(start)){  
 list.add(0,start);  
 results.add(new ArrayList<String>(list));  
 list.remove(0);  
 return;  
 }  
 list.add(0,word);  
 if (map.get(word)!=null)  
 for (String s:map.get(word))  
 backTrace(s,start,list);  
 list.remove(0);  
 }  
}  
  
  
*127,Word Ladder:***Python\_solution:**Share my two Python solutions: a very concise one (12 lines, ~160ms) and an optimized solution(~100ms)   
class Solution:  
 # @param {string} beginWord  
 # @param {string} endWord  
 # @param {set<string>} wordDict  
 # @return {integer}  
 def ladderLength(self, beginWord, endWord, wordDict):  
 length = 2  
 front, back = set([beginWord]), set([endWord])  
 wordDict.discard(beginWord)  
 while front:  
 # generate all valid transformations  
 front = wordDict & (set(word[:index] + ch + word[index+1:] for word in front   
 for index in range(len(beginWord)) for ch in 'abcdefghijklmnopqrstuvwxyz'))  
 if front & back:  
 # there are common elements in front and back, done  
 return length  
 length += 1  
 if len(front) > len(back):  
 # swap front and back for better performance (fewer choices in generating nextSet)  
 front, back = back, front  
 # remove transformations from wordDict to avoid cycle  
 wordDict -= front  
 return 0  
  
**Best\_solution:**Easy 76ms C++ Solution using BFS   
start = "hit"  
  
*128,Longest Consecutive Sequence:***Python\_solution:**Python O(n) solution using sets   
class Solution:  
 # @param num, a list of integer  
 # @return an integer  
 def longestConsecutive(self, num):  
 num=set(num)  
 maxLen=0  
 while num:  
 n=num.pop()  
 i=n+1  
 l1=0  
 l2=0  
 while i in num:  
 num.remove(i)  
 i+=1  
 l1+=1  
 i=n-1  
 while i in num:  
 num.remove(i)  
 i-=1  
 l2+=1  
 maxLen=max(maxLen,l1+l2+1)  
 return maxLen  
**Best\_solution:**My really simple Java O(n) solution - Accepted   
public int longestConsecutive(int[] num) {  
 int res = 0;  
 HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();  
 for (int n : num) {  
 if (!map.containsKey(n)) {  
 int left = (map.containsKey(n - 1)) ? map.get(n - 1) : 0;  
 int right = (map.containsKey(n + 1)) ? map.get(n + 1) : 0;  
 // sum: length of the sequence n is in  
 int sum = left + right + 1;  
 map.put(n, sum);  
   
 // keep track of the max length   
 res = Math.max(res, sum);  
   
 // extend the length to the boundary(s)  
 // of the sequence  
 // will do nothing if n has no neighbors  
 map.put(n - left, sum);  
 map.put(n + right, sum);  
 }  
 else {  
 // duplicates  
 continue;  
 }  
 }  
 return res;  
}  
  
*129,Sum Root to Leaf Numbers:***Python\_solution:**Python solutions (dfs+stack, bfs+queue, dfs recursively).   
# dfs + stack  
def sumNumbers1(self, root):  
 if not root:  
 return 0  
 stack, res = [(root, root.val)], 0  
 while stack:  
 node, value = stack.pop()  
 if node:  
 if not node.left and not node.right:  
 res += value  
 if node.right:  
 stack.append((node.right, value\*10+node.right.val))  
 if node.left:  
 stack.append((node.left, value\*10+node.left.val))  
 return res  
   
# bfs + queue  
def sumNumbers2(self, root):  
 if not root:  
 return 0  
 queue, res = collections.deque([(root, root.val)]), 0  
 while queue:  
 node, value = queue.popleft()  
 if node:  
 if not node.left and not node.right:  
 res += value  
 if node.left:  
 queue.append((node.left, value\*10+node.left.val))  
 if node.right:  
 queue.append((node.right, value\*10+node.right.val))  
 return res  
   
# recursively   
def sumNumbers(self, root):  
 self.res = 0  
 self.dfs(root, 0)  
 return self.res  
   
def dfs(self, root, value):  
 if root:  
 #if not root.left and not root.right:  
 # self.res += value\*10 + root.val  
 self.dfs(root.left, value\*10+root.val)  
 #if not root.left and not root.right:  
 # self.res += value\*10 + root.val  
 self.dfs(root.right, value\*10+root.val)  
 if not root.left and not root.right:  
 self.res += value\*10 + root.val  
**Best\_solution:**Short Java solution. Recursion.   
public int sumNumbers(TreeNode root) {  
 return sum(root, 0);  
}  
  
public int sum(TreeNode n, int s){  
 if (n == null) return 0;  
 if (n.right == null && n.left == null) return s\*10 + n.val;  
 return sum(n.left, s\*10 + n.val) + sum(n.right, s\*10 + n.val);  
}  
  
*130,Surrounded Regions:***Python\_solution:**9 lines, Python 148 ms   
def solve(self, board):  
 if not any(board): return  
  
 m, n = len(board), len(board[0])  
 save = [ij for k in range(m+n) for ij in ((0, k), (m-1, k), (k, 0), (k, n-1))]  
 while save:  
 i, j = save.pop()  
 if 0 <= i < m and 0 <= j < n and board[i][j] == 'O':  
 board[i][j] = 'S'  
 save += (i, j-1), (i, j+1), (i-1, j), (i+1, j)  
  
 board[:] = [['XO'[c == 'S'] for c in row] for row in board]  
  
**Best\_solution:**A really simple and readable C++ solution，only cost 12ms   
 X X X X X X X X X X X X  
 X X O X -> X X O X -> X X X X  
 X O X X X 1 X X X O X X  
 X O X X X 1 X X X O X X  
   
  
class Solution {  
public:  
 void solve(vector<vector<char>>& board) {  
 int i,j;  
 int row=board.size();  
 if(!row)  
 return;  
 int col=board[0].size();  
  
 for(i=0;i<row;i++){  
 check(board,i,0,row,col);  
 if(col>1)  
 check(board,i,col-1,row,col);  
 }  
 for(j=1;j+1<col;j++){  
 check(board,0,j,row,col);  
 if(row>1)  
 check(board,row-1,j,row,col);  
 }  
 for(i=0;i<row;i++)  
 for(j=0;j<col;j++)  
 if(board[i][j]=='O')  
 board[i][j]='X';  
 for(i=0;i<row;i++)  
 for(j=0;j<col;j++)  
 if(board[i][j]=='1')  
 board[i][j]='O';  
 }  
 void check(vector<vector<char> >&vec,int i,int j,int row,int col){  
 if(vec[i][j]=='O'){  
 vec[i][j]='1';  
 if(i>1)  
 check(vec,i-1,j,row,col);  
 if(j>1)  
 check(vec,i,j-1,row,col);  
 if(i+1<row)  
 check(vec,i+1,j,row,col);  
 if(j+1<col)  
 check(vec,i,j+1,row,col);  
 }  
 }  
};  
  
*131,Palindrome Partitioning:***Python\_solution:**1-liner Python, Ruby   
def partition(self, s):  
 return [[s[:i]] + rest  
 for i in xrange(1, len(s)+1)  
 if s[:i] == s[i-1::-1]  
 for rest in self.partition(s[i:])] or [[]]  
  
**Best\_solution:**Java: Backtracking solution.   
public class Solution {  
 List<List<String>> resultLst;  
 ArrayList<String> currLst;  
 public List<List<String>> partition(String s) {  
 resultLst = new ArrayList<List<String>>();  
 currLst = new ArrayList<String>();  
 backTrack(s,0);  
 return resultLst;  
 }  
 public void backTrack(String s, int l){  
 if(currLst.size()>0 //the initial str could be palindrome  
 && l>=s.length()){  
 List<String> r = (ArrayList<String>) currLst.clone();  
 resultLst.add(r);  
 }  
 for(int i=l;i<s.length();i++){  
 if(isPalindrome(s,l,i)){  
 if(l==i)  
 currLst.add(Character.toString(s.charAt(i)));  
 else  
 currLst.add(s.substring(l,i+1));  
 backTrack(s,i+1);  
 currLst.remove(currLst.size()-1);  
 }  
 }  
 }  
 public boolean isPalindrome(String str, int l, int r){  
 if(l==r) return true;  
 while(l<r){  
 if(str.charAt(l)!=str.charAt(r)) return false;  
 l++;r--;  
 }  
 return true;  
 }  
}  
  
  
*132,Palindrome Partitioning II:***Python\_solution:**56 ms python with explanation   
def minCut(self, s):  
 # acceleration  
 if s == s[::-1]: return 0  
 for i in range(1, len(s)):  
 if s[:i] == s[:i][::-1] and s[i:] == s[i:][::-1]:  
 return 1  
 # algorithm  
 cut = [x for x in range(-1,len(s))] # cut numbers in worst case (no palindrome)  
 for i in range(len(s)):  
 r1, r2 = 0, 0  
 # use i as origin, and gradually enlarge radius if a palindrome exists  
 # odd palindrome  
 while i-r1 >= 0 and i+r1 < len(s) and s[i-r1] == s[i+r1]:  
 cut[i+r1+1] = min(cut[i+r1+1], cut[i-r1]+1)  
 r1 += 1  
 # even palindrome  
 while i-r2 >= 0 and i+r2+1 < len(s) and s[i-r2] == s[i+r2+1]:  
 cut[i+r2+2] = min(cut[i+r2+2], cut[i-r2]+1)  
 r2 += 1  
 return cut[-1]  
  
**Best\_solution:**My solution does not need a table for palindrome, is it right ? It uses only O(n) space.   
class Solution {  
public:  
 int minCut(string s) {  
 int n = s.size();  
 vector<int> cut(n+1, 0); // number of cuts for the first k characters  
 for (int i = 0; i <= n; i++) cut[i] = i-1;  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; i-j >= 0 && i+j < n && s[i-j]==s[i+j] ; j++) // odd length palindrome  
 cut[i+j+1] = min(cut[i+j+1],1+cut[i-j]);  
  
 for (int j = 1; i-j+1 >= 0 && i+j < n && s[i-j+1] == s[i+j]; j++) // even length palindrome  
 cut[i+j+1] = min(cut[i+j+1],1+cut[i-j+1]);  
 }  
 return cut[n];  
 }  
};  
  
*133,Clone Graph:***Python\_solution:**Python DFS short solution   
def cloneGraph(self, node):  
 if not node:  
 return node  
 root = UndirectedGraphNode(node.label)  
 stack = [node]  
 visit = {}  
 visit[node.label] = root  
 while stack:  
 top = stack.pop()  
   
 for n in top.neighbors:  
 if n.label not in visit:  
 stack.append(n)  
 visit[n.label] = UndirectedGraphNode(n.label)  
 visit[top.label].neighbors.append(visit[n.label])  
   
 return root  
**Best\_solution:**Depth First Simple Java Solution   
public class Solution {  
 private HashMap<Integer, UndirectedGraphNode> map = new HashMap<>();  
 public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {  
 return clone(node);  
 }  
  
 private UndirectedGraphNode clone(UndirectedGraphNode node) {  
 if (node == null) return null;  
   
 if (map.containsKey(node.label)) {  
 return map.get(node.label);  
 }  
 UndirectedGraphNode clone = new UndirectedGraphNode(node.label);  
 map.put(clone.label, clone);  
 for (UndirectedGraphNode neighbor : node.neighbors) {  
 clone.neighbors.add(clone(neighbor));  
 }  
 return clone;  
 }  
}  
  
*134,Gas Station:***Python\_solution:**Possibly the MOST easiest approach, O(N), one variable, Python   
def canCompleteCircuit(self, gas, cost):  
 """  
 :type gas: List[int]  
 :type cost: List[int]  
 :rtype: int  
 """  
 if len(gas) == 0 or len(cost) == 0 or sum(gas) < sum(cost):  
 return -1  
 position = 0  
 balance = 0 # current tank balance  
 for i in range(len(gas)):  
 balance += gas[i] - cost[i] # update balance  
 if balance < 0: # balance drops to negative, reset the start position  
 balance = 0  
 position = i+1  
 return position  
**Best\_solution:**Share some of my ideas.   
class Solution {  
public:  
 int canCompleteCircuit(vector<int> &gas, vector<int> &cost) {  
 int start(0),total(0),tank(0);  
 //if car fails at 'start', record the next station  
 for(int i=0;i<gas.size();i++) if((tank=tank+gas[i]-cost[i])<0) {start=i+1;total+=tank;tank=0;}  
 return (total+tank<0)? -1:start;  
 }  
};  
  
*135,Candy:***Best\_solution:**A simple solution   
 int candy(vector<int> &ratings)  
 {  
 int size=ratings.size();  
 if(size<=1)  
 return size;  
 vector<int> num(size,1);  
 for (int i = 1; i < size; i++)  
 {  
 if(ratings[i]>ratings[i-1])  
 num[i]=num[i-1]+1;  
 }  
 for (int i= size-1; i>0 ; i--)  
 {  
 if(ratings[i-1]>ratings[i])  
 num[i-1]=max(num[i]+1,num[i-1]);  
 }  
 int result=0;  
 for (int i = 0; i < size; i++)  
 {  
 result+=num[i];  
 // cout<<num[i]<<" ";  
 }  
 return result;  
 }  
  
*136,Single Number:***Python\_solution:**Python different solutions.   
def singleNumber1(self, nums):  
 dic = {}  
 for num in nums:  
 dic[num] = dic.get(num, 0)+1  
 for key, val in dic.items():  
 if val == 1:  
 return key  
  
def singleNumber2(self, nums):  
 res = 0  
 for num in nums:  
 res ^= num  
 return res  
   
def singleNumber3(self, nums):  
 return 2\*sum(set(nums))-sum(nums)  
   
def singleNumber4(self, nums):  
 return reduce(lambda x, y: x ^ y, nums)  
   
def singleNumber(self, nums):  
 return reduce(operator.xor, nums)  
**Best\_solution:**My O(n) solution using XOR   
int singleNumber(int A[], int n) {  
 int result = 0;  
 for (int i = 0; i<n; i++)  
 {  
 result ^=A[i];  
 }  
 return result;  
}  
  
  
*137,Single Number II:***Python\_solution:**Python bitwise solution   
class Solution:  
# @param A, a list of integer  
# @return an integer  
def singleNumber(self, A):  
 ans = 0  
 for i in xrange(0,32):  
 count = 0  
 for a in A:  
 if ((a >> i) & 1):  
 count+=1  
 ans |= ((count%3) << i)  
 return self.convert(ans)  
   
def convert(self,x):  
 if x >= 2\*\*31:  
 x -= 2\*\*32  
 return x  
**Best\_solution:**Challenge me , thx   
public int singleNumber(int[] A) {  
 int ones = 0, twos = 0;  
 for(int i = 0; i < A.length; i++){  
 ones = (ones ^ A[i]) & ~twos;  
 twos = (twos ^ A[i]) & ~ones;  
 }  
 return ones;  
}  
  
*138,Copy List with Random Pointer:***Python\_solution:**Clear and short python O(2n) and O(n) solution   
class Solution:  
# @param head, a RandomListNode  
# @return a RandomListNode  
def copyRandomList(self, head):  
 dic = dict()  
 m = n = head  
 while m:  
 dic[m] = RandomListNode(m.label)  
 m = m.next  
 while n:  
 dic[n].next = dic.get(n.next)  
 dic[n].random = dic.get(n.random)  
 n = n.next  
 return dic.get(head)  
  
**Best\_solution:**A solution with constant space complexity O(1) and linear time complexity O(N)   
O(N)  
  
*139,Word Break:***Python\_solution:**Simple DP solution in Python with description   
def word\_break(s, words):  
 d = [False] \* len(s)   
 for i in range(len(s)):  
 for w in words:  
 if w == s[i-len(w)+1:i+1] and (d[i-len(w)] or i-len(w) == -1):  
 d[i] = True  
 return d[-1]  
**Best\_solution:**Java implementation using DP in two ways   
public class Solution {  
 public boolean wordBreak(String s, Set<String> dict) {  
   
 boolean[] f = new boolean[s.length() + 1];  
   
 f[0] = true;  
   
   
 /\* First DP  
 for(int i = 1; i <= s.length(); i++){  
 for(String str: dict){  
 if(str.length() <= i){  
 if(f[i - str.length()]){  
 if(s.substring(i-str.length(), i).equals(str)){  
 f[i] = true;  
 break;  
 }  
 }  
 }  
 }  
 }\*/  
   
 //Second DP  
 for(int i=1; i <= s.length(); i++){  
 for(int j=0; j < i; j++){  
 if(f[j] && dict.contains(s.substring(j, i))){  
 f[i] = true;  
 break;  
 }  
 }  
 }  
   
 return f[s.length()];  
 }  
}  
  
*140,Word Break II:***Python\_solution:**9 lines Python, 10 lines C++   
def wordBreak(self, s, wordDict):  
 memo = {len(s): ['']}  
 def sentences(i):  
 if i not in memo:  
 memo[i] = [s[i:j] + (tail and ' ' + tail)  
 for j in range(i+1, len(s)+1)  
 if s[i:j] in wordDict  
 for tail in sentences(j)]  
 return memo[i]  
 return sentences(0)  
  
**Best\_solution:**My concise JAVA solution based on memorized DFS   
public List<String> wordBreak(String s, Set<String> wordDict) {  
 return DFS(s, wordDict, new HashMap<String, LinkedList<String>>());  
}   
  
// DFS function returns an array including all substrings derived from s.  
List<String> DFS(String s, Set<String> wordDict, HashMap<String, LinkedList<String>>map) {  
 if (map.containsKey(s))   
 return map.get(s);  
   
 LinkedList<String>res = new LinkedList<String>();   
 if (s.length() == 0) {  
 res.add("");  
 return res;  
 }   
 for (String word : wordDict) {  
 if (s.startsWith(word)) {  
 List<String>sublist = DFS(s.substring(word.length()), wordDict, map);  
 for (String sub : sublist)   
 res.add(word + (sub.isEmpty() ? "" : " ") + sub);   
 }  
 }   
 map.put(s, res);  
 return res;  
}  
  
*141,Linked List Cycle:***Python\_solution:**Except-ionally fast Python   
def hasCycle(self, head):  
 try:  
 slow = head  
 fast = head.next  
 while slow is not fast:  
 slow = slow.next  
 fast = fast.next.next  
 return True  
 except:  
 return False  
**Best\_solution:**O(1) Space Solution   
public boolean hasCycle(ListNode head) {  
 if(head==null) return false;  
 ListNode walker = head;  
 ListNode runner = head;  
 while(runner.next!=null && runner.next.next!=null) {  
 walker = walker.next;  
 runner = runner.next.next;  
 if(walker==runner) return true;  
 }  
 return false;  
}  
  
  
*142,Linked List Cycle II:***Python\_solution:**Share my python solution with detailed explanation   
 Consider the following linked list, where E is the cylce entry and X, the crossing point of fast and slow.  
 H: distance from head to cycle entry E  
 D: distance from E to X  
 L: cycle length  
 \_\_\_\_\_  
 / \  
 head\_\_\_\_\_H\_\_\_\_\_\_E \  
 \ /  
 X\_\_\_\_\_/   
   
   
 If fast and slow both start at head, when fast catches slow, slow has traveled H+D and fast 2(H+D).   
 Assume fast has traveled n loops in the cycle, we have:  
 2H + 2D = H + D + L --> H + D = nL --> H = nL - D  
 Thus if two pointers start from head and X, respectively, one first reaches E, the other also reaches E.   
 In my solution, since fast starts at head.next, we need to move slow one step forward in the beginning of part 2  
  
class Solution:  
 # @param head, a ListNode  
 # @return a list node  
 def detectCycle(self, head):  
 try:  
 fast = head.next  
 slow = head  
 while fast is not slow:  
 fast = fast.next.next  
 slow = slow.next  
 except:  
 # if there is an exception, we reach the end and there is no cycle  
 return None  
  
 # since fast starts at head.next, we need to move slow one step forward  
 slow = slow.next  
 while head is not slow:  
 head = head.next  
 slow = slow.next  
  
 return head  
  
**Best\_solution:**O(n) solution by using two pointers without change anything   
 ListNode \*detectCycle(ListNode \*head) {  
 if (head == NULL || head->next == NULL) return NULL;  
   
 ListNode\* firstp = head;  
 ListNode\* secondp = head;  
 bool isCycle = false;  
   
 while(firstp != NULL && secondp != NULL) {  
 firstp = firstp->next;  
 if (secondp->next == NULL) return NULL;  
 secondp = secondp->next->next;  
 if (firstp == secondp) { isCycle = true; break; }  
 }  
   
 if(!isCycle) return NULL;  
 firstp = head;  
 while( firstp != secondp) {  
 firstp = firstp->next;  
 secondp = secondp->next;  
 }  
  
 return firstp;  
}  
  
*143,Reorder List:***Python\_solution:**A python solution O(n) time, O(1) space   
# Splits in place a list in two halves, the first half is >= in size than the second.  
# @return A tuple containing the heads of the two halves  
def \_splitList(head):  
 fast = head  
 slow = head  
 while fast and fast.next:  
 slow = slow.next  
 fast = fast.next  
 fast = fast.next  
  
 middle = slow.next  
 slow.next = None  
  
 return head, middle  
  
# Reverses in place a list.  
# @return Returns the head of the new reversed list  
def \_reverseList(head):  
  
 last = None  
 currentNode = head  
  
 while currentNode:  
 nextNode = currentNode.next  
 currentNode.next = last  
 last = currentNode  
 currentNode = nextNode  
  
 return last  
  
# Merges in place two lists  
# @return The newly merged list.  
def \_mergeLists(a, b):  
  
 tail = a  
 head = a  
  
 a = a.next  
 while b:  
 tail.next = b  
 tail = tail.next  
 b = b.next  
 if a:  
 a, b = b, a  
   
 return head  
  
  
class Solution:  
  
 # @param head, a ListNode  
 # @return nothing  
 def reorderList(self, head):  
  
 if not head or not head.next:  
 return  
  
 a, b = \_splitList(head)  
 b = \_reverseList(b)  
 head = \_mergeLists(a, b)  
**Best\_solution:**Java solution with 3 steps   
public void reorderList(ListNode head) {  
 if(head==null||head.next==null) return;  
   
 //Find the middle of the list  
 ListNode p1=head;  
 ListNode p2=head;  
 while(p2.next!=null&&p2.next.next!=null){   
 p1=p1.next;  
 p2=p2.next.next;  
 }  
   
 //Reverse the half after middle 1->2->3->4->5->6 to 1->2->3->6->5->4  
 ListNode preMiddle=p1;  
 ListNode preCurrent=p1.next;  
 while(preCurrent.next!=null){  
 ListNode current=preCurrent.next;  
 preCurrent.next=current.next;  
 current.next=preMiddle.next;  
 preMiddle.next=current;  
 }  
   
 //Start reorder one by one 1->2->3->6->5->4 to 1->6->2->5->3->4  
 p1=head;  
 p2=preMiddle.next;  
 while(p1!=preMiddle){  
 preMiddle.next=p2.next;  
 p2.next=p1.next;  
 p1.next=p2;  
 p1=p2.next;  
 p2=preMiddle.next;  
 }  
 }  
  
*144,Binary Tree Preorder Traversal:***Python\_solution:**Very simple iterative Python solution   
def preorderTraversal(self, root):  
 ret = []  
 stack = [root]  
 while stack:  
 node = stack.pop()  
 if node:  
 ret.append(node.val)  
 stack.append(node.right)  
 stack.append(node.left)  
 return ret  
**Best\_solution:**Accepted iterative solution in Java using stack.   
public List<Integer> preorderTraversal(TreeNode node) {  
 List<Integer> list = new LinkedList<Integer>();  
 Stack<TreeNode> rights = new Stack<TreeNode>();  
 while(node != null) {  
 list.add(node.val);  
 if (node.right != null) {  
 rights.push(node.right);  
 }  
 node = node.left;  
 if (node == null && !rights.isEmpty()) {  
 node = rights.pop();  
 }  
 }  
 return list;  
}  
  
*145,Binary Tree Postorder Traversal:***Python\_solution:**Share my two Python iterative solutions, post-order and modified preorder then reverse   
class Solution:  
 # @param {TreeNode} root  
 # @return {integer[]}  
 def postorderTraversal(self, root):  
 traversal, stack = [], [(root, False)]  
 while stack:  
 node, visited = stack.pop()  
 if node:  
 if visited:  
 # add to result if visited  
 traversal.append(node.val)  
 else:  
 # post-order  
 stack.append((node, True))  
 stack.append((node.right, False))  
 stack.append((node.left, False))  
  
 return traversal  
  
**Best\_solution:**Preorder, Inorder, and Postorder Iteratively Summarization   
public List<Integer> preorderTraversal(TreeNode root) {  
 List<Integer> result = new ArrayList<>();  
 Deque<TreeNode> stack = new ArrayDeque<>();  
 TreeNode p = root;  
 while(!stack.isEmpty() || p != null) {  
 if(p != null) {  
 stack.push(p);  
 result.add(p.val); // Add before going to children  
 p = p.left;  
 } else {  
 TreeNode node = stack.pop();  
 p = node.right;   
 }  
 }  
 return result;  
}  
  
  
*146,LRU Cache:***Python\_solution:**Python Dict + Double LinkedList   
class Node:  
def \_\_init\_\_(self, k, v):  
 self.key = k  
 self.val = v  
 self.prev = None  
 self.next = None  
  
class LRUCache:  
def \_\_init\_\_(self, capacity):  
 self.capacity = capacity  
 self.dic = dict()  
 self.head = Node(0, 0)  
 self.tail = Node(0, 0)  
 self.head.next = self.tail  
 self.tail.prev = self.head  
  
def get(self, key):  
 if key in self.dic:  
 n = self.dic[key]  
 self.\_remove(n)  
 self.\_add(n)  
 return n.val  
 return -1  
  
def set(self, key, value):  
 if key in self.dic:  
 self.\_remove(self.dic[key])  
 n = Node(key, value)  
 self.\_add(n)  
 self.dic[key] = n  
 if len(self.dic) > self.capacity:  
 n = self.head.next  
 self.\_remove(n)  
 del self.dic[n.key]  
  
def \_remove(self, node):  
 p = node.prev  
 n = node.next  
 p.next = n  
 n.prev = p  
  
def \_add(self, node):  
 p = self.tail.prev  
 p.next = node  
 self.tail.prev = node  
 node.prev = p  
 node.next = self.tail  
**Best\_solution:**[Java] Hashtable + Double linked list (with a touch of pseudo nodes)   
class DLinkedNode {  
 int key;  
 int value;  
 DLinkedNode pre;  
 DLinkedNode post;  
}  
  
/\*\*  
 \* Always add the new node right after head;  
 \*/  
private void addNode(DLinkedNode node){  
 node.pre = head;  
 node.post = head.post;  
   
 head.post.pre = node;  
 head.post = node;  
}  
  
/\*\*  
 \* Remove an existing node from the linked list.  
 \*/  
private void removeNode(DLinkedNode node){  
 DLinkedNode pre = node.pre;  
 DLinkedNode post = node.post;  
   
 pre.post = post;  
 post.pre = pre;  
}  
  
/\*\*  
 \* Move certain node in between to the head.  
 \*/  
private void moveToHead(DLinkedNode node){  
 this.removeNode(node);  
 this.addNode(node);  
}  
  
// pop the current tail.   
private DLinkedNode popTail(){  
 DLinkedNode res = tail.pre;  
 this.removeNode(res);  
 return res;  
}  
  
private Hashtable<Integer, DLinkedNode>   
 cache = new Hashtable<Integer, DLinkedNode>();  
private int count;  
private int capacity;  
private DLinkedNode head, tail;  
  
public LRUCache(int capacity) {  
 this.count = 0;  
 this.capacity = capacity;  
  
 head = new DLinkedNode();  
 head.pre = null;  
   
 tail = new DLinkedNode();  
 tail.post = null;  
   
 head.post = tail;  
 tail.pre = head;  
}  
  
public int get(int key) {  
   
 DLinkedNode node = cache.get(key);  
 if(node == null){  
 return -1; // should raise exception here.  
 }  
   
 // move the accessed node to the head;  
 this.moveToHead(node);  
   
 return node.value;  
}  
  
  
public void set(int key, int value) {  
 DLinkedNode node = cache.get(key);  
   
 if(node == null){  
   
 DLinkedNode newNode = new DLinkedNode();  
 newNode.key = key;  
 newNode.value = value;  
   
 this.cache.put(key, newNode);  
 this.addNode(newNode);  
   
 ++count;  
   
 if(count > capacity){  
 // pop the tail  
 DLinkedNode tail = this.popTail();  
 this.cache.remove(tail.key);  
 --count;  
 }  
 }else{  
 // update the value.  
 node.value = value;  
 this.moveToHead(node);  
 }  
   
}  
  
*147,Insertion Sort List:***Python\_solution:**Python time limit is too tight   
class Solution:  
# @param head, a ListNode  
# @return a ListNode  
def insertionSortList(self, head):  
 srt = None  
 while head:  
 node = head  
 head = head.next  
 node.next = None  
 srt = self.insertTo(srt, node)  
 return srt  
   
def insertTo(self, head, node):  
 node.next = head  
 head = node  
 while node.next and node.val > node.next.val:  
 node.val, node.next.val = node.next.val, node.val  
 node = node.next  
 return head  
  
**Best\_solution:**An easy and clear way to sort ( O(1) space )   
public ListNode insertionSortList(ListNode head) {  
 if( head == null ){  
 return head;  
 }  
   
 ListNode helper = new ListNode(0); //new starter of the sorted list  
 ListNode cur = head; //the node will be inserted  
 ListNode pre = helper; //insert node between pre and pre.next  
 ListNode next = null; //the next node will be inserted  
 //not the end of input list  
 while( cur != null ){  
 next = cur.next;  
 //find the right place to insert  
 while( pre.next != null && pre.next.val < cur.val ){  
 pre = pre.next;  
 }  
 //insert between pre and pre.next  
 cur.next = pre.next;  
 pre.next = cur;  
 pre = helper;  
 cur = next;  
 }  
   
 return helper.next;  
 }  
  
*148,Sort List:***Python\_solution:**Clean python code   
class Solution(object):  
 def merge(self, h1, h2):  
 dummy = tail = ListNode(None)  
 while h1 and h2:  
 if h1.val < h2.val:  
 tail.next, tail, h1 = h1, h1, h1.next  
 else:  
 tail.next, tail, h2 = h2, h2, h2.next  
   
 tail.next = h1 or h2  
 return dummy.next  
   
 def sortList(self, head):  
 if not head or not head.next:  
 return head  
   
 pre, slow, fast = None, head, head  
 while fast and fast.next:  
 pre, slow, fast = slow, slow.next, fast.next.next  
 pre.next = None  
  
 return self.merge(\*map(self.sortList, (head, slow)))  
**Best\_solution:**Java merge sort solution   
public class Solution {  
   
 public ListNode sortList(ListNode head) {  
 if (head == null || head.next == null)  
 return head;  
   
 // step 1. cut the list to two halves  
 ListNode prev = null, slow = head, fast = head;  
   
 while (fast != null && fast.next != null) {  
 prev = slow;  
 slow = slow.next;  
 fast = fast.next.next;  
 }  
   
 prev.next = null;  
   
 // step 2. sort each half  
 ListNode l1 = sortList(head);  
 ListNode l2 = sortList(slow);  
   
 // step 3. merge l1 and l2  
 return merge(l1, l2);  
 }  
   
 ListNode merge(ListNode l1, ListNode l2) {  
 ListNode l = new ListNode(0), p = l;  
   
 while (l1 != null && l2 != null) {  
 if (l1.val < l2.val) {  
 p.next = l1;  
 l1 = l1.next;  
 } else {  
 p.next = l2;  
 l2 = l2.next;  
 }  
 p = p.next;  
 }  
   
 if (l1 != null)  
 p.next = l1;  
   
 if (l2 != null)  
 p.next = l2;  
   
 return l.next;  
 }  
  
}  
  
*149,Max Points on a Line:***Python\_solution:**Python 68 ms code   
 def maxPoints(self, points):  
 l = len(points)  
 m = 0  
 for i in range(l):  
 dic = {'i': 1}  
 same = 0  
 for j in range(i+1, l):  
 tx, ty = points[j].x, points[j].y  
 if tx == points[i].x and ty == points[i].y:   
 same += 1  
 continue  
 if points[i].x == tx: slope = 'i'  
 else:slope = (points[i].y-ty) \* 1.0 /(points[i].x-tx)  
 if slope not in dic: dic[slope] = 1  
 dic[slope] += 1  
 m = max(m, max(dic.values()) + same)  
 return m  
**Best\_solution:**A java solution with notes   
 /\*  
 \* A line is determined by two factors,say y=ax+b  
 \*   
 \* If two points(x1,y1) (x2,y2) are on the same line(Of course).   
  
 \* Consider the gap between two points.  
  
 \* We have (y2-y1)=a(x2-x1),a=(y2-y1)/(x2-x1) a is a rational, b is canceled since b is a constant  
  
 \* If a third point (x3,y3) are on the same line. So we must have y3=ax3+b  
  
 \* Thus,(y3-y1)/(x3-x1)=(y2-y1)/(x2-x1)=a  
  
 \* Since a is a rational, there exists y0 and x0, y0/x0=(y3-y1)/(x3-x1)=(y2-y1)/(x2-x1)=a  
  
 \* So we can use y0&x0 to track a line;  
 \*/  
   
 public class Solution{  
 public int maxPoints(Point[] points) {  
 if (points==null) return 0;  
 if (points.length<=2) return points.length;  
   
 Map<Integer,Map<Integer,Integer>> map = new HashMap<Integer,Map<Integer,Integer>>();  
 int result=0;  
 for (int i=0;i<points.length;i++){   
 map.clear();  
 int overlap=0,max=0;  
 for (int j=i+1;j<points.length;j++){  
 int x=points[j].x-points[i].x;  
 int y=points[j].y-points[i].y;  
 if (x==0&&y==0){  
 overlap++;  
 continue;  
 }  
 int gcd=generateGCD(x,y);  
 if (gcd!=0){  
 x/=gcd;  
 y/=gcd;  
 }  
   
 if (map.containsKey(x)){  
 if (map.get(x).containsKey(y)){  
 map.get(x).put(y, map.get(x).get(y)+1);  
 }else{  
 map.get(x).put(y, 1);  
 }   
 }else{  
 Map<Integer,Integer> m = new HashMap<Integer,Integer>();  
 m.put(y, 1);  
 map.put(x, m);  
 }  
 max=Math.max(max, map.get(x).get(y));  
 }  
 result=Math.max(result, max+overlap+1);  
 }  
 return result;  
   
   
 }  
 private int generateGCD(int a,int b){  
   
 if (b==0) return a;  
 else return generateGCD(b,a%b);  
   
 }  
 }  
  
*150,Evaluate Reverse Polish Notation:***Python\_solution:**Python solution with comments (don't use eval() function).   
def evalRPN(self, tokens):  
 stack = []  
 for t in tokens:  
 if t not in ["+", "-", "\*", "/"]:  
 stack.append(int(t))  
 else:  
 r, l = stack.pop(), stack.pop()  
 if t == "+":  
 stack.append(l+r)  
 elif t == "-":  
 stack.append(l-r)  
 elif t == "\*":  
 stack.append(l\*r)  
 else:  
 # here take care of the case like "1/-22",  
 # in Python 2.x, it returns -1, while in   
 # Leetcode it should return 0  
 if l\*r < 0 and l % r != 0:  
 stack.append(l/r+1)  
 else:  
 stack.append(l/r)  
 return stack.pop()  
**Best\_solution:**6/（-132）= 0 or -1   
if a % b != 0 and a / b < 0: temp = a / b + 1 else: temp = a / b  
  
*151,Reverse Words in a String:***Python\_solution:**My Accept Answer of Python with one line   
class Solution:  
# @param s, a string  
# @return a string  
def reverseWords(self, s):  
 return " ".join(s.strip().split()[::-1])  
**Best\_solution:**In place simple solution   
void reverseWords(string &s) {  
 reverse(s.begin(), s.end());  
 int storeIndex = 0;  
 for (int i = 0; i < s.size(); i++) {  
 if (s[i] != ' ') {  
 if (storeIndex != 0) s[storeIndex++] = ' ';  
 int j = i;  
 while (j < s.size() && s[j] != ' ') { s[storeIndex++] = s[j++]; }  
 reverse(s.begin() + storeIndex - (j - i), s.begin() + storeIndex);  
 i = j;  
 }  
 }  
 s.erase(s.begin() + storeIndex, s.end());  
}  
  
*152,Maximum Product Subarray:***Python\_solution:**In Python, can it be more concise?   
def maxProduct(nums):  
 maximum=big=small=nums[0]  
 for n in nums[1:]:  
 big, small=max(n, n\*big, n\*small), min(n, n\*big, n\*small)  
 maximum=max(maximum, big)  
 return maximum  
**Best\_solution:**Possibly simplest solution with O(n) time complexity   
int maxProduct(int A[], int n) {  
 // store the result that is the max we have found so far  
 int r = A[0];  
  
 // imax/imin stores the max/min product of  
 // subarray that ends with the current number A[i]  
 for (int i = 1, imax = r, imin = r; i < n; i++) {  
 // multiplied by a negative makes big number smaller, small number bigger  
 // so we redefine the extremums by swapping them  
 if (A[i] < 0)  
 swap(imax, imin);  
  
 // max/min product for the current number is either the current number itself  
 // or the max/min by the previous number times the current one  
 imax = max(A[i], imax \* A[i]);  
 imin = min(A[i], imin \* A[i]);  
  
 // the newly computed max value is a candidate for our global result  
 r = max(r, imax);  
 }  
 return r;  
}  
  
  
*153,Find Minimum in Rotated Sorted Array:***Python\_solution:**9-line python clean code   
class Solution(object):  
 def findMin(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 i = 0  
 j = len(nums) - 1  
 while i < j:  
 m = i + (j - i) / 2  
 if nums[m] > nums[j]:  
 i = m + 1  
 else:  
 j = m  
 return nums[i]  
**Best\_solution:**Compact and clean C++ solution   
 int findMin(vector<int> &num) {  
 int start=0,end=num.size()-1;  
   
 while (start<end) {  
 if (num[start]<num[end])  
 return num[start];  
   
 int mid = (start+end)/2;  
   
 if (num[mid]>=num[start]) {  
 start = mid+1;  
 } else {  
 end = mid;  
 }  
 }  
   
 return num[start];  
 }  
  
  
*154,Find Minimum in Rotated Sorted Array II:***Python\_solution:**Python solution. Worst case O(N)   
def findMin(self, nums):  
 beg = 0  
 end = len(nums)-1  
 while beg <= end:  
 while beg < end and nums[beg] == nums[beg + 1]:  
 beg += 1  
 while end > beg and nums[end] == nums[end - 1]:  
 end -= 1  
 if beg == end:  
 return nums[beg]  
   
 mid = (beg+end)/2  
 if nums[mid] > nums[end]:  
 beg = mid + 1  
 else:  
 end = mid  
   
   
 return nums[beg]  
**Best\_solution:**My pretty simple code to solve it   
class Solution {  
public:  
 int findMin(vector<int> &num) {  
 int lo = 0;  
 int hi = num.size() - 1;  
 int mid = 0;  
   
 while(lo < hi) {  
 mid = lo + (hi - lo) / 2;  
   
 if (num[mid] > num[hi]) {  
 lo = mid + 1;  
 }  
 else if (num[mid] < num[hi]) {  
 hi = mid;  
 }  
 else { // when num[mid] and num[hi] are same  
 hi--;  
 }  
 }  
 return num[lo];  
 }  
};  
  
  
*155,Min Stack:***Python\_solution:**My Python solution   
class MinStack:  
  
def \_\_init\_\_(self):  
 self.q = []  
  
# @param x, an integer  
# @return an integer  
def push(self, x):  
 curMin = self.getMin()  
 if curMin == None or x < curMin:  
 curMin = x  
 self.q.append((x, curMin));  
  
# @return nothing  
def pop(self):  
 self.q.pop()  
  
  
# @return an integer  
def top(self):  
 if len(self.q) == 0:  
 return None  
 else:  
 return self.q[len(self.q) - 1][0]  
  
  
# @return an integer  
def getMin(self):  
 if len(self.q) == 0:  
 return None  
 else:  
 return self.q[len(self.q) - 1][1]  
**Best\_solution:**Share my Java solution with ONLY ONE stack   
public class MinStack {  
 long min;  
 Stack<Long> stack;  
  
 public MinStack(){  
 stack=new Stack<>();  
 }  
   
 public void push(int x) {  
 if (stack.isEmpty()){  
 stack.push(0L);  
 min=x;  
 }else{  
 stack.push(x-min);//Could be negative if min value needs to change  
 if (x<min) min=x;  
 }  
 }  
  
 public void pop() {  
 if (stack.isEmpty()) return;  
   
 long pop=stack.pop();  
   
 if (pop<0) min=min-pop;//If negative, increase the min value  
   
 }  
  
 public int top() {  
 long top=stack.peek();  
 if (top>0){  
 return (int)(top+min);  
 }else{  
 return (int)(min);  
 }  
 }  
  
 public int getMin() {  
 return (int)min;  
 }  
}  
  
*160,Intersection of Two Linked Lists:***Python\_solution:**Concise python code with comments   
class Solution:  
 # @param two ListNodes  
 # @return the intersected ListNode  
 def getIntersectionNode(self, headA, headB):  
 if headA is None or headB is None:  
 return None  
  
 pa = headA # 2 pointers  
 pb = headB  
  
 while pa is not pb:  
 # if either pointer hits the end, switch head and continue the second traversal,   
 # if not hit the end, just move on to next  
 pa = headB if pa is None else pa.next  
 pb = headA if pb is None else pb.next  
  
 return pa # only 2 ways to get out of the loop, they meet or the both hit the end=None  
  
# the idea is if you switch head, the possible difference between length would be countered.   
# On the second traversal, they either hit or miss.   
# if they meet, pa or pb would be the node we are looking for,   
# if they didn't meet, they will hit the end at the same iteration, pa == pb == None, return either one of them is the same,None  
**Best\_solution:**Java solution without knowing the difference in len!   
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {  
 //boundary check  
 if(headA == null || headB == null) return null;  
   
 ListNode a = headA;  
 ListNode b = headB;  
   
 //if a & b have different len, then we will stop the loop after second iteration  
 while( a != b){  
 //for the end of first iteration, we just reset the pointer to the head of another linkedlist  
 a = a == null? headB : a.next;  
 b = b == null? headA : b.next;   
 }  
   
 return a;  
}  
  
*162,Find Peak Element:***Python\_solution:**My clean and readable python solution   
Basic Idea: Binary search  
  
Elaboration:   
 if an element(not the right-most one) is smaller than its right neighbor, then there must be a peak element on its right, because the elements on its right is either   
 1. always increasing -> the right-most element is the peak  
 2. always decreasing -> the left-most element is the peak  
 3. first increasing then decreasing -> the pivot point is the peak  
 4. first decreasing then increasing -> the left-most element is the peak   
  
 Therefore, we can find the peak only on its right elements( cut the array to half)  
  
 The same idea applies to that an element(not the left-most one) is smaller than its left neighbor.  
  
  
  
Conditions:  
 1. array length is 1 -> return the only index   
 2. array length is 2 -> return the bigger number's index   
 3. array length is bigger than 2 ->   
 (1) find mid, compare it with its left and right neighbors   
 (2) return mid if nums[mid] greater than both neighbors  
 (3) take the right half array if nums[mid] smaller than right neighbor  
 (4) otherwise, take the left half  
  
Run time: O(logn)  
Memory: constant  
Test cases:   
 [1]  
 [1,2]  
 [2,1]  
 [1,2,3]  
 [3,2,1]  
 [2,1,3]  
  
  
def findPeakElement(self, nums):  
 left = 0  
 right = len(nums)-1  
  
 # handle condition 3  
 while left < right-1:  
 mid = (left+right)/2  
 if nums[mid] > nums[mid+1] and nums[mid] > nums[mid-1]:  
 return mid  
   
 if nums[mid] < nums[mid+1]:  
 left = mid+1  
 else:  
 right = mid-1  
   
 #handle condition 1 and 2  
 return left if nums[left] >= nums[right] else right  
**Best\_solution:**Find the maximum by binary search (recursion and iteration)   
class Solution {  
public:  
  
int findPeakElement(const vector<int> &num) {  
 return Helper(num, 0, num.size()-1);  
}  
int Helper(const vector<int> &num, int low, int high)  
{  
 if(low == high)  
 return low;  
 else  
 {  
 int mid1 = (low+high)/2;  
 int mid2 = mid1+1;  
 if(num[mid1] > num[mid2])  
 return Helper(num, low, mid1);  
 else  
 return Helper(num, mid2, high);  
 }  
}  
};  
  
  
*164,Maximum Gap:***Python\_solution:**Python bucket sort from official solution   
class Solution:  
# @param num, a list of integer  
# @return an integer  
def maximumGap(self, num):  
 if len(num) < 2 or min(num) == max(num):  
 return 0  
 a, b = min(num), max(num)  
 size = math.ceil((b-a)/(len(num)-1))  
 bucket = [[None, None] for \_ in range((b-a)//size+1)]  
 for n in num:  
 b = bucket[(n-a)//size]  
 b[0] = n if b[0] is None else min(b[0], n)  
 b[1] = n if b[1] is None else max(b[1], n)  
 bucket = [b for b in bucket if b[0] is not None]  
 return max(bucket[i][0]-bucket[i-1][1] for i in range(1, len(bucket)))  
  
**Best\_solution:**[bucket sort] JAVA solution with explanation, O(N) time and space   
public class Solution {  
public int maximumGap(int[] num) {  
 if (num == null || num.length < 2)  
 return 0;  
 // get the max and min value of the array  
 int min = num[0];  
 int max = num[0];  
 for (int i:num) {  
 min = Math.min(min, i);  
 max = Math.max(max, i);  
 }  
 // the minimum possibale gap, ceiling of the integer division  
 int gap = (int)Math.ceil((double)(max - min)/(num.length - 1));  
 int[] bucketsMIN = new int[num.length - 1]; // store the min value in that bucket  
 int[] bucketsMAX = new int[num.length - 1]; // store the max value in that bucket  
 Arrays.fill(bucketsMIN, Integer.MAX\_VALUE);  
 Arrays.fill(bucketsMAX, Integer.MIN\_VALUE);  
 // put numbers into buckets  
 for (int i:num) {  
 if (i == min || i == max)  
 continue;  
 int idx = (i - min) / gap; // index of the right position in the buckets  
 bucketsMIN[idx] = Math.min(i, bucketsMIN[idx]);  
 bucketsMAX[idx] = Math.max(i, bucketsMAX[idx]);  
 }  
 // scan the buckets for the max gap  
 int maxGap = Integer.MIN\_VALUE;  
 int previous = min;  
 for (int i = 0; i < num.length - 1; i++) {  
 if (bucketsMIN[i] == Integer.MAX\_VALUE && bucketsMAX[i] == Integer.MIN\_VALUE)  
 // empty bucket  
 continue;  
 // min value minus the previous value is the current gap  
 maxGap = Math.max(maxGap, bucketsMIN[i] - previous);  
 // update previous bucket value  
 previous = bucketsMAX[i];  
 }  
 maxGap = Math.max(maxGap, max - previous); // updata the final max value gap  
 return maxGap;  
}  
  
  
*165,Compare Version Numbers:***Python\_solution:**2-4 lines Python, 3 different ways   
izip\_longest  
**Best\_solution:**Accepted small Java solution.   
public int compareVersion(String version1, String version2) {  
 String[] levels1 = version1.split("\\.");  
 String[] levels2 = version2.split("\\.");  
   
 int length = Math.max(levels1.length, levels2.length);  
 for (int i=0; i<length; i++) {  
 Integer v1 = i < levels1.length ? Integer.parseInt(levels1[i]) : 0;  
 Integer v2 = i < levels2.length ? Integer.parseInt(levels2[i]) : 0;  
 int compare = v1.compareTo(v2);  
 if (compare != 0) {  
 return compare;  
 }  
 }  
   
 return 0;  
}  
  
*166,Fraction to Recurring Decimal:***Python\_solution:**Do not use python as cpp, here's a short version python code   
class Solution:  
# @return a string  
def fractionToDecimal(self, numerator, denominator):  
 n, remainder = divmod(abs(numerator), abs(denominator))  
 sign = '-' if numerator\*denominator < 0 else ''  
 result = [sign+str(n), '.']  
 stack = []  
 while remainder not in stack:  
 stack.append(remainder)  
 n, remainder = divmod(remainder\*10, abs(denominator))  
 result.append(str(n))  
  
 idx = stack.index(remainder)  
 result.insert(idx+2, '(')  
 result.append(')')  
 return ''.join(result).replace('(0)', '').rstrip('.')  
  
**Best\_solution:**My clean Java solution   
public class Solution {  
 public String fractionToDecimal(int numerator, int denominator) {  
 if (numerator == 0) {  
 return "0";  
 }  
 StringBuilder res = new StringBuilder();  
 // "+" or "-"  
 res.append(((numerator > 0) ^ (denominator > 0)) ? "-" : "");  
 long num = Math.abs((long)numerator);  
 long den = Math.abs((long)denominator);  
   
 // integral part  
 res.append(num / den);  
 num %= den;  
 if (num == 0) {  
 return res.toString();  
 }  
   
 // fractional part  
 res.append(".");  
 HashMap<Long, Integer> map = new HashMap<Long, Integer>();  
 map.put(num, res.length());  
 while (num != 0) {  
 num \*= 10;  
 res.append(num / den);  
 num %= den;  
 if (map.containsKey(num)) {  
 int index = map.get(num);  
 res.insert(index, "(");  
 res.append(")");  
 break;  
 }  
 else {  
 map.put(num, res.length());  
 }  
 }  
 return res.toString();  
 }  
}  
  
*167,Two Sum II - Input array is sorted:***Python\_solution:**Python different solutions (two-pointer, dictionary, binary search).   
# two-pointer  
def twoSum1(self, numbers, target):  
 l, r = 0, len(numbers)-1  
 while l < r:  
 s = numbers[l] + numbers[r]  
 if s == target:  
 return [l+1, r+1]  
 elif s < target:  
 l += 1  
 else:  
 r -= 1  
   
# dictionary   
def twoSum2(self, numbers, target):  
 dic = {}  
 for i, num in enumerate(numbers):  
 if target-num in dic:  
 return [dic[target-num]+1, i+1]  
 dic[num] = i  
   
# binary search   
def twoSum(self, numbers, target):  
 for i in xrange(len(numbers)):  
 l, r = i+1, len(numbers)-1  
 tmp = target - numbers[i]  
 while l <= r:  
 mid = l + (r-l)//2  
 if numbers[mid] == tmp:  
 return [i+1, mid+1]  
 elif numbers[mid] < tmp:  
 l = mid+1  
 else:  
 r = mid-1  
**Best\_solution:**Share my java AC solution.   
public int[] twoSum(int[] num, int target) {  
 int[] indice = new int[2];  
 if (num == null || num.length < 2) return indice;  
 int left = 0, right = num.length - 1;  
 while (left < right) {  
 int v = num[left] + num[right];  
 if (v == target) {  
 indice[0] = left + 1;  
 indice[1] = right + 1;  
 break;  
 } else if (v > target) {  
 right --;  
 } else {  
 left ++;  
 }  
 }  
 return indice;  
}  
  
*168,Excel Sheet Column Title:***Python\_solution:**My 1 lines code in Java, C++, and Python   
return n == 0 ? "" : convertToTitle(--n / 26) + (char)('A' + (n % 26));  
  
**Best\_solution:**My 1 lines code in Java, C++, and Python   
return n == 0 ? "" : convertToTitle(--n / 26) + (char)('A' + (n % 26));  
  
  
*169,Majority Element:***Python\_solution:**One line solution in Python   
return sorted(num)[len(num)/2]  
**Best\_solution:**O(n) time O(1) space fastest solution   
public class Solution {  
 public int majorityElement(int[] num) {  
  
 int major=num[0], count = 1;  
 for(int i=1; i<num.length;i++){  
 if(count==0){  
 count++;  
 major=num[i];  
 }else if(major==num[i]){  
 count++;  
 }else count--;  
   
 }  
 return major;  
 }  
}  
  
*171,Excel Sheet Column Number:***Python\_solution:**One line python code using Map/Reduce   
def titleToNumber(self, s):  
 return reduce(lambda x,y:x\*26+y,map(lambda x:ord(x)-ord('A')+1,s))  
**Best\_solution:**My solutions in 3 languages, does any one have one line solution in Java or C++?   
int result = 0;  
for (int i = 0; i < s.length(); result = result \* 26 + (s.charAt(i) - 'A' + 1), i++);  
return result;  
  
  
*172,Factorial Trailing Zeroes:***Python\_solution:**O(log5\_n) solution, python.   
 def trailingZeroes(self, n):  
 r = 0  
 while n > 0:  
 n /= 5  
 r += n  
 return r  
**Best\_solution:**My one-line solutions in 3 languages   
 return n == 0 ? 0 : n / 5 + trailingZeroes(n / 5);  
  
  
*173,Binary Search Tree Iterator:***Python\_solution:**Two Python solutions, stack and generator   
def \_\_init\_\_(self, root):  
 self.stack = []  
 while root:  
 self.stack.append(root)  
 root = root.left  
  
# @return a boolean, whether we have a next smallest number  
def hasNext(self):  
 return len(self.stack) > 0  
  
# @return an integer, the next smallest number  
def next(self):  
 node = self.stack.pop()  
 x = node.right  
 while x:  
 self.stack.append(x)  
 x = x.left  
 return node.val  
  
**Best\_solution:**My solutions in 3 languages with Stack   
public class BSTIterator {  
 private Stack<TreeNode> stack = new Stack<TreeNode>();  
   
 public BSTIterator(TreeNode root) {  
 pushAll(root);  
 }  
  
 /\*\* @return whether we have a next smallest number \*/  
 public boolean hasNext() {  
 return !stack.isEmpty();  
 }  
  
 /\*\* @return the next smallest number \*/  
 public int next() {  
 TreeNode tmpNode = stack.pop();  
 pushAll(tmpNode.right);  
 return tmpNode.val;  
 }  
   
 private void pushAll(TreeNode node) {  
 for (; node != null; stack.push(node), node = node.left);  
 }  
}  
  
  
*174,Dungeon Game:***Python\_solution:**6 lines Python, 8 lines Ruby   
def calculateMinimumHP(self, dungeon):  
 n = len(dungeon[0])  
 need = [2\*\*31] \* (n-1) + [1]  
 for row in dungeon[::-1]:  
 for j in range(n)[::-1]:  
 need[j] = max(min(need[j:j+2]) - row[j], 1)  
 return need[0]  
  
**Best\_solution:**C++ DP solution   
class Solution {  
public:  
 int calculateMinimumHP(vector<vector<int> > &dungeon) {  
 int M = dungeon.size();  
 int N = dungeon[0].size();  
 // hp[i][j] represents the min hp needed at position (i, j)  
 // Add dummy row and column at bottom and right side  
 vector<vector<int> > hp(M + 1, vector<int>(N + 1, INT\_MAX));  
 hp[M][N - 1] = 1;  
 hp[M - 1][N] = 1;  
 for (int i = M - 1; i >= 0; i--) {  
 for (int j = N - 1; j >= 0; j--) {  
 int need = min(hp[i + 1][j], hp[i][j + 1]) - dungeon[i][j];  
 hp[i][j] = need <= 0 ? 1 : need;  
 }  
 }  
 return hp[0][0];  
 }  
};  
  
*175,Combine Two Tables:***Best\_solution:**Its a simple question of Left Join. My solution attached   
SELECT Person.FirstName, Person.LastName, Address.City, Address.State from Person LEFT JOIN Address on Person.PersonId = Address.PersonId;  
  
*176,Second Highest Salary:***Best\_solution:**Simple query which handles the NULL situation   
guaranteed  
  
*177,Nth Highest Salary:***Best\_solution:**Accpted Solution for the Nth Highest Salary   
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT  
BEGIN  
DECLARE M INT;  
SET M=N-1;  
 RETURN (  
 # Write your MySQL query statement below.  
 SELECT DISTINCT Salary FROM Employee ORDER BY Salary DESC LIMIT M, 1  
 );  
END  
  
*178,Rank Scores:***Best\_solution:**Simple, Short, Fast   
SELECT  
 Score,  
 @rank := @rank + (@prev <> (@prev := Score)) Rank  
FROM  
 Scores,  
 (SELECT @rank := 0, @prev := -1) init  
ORDER BY Score desc  
  
  
*179,Largest Number:***Python\_solution:**My 3-lines code in Java and Python   
public class Solution {  
 public String largestNumber(int[] num) {  
 String[] array = Arrays.stream(num).mapToObj(String::valueOf).toArray(String[]::new);  
 Arrays.sort(array, (String s1, String s2) -> (s2 + s1).compareTo(s1 + s2));  
 return Arrays.stream(array).reduce((x, y) -> x.equals("0") ? y : x + y).get();  
 }  
}  
  
**Best\_solution:**My Java Solution to share   
String s1 = "9";  
String s2 = "31";  
  
String case1 = s1 + s2; // 931  
String case2 = s2 + s1; // 319  
  
  
  
*180,Consecutive Numbers:***Best\_solution:**Simple solution   
Select DISTINCT l1.Num from Logs l1, Logs l2, Logs l3   
where l1.Id=l2.Id-1 and l2.Id=l3.Id-1   
and l1.Num=l2.Num and l2.Num=l3.Num  
  
*181,Employees Earning More Than Their Managers:***Best\_solution:**A straightforward method   
select E1.Name   
from Employee as E1, Employee as E2   
where E1.ManagerId = E2.Id and E1.Salary > E2.Salary  
  
*182,Duplicate Emails:***Best\_solution:**I have this Simple Approach, anybody has some other way   
 SELECT DISTINCT a.Email  
 FROM Person a JOIN Person b  
 ON (a.Email = b.Email)  
 WHERE a.Id <> b.Id  
  
  
*183,Customers Who Never Order:***Best\_solution:**Three accepted solutions   
SELECT A.Name from Customers A  
WHERE NOT EXISTS (SELECT 1 FROM Orders B WHERE A.Id = B.CustomerId)  
  
SELECT A.Name from Customers A  
LEFT JOIN Orders B on a.Id = B.CustomerId  
WHERE b.CustomerId is NULL  
  
SELECT A.Name from Customers A  
WHERE A.Id NOT IN (SELECT B.CustomerId from Orders B)  
  
*184,Department Highest Salary:***Best\_solution:**Three accpeted solutions   
SELECT D.Name AS Department ,E.Name AS Employee ,E.Salary   
FROM  
 Employee E,  
 (SELECT DepartmentId,max(Salary) as max FROM Employee GROUP BY DepartmentId) T,  
 Department D  
WHERE E.DepartmentId = T.DepartmentId   
 AND E.Salary = T.max  
 AND E.DepartmentId = D.id  
  
SELECT D.Name,A.Name,A.Salary   
FROM   
 Employee A,  
 Department D   
WHERE A.DepartmentId = D.Id   
 AND NOT EXISTS   
 (SELECT 1 FROM Employee B WHERE B.Salary > A.Salary AND A.DepartmentId = B.DepartmentId)   
  
SELECT D.Name AS Department ,E.Name AS Employee ,E.Salary   
from   
 Employee E,  
 Department D   
WHERE E.DepartmentId = D.id   
 AND (DepartmentId,Salary) in   
 (SELECT DepartmentId,max(Salary) as max FROM Employee GROUP BY DepartmentId)  
  
*185,Department Top Three Salaries:***Best\_solution:**Accepted solution without group by or order by   
select d.Name Department, e1.Name Employee, e1.Salary  
from Employee e1   
join Department d  
on e1.DepartmentId = d.Id  
where 3 > (select count(distinct(e2.Salary))   
 from Employee e2   
 where e2.Salary > e1.Salary   
 and e1.DepartmentId = e2.DepartmentId  
 );  
  
*187,Repeated DNA Sequences:***Python\_solution:**4 lines Python solution   
class Solution:  
 # @param s, a string  
 # @return a list of strings  
 def findRepeatedDnaSequences(self, s):  
 sequences = collections.defaultdict(int) #set '0' as the default value for non-existing keys  
 for i in range(len(s)):  
 sequences[s[i:i+10]] += 1#add 1 to the count  
 return [key for key, value in sequences.iteritems() if value > 1] #extract the relevant keys  
**Best\_solution:**Clean Java solution (hashmap + bits manipulation)   
public List<String> findRepeatedDnaSequences(String s) {  
 Set<Integer> words = new HashSet<>();  
 Set<Integer> doubleWords = new HashSet<>();  
 List<String> rv = new ArrayList<>();  
 char[] map = new char[26];  
 //map['A' - 'A'] = 0;  
 map['C' - 'A'] = 1;  
 map['G' - 'A'] = 2;  
 map['T' - 'A'] = 3;  
  
 for(int i = 0; i < s.length() - 9; i++) {  
 int v = 0;  
 for(int j = i; j < i + 10; j++) {  
 v <<= 2;  
 v |= map[s.charAt(j) - 'A'];  
 }  
 if(!words.add(v) && doubleWords.add(v)) {  
 rv.add(s.substring(i, i + 10));  
 }  
 }  
 return rv;  
}  
  
*188,Best Time to Buy and Sell Stock IV:***Python\_solution:**Well explained Python DP with comments   
def maxProfit4(self, k, prices):  
 n = len(prices)  
 if n < 2:  
 return 0  
 # k is big enougth to cover all ramps.  
 if k >= n / 2:  
 return sum(i - j  
 for i, j in zip(prices[1:], prices[:-1]) if i - j > 0)  
 globalMax = [[0] \* n for \_ in xrange(k + 1)]  
 for i in xrange(1, k + 1):  
 # The max profit with i transations and selling stock on day j.  
 localMax = [0] \* n  
 for j in xrange(1, n):  
 profit = prices[j] - prices[j - 1]  
 localMax[j] = max(  
 # We have made max profit with (i - 1) transations in  
 # (j - 1) days.  
 # For the last transation, we buy stock on day (j - 1)  
 # and sell it on day j.  
 globalMax[i - 1][j - 1] + profit,  
 # We have made max profit with (i - 1) transations in  
 # (j - 1) days.  
 # For the last transation, we buy stock on day j and  
 # sell it on the same day, so we have 0 profit, apparently  
 # we do not have to add it.  
 globalMax[i - 1][j - 1], # + 0,  
 # We have made profit in (j - 1) days.  
 # We want to cancel the day (j - 1) sale and sell it on  
 # day j.  
 localMax[j - 1] + profit)  
 globalMax[i][j] = max(globalMax[i][j - 1], localMax[j])  
 return globalMax[k][-1]  
**Best\_solution:**A Concise DP Solution in Java   
 public int maxProfit(int k, int[] prices) {  
 int len = prices.length;  
 if (k >= len / 2) return quickSolve(prices);  
   
 int[][] t = new int[k + 1][len];  
 for (int i = 1; i <= k; i++) {  
 int tmpMax = -prices[0];  
 for (int j = 1; j < len; j++) {  
 t[i][j] = Math.max(t[i][j - 1], prices[j] + tmpMax);  
 tmpMax = Math.max(tmpMax, t[i - 1][j - 1] - prices[j]);  
 }  
 }  
 return t[k][len - 1];  
 }  
   
  
 private int quickSolve(int[] prices) {  
 int len = prices.length, profit = 0;  
 for (int i = 1; i < len; i++)  
 // as long as there is a price gap, we gain a profit.  
 if (prices[i] > prices[i - 1]) profit += prices[i] - prices[i - 1];  
 return profit;  
 }  
  
*189,Rotate Array:***Python\_solution:**My solution by using Python   
class Solution:  
 # @param nums, a list of integer  
 # @param k, num of steps  
 # @return nothing, please modify the nums list in-place.  
 def rotate(self, nums, k):  
 n = len(nums)  
 k = k % n  
 nums[:] = nums[n-k:] + nums[:n-k]  
  
**Best\_solution:**Easy to read Java solution   
public void rotate(int[] nums, int k) {  
 k %= nums.length;  
 reverse(nums, 0, nums.length - 1);  
 reverse(nums, 0, k - 1);  
 reverse(nums, k, nums.length - 1);  
}  
  
public void reverse(int[] nums, int start, int end) {  
 while (start < end) {  
 int temp = nums[start];  
 nums[start] = nums[end];  
 nums[end] = temp;  
 start++;  
 end--;  
 }  
}  
  
*190,Reverse Bits:***Python\_solution:**Python AC with 63ms, 3lines   
class Solution:  
 # @param n, an integer  
 # @return an integer  
 def reverseBits(self, n):  
 oribin='{0:032b}'.format(n)  
 reversebin=oribin[::-1]  
 return int(reversebin,2)  
**Best\_solution:**O(1) bit operation C++ solution (8ms)   
class Solution {  
public:  
 uint32\_t reverseBits(uint32\_t n) {  
 n = (n >> 16) | (n << 16);  
 n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);  
 n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);  
 n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);  
 n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);  
 return n;  
 }  
};  
  
  
*191,Number of 1 Bits:***Python\_solution:**[Python] 2 solutions. One naive solution with built-in functions. One trick with bit operation   
def hammingWeight(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 return bin(n).count('1')  
  
**Best\_solution:**Simple Java Solution, Bit Shifting   
public static int hammingWeight(int n) {  
 int ones = 0;  
 while(n!=0) {  
 ones = ones + (n & 1);  
 n = n>>>1;  
 }  
 return ones;  
}  
  
  
*192,Word Frequency:***Best\_solution:**My simple solution (one line with pipe)   
cat words.txt | tr -s ' ' '\n' | sort | uniq -c | sort -r | awk '{ print $2, $1 }'  
  
  
*193,Valid Phone Numbers:***Best\_solution:**Three different solutions using grep, sed, and awk   
grep  
  
*194,Transpose File:***Best\_solution:**AC solution using awk and statement just like C.   
for  
  
*195,Tenth Line:***Best\_solution:**Share four different solutions   
# Solution 1  
cnt=0  
while read line && [ $cnt -le 10 ]; do  
 let 'cnt = cnt + 1'  
 if [ $cnt -eq 10 ]; then  
 echo $line  
 exit 0  
 fi  
done < file.txt  
  
# Solution 2  
awk 'FNR == 10 {print }' file.txt  
# OR  
awk 'NR == 10' file.txt  
  
# Solution 3  
sed -n 10p file.txt  
  
# Solution 4  
tail -n+10 file.txt|head -1  
  
*196,Delete Duplicate Emails:***Best\_solution:**Simple Solution   
DELETE FROM Person  
 WHERE Id IN  
 (SELECT P1.Id FROM Person AS P1, Person AS P2   
 WHERE P1.Id > P2.Id AND P1.Email = P2.Email);  
  
  
*197,Rising Temperature:***Best\_solution:**Simple Solution   
SELECT wt1.Id   
FROM Weather wt1, Weather wt2  
WHERE wt1.Temperature > wt2.Temperature AND   
 TO\_DAYS(wt1.DATE)-TO\_DAYS(wt2.DATE)=1;  
  
  
*198,House Robber:***Python\_solution:**Python solution, 3 lines.   
f(0) = nums[0]  
f(1) = max(num[0], num[1])  
f(k) = max( f(k-2) + nums[k], f(k-1) )  
  
**Best\_solution:**C 1ms, O(1)space, very simple solution   
#define max(a, b) ((a)>(b)?(a):(b))  
int rob(int num[], int n) {  
 int a = 0;  
 int b = 0;  
   
 for (int i=0; i<n; i++)  
 {  
 if (i%2==0)  
 {  
 a = max(a+num[i], b);  
 }  
 else  
 {  
 b = max(a, b+num[i]);  
 }  
 }  
   
 return max(a, b);  
}  
  
*199,Binary Tree Right Side View:***Python\_solution:**5-9 Lines Python, 48+ ms   
def rightSideView(self, root):  
 if not root:  
 return []  
 right = self.rightSideView(root.right)  
 left = self.rightSideView(root.left)  
 return [root.val] + right + left[len(right):]  
  
**Best\_solution:**My simple accepted solution(JAVA)   
public class Solution {  
 public List<Integer> rightSideView(TreeNode root) {  
 List<Integer> result = new ArrayList<Integer>();  
 rightView(root, result, 0);  
 return result;  
 }  
   
 public void rightView(TreeNode curr, List<Integer> result, int currDepth){  
 if(curr == null){  
 return;  
 }  
 if(currDepth == result.size()){  
 result.add(curr.val);  
 }  
   
 rightView(curr.right, result, currDepth + 1);  
 rightView(curr.left, result, currDepth + 1);  
   
 }  
}  
  
*200,Number of Islands:***Python\_solution:**7 lines Python, ~14 lines Java   
def numIslands(self, grid):  
 def sink(i, j):  
 if 0 <= i < len(grid) and 0 <= j < len(grid[i]) and grid[i][j] == '1':  
 grid[i][j] = '0'  
 map(sink, (i+1, i-1, i, i), (j, j, j+1, j-1))  
 return 1  
 return 0  
 return sum(sink(i, j) for i in range(len(grid)) for j in range(len(grid[i])))  
  
**Best\_solution:**Very concise Java AC solution   
public class Solution {  
  
private int n;  
private int m;  
  
public int numIslands(char[][] grid) {  
 int count = 0;  
 n = grid.length;  
 if (n == 0) return 0;  
 m = grid[0].length;  
 for (int i = 0; i < n; i++){  
 for (int j = 0; j < m; j++)  
 if (grid[i][j] == '1') {  
 DFSMarking(grid, i, j);  
 ++count;  
 }  
 }   
 return count;  
}  
  
private void DFSMarking(char[][] grid, int i, int j) {  
 if (i < 0 || j < 0 || i >= n || j >= m || grid[i][j] != '1') return;  
 grid[i][j] = '0';  
 DFSMarking(grid, i + 1, j);  
 DFSMarking(grid, i - 1, j);  
 DFSMarking(grid, i, j + 1);  
 DFSMarking(grid, i, j - 1);  
}  
  
  
*201,Bitwise AND of Numbers Range:***Python\_solution:**Java/Python easy solution with explanation   
4 & 7 = 0b100 & 0b111 = 0b100  
5 & 7 = 0b101 & 0b111 = 0b101  
5 & 6 = 0b101 & 0b110 = 0b100  
  
**Best\_solution:**Bit operation solution(JAVA)   
public class Solution {  
 public int rangeBitwiseAnd(int m, int n) {  
 if(m == 0){  
 return 0;  
 }  
 int moveFactor = 1;  
 while(m != n){  
 m >>= 1;  
 n >>= 1;  
 moveFactor <<= 1;  
 }  
 return m \* moveFactor;  
 }  
}  
  
*202,Happy Number:***Python\_solution:**My Python Solution   
def isHappy(self, n):  
 mem = set()  
 while n != 1:  
 n = sum([int(i) \*\* 2 for i in str(n)])  
 if n in mem:  
 return False  
 else:  
 mem.add(n)  
 else:  
 return True  
**Best\_solution:**My solution in C( O(1) space and no magic math property involved )   
int digitSquareSum(int n) {  
 int sum = 0, tmp;  
 while (n) {  
 tmp = n % 10;  
 sum += tmp \* tmp;  
 n /= 10;  
 }  
 return sum;  
}  
  
bool isHappy(int n) {  
 int slow, fast;  
 slow = fast = n;  
 do {  
 slow = digitSquareSum(slow);  
 fast = digitSquareSum(fast);  
 fast = digitSquareSum(fast);  
 } while(slow != fast);  
 if (slow == 1) return 1;  
 else return 0;  
}  
  
*203,Remove Linked List Elements:***Python\_solution:**Python solution   
class Solution:  
# @param {ListNode} head  
# @param {integer} val  
# @return {ListNode}  
def removeElements(self, head, val):  
 dummy = ListNode(-1)  
 dummy.next = head  
 next = dummy  
   
 while next != None and next.next != None:  
 if next.next.val == val:  
 next.next = next.next.next  
 else:  
 next = next.next  
   
 return dummy.next  
**Best\_solution:**3 line recursive solution   
public ListNode removeElements(ListNode head, int val) {  
 if (head == null) return null;  
 head.next = removeElements(head.next, val);  
 return head.val == val ? head.next : head;  
}  
  
*204,Count Primes:***Python\_solution:**Fast Python Solution   
class Solution:  
# @param {integer} n  
# @return {integer}  
def countPrimes(self, n):  
 if n < 3:  
 return 0  
 primes = [True] \* n  
 primes[0] = primes[1] = False  
 for i in range(2, int(n \*\* 0.5) + 1):  
 if primes[i]:  
 primes[i \* i: n: i] = [False] \* len(primes[i \* i: n: i])  
 return sum(primes)  
**Best\_solution:**My simple Java solution   
public class Solution {  
 public int countPrimes(int n) {  
 boolean[] notPrime = new boolean[n];  
 int count = 0;  
 for (int i = 2; i < n; i++) {  
 if (notPrime[i] == false) {  
 count++;  
 for (int j = 2; i\*j < n; j++) {  
 notPrime[i\*j] = true;  
 }  
 }  
 }  
   
 return count;  
 }  
}  
  
*205,Isomorphic Strings:***Python\_solution:**Python different solutions (dictionary, etc).   
def isIsomorphic1(self, s, t):  
 d1, d2 = {}, {}  
 for i, val in enumerate(s):  
 d1[val] = d1.get(val, []) + [i]  
 for i, val in enumerate(t):  
 d2[val] = d2.get(val, []) + [i]  
 return sorted(d1.values()) == sorted(d2.values())  
   
def isIsomorphic2(self, s, t):  
 d1, d2 = [[] for \_ in xrange(256)], [[] for \_ in xrange(256)]  
 for i, val in enumerate(s):  
 d1[ord(val)].append(i)  
 for i, val in enumerate(t):  
 d2[ord(val)].append(i)  
 return sorted(d1) == sorted(d2)  
   
def isIsomorphic3(self, s, t):  
 return len(set(zip(s, t))) == len(set(s)) == len(set(t))  
   
def isIsomorphic4(self, s, t):   
 return [s.find(i) for i in s] == [t.find(j) for j in t]  
   
def isIsomorphic5(self, s, t):  
 return map(s.find, s) == map(t.find, t)  
  
def isIsomorphic(self, s, t):  
 d1, d2 = [0 for \_ in xrange(256)], [0 for \_ in xrange(256)]  
 for i in xrange(len(s)):  
 if d1[ord(s[i])] != d2[ord(t[i])]:  
 return False  
 d1[ord(s[i])] = i+1  
 d2[ord(t[i])] = i+1  
 return True  
**Best\_solution:**My 6 lines solution   
class Solution {  
public:  
 bool isIsomorphic(string s, string t) {  
 int m1[256] = {0}, m2[256] = {0}, n = s.size();  
 for (int i = 0; i < n; ++i) {  
 if (m1[s[i]] != m2[t[i]]) return false;  
 m1[s[i]] = i + 1;  
 m2[t[i]] = i + 1;  
 }  
 return true;  
 }  
};  
  
*206,Reverse Linked List:***Python\_solution:**Python Iterative and Recursive Solution   
class Solution:  
# @param {ListNode} head  
# @return {ListNode}  
def reverseList(self, head):  
 prev = None  
 while head:  
 curr = head  
 head = head.next  
 curr.next = prev  
 prev = curr  
 return prev  
  
**Best\_solution:**In-place iterative and recursive Java solution   
public ListNode reverseList(ListNode head) {  
 /\* iterative solution \*/  
 ListNode newHead = null;  
 while (head != null) {  
 ListNode next = head.next;  
 head.next = newHead;  
 newHead = head;  
 head = next;  
 }  
 return newHead;  
}  
  
public ListNode reverseList(ListNode head) {  
 /\* recursive solution \*/  
 return reverseListInt(head, null);  
}  
  
private ListNode reverseListInt(ListNode head, ListNode newHead) {  
 if (head == null)  
 return newHead;  
 ListNode next = head.next;  
 head.next = newHead;  
 return reverseListInt(next, head);  
}  
  
*207,Course Schedule:***Python\_solution:**Python 20 lines DFS solution sharing with explanation   
def canFinish(self, numCourses, prerequisites):  
 graph = [[] for \_ in xrange(numCourses)]  
 visit = [0 for \_ in xrange(numCourses)]  
 for x, y in prerequisites:  
 graph[x].append(y)  
 def dfs(i):  
 if visit[i] == -1:  
 return False  
 if visit[i] == 1:  
 return True  
 visit[i] = -1  
 for j in graph[i]:  
 if not dfs(j):  
 return False  
 visit[i] = 1  
 return True  
 for i in xrange(numCourses):  
 if not dfs(i):  
 return False  
 return True  
  
**Best\_solution:**18-22 lines C++ BFS/DFS Solutions   
prerequisites  
  
*208,Implement Trie (Prefix Tree):***Python\_solution:**AC Python Solution   
class TrieNode:  
# Initialize your data structure here.  
def \_\_init\_\_(self):  
 self.children = collections.defaultdict(TrieNode)  
 self.is\_word = False  
  
class Trie:  
  
def \_\_init\_\_(self):  
 self.root = TrieNode()  
  
def insert(self, word):  
 current = self.root  
 for letter in word:  
 current = current.children[letter]  
 current.is\_word = True  
  
def search(self, word):  
 current = self.root  
 for letter in word:  
 current = current.children.get(letter)  
 if current is None:  
 return False  
 return current.is\_word  
  
def startsWith(self, prefix):  
 current = self.root  
 for letter in prefix:  
 current = current.children.get(letter)  
 if current is None:  
 return False  
 return True  
**Best\_solution:**Maybe the code is not too much by using "next[26]", C++   
class TrieNode  
{  
public:  
 TrieNode \*next[26];  
 bool is\_word;  
   
 // Initialize your data structure here.  
 TrieNode(bool b = false)  
 {  
 memset(next, 0, sizeof(next));  
 is\_word = b;  
 }  
};  
  
class Trie  
{  
 TrieNode \*root;  
public:  
 Trie()  
 {  
 root = new TrieNode();  
 }  
  
 // Inserts a word into the trie.  
 void insert(string s)  
 {  
 TrieNode \*p = root;  
 for(int i = 0; i < s.size(); ++ i)  
 {  
 if(p -> next[s[i] - 'a'] == NULL)  
 p -> next[s[i] - 'a'] = new TrieNode();  
 p = p -> next[s[i] - 'a'];  
 }  
 p -> is\_word = true;  
 }  
  
 // Returns if the word is in the trie.  
 bool search(string key)  
 {  
 TrieNode \*p = find(key);  
 return p != NULL && p -> is\_word;  
 }  
  
 // Returns if there is any word in the trie  
 // that starts with the given prefix.  
 bool startsWith(string prefix)  
 {  
 return find(prefix) != NULL;  
 }  
  
private:  
 TrieNode\* find(string key)  
 {  
 TrieNode \*p = root;  
 for(int i = 0; i < key.size() && p != NULL; ++ i)  
 p = p -> next[key[i] - 'a'];  
 return p;  
 }  
};  
  
*209,Minimum Size Subarray Sum:***Python\_solution:**Python O(n) and O(n log n) solution   
class Solution:  
  
def minSubArrayLen(self, s, nums):  
 total = left = 0  
 result = len(nums) + 1  
 for right, n in enumerate(nums):  
 total += n  
 while total >= s:  
 result = min(result, right - left + 1)  
 total -= nums[left]  
 left += 1  
 return result if result <= len(nums) else 0  
  
**Best\_solution:**Accepted clean Java O(n) solution (two pointers)   
public int minSubArrayLen(int s, int[] a) {  
 if (a == null || a.length == 0)  
 return 0;  
   
 int i = 0, j = 0, sum = 0, min = Integer.MAX\_VALUE;  
   
 while (j < a.length) {  
 sum += a[j++];  
   
 while (sum >= s) {  
 min = Math.min(min, j - i);  
 sum -= a[i++];  
 }  
 }  
   
 return min == Integer.MAX\_VALUE ? 0 : min;  
}  
  
*210,Course Schedule II:***Python\_solution:**Python dfs, bfs solutions with comments.   
# BFS  
def findOrder1(self, numCourses, prerequisites):  
 dic = {i: set() for i in xrange(numCourses)}  
 neigh = collections.defaultdict(set)  
 for i, j in prerequisites:  
 dic[i].add(j)  
 neigh[j].add(i)  
 # queue stores the courses which have no prerequisites  
 queue = collections.deque([i for i in dic if not dic[i]])  
 count, res = 0, []  
 while queue:  
 node = queue.popleft()  
 res.append(node)  
 count += 1  
 for i in neigh[node]:  
 dic[i].remove(node)  
 if not dic[i]:  
 queue.append(i)  
 return res if count == numCourses else []  
   
# DFS  
def findOrder(self, numCourses, prerequisites):  
 dic = collections.defaultdict(set)  
 neigh = collections.defaultdict(set)  
 for i, j in prerequisites:  
 dic[i].add(j)  
 neigh[j].add(i)  
 stack = [i for i in xrange(numCourses) if not dic[i]]  
 res = []  
 while stack:  
 node = stack.pop()  
 res.append(node)  
 for i in neigh[node]:  
 dic[i].remove(node)  
 if not dic[i]:  
 stack.append(i)  
 dic.pop(node)  
 return res if not dic else []  
**Best\_solution:**Two AC solution in Java using BFS and DFS with explanation   
public int[] findOrder(int numCourses, int[][] prerequisites) {  
 int[] incLinkCounts = new int[numCourses];  
 List<List<Integer>> adjs = new ArrayList<>(numCourses);  
 initialiseGraph(incLinkCounts, adjs, prerequisites);  
 //return solveByBFS(incLinkCounts, adjs);  
 return solveByDFS(adjs);  
}  
  
  
*211,Add and Search Word - Data structure design:***Python\_solution:**Python 168ms-beat-100% solution   
class WordDictionary(object):  
 def \_\_init\_\_(self):  
 self.word\_dict = collections.defaultdict(list)  
   
  
 def addWord(self, word):  
 if word:  
 self.word\_dict[len(word)].append(word)  
  
 def search(self, word):  
 if not word:  
 return False  
 if '.' not in word:  
 return word in self.word\_dict[len(word)]  
 for v in self.word\_dict[len(word)]:  
 # match xx.xx.x with yyyyyyy  
 for i, ch in enumerate(word):  
 if ch != v[i] and ch != '.':  
 break  
 else:  
 return True  
 return False  
  
**Best\_solution:**My simple and clean Java code   
public class WordDictionary {  
 public class TrieNode {  
 public TrieNode[] children = new TrieNode[26];  
 public String item = "";  
 }  
   
 private TrieNode root = new TrieNode();  
  
 public void addWord(String word) {  
 TrieNode node = root;  
 for (char c : word.toCharArray()) {  
 if (node.children[c - 'a'] == null) {  
 node.children[c - 'a'] = new TrieNode();  
 }  
 node = node.children[c - 'a'];  
 }  
 node.item = word;  
 }  
  
 public boolean search(String word) {  
 return match(word.toCharArray(), 0, root);  
 }  
   
 private boolean match(char[] chs, int k, TrieNode node) {  
 if (k == chs.length) return !node.item.equals("");   
 if (chs[k] != '.') {  
 return node.children[chs[k] - 'a'] != null && match(chs, k + 1, node.children[chs[k] - 'a']);  
 } else {  
 for (int i = 0; i < node.children.length; i++) {  
 if (node.children[i] != null) {  
 if (match(chs, k + 1, node.children[i])) {  
 return true;  
 }  
 }  
 }  
 }  
 return false;  
 }  
}  
  
*212,Word Search II:***Python\_solution:**Python code use trie and dfs 380ms   
class Solution:  
 # @param {character[][]} board  
 # @param {string[]} words  
 # @return {string[]}  
 def findWords(self, board, words):  
 #make trie  
 trie={}  
 for w in words:  
 t=trie  
 for c in w:  
 if c not in t:  
 t[c]={}  
 t=t[c]  
 t['#']='#'  
 self.res=set()  
 self.used=[[False]\*len(board[0]) for \_ in range(len(board))]  
 for i in range(len(board)):  
 for j in range(len(board[0])):  
 self.find(board,i,j,trie,'')  
 return list(self.res)  
   
 def find(self,board,i,j,trie,pre):  
 if '#' in trie:  
 self.res.add(pre)  
 if i<0 or i>=len(board) or j<0 or j>=len(board[0]):  
 return  
 if not self.used[i][j] and board[i][j] in trie:  
 self.used[i][j]=True  
 self.find(board,i+1,j,trie[board[i][j]],pre+board[i][j])  
 self.find(board,i,j+1,trie[board[i][j]],pre+board[i][j])  
 self.find(board,i-1,j,trie[board[i][j]],pre+board[i][j])  
 self.find(board,i,j-1,trie[board[i][j]],pre+board[i][j])  
 self.used[i][j]=False  
**Best\_solution:**Java 15ms Easiest Solution (100.00%)   
Backtracking (dfs)  
  
*213,House Robber II:***Python\_solution:**My Python Solution   
class Solution(object):  
 def rob(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 n = len(nums)  
 if n == 0: return 0  
 if n < 4: return max(nums)  
  
 first, second = 0, 0  
 for i in nums[:-1]: first, second = second, max(first + i, second)  
 result = second  
  
 first, second = 0, 0  
 for i in nums[1:]: first, second = second, max(first + i, second)  
 return max(result, second)  
**Best\_solution:**Simple AC solution in Java in O(n) with explanation   
private int rob(int[] num, int lo, int hi) {  
 int include = 0, exclude = 0;  
 for (int j = lo; j <= hi; j++) {  
 int i = include, e = exclude;  
 include = e + num[j];  
 exclude = Math.max(e, i);  
 }  
 return Math.max(include, exclude);  
}  
  
  
*214,Shortest Palindrome:***Python\_solution:**Python solution(KMP)   
class Solution:  
# @param {string} s  
# @return {string}  
def shortestPalindrome(self, s):  
 A=s+"\*"+s[::-1]  
 cont=[0]  
 for i in range(1,len(A)):  
 index=cont[i-1]  
 while(index>0 and A[index]!=A[i]):  
 index=cont[index-1]  
 cont.append(index+(1 if A[index]==A[i] else 0))  
 return s[cont[-1]:][::-1]+s  
**Best\_solution:**Clean KMP solution with super detailed explanation   
public String shortestPalindrome(String s) {  
 String temp = s + "#" + new StringBuilder(s).reverse().toString();  
 int[] table = getTable(temp);  
   
 //get the maximum palin part in s starts from 0  
 return new StringBuilder(s.substring(table[table.length - 1])).reverse().toString() + s;  
}  
  
public int[] getTable(String s){  
 //get lookup table  
 int[] table = new int[s.length()];  
   
 //pointer that points to matched char in prefix part  
   
 int index = 0;  
 //skip index 0, we will not match a string with itself  
 for(int i = 1; i < s.length(); i++){  
 if(s.charAt(index) == s.charAt(i)){  
 //we can extend match in prefix and postfix  
 table[i] = table[i-1] + 1;  
 index ++;  
 }else{  
 //match failed, we try to match a shorter substring  
   
 //by assigning index to table[i-1], we will shorten the match string length, and jump to the   
 //prefix part that we used to match postfix ended at i - 1  
 index = table[i-1];  
   
 while(index > 0 && s.charAt(index) != s.charAt(i)){  
 //we will try to shorten the match string length until we revert to the beginning of match (index 1)  
 index = table[index-1];  
 }  
   
 //when we are here may either found a match char or we reach the boundary and still no luck  
 //so we need check char match  
 if(s.charAt(index) == s.charAt(i)){  
 //if match, then extend one char   
 index ++ ;  
 }  
   
 table[i] = index;  
 }  
   
 }  
   
 return table;  
}  
  
  
*215,Kth Largest Element in an Array:***Python\_solution:**Python different solutions with comments (bubble sort, selection sort, heap sort and quick sort).   
# O(nlgn) time  
def findKthLargest1(self, nums, k):  
 return sorted(nums, reverse=True)[k-1]  
   
# O(nk) time, bubble sort idea, TLE  
def findKthLargest2(self, nums, k):  
 for i in xrange(k):  
 for j in xrange(len(nums)-i-1):  
 if nums[j] > nums[j+1]:  
 # exchange elements, time consuming  
 nums[j], nums[j+1] = nums[j+1], nums[j]  
 return nums[len(nums)-k]  
   
# O(nk) time, selection sort idea  
def findKthLargest3(self, nums, k):  
 for i in xrange(len(nums), len(nums)-k, -1):  
 tmp = 0  
 for j in xrange(i):  
 if nums[j] > nums[tmp]:  
 tmp = j  
 nums[tmp], nums[i-1] = nums[i-1], nums[tmp]  
 return nums[len(nums)-k]  
   
# O(k+(n-k)lgk) time, min-heap  
def findKthLargest4(self, nums, k):  
 heap = []  
 for num in nums:  
 heapq.heappush(heap, num)  
 for \_ in xrange(len(nums)-k):  
 heapq.heappop(heap)  
 return heapq.heappop(heap)  
  
# O(k+(n-k)lgk) time, min-heap   
def findKthLargest5(self, nums, k):  
 return heapq.nlargest(k, nums)[k-1]  
   
# O(n) time, quick selection  
def findKthLargest(self, nums, k):  
 # convert the kth largest to smallest  
 return self.findKthSmallest(nums, len(nums)+1-k)  
   
def findKthSmallest(self, nums, k):  
 if nums:  
 pos = self.partition(nums, 0, len(nums)-1)  
 if k > pos+1:  
 return self.findKthSmallest(nums[pos+1:], k-pos-1)  
 elif k < pos+1:  
 return self.findKthSmallest(nums[:pos], k)  
 else:  
 return nums[pos]  
   
# choose the right-most element as pivot   
def partition(self, nums, l, r):  
 low = l  
 while l < r:  
 if nums[l] < nums[r]:  
 nums[l], nums[low] = nums[low], nums[l]  
 low += 1  
 l += 1  
 nums[low], nums[r] = nums[r], nums[low]  
 return low  
**Best\_solution:**Solution explained   
public int findKthLargest(int[] nums, int k) {  
 final int N = nums.length;  
 Arrays.sort(nums);  
 return nums[N - k];  
}  
  
  
*216,Combination Sum III:***Python\_solution:**Concise python solution using DFS   
class Solution:  
 # @param {integer} k  
 # @param {integer} n  
 # @return {integer[][]}  
 def combinationSum3(self, k, n):  
 if n > sum([i for i in range(1, 11)]):  
 return []  
  
 res = []  
 self.sum\_help(k, n, 1, [], res)  
 return res  
  
  
 def sum\_help(self, k, n, curr, arr, res):  
 if len(arr) == k:  
 if sum(arr) == n:  
 res.append(list(arr))  
 return  
  
 if len(arr) > k or curr > 9:  
 return  
   
 for i in range(curr, 10):  
 arr.append(i)  
 self.sum\_help(k, n, i + 1, arr, res)  
 arr.pop()  
**Best\_solution:**Simple and clean Java code, backtracking.   
 public List<List<Integer>> combinationSum3(int k, int n) {  
 List<List<Integer>> ans = new ArrayList<>();  
 combination(ans, new ArrayList<Integer>(), k, 1, n);  
 return ans;  
}  
  
private void combination(List<List<Integer>> ans, List<Integer> comb, int k, int start, int n) {  
 if (comb.size() == k && n == 0) {  
 List<Integer> li = new ArrayList<Integer>(comb);  
 ans.add(li);  
 return;  
 }  
 for (int i = start; i <= 9; i++) {  
 comb.add(i);  
 combination(ans, comb, k, i+1, n-i);  
 comb.remove(comb.size() - 1);  
 }  
}  
  
*217,Contains Duplicate:***Python\_solution:**One line solution in python   
class Solution(object):  
def containsDuplicate(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: bool  
 """  
 return len(nums) != len(set(nums))  
**Best\_solution:**Possible solutions.   
public boolean containsDuplicate(int[] nums) {  
  
 for(int i = 0; i < nums.length; i++) {  
 for(int j = i + 1; j < nums.length; j++) {  
 if(nums[i] == nums[j]) {  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
  
  
*218,The Skyline Problem:***Python\_solution:**14 line python code, straightforward & easy to understand   
class Solution(object):  
def getSkyline(self, buildings):  
 """  
 :type buildings: List[List[int]]  
 :rtype: List[List[int]]  
 """  
 def addsky(pos, hei):  
 if sky[-1][1] != hei:  
 sky.append([pos, hei])  
  
 sky = [[-1,0]]  
   
 # possible corner positions  
 position = set([b[0] for b in buildings] + [b[1] for b in buildings])  
   
 # live buildings  
 live = []  
   
 i = 0  
   
 for t in sorted(position):  
   
 # add the new buildings whose left side is lefter than position t  
 while i < len(buildings) and buildings[i][0] <= t:  
 heappush(live, (-buildings[i][2], buildings[i][1]))  
 i += 1  
   
 # remove the past buildings whose right side is lefter than position t  
 while live and live[0][1] <= t:  
 heappop(live)  
   
 # pick the highest existing building at this moment  
 h = -live[0][0] if live else 0  
 addsky(t, h)  
  
 return sky[1:]  
**Best\_solution:**(Guaranteed) Really Detailed and Good (Perfect) Explanation of The Skyline Problem   
None  
  
*219,Contains Duplicate II:***Python\_solution:**Python concise solution with dictionary.   
def containsNearbyDuplicate(self, nums, k):  
 dic = {}  
 for i, v in enumerate(nums):  
 if v in dic and i - dic[v] <= k:  
 return True  
 dic[v] = i  
 return False  
**Best\_solution:**Simple Java solution   
public boolean containsNearbyDuplicate(int[] nums, int k) {  
 Set<Integer> set = new HashSet<Integer>();  
 for(int i = 0; i < nums.length; i++){  
 if(i > k) set.remove(nums[i-k-1]);  
 if(!set.add(nums[i])) return true;  
 }  
 return false;  
 }  
  
*220,Contains Duplicate III:***Python\_solution:**Java/Python one pass solution, O(n) time O(n) space using buckets   
(1) the two in the same bucket  
(2) the two in neighbor buckets  
  
**Best\_solution:**AC O(N) solution in Java using buckets with explanation   
 public class Solution {  
 public boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {  
 if (k < 1 || t < 0) return false;  
 Map<Long, Long> map = new HashMap<>();  
 for (int i = 0; i < nums.length; i++) {  
 long remappedNum = (long) nums[i] - Integer.MIN\_VALUE;  
 long bucket = remappedNum / ((long) t + 1);  
 if (map.containsKey(bucket)  
 || (map.containsKey(bucket - 1) && remappedNum - map.get(bucket - 1) <= t)  
 || (map.containsKey(bucket + 1) && map.get(bucket + 1) - remappedNum <= t))  
 return true;  
 if (map.entrySet().size() >= k) {  
 long lastBucket = ((long) nums[i - k] - Integer.MIN\_VALUE) / ((long) t + 1);  
 map.remove(lastBucket);  
 }  
 map.put(bucket, remappedNum);  
 }  
 return false;  
 }  
}  
  
  
*221,Maximal Square:***Python\_solution:**9-lines Python DP solution with explaination   
def maximalSquare(self, matrix):  
 dp, maxArea = [[0 for \_1\_ in range(len(matrix[0]))] for \_\_\_ in range(len(matrix))], 0  
 for i in xrange(0, len(matrix)):  
 for j in xrange(0, len(matrix[0])):  
 if i == 0 or j == 0:  
 dp[i][j] = int(matrix[i][j])  
 elif int(matrix[i][j]) == 1:  
 dp[i][j] = min(dp[i - 1][j - 1], dp[i][j - 1], dp[i - 1][j]) + 1  
 maxArea = max(maxArea, dp[i][j])  
 return maxArea\*maxArea  
  
**Best\_solution:**Easy DP solution in C++ with detailed explanations (8ms, O(n^2) time and O(n) space)   
(i, j)  
  
*222,Count Complete Tree Nodes:***Python\_solution:**My python solution in O(lgn \* lgn) time   
 class Solution:  
 # @param {TreeNode} root  
 # @return {integer}  
 def countNodes(self, root):  
 if not root:  
 return 0  
 leftDepth = self.getDepth(root.left)  
 rightDepth = self.getDepth(root.right)  
 if leftDepth == rightDepth:  
 return pow(2, leftDepth) + self.countNodes(root.right)  
 else:  
 return pow(2, rightDepth) + self.countNodes(root.left)  
   
 def getDepth(self, root):  
 if not root:  
 return 0  
 return 1 + self.getDepth(root.left)  
**Best\_solution:**Concise Java solutions O(log(n)^2)   
class Solution {  
 int height(TreeNode root) {  
 return root == null ? -1 : 1 + height(root.left);  
 }  
 public int countNodes(TreeNode root) {  
 int h = height(root);  
 return h < 0 ? 0 :  
 height(root.right) == h-1 ? (1 << h) + countNodes(root.right)  
 : (1 << h-1) + countNodes(root.left);  
 }  
}  
  
  
*223,Rectangle Area:***Python\_solution:**My python solutions   
class Solution:  
 def computeArea(self, A, B, C, D, E, F, G, H):  
 areaA = (C - A) \* (D - B)  
 areaB = (G - E) \* (H - F)  
 l = max(0, min(C, G) - max(A, E))  
 h = max(0, min(D, H) - max(B, F))  
 return areaA + areaB - l \* h  
**Best\_solution:**Just another short way   
right  
  
*224,Basic Calculator:***Python\_solution:**Easy 18 lines C++, 16 lines Python   
+  
**Best\_solution:**Iterative Java solution with stack   
public int calculate(String s) {  
 Stack<Integer> stack = new Stack<Integer>();  
 int result = 0;  
 int number = 0;  
 int sign = 1;  
 for(int i = 0; i < s.length(); i++){  
 char c = s.charAt(i);  
 if(Character.isDigit(c)){  
 number = 10 \* number + (int)(c - '0');  
 }else if(c == '+'){  
 result += sign \* number;  
 number = 0;  
 sign = 1;  
 }else if(c == '-'){  
 result += sign \* number;  
 number = 0;  
 sign = -1;  
 }else if(c == '('){  
 //we push the result first, then sign;  
 stack.push(result);  
 stack.push(sign);  
 //reset the sign and result for the value in the parenthesis  
 sign = 1;   
 result = 0;  
 }else if(c == ')'){  
 result += sign \* number;   
 number = 0;  
 result \*= stack.pop(); //stack.pop() is the sign before the parenthesis  
 result += stack.pop(); //stack.pop() now is the result calculated before the parenthesis  
   
 }  
 }  
 if(number != 0) result += sign \* number;  
 return result;  
}  
  
*225,Implement Stack using Queues:***Python\_solution:**Concise 1 Queue - Java, C++, Python   
class Stack {  
 queue<int> q;  
public:  
 void push(int x) {  
 q.push(x);  
 for (int i=1; i<q.size(); i++) {  
 q.push(q.front());  
 q.pop();  
 }  
 }  
  
 void pop() {  
 q.pop();  
 }  
  
 int top() {  
 return q.front();  
 }  
  
 bool empty() {  
 return q.empty();  
 }  
};  
  
**Best\_solution:**A simple C++ solution   
class Stack {  
public:  
 queue<int> que;  
 // Push element x onto stack.  
 void push(int x) {  
 que.push(x);  
 for(int i=0;i<que.size()-1;++i){  
 que.push(que.front());  
 que.pop();  
 }  
 }  
  
 // Removes the element on top of the stack.  
 void pop() {  
 que.pop();  
 }  
  
 // Get the top element.  
 int top() {  
 return que.front();  
 }  
  
 // Return whether the stack is empty.  
 bool empty() {  
 return que.empty();  
 }  
};  
  
*226,Invert Binary Tree:***Python\_solution:**3-4 lines Python   
def invertTree(self, root):  
 if root:  
 root.left, root.right = self.invertTree(root.right), self.invertTree(root.left)  
 return root  
  
**Best\_solution:**Straightforward DFS recursive, iterative, BFS solutions   
public class Solution {  
 public TreeNode invertTree(TreeNode root) {  
   
 if (root == null) {  
 return null;  
 }  
  
 final TreeNode left = root.left,  
 right = root.right;  
 root.left = invertTree(right);  
 root.right = invertTree(left);  
 return root;  
 }  
}  
  
  
*227,Basic Calculator II:***Python\_solution:**Python short solution with stack.   
def calculate(self, s):  
 if not s:  
 return "0"  
 stack, num, sign = [], 0, "+"  
 for i in xrange(len(s)):  
 if s[i].isdigit():  
 num = num\*10+ord(s[i])-ord("0")  
 if (not s[i].isdigit() and not s[i].isspace()) or i == len(s)-1:  
 if sign == "-":  
 stack.append(-num)  
 elif sign == "+":  
 stack.append(num)  
 elif sign == "\*":  
 stack.append(stack.pop()\*num)  
 else:  
 tmp = stack.pop()  
 if tmp//num < 0 and tmp%num != 0:  
 stack.append(tmp//num+1)  
 else:  
 stack.append(tmp//num)  
 sign = s[i]  
 num = 0  
 return sum(stack)  
**Best\_solution:**Share my java solution   
public class Solution {  
public int calculate(String s) {  
 int len;  
 if(s==null || (len = s.length())==0) return 0;  
 Stack<Integer> stack = new Stack<Integer>();  
 int num = 0;  
 char sign = '+';  
 for(int i=0;i<len;i++){  
 if(Character.isDigit(s.charAt(i))){  
 num = num\*10+s.charAt(i)-'0';  
 }  
 if((!Character.isDigit(s.charAt(i)) &&' '!=s.charAt(i)) || i==len-1){  
 if(sign=='-'){  
 stack.push(-num);  
 }  
 if(sign=='+'){  
 stack.push(num);  
 }  
 if(sign=='\*'){  
 stack.push(stack.pop()\*num);  
 }  
 if(sign=='/'){  
 stack.push(stack.pop()/num);  
 }  
 sign = s.charAt(i);  
 num = 0;  
 }  
 }  
  
 int re = 0;  
 for(int i:stack){  
 re += i;  
 }  
 return re;  
}  
  
  
*228,Summary Ranges:***Python\_solution:**6 lines in Python   
def summaryRanges(self, nums):  
 ranges = []  
 for n in nums:  
 if not ranges or n > ranges[-1][-1] + 1:  
 ranges += [],  
 ranges[-1][1:] = n,  
 return ['->'.join(map(str, r)) for r in ranges]  
  
**Best\_solution:**Accepted JAVA solution--easy to understand   
List<String> list=new ArrayList();  
 if(nums.length==1){  
 list.add(nums[0]+"");  
 return list;  
 }  
 for(int i=0;i<nums.length;i++){  
 int a=nums[i];  
 while(i+1<nums.length&&(nums[i+1]-nums[i])==1){  
 i++;  
 }  
 if(a!=nums[i]){  
 list.add(a+"->"+nums[i]);  
 }else{  
 list.add(a+"");  
 }  
 }  
 return list;  
  
*229,Majority Element II:***Python\_solution:**Clear O(n) solution in python, no data structure or sort.   
class Solution:  
 # @param {integer[]} nums  
 # @return {integer[]}  
 def majorityElement(self, nums):  
 a, b, ca, cb = 0, 1, 0, 0  
 for num in nums:  
 if a == num:  
 ca += 1  
 elif b == num:  
 cb += 1  
 elif ca == 0:  
 a, ca = num, 1  
 elif cb == 0:  
 b, cb = num, 1  
 else:  
 ca -= 1  
 cb -= 1  
 ca = len([0 for num in nums if num == a])  
 cb = len([0 for num in nums if num == b])  
 res = []  
 if ca > len(nums) / 3:  
 res.append(a)  
 if cb > len(nums) / 3:  
 res.append(b)  
 return res  
**Best\_solution:**Boyer-Moore Majority Vote algorithm and my elaboration   
class Solution:  
# @param {integer[]} nums  
# @return {integer[]}  
def majorityElement(self, nums):  
 if not nums:  
 return []  
 count1, count2, candidate1, candidate2 = 0, 0, 0, 1  
 for n in nums:  
 if n == candidate1:  
 count1 += 1  
 elif n == candidate2:  
 count2 += 1  
 elif count1 == 0:  
 candidate1, count1 = n, 1  
 elif count2 == 0:  
 candidate2, count2 = n, 1  
 else:  
 count1, count2 = count1 - 1, count2 - 1  
 return [n for n in (candidate1, candidate2)  
 if nums.count(n) > len(nums) // 3]  
  
*230,Kth Smallest Element in a BST:***Python\_solution:**3 ways implemented in JAVA (Python): Binary Search, in-order iterative & recursive   
 public int kthSmallest(TreeNode root, int k) {  
 int count = countNodes(root.left);  
 if (k <= count) {  
 return kthSmallest(root.left, k);  
 } else if (k > count + 1) {  
 return kthSmallest(root.right, k-1-count); // 1 is counted as current node  
 }  
   
 return root.val;  
 }  
   
 public int countNodes(TreeNode n) {  
 if (n == null) return 0;  
   
 return 1 + countNodes(n.left) + countNodes(n.right);  
 }  
  
**Best\_solution:**3 ways implemented in JAVA (Python): Binary Search, in-order iterative & recursive   
 public int kthSmallest(TreeNode root, int k) {  
 int count = countNodes(root.left);  
 if (k <= count) {  
 return kthSmallest(root.left, k);  
 } else if (k > count + 1) {  
 return kthSmallest(root.right, k-1-count); // 1 is counted as current node  
 }  
   
 return root.val;  
 }  
   
 public int countNodes(TreeNode n) {  
 if (n == null) return 0;  
   
 return 1 + countNodes(n.left) + countNodes(n.right);  
 }  
  
  
*231,Power of Two:***Python\_solution:**Python one line solution   
class Solution(object):  
 def isPowerOfTwo(self, n):  
 """  
 :type n: int  
 :rtype: bool  
 """  
 return n > 0 and not (n & n-1)  
**Best\_solution:**Using n&(n-1) trick   
class Solution {  
public:  
 bool isPowerOfTwo(int n) {  
 if(n<=0) return false;  
 return !(n&(n-1));  
 }  
};  
  
*232,Implement Queue using Stacks:***Python\_solution:**Share my python solution (32ms)   
class Queue(object):  
 def \_\_init\_\_(self):  
 """  
 initialize your data structure here.  
 """  
 self.inStack, self.outStack = [], []  
  
 def push(self, x):  
 """  
 :type x: int  
 :rtype: nothing  
 """  
 self.inStack.append(x)  
  
 def pop(self):  
 """  
 :rtype: nothing  
 """  
 self.move()  
 self.outStack.pop()  
  
 def peek(self):  
 """  
 :rtype: int  
 """  
 self.move()  
 return self.outStack[-1]  
  
 def empty(self):  
 """  
 :rtype: bool  
 """  
 return (not self.inStack) and (not self.outStack)   
   
 def move(self):  
 """  
 :rtype nothing  
 """  
 if not self.outStack:  
 while self.inStack:  
 self.outStack.append(self.inStack.pop())  
**Best\_solution:**Short O(1) amortized, C++ / Java / Ruby   
peek  
  
*233,Number of Digit One:***Python\_solution:**4+ lines, O(log n), C++/Java/Python   
int countDigitOne(int n) {  
 int ones = 0;  
 for (long long m = 1; m <= n; m \*= 10)  
 ones += (n/m + 8) / 10 \* m + (n/m % 10 == 1) \* (n%m + 1);  
 return ones;  
}  
  
**Best\_solution:**4+ lines, O(log n), C++/Java/Python   
int countDigitOne(int n) {  
 int ones = 0;  
 for (long long m = 1; m <= n; m \*= 10)  
 ones += (n/m + 8) / 10 \* m + (n/m % 10 == 1) \* (n%m + 1);  
 return ones;  
}  
  
  
*234,Palindrome Linked List:***Python\_solution:**Python easy to understand solution with comments (operate nodes directly).   
def isPalindrome(self, head):  
 fast = slow = head  
 # find the mid node  
 while fast and fast.next:  
 fast = fast.next.next  
 slow = slow.next  
 # reverse the second half  
 node = None  
 while slow:  
 nxt = slow.next  
 slow.next = node  
 node = slow  
 slow = nxt  
 # compare the first and second half nodes  
 while node: # while node and head:  
 if node.val != head.val:  
 return False  
 node = node.next  
 head = head.next  
 return True  
**Best\_solution:**Reversing a list is not considered "O(1) space"   
# reverse in place  
def reverse(ls):  
 for i in xrange(len(ls)//2):  
 ls[i], ls[-(i+1)] = ls[-(i+1)], ls[i]  
 return ls  
  
*235,Lowest Common Ancestor of a Binary Search Tree:***Python\_solution:**Python Iterative Solution   
class Solution:  
  
def lowestCommonAncestor(self, root, p, q):  
 while root:  
 if root.val > p.val and root.val > q.val:  
 root = root.left  
 elif root.val < p.val and root.val < q.val:  
 root = root.right  
 else:  
 return root  
**Best\_solution:**3 lines with O(1) space, 1-Liners, Alternatives   
def lowestCommonAncestor(self, root, p, q):  
 while (root.val - p.val) \* (root.val - q.val) > 0:  
 root = (root.left, root.right)[p.val > root.val]  
 return root  
  
  
*236,Lowest Common Ancestor of a Binary Tree:***Python\_solution:**4 lines C++/Java/Python/Ruby   
TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {  
 if (!root || root == p || root == q) return root;  
 TreeNode\* left = lowestCommonAncestor(root->left, p, q);  
 TreeNode\* right = lowestCommonAncestor(root->right, p, q);  
 return !left ? right : !right ? left : root;  
}  
  
**Best\_solution:**4 lines C++/Java/Python/Ruby   
TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {  
 if (!root || root == p || root == q) return root;  
 TreeNode\* left = lowestCommonAncestor(root->left, p, q);  
 TreeNode\* right = lowestCommonAncestor(root->right, p, q);  
 return !left ? right : !right ? left : root;  
}  
  
  
*237,Delete Node in a Linked List:***Python\_solution:**1-3 lines, C++/Java/Python/C/C#/JavaScript/Ruby   
void deleteNode(ListNode\* node) {  
 \*node = \*node->next;  
}  
  
**Best\_solution:**1-3 lines, C++/Java/Python/C/C#/JavaScript/Ruby   
void deleteNode(ListNode\* node) {  
 \*node = \*node->next;  
}  
  
  
*238,Product of Array Except Self:***Python\_solution:**Python solution (Accepted), O(n) time, O(1) space   
class Solution:  
 # @param {integer[]} nums  
 # @return {integer[]}  
 def productExceptSelf(self, nums):  
 p = 1  
 n = len(nums)  
 output = []  
 for i in range(0,n):  
 output.append(p)  
 p = p \* nums[i]  
 p = 1  
 for i in range(n-1,-1,-1):  
 output[i] = output[i] \* p  
 p = p \* nums[i]  
 return output  
**Best\_solution:**Simple Java solution in O(n) without extra space   
public class Solution {  
public int[] productExceptSelf(int[] nums) {  
 int n = nums.length;  
 int[] res = new int[n];  
 res[0] = 1;  
 for (int i = 1; i < n; i++) {  
 res[i] = res[i - 1] \* nums[i - 1];  
 }  
 int right = 1;  
 for (int i = n - 1; i >= 0; i--) {  
 res[i] \*= right;  
 right \*= nums[i];  
 }  
 return res;  
}  
  
  
*239,Sliding Window Maximum:***Python\_solution:**9 lines Ruby, 11 lines Python, O(n)   
d  
**Best\_solution:**Java O(n) solution using deque with explanation   
public int[] maxSlidingWindow(int[] a, int k) {   
 if (a == null || k <= 0) {  
 return new int[0];  
 }  
 int n = a.length;  
 int[] r = new int[n-k+1];  
 int ri = 0;  
 // store index  
 Deque<Integer> q = new ArrayDeque<>();  
 for (int i = 0; i < a.length; i++) {  
 // remove numbers out of range k  
 while (!q.isEmpty() && q.peek() < i - k + 1) {  
 q.poll();  
 }  
 // remove smaller numbers in k range as they are useless  
 while (!q.isEmpty() && a[q.peekLast()] < a[i]) {  
 q.pollLast();  
 }  
 // q contains index... r contains content  
 q.offer(i);  
 if (i >= k - 1) {  
 r[ri++] = a[q.peek()];  
 }  
 }  
 return r;  
 }  
  
*240,Search a 2D Matrix II:***Python\_solution:**6-9 lines C++/Python Solutions with Explanations   
target  
**Best\_solution:**My concise O(m+n) Java solution   
public class Solution {  
 public boolean searchMatrix(int[][] matrix, int target) {  
 if(matrix == null || matrix.length < 1 || matrix[0].length <1) {  
 return false;  
 }  
 int col = matrix[0].length-1;  
 int row = 0;  
 while(col >= 0 && row <= matrix.length-1) {  
 if(target == matrix[row][col]) {  
 return true;  
 } else if(target < matrix[row][col]) {  
 col--;  
 } else if(target > matrix[row][col]) {  
 row++;  
 }  
 }  
 return false;  
 }  
}  
  
*241,Different Ways to Add Parentheses:***Python\_solution:**1-11 lines Python, 9 lines C++   
def diffWaysToCompute(self, input):  
 tokens = re.split('(\D)', input)  
 nums = map(int, tokens[::2])  
 ops = map({'+': operator.add, '-': operator.sub, '\*': operator.mul}.get, tokens[1::2])  
 def build(lo, hi):  
 if lo == hi:  
 return [nums[lo]]  
 return [ops[i](a, b)  
 for i in xrange(lo, hi)  
 for a in build(lo, i)  
 for b in build(i + 1, hi)]  
 return build(0, len(nums) - 1)  
  
**Best\_solution:**A recursive Java solution (284 ms)   
public class Solution {  
 public List<Integer> diffWaysToCompute(String input) {  
 List<Integer> ret = new LinkedList<Integer>();  
 for (int i=0; i<input.length(); i++) {  
 if (input.charAt(i) == '-' ||  
 input.charAt(i) == '\*' ||  
 input.charAt(i) == '+' ) {  
 String part1 = input.substring(0, i);  
 String part2 = input.substring(i+1);  
 List<Integer> part1Ret = diffWaysToCompute(part1);  
 List<Integer> part2Ret = diffWaysToCompute(part2);  
 for (Integer p1 : part1Ret) {  
 for (Integer p2 : part2Ret) {  
 int c = 0;  
 switch (input.charAt(i)) {  
 case '+': c = p1+p2;  
 break;  
 case '-': c = p1-p2;  
 break;  
 case '\*': c = p1\*p2;  
 break;  
 }  
 ret.add(c);  
 }  
 }  
 }  
 }  
 if (ret.size() == 0) {  
 ret.add(Integer.valueOf(input));  
 }  
 return ret;  
 }  
}  
  
*242,Valid Anagram:***Python\_solution:**Python solutions (sort and dictionary).   
def isAnagram1(self, s, t):  
 dic1, dic2 = {}, {}  
 for item in s:  
 dic1[item] = dic1.get(item, 0) + 1  
 for item in t:  
 dic2[item] = dic2.get(item, 0) + 1  
 return dic1 == dic2  
   
def isAnagram2(self, s, t):  
 dic1, dic2 = [0]\*26, [0]\*26  
 for item in s:  
 dic1[ord(item)-ord('a')] += 1  
 for item in t:  
 dic2[ord(item)-ord('a')] += 1  
 return dic1 == dic2  
   
def isAnagram3(self, s, t):  
 return sorted(s) == sorted(t)  
**Best\_solution:**Accepted Java O(n) solution in 5 lines   
public class Solution {  
 public boolean isAnagram(String s, String t) {  
 int[] alphabet = new int[26];  
 for (int i = 0; i < s.length(); i++) alphabet[s.charAt(i) - 'a']++;  
 for (int i = 0; i < t.length(); i++) alphabet[t.charAt(i) - 'a']--;  
 for (int i : alphabet) if (i != 0) return false;  
 return true;  
 }  
}  
  
*257,Binary Tree Paths:***Python\_solution:**Python solutions (dfs+stack, bfs+queue, dfs recursively).   
# dfs + stack  
def binaryTreePaths1(self, root):  
 if not root:  
 return []  
 res, stack = [], [(root, "")]  
 while stack:  
 node, ls = stack.pop()  
 if not node.left and not node.right:  
 res.append(ls+str(node.val))  
 if node.right:  
 stack.append((node.right, ls+str(node.val)+"->"))  
 if node.left:  
 stack.append((node.left, ls+str(node.val)+"->"))  
 return res  
   
# bfs + queue  
def binaryTreePaths2(self, root):  
 if not root:  
 return []  
 res, queue = [], collections.deque([(root, "")])  
 while queue:  
 node, ls = queue.popleft()  
 if not node.left and not node.right:  
 res.append(ls+str(node.val))  
 if node.left:  
 queue.append((node.left, ls+str(node.val)+"->"))  
 if node.right:  
 queue.append((node.right, ls+str(node.val)+"->"))  
 return res  
   
# dfs recursively  
def binaryTreePaths(self, root):  
 if not root:  
 return []  
 res = []  
 self.dfs(root, "", res)  
 return res  
  
def dfs(self, root, ls, res):  
 if not root.left and not root.right:  
 res.append(ls+str(root.val))  
 if root.left:  
 self.dfs(root.left, ls+str(root.val)+"->", res)  
 if root.right:  
 self.dfs(root.right, ls+str(root.val)+"->", res)  
**Best\_solution:**Accepted Java simple solution in 8 lines   
public List<String> binaryTreePaths(TreeNode root) {  
 List<String> answer = new ArrayList<String>();  
 if (root != null) searchBT(root, "", answer);  
 return answer;  
}  
private void searchBT(TreeNode root, String path, List<String> answer) {  
 if (root.left == null && root.right == null) answer.add(path + root.val);  
 if (root.left != null) searchBT(root.left, path + root.val + "->", answer);  
 if (root.right != null) searchBT(root.right, path + root.val + "->", answer);  
}  
  
*258,Add Digits:***Python\_solution:**3 methods for python with explains   
 class Solution(object):  
 def addDigits(self, num):  
 """  
 :type num: int  
 :rtype: int  
 """  
 while(num >= 10):  
 temp = 0  
 while(num > 0):  
 temp += num % 10  
 num /= 10  
 num = temp  
 return num  
  
**Best\_solution:**Accepted C++ O(1)-time O(1)-space 1-Line Solution with Detail Explanations   
https://en.wikipedia.org/wiki/Digital\_root#Congruence\_formula  
  
  
*260,Single Number III:***Python\_solution:**Easy Python O(n) - O(1) solution   
class Solution(object):  
 def singleNumber(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: List[int]  
 """  
 xor = 0  
 a = 0  
 b = 0  
 for num in nums:  
 xor ^= num  
 mask = 1  
 while(xor&mask == 0):  
 mask = mask << 1  
 for num in nums:  
 if num&mask:  
 a ^= num  
 else:  
 b ^= num  
 return [a, b]  
**Best\_solution:**Accepted C++/Java O(n)-time O(1)-space Easy Solution with Detail Explanations   
diff == numeric\_limits<int>::min()  
  
*262,Trips and Users:***Best\_solution:**Sharing my solution,   
select   
t.Request\_at Day,   
round(sum(case when t.Status like 'cancelled\_%' then 1 else 0 end)/count(\*),2) Rate  
from Trips t   
inner join Users u   
on t.Client\_Id = u.Users\_Id and u.Banned='No'  
where t.Request\_at between '2013-10-01' and '2013-10-03'  
group by t.Request\_at  
  
*263,Ugly Number:***Python\_solution:**My python solution   
def isUgly(self, num):  
 """  
 :type num: int  
 :rtype: bool  
 """  
 if num <= 0:  
 return False  
 for x in [2, 3, 5]:  
 while num % x == 0:  
 num = num / x  
 return num == 1  
**Best\_solution:**2-4 lines, every language   
for (int i=2; i<6 && num; i++)  
 while (num % i == 0)  
 num /= i;  
return num == 1;  
  
  
*264,Ugly Number II:***Python\_solution:**My expressive Python solution   
def nthUglyNumber(self, n):  
 ugly = [1]  
 i2, i3, i5 = 0, 0, 0  
 while n > 1:  
 u2, u3, u5 = 2 \* ugly[i2], 3 \* ugly[i3], 5 \* ugly[i5]  
 umin = min((u2, u3, u5))  
 if umin == u2:  
 i2 += 1  
 if umin == u3:  
 i3 += 1  
 if umin == u5:  
 i5 += 1  
 ugly.append(umin)  
 n -= 1  
 return ugly[-1]  
**Best\_solution:**My 16ms C++ DP solution with short explanation   
class Solution {  
public:  
 int nthUglyNumber(int n) {  
 if(n <= 0) return false; // get rid of corner cases   
 if(n == 1) return true; // base case  
 int t2 = 0, t3 = 0, t5 = 0; //pointers for 2, 3, 5  
 vector<int> k(n);  
 k[0] = 1;  
 for(int i = 1; i < n ; i ++)  
 {  
 k[i] = min(k[t2]\*2,min(k[t3]\*3,k[t5]\*5));  
 if(k[i] == k[t2]\*2) t2++;   
 if(k[i] == k[t3]\*3) t3++;  
 if(k[i] == k[t5]\*5) t5++;  
 }  
 return k[n-1];  
 }  
};  
  
*268,Missing Number:***Python\_solution:**1+ lines Ruby, Python, Java, C++   
def missing\_number(nums)  
 (n = nums.size) \* (n+1) / 2 - nums.reduce(:+)  
end  
  
**Best\_solution:**4 Line Simple Java Bit Manipulate Solution with Explaination   
public int missingNumber(int[] nums) {  
  
 int xor = 0, i = 0;  
 for (i = 0; i < nums.length; i++) {  
 xor = xor ^ i ^ nums[i];  
 }  
  
 return xor ^ i;  
}  
  
*273,Integer to English Words:***Python\_solution:**Recursive Python   
def numberToWords(self, num):  
 to19 = 'One Two Three Four Five Six Seven Eight Nine Ten Eleven Twelve ' \  
 'Thirteen Fourteen Fifteen Sixteen Seventeen Eighteen Nineteen'.split()  
 tens = 'Twenty Thirty Forty Fifty Sixty Seventy Eighty Ninety'.split()  
 def words(n):  
 if n < 20:  
 return to19[n-1:n]  
 if n < 100:  
 return [tens[n/10-2]] + words(n%10)  
 if n < 1000:  
 return [to19[n/100-1]] + ['Hundred'] + words(n%100)  
 for p, w in enumerate(('Thousand', 'Million', 'Billion'), 1):  
 if n < 1000\*\*(p+1):  
 return words(n/1000\*\*p) + [w] + words(n%1000\*\*p)  
 return ' '.join(words(num)) or 'Zero'  
**Best\_solution:**My clean Java solution, very easy to understand   
private final String[] LESS\_THAN\_20 = {"", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};  
private final String[] TENS = {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};  
private final String[] THOUSANDS = {"", "Thousand", "Million", "Billion"};  
  
public String numberToWords(int num) {  
 if (num == 0) return "Zero";  
  
 int i = 0;  
 String words = "";  
   
 while (num > 0) {  
 if (num % 1000 != 0)  
 words = helper(num % 1000) +THOUSANDS[i] + " " + words;  
 num /= 1000;  
 i++;  
 }  
   
 return words.trim();  
}  
  
private String helper(int num) {  
 if (num == 0)  
 return "";  
 else if (num < 20)  
 return LESS\_THAN\_20[num] + " ";  
 else if (num < 100)  
 return TENS[num / 10] + " " + helper(num % 10);  
 else  
 return LESS\_THAN\_20[num / 100] + " Hundred " + helper(num % 100);  
}  
  
*274,H-Index:***Python\_solution:**Python O(n lgn) time with sort, O(n) time with O(n) space   
def hIndex(self, citations):  
 citations.sort()  
 n = len(citations)  
 for i in xrange(n):  
 if citations[i] >= (n-i):  
 return n-i  
 return 0  
  
**Best\_solution:**My O(n) time solution use Java   
public class Solution {  
 // 9.3 70 years diaoZhaTian China jiaYou   
 public int hIndex(int[] citations) {  
 int length = citations.length;  
 if (length == 0) {  
 return 0;  
 }  
   
 int[] array2 = new int[length + 1];  
 for (int i = 0; i < length; i++) {  
 if (citations[i] > length) {  
 array2[length] += 1;  
 } else {  
 array2[citations[i]] += 1;  
 }  
 }  
 int t = 0;  
 int result = 0;  
  
 for (int i = length; i >= 0; i--) {  
 t = t + array2[i];  
 if (t >= i) {  
 return i;  
 }  
 }  
 return 0;  
 }  
}  
  
*275,H-Index II:***Python\_solution:**O(logN)-time O(1)-space Easy Solution with Detailed Explanations (C++/Java/Python)   
index  
**Best\_solution:**Standard binary search   
class Solution {  
public:  
 int hIndex(vector<int>& citations) {  
 int left=0, len = citations.size(), right= len-1, mid;  
 while(left<=right)  
 {  
 mid=(left+right)>>1;  
 if(citations[mid]== (len-mid)) return citations[mid];  
 else if(citations[mid] > (len-mid)) right = mid - 1;  
 else left = mid + 1;  
 }  
 return len - (right+1);  
 }  
};  
  
  
*278,First Bad Version:***Python\_solution:**1-liner in Ruby / Python   
def first\_bad\_version(n)  
 (1..n).bsearch { |i| is\_bad\_version(i) }  
end  
  
**Best\_solution:**O(lgN) simple Java solution   
public int firstBadVersion(int n) {  
 int start = 1, end = n;  
 while (start < end) {  
 int mid = start + (end-start) / 2;  
 if (!isBadVersion(mid)) start = mid + 1;  
 else end = mid;   
 }   
 return start;  
}  
  
*279,Perfect Squares:***Python\_solution:**Short Python solution using BFS   
def numSquares(self, n):  
 if n < 2:  
 return n  
 lst = []  
 i = 1  
 while i \* i <= n:  
 lst.append( i \* i )  
 i += 1  
 cnt = 0  
 toCheck = {n}  
 while toCheck:  
 cnt += 1  
 temp = set()  
 for x in toCheck:  
 for y in lst:  
 if x == y:  
 return cnt  
 if x < y:  
 break  
 temp.add(x-y)  
 toCheck = temp  
  
 return cnt  
  
**Best\_solution:**Summary of 4 different solutions (BFS, DP, static DP and mathematics)   
class Solution   
{  
public:  
 int numSquares(int n)   
 {  
 if (n <= 0)  
 {  
 return 0;  
 }  
   
 // cntPerfectSquares[i] = the least number of perfect square numbers   
 // which sum to i. Note that cntPerfectSquares[0] is 0.  
 vector<int> cntPerfectSquares(n + 1, INT\_MAX);  
 cntPerfectSquares[0] = 0;  
 for (int i = 1; i <= n; i++)  
 {  
 // For each i, it must be the sum of some number (i - j\*j) and   
 // a perfect square number (j\*j).  
 for (int j = 1; j\*j <= i; j++)  
 {  
 cntPerfectSquares[i] =   
 min(cntPerfectSquares[i], cntPerfectSquares[i - j\*j] + 1);  
 }  
 }  
   
 return cntPerfectSquares.back();  
 }  
};  
  
  
*282,Expression Add Operators:***Python\_solution:**Clean Python DFS with comments   
def addOperators(self, num, target):  
 res, self.target = [], target  
 for i in range(1,len(num)+1):  
 if i == 1 or (i > 1 and num[0] != "0"): # prevent "00\*" as a number  
 self.dfs(num[i:], num[:i], int(num[:i]), int(num[:i]), res) # this step put first number in the string  
 return res  
  
def dfs(self, num, temp, cur, last, res):  
 if not num:  
 if cur == self.target:  
 res.append(temp)  
 return  
 for i in range(1, len(num)+1):  
 val = num[:i]  
 if i == 1 or (i > 1 and num[0] != "0"): # prevent "00\*" as a number  
 self.dfs(num[i:], temp + "+" + val, cur+int(val), int(val), res)  
 self.dfs(num[i:], temp + "-" + val, cur-int(val), -int(val), res)  
 self.dfs(num[i:], temp + "\*" + val, cur-last+last\*int(val), last\*int(val), res)  
**Best\_solution:**Java Standard Backtrace AC Solutoin, short and clear   
public class Solution {  
 public List<String> addOperators(String num, int target) {  
 List<String> rst = new ArrayList<String>();  
 if(num == null || num.length() == 0) return rst;  
 helper(rst, "", num, target, 0, 0, 0);  
 return rst;  
 }  
 public void helper(List<String> rst, String path, String num, int target, int pos, long eval, long multed){  
 if(pos == num.length()){  
 if(target == eval)  
 rst.add(path);  
 return;  
 }  
 for(int i = pos; i < num.length(); i++){  
 if(i != pos && num.charAt(pos) == '0') break;  
 long cur = Long.parseLong(num.substring(pos, i + 1));  
 if(pos == 0){  
 helper(rst, path + cur, num, target, i + 1, cur, cur);  
 }  
 else{  
 helper(rst, path + "+" + cur, num, target, i + 1, eval + cur , cur);  
   
 helper(rst, path + "-" + cur, num, target, i + 1, eval -cur, -cur);  
   
 helper(rst, path + "\*" + cur, num, target, i + 1, eval - multed + multed \* cur, multed \* cur );  
 }  
 }  
 }  
}  
  
*283,Move Zeroes:***Python\_solution:**Share my one line python solution   
class Solution(object):  
 def moveZeroes(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: void Do not return anything, modify nums in-place instead.  
 """  
 length = len(nums);  
 lastIndex = 0;  
 for p1 in range(0,length) :  
 if nums[p1] != 0 :  
 nums[lastIndex] = nums[p1];  
 lastIndex = lastIndex + 1;  
 while lastIndex < length :  
 nums[lastIndex] = 0;  
 lastIndex = lastIndex + 1;  
**Best\_solution:**Simple O(N) Java Solution Using Insert Index   
// Shift non-zero values as far forward as possible  
// Fill remaining space with zeros  
  
public void moveZeroes(int[] nums) {  
 if (nums == null || nums.length == 0) return;   
  
 int insertPos = 0;  
 for (int num: nums) {  
 if (num != 0) nums[insertPos++] = num;  
 }   
  
 while (insertPos < nums.length) {  
 nums[insertPos++] = 0;  
 }  
}  
  
*284,Peeking Iterator:***Best\_solution:**Concise Java Solution   
class PeekingIterator implements Iterator<Integer> {   
 private Integer next = null;  
 private Iterator<Integer> iter;  
  
 public PeekingIterator(Iterator<Integer> iterator) {  
 // initialize any member here.  
 iter = iterator;  
 if (iter.hasNext())  
 next = iter.next();  
 }  
   
 // Returns the next element in the iteration without advancing the iterator.   
 public Integer peek() {  
 return next;   
 }  
  
 // hasNext() and next() should behave the same as in the Iterator interface.  
 // Override them if needed.  
 @Override  
 public Integer next() {  
 Integer res = next;  
 next = iter.hasNext() ? iter.next() : null;  
 return res;   
 }  
  
 @Override  
 public boolean hasNext() {  
 return next != null;  
 }  
}  
  
  
*287,Find the Duplicate Number:***Python\_solution:**Python same solution as #142 Linked List Cycle II   
def findDuplicate(self, nums):  
 slow = fast = finder = 0  
 while True:  
 slow = nums[slow]  
 fast = nums[nums[fast]]  
 if slow == fast:  
 while finder != slow:  
 finder = nums[finder]  
 slow = nums[slow]  
 return finder  
**Best\_solution:**My easy understood solution with O(n) time and O(1) space without modifying the array. With clear explanation.   
int findDuplicate3(vector<int>& nums)  
{  
 if (nums.size() > 1)  
 {  
 int slow = nums[0];  
 int fast = nums[nums[0]];  
 while (slow != fast)  
 {  
 slow = nums[slow];  
 fast = nums[nums[fast]];  
 }  
  
 fast = 0;  
 while (fast != slow)  
 {  
 fast = nums[fast];  
 slow = nums[slow];  
 }  
 return slow;  
 }  
 return -1;  
}  
  
*289,Game of Life:***Python\_solution:**Python solution, easy to understand..   
def gameOfLife(self, board):  
 m,n = len(board), len(board[0])  
 for i in range(m):  
 for j in range(n):  
 if board[i][j] == 0 or board[i][j] == 2:  
 if self.nnb(board,i,j) == 3:  
 board[i][j] = 2  
 else:  
 if self.nnb(board,i,j) < 2 or self.nnb(board,i,j) >3:  
 board[i][j] = 3  
 for i in range(m):  
 for j in range(n):  
 if board[i][j] == 2: board[i][j] = 1  
 if board[i][j] == 3: board[i][j] = 0  
   
def nnb(self, board, i, j):  
 m,n = len(board), len(board[0])  
 count = 0  
 if i-1 >= 0 and j-1 >= 0: count += board[i-1][j-1]%2  
 if i-1 >= 0: count += board[i-1][j]%2  
 if i-1 >= 0 and j+1 < n: count += board[i-1][j+1]%2  
 if j-1 >= 0: count += board[i][j-1]%2  
 if j+1 < n: count += board[i][j+1]%2  
 if i+1 < m and j-1 >= 0: count += board[i+1][j-1]%2  
 if i+1 < m: count += board[i+1][j]%2  
 if i+1 < m and j+1 < n: count += board[i+1][j+1]%2  
 return count  
**Best\_solution:**Easiest JAVA solution with explanation   
[2nd bit, 1st bit] = [next state, current state]  
  
- 00 dead (next) <- dead (current)  
- 01 dead (next) <- live (current)   
- 10 live (next) <- dead (current)   
- 11 live (next) <- live (current)   
  
  
*290,Word Pattern:***Python\_solution:**Short in Python   
def wordPattern(self, pattern, str):  
 s = pattern  
 t = str.split()  
 return map(s.find, s) == map(t.index, t)  
  
**Best\_solution:**8 lines simple Java   
public boolean wordPattern(String pattern, String str) {  
 String[] words = str.split(" ");  
 if (words.length != pattern.length())  
 return false;  
 Map index = new HashMap();  
 for (Integer i=0; i<words.length; ++i)  
 if (index.put(pattern.charAt(i), i) != index.put(words[i], i))  
 return false;  
 return true;  
}  
  
  
*292,Nim Game:***Best\_solution:**Theorem: all 4s shall be false   
n = 4  
  
*295,Find Median from Data Stream:***Python\_solution:**Short simple Java/C++/Python, O(log n) + O(1)   
small  
**Best\_solution:**Short simple Java/C++/Python, O(log n) + O(1)   
small  
  
*297,Serialize and Deserialize Binary Tree:***Python\_solution:**Recursive preorder, Python and C++, O(n)   
class Codec:  
  
 def serialize(self, root):  
 def doit(node):  
 if node:  
 vals.append(str(node.val))  
 doit(node.left)  
 doit(node.right)  
 else:  
 vals.append('#')  
 vals = []  
 doit(root)  
 return ' '.join(vals)  
  
 def deserialize(self, data):  
 def doit():  
 val = next(vals)  
 if val == '#':  
 return None  
 node = TreeNode(int(val))  
 node.left = doit()  
 node.right = doit()  
 return node  
 vals = iter(data.split())  
 return doit()  
  
**Best\_solution:**Easy to understand Java Solution   
public class Codec {  
 private static final String spliter = ",";  
 private static final String NN = "X";  
  
 // Encodes a tree to a single string.  
 public String serialize(TreeNode root) {  
 StringBuilder sb = new StringBuilder();  
 buildString(root, sb);  
 return sb.toString();  
 }  
  
 private void buildString(TreeNode node, StringBuilder sb) {  
 if (node == null) {  
 sb.append(NN).append(spliter);  
 } else {  
 sb.append(node.val).append(spliter);  
 buildString(node.left, sb);  
 buildString(node.right,sb);  
 }  
 }  
 // Decodes your encoded data to tree.  
 public TreeNode deserialize(String data) {  
 Deque<String> nodes = new LinkedList<>();  
 nodes.addAll(Arrays.asList(data.split(spliter)));  
 return buildTree(nodes);  
 }  
   
 private TreeNode buildTree(Deque<String> nodes) {  
 String val = nodes.remove();  
 if (val.equals(NN)) return null;  
 else {  
 TreeNode node = new TreeNode(Integer.valueOf(val));  
 node.left = buildTree(nodes);  
 node.right = buildTree(nodes);  
 return node;  
 }  
 }  
}  
  
*299,Bulls and Cows:***Python\_solution:**Python 3 lines solution   
Counter  
**Best\_solution:**One pass Java solution   
secret  
  
*300,Longest Increasing Subsequence:***Python\_solution:**Java/Python Binary search O(nlogn) time with explanation   
tails  
**Best\_solution:**Short Java solution using DP O(n log n)   
public class Solution {  
 public int lengthOfLIS(int[] nums) {   
 int[] dp = new int[nums.length];  
 int len = 0;  
  
 for(int x : nums) {  
 int i = Arrays.binarySearch(dp, 0, len, x);  
 if(i < 0) i = -(i + 1);  
 dp[i] = x;  
 if(i == len) len++;  
 }  
  
 return len;  
 }  
}  
  
*301,Remove Invalid Parentheses:***Python\_solution:**Short Python BFS   
eval  
**Best\_solution:**Easy, Short, Concise and Fast Java DFS 3 ms solution   
public List<String> removeInvalidParentheses(String s) {  
 List<String> ans = new ArrayList<>();  
 remove(s, ans, 0, 0, new char[]{'(', ')'});  
 return ans;  
}  
  
public void remove(String s, List<String> ans, int last\_i, int last\_j, char[] par) {  
 for (int stack = 0, i = last\_i; i < s.length(); ++i) {  
 if (s.charAt(i) == par[0]) stack++;  
 if (s.charAt(i) == par[1]) stack--;  
 if (stack >= 0) continue;  
 for (int j = last\_j; j <= i; ++j)  
 if (s.charAt(j) == par[1] && (j == last\_j || s.charAt(j - 1) != par[1]))  
 remove(s.substring(0, j) + s.substring(j + 1, s.length()), ans, i, j, par);  
 return;  
 }  
 String reversed = new StringBuilder(s).reverse().toString();  
 if (par[0] == '(') // finished left to right  
 remove(reversed, ans, 0, 0, new char[]{')', '('});  
 else // finished right to left  
 ans.add(reversed);  
}  
  
*303,Range Sum Query - Immutable:***Python\_solution:**5-lines C++, 4-lines Python   
accu  
**Best\_solution:**Java simple O(n) init and O(1) query solution   
int[] nums;  
  
public NumArray(int[] nums) {  
 for(int i = 1; i < nums.length; i++)  
 nums[i] += nums[i - 1];  
   
 this.nums = nums;  
}  
  
public int sumRange(int i, int j) {  
 if(i == 0)  
 return nums[j];  
   
 return nums[j] - nums[i - 1];  
}  
  
  
*304,Range Sum Query 2D - Immutable:***Python\_solution:**Sharing My Python solution   
class NumMatrix(object):  
 def \_\_init\_\_(self, matrix):  
 if matrix is None or not matrix:  
 return  
 n, m = len(matrix), len(matrix[0])  
 self.sums = [ [0 for j in xrange(m+1)] for i in xrange(n+1) ]  
 for i in xrange(1, n+1):  
 for j in xrange(1, m+1):  
 self.sums[i][j] = matrix[i-1][j-1] + self.sums[i][j-1] + self.sums[i-1][j] - self.sums[i-1][j-1]  
   
  
 def sumRegion(self, row1, col1, row2, col2):  
 row1, col1, row2, col2 = row1+1, col1+1, row2+1, col2+1  
 return self.sums[row2][col2] - self.sums[row2][col1-1] - self.sums[row1-1][col2] + self.sums[row1-1][col1-1]  
**Best\_solution:**Clean C++ Solution and Explaination - O(mn) space with O(1) time   
sums[row+1][col+1]  
  
*306,Additive Number:***Python\_solution:**Python solution   
def isAdditiveNumber(self, num):  
 n = len(num)  
 for i, j in itertools.combinations(range(1, n), 2):  
 a, b = num[:i], num[i:j]  
 if b != str(int(b)):  
 continue  
 while j < n:  
 c = str(int(a) + int(b))  
 if not num.startswith(c, j):  
 break  
 j += len(c)  
 a, b = b, c  
 if j == n:  
 return True  
 return False  
**Best\_solution:**Java Recursive and Iterative Solutions   
i  
  
*307,Range Sum Query - Mutable:***Python\_solution:**"0 lines" Python   
class NumArray(object):  
 def \_\_init\_\_(self, nums):  
 self.update = nums.\_\_setitem\_\_  
 self.sumRange = lambda i, j: sum(nums[i:j+1])  
  
**Best\_solution:**17 ms Java solution with segment tree   
public class NumArray {  
  
 class SegmentTreeNode {  
 int start, end;  
 SegmentTreeNode left, right;  
 int sum;  
  
 public SegmentTreeNode(int start, int end) {  
 this.start = start;  
 this.end = end;  
 this.left = null;  
 this.right = null;  
 this.sum = 0;  
 }  
 }  
   
 SegmentTreeNode root = null;  
   
 public NumArray(int[] nums) {  
 root = buildTree(nums, 0, nums.length-1);  
 }  
  
 private SegmentTreeNode buildTree(int[] nums, int start, int end) {  
 if (start > end) {  
 return null;  
 } else {  
 SegmentTreeNode ret = new SegmentTreeNode(start, end);  
 if (start == end) {  
 ret.sum = nums[start];  
 } else {  
 int mid = start + (end - start) / 2;   
 ret.left = buildTree(nums, start, mid);  
 ret.right = buildTree(nums, mid + 1, end);  
 ret.sum = ret.left.sum + ret.right.sum;  
 }   
 return ret;  
 }  
 }  
   
 void update(int i, int val) {  
 update(root, i, val);  
 }  
   
 void update(SegmentTreeNode root, int pos, int val) {  
 if (root.start == root.end) {  
 root.sum = val;  
 } else {  
 int mid = root.start + (root.end - root.start) / 2;  
 if (pos <= mid) {  
 update(root.left, pos, val);  
 } else {  
 update(root.right, pos, val);  
 }  
 root.sum = root.left.sum + root.right.sum;  
 }  
 }  
  
 public int sumRange(int i, int j) {  
 return sumRange(root, i, j);  
 }  
   
 public int sumRange(SegmentTreeNode root, int start, int end) {  
 if (root.end == end && root.start == start) {  
 return root.sum;  
 } else {  
 int mid = root.start + (root.end - root.start) / 2;  
 if (end <= mid) {  
 return sumRange(root.left, start, end);  
 } else if (start >= mid+1) {  
 return sumRange(root.right, start, end);  
 } else {   
 return sumRange(root.right, mid+1, end) + sumRange(root.left, start, mid);  
 }  
 }  
 }  
}  
  
*309,Best Time to Buy and Sell Stock with Cooldown:***Python\_solution:**4-line Python solution, 52 ms   
notHold (stock)  
**Best\_solution:**Share my thinking process   
buy  
  
*310,Minimum Height Trees:***Python\_solution:**Share my Accepted BFS Python Code with O(n) Time   
def findMinHeightTrees(self, n, edges):  
 """  
 :type n: int  
 :type edges: List[List[int]]  
 :rtype: List[int]  
 """  
 if n == 1: return [0]  
 neighbors = collections.defaultdict(list)  
 degrees = collections.defaultdict(int)  
 for u, v in edges:  
 neighbors[u].append(v)  
 neighbors[v].append(u)  
 degrees[u] += 1  
 degrees[v] += 1  
   
 # First find the leaves  
 preLevel, unvisited = [], set(range(n))  
 for i in range(n):  
 if degrees[i] == 1: preLevel.append(i)  
   
 while len(unvisited) > 2:  
 thisLevel = []  
 for u in preLevel:  
 unvisited.remove(u)  
 for v in neighbors[u]:  
 if v in unvisited:   
 degrees[v] -= 1  
 if degrees[v] == 1: thisLevel += [v]  
 preLevel = thisLevel  
   
 return preLevel  
**Best\_solution:**Share some thoughts   
n  
  
*312,Burst Balloons:***Python\_solution:**Python DP N^3 Solutions   
dp[i][j] = max(dp[i][j], nums[i] \* nums[k] \* nums[j] + dp[i][k] + dp[k][j]) # i < k < j  
  
**Best\_solution:**Share some analysis and explanations   
nums[i-1]\*nums[i]\*nums[i+1]  
  
*313,Super Ugly Number:***Python\_solution:**Python, generators on a heap   
heapq.merge  
**Best\_solution:**Java three methods, 23ms, 36 ms, 58ms(with heap), performance explained   
public int nthSuperUglyNumberI(int n, int[] primes) {  
 int[] ugly = new int[n];  
 int[] idx = new int[primes.length];  
  
 ugly[0] = 1;  
 for (int i = 1; i < n; i++) {  
 //find next  
 ugly[i] = Integer.MAX\_VALUE;  
 for (int j = 0; j < primes.length; j++)  
 ugly[i] = Math.min(ugly[i], primes[j] \* ugly[idx[j]]);  
   
 //slip duplicate  
 for (int j = 0; j < primes.length; j++) {  
 while (primes[j] \* ugly[idx[j]] <= ugly[i]) idx[j]++;  
 }  
 }  
  
 return ugly[n - 1];  
}  
  
  
*315,Count of Smaller Numbers After Self:***Python\_solution:**3 ways (Segment Tree, Binary Indexed Tree, Binary Search Tree) clean python code   
class SegmentTreeNode(object):  
 def \_\_init\_\_(self, val, start, end):  
 self.val = val  
 self.start = start  
 self.end = end  
 self.children = []  
  
  
class SegmentTree(object):  
 def \_\_init\_\_(self, n):  
 self.root = self.build(0, n - 1)  
  
 def build(self, start, end):  
 if start > end:  
 return  
  
 root = SegmentTreeNode(0, start, end)  
 if start == end:  
 return root  
  
 mid = start + end >> 1  
 root.children = filter(None, [  
 self.build(start, end)  
 for start, end in ((start, mid), (mid + 1, end))])  
 return root  
  
 def update(self, i, val, root=None):  
 root = root or self.root  
 if i < root.start or i > root.end:  
 return root.val  
  
 if i == root.start == root.end:  
 root.val += val  
 return root.val  
  
 root.val = sum([self.update(i, val, c) for c in root.children])  
 return root.val  
  
 def sum(self, start, end, root=None):  
 root = root or self.root  
 if end < root.start or start > root.end:  
 return 0  
  
 if start <= root.start and end >= root.end:  
 return root.val  
  
 return sum([self.sum(start, end, c) for c in root.children])  
  
  
class Solution(object):  
 def countSmaller(self, nums):  
 hashTable = {v: i for i, v in enumerate(sorted(set(nums)))}  
  
 tree, r = SegmentTree(len(hashTable)), []  
 for i in xrange(len(nums) - 1, -1, -1):  
 r.append(tree.sum(0, hashTable[nums[i]] - 1))  
 tree.update(hashTable[nums[i]], 1)  
 return r[::-1]  
  
**Best\_solution:**9ms short Java BST solution get answer when building BST   
 1(0, 1)  
 \  
 6(3, 1)  
 /  
 2(0, 2)  
 \  
 3(0, 1)  
  
  
*316,Remove Duplicate Letters:***Python\_solution:**Some Python solutions   
def removeDuplicateLetters(self, s):  
 for c in sorted(set(s)):  
 suffix = s[s.index(c):]  
 if set(suffix) == set(s):  
 return c + self.removeDuplicateLetters(suffix.replace(c, ''))  
 return ''  
  
**Best\_solution:**A short O(n) recursive greedy solution   
public class Solution {  
 public String removeDuplicateLetters(String s) {  
 int[] cnt = new int[26];  
 int pos = 0; // the position for the smallest s[i]  
 for (int i = 0; i < s.length(); i++) cnt[s.charAt(i) - 'a']++;  
 for (int i = 0; i < s.length(); i++) {  
 if (s.charAt(i) < s.charAt(pos)) pos = i;  
 if (--cnt[s.charAt(i) - 'a'] == 0) break;  
 }  
 return s.length() == 0 ? "" : s.charAt(pos) + removeDuplicateLetters(s.substring(pos + 1).replaceAll("" + s.charAt(pos), ""));  
 }  
}  
  
*318,Maximum Product of Word Lengths:***Python\_solution:**Python solution, beats 99.67%   
class Solution(object):  
 def maxProduct(self, words):  
 d = {}  
 for w in words:  
 mask = 0  
 for c in set(w):  
 mask |= (1 << (ord(c) - 97))  
 d[mask] = max(d.get(mask, 0), len(w))  
 return max([d[x] \* d[y] for x in d for y in d if not x & y] or [0])  
**Best\_solution:**JAVA----------Easy Version To Understand!!!!!!!!!!!!!!!!!   
 public static int maxProduct(String[] words) {  
 if (words == null || words.length == 0)  
 return 0;  
 int len = words.length;  
 int[] value = new int[len];  
 for (int i = 0; i < len; i++) {  
 String tmp = words[i];  
 value[i] = 0;  
 for (int j = 0; j < tmp.length(); j++) {  
 value[i] |= 1 << (tmp.charAt(j) - 'a');  
 }  
 }  
 int maxProduct = 0;  
 for (int i = 0; i < len; i++)  
 for (int j = i + 1; j < len; j++) {  
 if ((value[i] & value[j]) == 0 && (words[i].length() \* words[j].length() > maxProduct))  
 maxProduct = words[i].length() \* words[j].length();  
 }  
 return maxProduct;  
}  
  
*319,Bulb Switcher:***Best\_solution:**Math solution..   
int bulbSwitch(int n) {  
 return sqrt(n);  
}  
  
  
*321,Create Maximum Number:***Python\_solution:**Short Python / Ruby / C++   
def maxNumber(self, nums1, nums2, k):  
  
 def prep(nums, k):  
 drop = len(nums) - k  
 out = []  
 for num in nums:  
 while drop and out and out[-1] < num:  
 out.pop()  
 drop -= 1  
 out.append(num)  
 return out[:k]  
  
 def merge(a, b):  
 return [max(a, b).pop(0) for \_ in a+b]  
  
 return max(merge(prep(nums1, i), prep(nums2, k-i))  
 for i in range(k+1)  
 if i <= len(nums1) and k-i <= len(nums2))  
  
**Best\_solution:**Share my greedy solution   
public int[] maxNumber(int[] nums1, int[] nums2, int k) {  
 int n = nums1.length;  
 int m = nums2.length;  
 int[] ans = new int[k];  
 for (int i = Math.max(0, k - m); i <= k && i <= n; ++i) {  
 int[] candidate = merge(maxArray(nums1, i), maxArray(nums2, k - i), k);  
 if (greater(candidate, 0, ans, 0)) ans = candidate;  
 }  
 return ans;  
}  
private int[] merge(int[] nums1, int[] nums2, int k) {  
 int[] ans = new int[k];  
 for (int i = 0, j = 0, r = 0; r < k; ++r)  
 ans[r] = greater(nums1, i, nums2, j) ? nums1[i++] : nums2[j++];  
 return ans;  
}  
public boolean greater(int[] nums1, int i, int[] nums2, int j) {  
 while (i < nums1.length && j < nums2.length && nums1[i] == nums2[j]) {  
 i++;  
 j++;  
 }  
 return j == nums2.length || (i < nums1.length && nums1[i] > nums2[j]);  
}  
public int[] maxArray(int[] nums, int k) {  
 int n = nums.length;  
 int[] ans = new int[k];  
 for (int i = 0, j = 0; i < n; ++i) {  
 while (n - i + j > k && j > 0 && ans[j - 1] < nums[i]) j--;  
 if (j < k) ans[j++] = nums[i];  
 }  
 return ans;  
}  
  
  
*322,Coin Change:***Python\_solution:**Fast Python BFS Solution   
class Solution(object):  
 def coinChange(self, coins, amount):  
 """  
 :type coins: List[int]  
 :type amount: int  
 :rtype: int  
 """  
 if amount == 0:  
 return 0  
 value1 = [0]  
 value2 = []  
 nc = 0  
 visited = [False]\*(amount+1)  
 visited[0] = True  
 while value1:  
 nc += 1  
 for v in value1:  
 for coin in coins:  
 newval = v + coin  
 if newval == amount:  
 return nc  
 elif newval > amount:  
 continue  
 elif not visited[newval]:  
 visited[newval] = True  
 value2.append(newval)  
 value1, value2 = value2, []  
 return -1  
  
**Best\_solution:**[C++] O(n\*amount) time O(amount) space DP solution   
class Solution {  
public:  
 int coinChange(vector<int>& coins, int amount) {  
 int Max = amount + 1;  
 vector<int> dp(amount + 1, Max);  
 dp[0] = 0;  
 for (int i = 1; i <= amount; i++) {  
 for (int j = 0; j < coins.size(); j++) {  
 if (coins[j] <= i) {  
 dp[i] = min(dp[i], dp[i - coins[j]] + 1);  
 }  
 }  
 }  
 return dp[amount] > amount ? -1 : dp[amount];  
 }  
};  
  
*324,Wiggle Sort II:***Python\_solution:**3 lines Python, with Explanation / Proof   
def wiggleSort(self, nums):  
 nums.sort()  
 half = len(nums[::2])  
 nums[::2], nums[1::2] = nums[:half][::-1], nums[half:][::-1]  
  
**Best\_solution:**O(n)+O(1) after median --- Virtual Indexing   
void wiggleSort(vector<int>& nums) {  
 int n = nums.size();  
   
 // Find a median.  
 auto midptr = nums.begin() + n / 2;  
 nth\_element(nums.begin(), midptr, nums.end());  
 int mid = \*midptr;  
   
 // Index-rewiring.  
 #define A(i) nums[(1+2\*(i)) % (n|1)]  
  
 // 3-way-partition-to-wiggly in O(n) time with O(1) space.  
 int i = 0, j = 0, k = n - 1;  
 while (j <= k) {  
 if (A(j) > mid)  
 swap(A(i++), A(j++));  
 else if (A(j) < mid)  
 swap(A(j), A(k--));  
 else  
 j++;  
 }  
}  
  
  
*326,Power of Three:***Python\_solution:**Python O(1) Solution 96.6%   
class Solution(object):  
 def isPowerOfThree(self, n):  
 return n > 0 and 1162261467 % n == 0  
**Best\_solution:**1 line java solution without loop / recursion   
public class Solution {  
public boolean isPowerOfThree(int n) {  
 // 1162261467 is 3^19, 3^20 is bigger than int   
 return ( n>0 && 1162261467%n==0);  
}  
  
  
*327,Count of Range Sum:***Best\_solution:**Share my solution   
S[i]  
  
*328,Odd Even Linked List:***Python\_solution:**Clear Python Solution   
def oddEvenList(self, head):  
 dummy1 = odd = ListNode(0)  
 dummy2 = even = ListNode(0)  
 while head:  
 odd.next = head  
 even.next = head.next  
 odd = odd.next  
 even = even.next  
 head = head.next.next if even else None  
 odd.next = dummy2.next  
 return dummy1.next  
**Best\_solution:**Simple O(N) time, O(1), space Java solution.   
public class Solution {  
public ListNode oddEvenList(ListNode head) {  
 if (head != null) {  
   
 ListNode odd = head, even = head.next, evenHead = even;   
   
 while (even != null && even.next != null) {  
 odd.next = odd.next.next;   
 even.next = even.next.next;   
 odd = odd.next;  
 even = even.next;  
 }  
 odd.next = evenHead;   
 }  
 return head;  
}}  
  
*329,Longest Increasing Path in a Matrix:***Python\_solution:**Python solution, memoization dp, 288ms   
dp  
**Best\_solution:**15ms Concise Java Solution   
DFS  
  
*330,Patching Array:***Python\_solution:**Simple 9-line Python Solution   
class Solution(object):  
 def minPatches(self, nums, n):  
 """  
 :type nums: List[int]  
 :type n: int  
 :rtype: int  
 """  
 miss, i, added = 1, 0, 0  
 while miss <= n:  
 if i < len(nums) and nums[i] <= miss:  
 miss += nums[i]  
 i += 1  
 else:  
 miss += miss  
 added += 1  
 return added  
**Best\_solution:**Solution + explanation   
int minPatches(vector<int>& nums, int n) {  
 long miss = 1, added = 0, i = 0;  
 while (miss <= n) {  
 if (i < nums.size() && nums[i] <= miss) {  
 miss += nums[i++];  
 } else {  
 miss += miss;  
 added++;  
 }  
 }  
 return added;  
}  
  
  
*331,Verify Preorder Serialization of a Binary Tree:***Python\_solution:**The simplest python solution with explanation (no stack, no recursion)   
class Solution(object):  
 def isValidSerialization(self, preorder):  
 """  
 :type preorder: str  
 :rtype: bool  
 """  
 # remember how many empty slots we have  
 # non-null nodes occupy one slot but create two new slots  
 # null nodes occupy one slot  
   
 p = preorder.split(',')  
   
 #initially we have one empty slot to put the root in it  
 slot = 1  
 for node in p:  
   
 # no empty slot to put the current node  
 if slot == 0:  
 return False  
   
 # a null node?  
 if node == '#':  
 # ocuppy slot  
 slot -= 1  
 else:  
 # create new slot  
 slot += 1  
   
 #we don't allow empty slots at the end  
 return slot==0  
**Best\_solution:**7 lines Easy Java Solution   
diff  
  
*332,Reconstruct Itinerary:***Python\_solution:**Short Ruby / Python / Java / C++   
def find\_itinerary(tickets)  
 tickets = tickets.sort.reverse.group\_by(&:first)  
 route = []  
 visit = -> airport {  
 visit[tickets[airport].pop()[1]] while (tickets[airport] || []).any?  
 route << airport  
 }  
 visit["JFK"]  
 route.reverse  
end  
  
**Best\_solution:**Short Ruby / Python / Java / C++   
def find\_itinerary(tickets)  
 tickets = tickets.sort.reverse.group\_by(&:first)  
 route = []  
 visit = -> airport {  
 visit[tickets[airport].pop()[1]] while (tickets[airport] || []).any?  
 route << airport  
 }  
 visit["JFK"]  
 route.reverse  
end  
  
  
*334,Increasing Triplet Subsequence:***Python\_solution:**Python Easy O(n) Solution   
def increasingTriplet(nums):  
 first = second = float('inf')  
 for n in nums:  
 if n <= first:  
 first = n  
 elif n <= second:  
 second = n  
 else:  
 return True  
 return False  
**Best\_solution:**Clean and short, with comments, C++   
bool increasingTriplet(vector<int>& nums) {  
 int c1 = INT\_MAX, c2 = INT\_MAX;  
 for (int x : nums) {  
 if (x <= c1) {  
 c1 = x; // c1 is min seen so far (it's a candidate for 1st element)  
 } else if (x <= c2) { // here when x > c1, i.e. x might be either c2 or c3  
 c2 = x; // x is better than the current c2, store it  
 } else { // here when we have/had c1 < c2 already and x > c2  
 return true; // the increasing subsequence of 3 elements exists  
 }  
 }  
 return false;  
}  
  
*335,Self Crossing:***Python\_solution:**Another python...   
def isSelfCrossing(self, x):  
 return any(d >= b > 0 and (a >= c or a >= c-e >= 0 and f >= d-b)  
 for a, b, c, d, e, f in ((x[i:i+6] + [0] \* 6)[:6]  
 for i in xrange(len(x))))  
  
**Best\_solution:**Java Oms with explanation   
// Categorize the self-crossing scenarios, there are 3 of them:   
// 1. Fourth line crosses first line and works for fifth line crosses second line and so on...  
// 2. Fifth line meets first line and works for the lines after  
// 3. Sixth line crosses first line and works for the lines after  
public class Solution {  
 public boolean isSelfCrossing(int[] x) {  
 int l = x.length;  
 if(l <= 3) return false;  
   
 for(int i = 3; i < l; i++){  
 if(x[i] >= x[i-2] && x[i-1] <= x[i-3]) return true; //Fourth line crosses first line and onward  
 if(i >=4)  
 {  
 if(x[i-1] == x[i-3] && x[i] + x[i-4] >= x[i-2]) return true; // Fifth line meets first line and onward  
 }  
 if(i >=5)  
 {  
 if(x[i-2] - x[i-4] >= 0 && x[i] >= x[i-2] - x[i-4] && x[i-1] >= x[i-3] - x[i-5] && x[i-1] <= x[i-3]) return true; // Sixth line crosses first line and onward  
 }  
 }  
 return false;  
 }  
}  
  
*336,Palindrome Pairs:***Python\_solution:**Python solution~   
 wordict = {}  
 res = []   
 for i in range(len(words)):  
 wordict[words[i]] = i  
 for i in range(len(words)):  
 for j in range(len(words[i])+1):  
 tmp1 = words[i][:j]  
 tmp2 = words[i][j:]  
 if tmp1[::-1] in wordict and wordict[tmp1[::-1]]!=i and tmp2 == tmp2[::-1]:  
 res.append([i,wordict[tmp1[::-1]]])  
 if j!=0 and tmp2[::-1] in wordict and wordict[tmp2[::-1]]!=i and tmp1 == tmp1[::-1]:  
 res.append([wordict[tmp2[::-1]],i])  
   
 return res  
**Best\_solution:**150 ms 45 lines JAVA solution   
public List<List<Integer>> palindromePairs(String[] words) {  
 List<List<Integer>> ret = new ArrayList<>();   
 if (words == null || words.length < 2) return ret;  
 Map<String, Integer> map = new HashMap<String, Integer>();  
 for (int i=0; i<words.length; i++) map.put(words[i], i);  
 for (int i=0; i<words.length; i++) {  
 // System.out.println(words[i]);  
 for (int j=0; j<=words[i].length(); j++) { // notice it should be "j <= words[i].length()"  
 String str1 = words[i].substring(0, j);  
 String str2 = words[i].substring(j);  
 if (isPalindrome(str1)) {  
 String str2rvs = new StringBuilder(str2).reverse().toString();  
 if (map.containsKey(str2rvs) && map.get(str2rvs) != i) {  
 List<Integer> list = new ArrayList<Integer>();  
 list.add(map.get(str2rvs));  
 list.add(i);  
 ret.add(list);  
 // System.out.printf("isPal(str1): %s\n", list.toString());  
 }  
 }  
 if (isPalindrome(str2)) {  
 String str1rvs = new StringBuilder(str1).reverse().toString();  
 // check "str.length() != 0" to avoid duplicates  
 if (map.containsKey(str1rvs) && map.get(str1rvs) != i && str2.length()!=0) {   
 List<Integer> list = new ArrayList<Integer>();  
 list.add(i);  
 list.add(map.get(str1rvs));  
 ret.add(list);  
 // System.out.printf("isPal(str2): %s\n", list.toString());  
 }  
 }  
 }  
 }  
 return ret;  
}  
  
private boolean isPalindrome(String str) {  
 int left = 0;  
 int right = str.length() - 1;  
 while (left <= right) {  
 if (str.charAt(left++) != str.charAt(right--)) return false;  
 }  
 return true;  
}  
  
  
*337,House Robber III:***Python\_solution:**C++, JAVA, PYTHON & explanation   
f1(node)  
**Best\_solution:**Step by step tackling of the problem   
root  
  
*338,Counting Bits:***Python\_solution:**Simple Python Solution   
def countBits(self, num):  
 """  
 :type num: int  
 :rtype: List[int]  
 """  
   
 iniArr = [0]  
 if num > 0:  
 amountToAdd = 1  
 while len(iniArr) < num + 1:  
 iniArr.extend([x+1 for x in iniArr])  
   
 return iniArr[0:num+1]  
  
**Best\_solution:**Three-Line Java Solution   
public int[] countBits(int num) {  
 int[] f = new int[num + 1];  
 for (int i=1; i<=num; i++) f[i] = f[i >> 1] + (i & 1);  
 return f;  
}  
  
*341,Flatten Nested List Iterator:***Python\_solution:**Real iterator in Python, Java, C++   
hasNext  
**Best\_solution:**Simple Java solution using a stack with explanation   
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 curr = stack.peek();  
 if(curr.isInteger()) {  
 return true;  
 }  
 stack.pop();  
 for(int i = curr.getList().size() - 1; i >= 0; i--) {  
 stack.push(curr.getList().get(i));  
 }  
 }  
 return false;  
 }  
}  
  
*342,Power of Four:***Python\_solution:**Python one line solution with explanations   
def isPowerOfFour(self, num):  
 return num != 0 and num &(num-1) == 0 and num & 1431655765== num  
  
**Best\_solution:**Java 1-line (cheating for the purpose of not using loops)   
 public boolean isPowerOfFour(int num) {  
 return num > 0 && (num&(num-1)) == 0 && (num & 0x55555555) != 0;  
 //0x55555555 is to get rid of those power of 2 but not power of 4  
 //so that the single 1 bit always appears at the odd position   
 }  
  
*343,Integer Break:***Python\_solution:**Python solution (40ms) with explanation   
class Solution(object):  
 def integerBreak(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 if n == 2:  
 return 1  
 if n == 3:  
 return 2  
 list\_3 = [3] \* (n/3) # generate a list of 3  
 mod\_3 = n%3  
 if mod\_3 == 1: # if a 1 is left, then add it to the first element to get a 4  
 list\_3[0] += 1  
 if mod\_3 == 2: # if a 2 is left, then put it into the list  
 list\_3.append(2)  
 return reduce(lambda a, b: a\*b, list\_3)  
  
**Best\_solution:**Why factor 2 or 3? The math behind this problem.   
None  
  
*344,Reverse String:***Python\_solution:**Python solution   
 class Solution(object):  
 def reverseString(self, s):  
 """  
 :type s: str  
 :rtype: str  
 """  
 return s[::-1]  
**Best\_solution:**[JAVA] Simple and Clean with Explanations [6 Solutions]   
public class Solution {  
 public String reverseString(String s) {  
 char[] word = s.toCharArray();  
 int i = 0;  
 int j = s.length() - 1;  
 while (i < j) {  
 char temp = word[i];  
 word[i] = word[j];  
 word[j] = temp;  
 i++;  
 j--;  
 }  
 return new String(word);  
 }  
}  
  
  
*345,Reverse Vowels of a String:***Python\_solution:**1-2 lines Python/Ruby   
def reverse\_vowels(s)  
 vowels = s.scan(/[aeiou]/i)  
 s.gsub(/[aeiou]/i) { vowels.pop }  
end  
  
**Best\_solution:**Java Standard Two Pointer Solution   
public class Solution {  
public String reverseVowels(String s) {  
 if(s == null || s.length()==0) return s;  
 String vowels = "aeiouAEIOU";  
 char[] chars = s.toCharArray();  
 int start = 0;  
 int end = s.length()-1;  
 while(start<end){  
   
 while(start<end && !vowels.contains(chars[start]+"")){  
 start++;  
 }  
   
 while(start<end && !vowels.contains(chars[end]+"")){  
 end--;  
 }  
   
 char temp = chars[start];  
 chars[start] = chars[end];  
 chars[end] = temp;  
   
 start++;  
 end--;  
 }  
 return new String(chars);  
}  
  
  
*347,Top K Frequent Elements:***Python\_solution:**1-line Python Solution using Counter with explanation   
import collections  
  
class Solution(object):  
 def topKFrequent(self, nums, k):  
 """  
 :type nums: List[int]  
 :type k: int  
 :rtype: List[int]  
 """  
 # Use Counter to extract the top k frequent elements  
 # most\_common(k) return a list of tuples, where the first item of the tuple is the element,  
 # and the second item of the tuple is the count  
 # Thus, the built-in zip function could be used to extract the first item from the tuples  
 return zip(\*collections.Counter(nums).most\_common(k))[0]  
**Best\_solution:**Java O(n) Solution - Bucket Sort   
public List<Integer> topKFrequent(int[] nums, int k) {  
  
 List<Integer>[] bucket = new List[nums.length + 1];  
 Map<Integer, Integer> frequencyMap = new HashMap<Integer, Integer>();  
  
 for (int n : nums) {  
 frequencyMap.put(n, frequencyMap.getOrDefault(n, 0) + 1);  
 }  
  
 for (int key : frequencyMap.keySet()) {  
 int frequency = frequencyMap.get(key);  
 if (bucket[frequency] == null) {  
 bucket[frequency] = new ArrayList<>();  
 }  
 bucket[frequency].add(key);  
 }  
  
 List<Integer> res = new ArrayList<>();  
  
 for (int pos = bucket.length - 1; pos >= 0 && res.size() < k; pos--) {  
 if (bucket[pos] != null) {  
 res.addAll(bucket[pos]);  
 }  
 }  
 return res;  
}  
  
*349,Intersection of Two Arrays:***Python\_solution:**Python code, 3 lines using set   
class Solution(object):  
def intersection(self, nums1, nums2):  
 """  
 :type nums1: List[int]  
 :type nums2: List[int]  
 :rtype: List[int]  
 """  
 nums1=set(nums1)  
 nums2=set(nums2)  
 return list(nums1&nums2)  
**Best\_solution:**Three Java Solutions   
public class Solution {  
 public int[] intersection(int[] nums1, int[] nums2) {  
 Set<Integer> set = new HashSet<>();  
 Set<Integer> intersect = new HashSet<>();  
 for (int i = 0; i < nums1.length; i++) {  
 set.add(nums1[i]);  
 }  
 for (int i = 0; i < nums2.length; i++) {  
 if (set.contains(nums2[i])) {  
 intersect.add(nums2[i]);  
 }  
 }  
 int[] result = new int[intersect.size()];  
 int i = 0;  
 for (Integer num : intersect) {  
 result[i++] = num;  
 }  
 return result;  
 }  
}  
  
  
*350,Intersection of Two Arrays II:***Python\_solution:**2 lines in Python   
from collections import Counter  
  
class Solution(object):  
 def intersect(self, nums1, nums2):  
 c1, c2 = Counter(nums1), Counter(nums2)  
 return sum([[num] \* min(c1[num], c2[num]) for num in c1 & c2], [])  
**Best\_solution:**Solution to 3rd follow-up question   
None  
  
*352,Data Stream as Disjoint Intervals:***Python\_solution:**Share my python solution using heap   
class SummaryRanges(object):  
  
 def \_\_init\_\_(self):  
 self.intervals = []  
   
 def addNum(self, val):  
 heapq.heappush(self.intervals, (val, Interval(val, val)))  
   
 def getIntervals(self):  
 stack = []  
 while self.intervals:  
 idx, cur = heapq.heappop(self.intervals)  
 if not stack:  
 stack.append((idx, cur))  
 else:  
 \_, prev = stack[-1]  
 if prev.end + 1 >= cur.start:  
 prev.end = max(prev.end, cur.end)  
 else:  
 stack.append((idx, cur))  
 self.intervals = stack  
 return list(map(lambda x: x[1], stack))  
**Best\_solution:**Java solution using TreeMap, real O(logN) per adding.   
public class SummaryRanges {  
 TreeMap<Integer, Interval> tree;  
  
 public SummaryRanges() {  
 tree = new TreeMap<>();  
 }  
  
 public void addNum(int val) {  
 if(tree.containsKey(val)) return;  
 Integer l = tree.lowerKey(val);  
 Integer h = tree.higherKey(val);  
 if(l != null && h != null && tree.get(l).end + 1 == val && h == val + 1) {  
 tree.get(l).end = tree.get(h).end;  
 tree.remove(h);  
 } else if(l != null && tree.get(l).end + 1 >= val) {  
 tree.get(l).end = Math.max(tree.get(l).end, val);  
 } else if(h != null && h == val + 1) {  
 tree.put(val, new Interval(val, tree.get(h).end));  
 tree.remove(h);  
 } else {  
 tree.put(val, new Interval(val, val));  
 }  
 }  
  
 public List<Interval> getIntervals() {  
 return new ArrayList<>(tree.values());  
 }  
}  
  
*354,Russian Doll Envelopes:***Python\_solution:**Python O(nlogn) O(n) solution, beats 97%, with explanation   
class Solution(object):  
 def maxEnvelopes(self, envs):  
 def liss(envs):  
 def lmip(envs, tails, k):  
 b, e = 0, len(tails) - 1  
 while b <= e:  
 m = (b + e) >> 1  
 if envs[tails[m]][1] >= k[1]:  
 e = m - 1  
 else:  
 b = m + 1  
 return b  
   
 tails = []  
 for i, env in enumerate(envs):  
 idx = lmip(envs, tails, env)  
 if idx >= len(tails):  
 tails.append(i)  
 else:  
 tails[idx] = i  
 return len(tails)  
   
   
 def f(x, y):  
 return -1 if (x[0] < y[0] or x[0] == y[0] and x[1] > y[1]) else 1  
   
 envs.sort(cmp=f)  
 return liss(envs)  
  
# Runtime: 100ms  
  
**Best\_solution:**Java NLogN Solution with Explanation   
public int maxEnvelopes(int[][] envelopes) {  
 if(envelopes == null || envelopes.length == 0   
 || envelopes[0] == null || envelopes[0].length != 2)  
 return 0;  
 Arrays.sort(envelopes, new Comparator<int[]>(){  
 public int compare(int[] arr1, int[] arr2){  
 if(arr1[0] == arr2[0])  
 return arr2[1] - arr1[1];  
 else  
 return arr1[0] - arr2[0];  
 }   
 });  
 int dp[] = new int[envelopes.length];  
 int len = 0;  
 for(int[] envelope : envelopes){  
 int index = Arrays.binarySearch(dp, 0, len, envelope[1]);  
 if(index < 0)  
 index = -(index + 1);  
 dp[index] = envelope[1];  
 if(index == len)  
 len++;  
 }  
 return len;  
}  
  
  
*357,Count Numbers with Unique Digits:***Python\_solution:**Simple Python solution, 90%   
class Solution(object):  
 def countNumbersWithUniqueDigits(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 choices = [9, 9, 8, 7, 6, 5, 4, 3, 2, 1]  
 ans, product = 1, 1  
   
 for i in range(n if n <= 10 else 10):  
 product \*= choices[i]  
 ans += product  
   
 return ans  
  
**Best\_solution:**JAVA DP O(1) solution.   
 public int countNumbersWithUniqueDigits(int n) {  
 if (n == 0) return 1;  
   
 int res = 10;  
 int uniqueDigits = 9;  
 int availableNumber = 9;  
 while (n-- > 1 && availableNumber > 0) {  
 uniqueDigits = uniqueDigits \* availableNumber;  
 res += uniqueDigits;  
 availableNumber--;  
 }  
 return res;  
 }  
  
*365,Water and Jug Problem:***Python\_solution:**A little explanation on GCD method. C++/Java/Python   
if x and y are coprime, then we can and only can reach every integer z in [0, x + y]. (1)  
  
**Best\_solution:**Math solution - Java solution   
public boolean canMeasureWater(int x, int y, int z) {  
 //limit brought by the statement that water is finallly in one or both buckets  
 if(x + y < z) return false;  
 //case x or y is zero  
 if( x == z || y == z || x + y == z ) return true;  
   
 //get GCD, then we can use the property of Bézout's identity  
 return z%GCD(x, y) == 0;  
}  
  
public int GCD(int a, int b){  
 while(b != 0 ){  
 int temp = b;  
 b = a%b;  
 a = temp;  
 }  
 return a;  
}  
  
  
*367,Valid Perfect Square:***Python\_solution:**Python solution using Newton's method   
class Solution(object):  
 def isPerfectSquare(self, num):  
 """  
 :type num: int  
 :rtype: bool  
 """  
 if num < 0: return False  
 if num <= 1: return True  
 n = num/2 # start guessing using n = num/2  
 while n\*n!= num:  
 inc = (num-n\*n)/(2\*n)  
 n += inc  
 if -1 <= inc <= 1: break  
 if n\*n < num: n+=1  
 if n\*n > num: n-=1  
 return n\*n == num  
  
**Best\_solution:**A square number is 1+3+5+7+..., JAVA code   
public boolean isPerfectSquare(int num) {  
 int i = 1;  
 while (num > 0) {  
 num -= i;  
 i += 2;  
 }  
 return num == 0;  
 }  
  
  
*368,Largest Divisible Subset:***Python\_solution:**4 lines in Python   
def largestDivisibleSubset(self, nums):  
 S = {-1: set()}  
 for x in sorted(nums):  
 S[x] = max((S[d] for d in S if x % d == 0), key=len) | {x}  
 return list(max(S.values(), key=len))  
  
**Best\_solution:**C++ Solution with Explanations   
class Solution {  
public:  
 vector<int> largestDivisibleSubset(vector<int>& nums) {  
 sort(nums.begin(), nums.end());  
   
 vector<int> T(nums.size(), 0);  
 vector<int> parent(nums.size(), 0);  
   
 int m = 0;  
 int mi = 0;  
   
 // for(int i = 0; i < nums.size(); ++i) // if extending by larger elements  
 for(int i = nums.size() - 1; i >= 0; --i) // iterate from end to start since it's easier to track the answer index  
 {  
 // for(int j = i; j >=0; --j) // if extending by larger elements  
 for(int j = i; j < nums.size(); ++j)  
 {  
 // if(nums[i] % nums[j] == 0 && T[i] < 1 + T[j]) // if extending by larger elements  
 // check every a[j] that is larger than a[i]  
 if(nums[j] % nums[i] == 0 && T[i] < 1 + T[j])  
 {  
 // if a[j] mod a[i] == 0, it means T[j] can form a larger subset by putting a[i] into T[j]  
 T[i] = 1 + T[j];  
 parent[i] = j;  
   
 if(T[i] > m)  
 {  
 m = T[i];  
 mi = i;  
 }  
 }  
 }  
 }  
   
 vector<int> ret;  
   
 for(int i = 0; i < m; ++i)  
 {  
 ret.push\_back(nums[mi]);  
 mi = parent[mi];  
 }  
  
 // sort(ret.begin(), ret.end()); // if we go by extending larger ends, the largest "answer" element will come first since the candidate element we observe will become larger and larger as i increases in the outermost "for" loop above.  
 // alternatively, we can sort nums in decreasing order obviously.   
   
 return ret;  
 }  
};  
  
*371,Sum of Two Integers:***Python\_solution:**Python solution with no "+-\*/%", completely bit manipulation guaranteed   
class Solution(object):  
 def getSum(self, a, b):  
 """  
 :type a: int  
 :type b: int  
 :rtype: int  
 """  
 # 32 bits integer max  
 MAX = 0x7FFFFFFF  
 # 32 bits interger min  
 MIN = 0x80000000  
 # mask to get last 32 bits  
 mask = 0xFFFFFFFF  
 while b != 0:  
 # ^ get different bits and & gets double 1s, << moves carry  
 a, b = (a ^ b) & mask, ((a & b) << 1) & mask  
 # if a is negative, get a's 32 bits complement positive first  
 # then get 32-bit positive's Python complement negative  
 return a if a <= MAX else ~(a ^ mask)  
  
**Best\_solution:**A summary: how to use bit manipulation to solve problems easily and efficiently   
int count\_one(int n) {  
 while(n) {  
 n = n&(n-1);  
 count++;  
 }  
 return count;  
}  
  
  
*372,Super Pow:***Python\_solution:**Math solusion based on Euler's theorem, power called only ONCE, C++/Java/1-line-Python   
a  
**Best\_solution:**C++ Clean and Short Solution   
class Solution {  
 const int base = 1337;  
 int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10  
 {  
 a %= base;  
 int result = 1;  
 for (int i = 0; i < k; ++i)  
 result = (result \* a) % base;  
 return result;  
 }  
public:  
 int superPow(int a, vector<int>& b) {  
 if (b.empty()) return 1;  
 int last\_digit = b.back();  
 b.pop\_back();  
 return powmod(superPow(a, b), 10) \* powmod(a, last\_digit) % base;  
 }  
};  
  
  
*373,Find K Pairs with Smallest Sums:***Python\_solution:**BFS Python 104ms with comments   
visited  
**Best\_solution:**Slow 1-liner to Fast solutions   
 2 4 6  
 +------------  
 1 | 3 5 7  
 7 | 9 11 13  
11 | 13 15 17  
  
  
*374,Guess Number Higher or Lower:***Python\_solution:**Standard binary search in Python   
class Solution(object):  
 def guessNumber(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 l, r = 1, n  
 while l + 1 < r:  
 m = l + (r - l) / 2  
 res = guess(m)  
 if res < 0:  
 r = m  
 elif res > 0:  
 l = m  
 else:  
 return m  
   
 if guess(l) == 0:  
 return l  
 if guess(r) == 0:  
 return r  
 return None  
  
**Best\_solution:**The key point is to read the problem carefully.   
@Nakanu   
  
  
*375,Guess Number Higher or Lower II:***Python\_solution:**Two Python solutions   
def getMoneyAmount(self, n):  
 need = [[0] \* (n+1) for \_ in range(n+1)]  
 for lo in range(n, 0, -1):  
 for hi in range(lo+1, n+1):  
 need[lo][hi] = min(x + max(need[lo][x-1], need[x+1][hi])  
 for x in range(lo, hi))  
 return need[1][n]  
  
**Best\_solution:**Simple DP solution with explanation~~   
public class Solution {  
 public int getMoneyAmount(int n) {  
 int[][] table = new int[n+1][n+1];  
 return DP(table, 1, n);  
 }  
   
 int DP(int[][] t, int s, int e){  
 if(s >= e) return 0;  
 if(t[s][e] != 0) return t[s][e];  
 int res = Integer.MAX\_VALUE;  
 for(int x=s; x<=e; x++){  
 int tmp = x + Math.max(DP(t, s, x-1), DP(t, x+1, e));  
 res = Math.min(res, tmp);  
 }  
 t[s][e] = res;  
 return res;  
 }  
}  
  
  
*376,Wiggle Subsequence:***Python\_solution:**3 lines O(n) Python with explanation/proof   
def wiggleMaxLength(self, nums):  
 nan = float('nan')  
 diffs = [a-b for a, b in zip([nan] + nums, nums + [nan]) if a-b]  
 return sum(not d\*e >= 0 for d, e in zip(diffs, diffs[1:]))  
  
**Best\_solution:**Very Simple Java Solution with detail explanation   
Step 1: First we check our requirement is to get small number. As 1<2 so the series will be  
 2,1  
  
  
*377,Combination Sum IV:***Python\_solution:**7-liner in Python, and follow-up question   
class Solution(object):  
 def combinationSum4(self, nums, target):  
 nums, combs = sorted(nums), [1] + [0] \* (target)  
 for i in range(target + 1):  
 for num in nums:  
 if num > i: break  
 if num == i: combs[i] += 1  
 if num < i: combs[i] += combs[i - num]  
 return combs[target]  
  
# 17 / 17 test cases passed.  
# Status: Accepted  
# Runtime: 116 ms  
  
**Best\_solution:**1ms Java DP Solution with Detailed Explanation   
target  
  
*378,Kth Smallest Element in a Sorted Matrix:***Python\_solution:**python one-line solution ...   
import heapq  
class Solution(object):  
 def kthSmallest(self, matrix, k):  
 return list(heapq.merge(\*matrix))[k-1]  
  
  
**Best\_solution:**Share my thoughts and Clean Java Code   
public class Solution {  
 public int kthSmallest(int[][] matrix, int k) {  
 int n = matrix.length;  
 PriorityQueue<Tuple> pq = new PriorityQueue<Tuple>();  
 for(int j = 0; j <= n-1; j++) pq.offer(new Tuple(0, j, matrix[0][j]));  
 for(int i = 0; i < k-1; i++) {  
 Tuple t = pq.poll();  
 if(t.x == n-1) continue;  
 pq.offer(new Tuple(t.x+1, t.y, matrix[t.x+1][t.y]));  
 }  
 return pq.poll().val;  
 }  
}  
  
class Tuple implements Comparable<Tuple> {  
 int x, y, val;  
 public Tuple (int x, int y, int val) {  
 this.x = x;  
 this.y = y;  
 this.val = val;  
 }  
   
 @Override  
 public int compareTo (Tuple that) {  
 return this.val - that.val;  
 }  
}  
  
  
*380,Insert Delete GetRandom O(1):***Python\_solution:**Simple solution in Python   
list.append()  
**Best\_solution:**Java solution using a HashMap and an ArrayList along with a follow-up. (131 ms)   
public class RandomizedSet {  
 ArrayList<Integer> nums;  
 HashMap<Integer, Integer> locs;  
 java.util.Random rand = new java.util.Random();  
 /\*\* Initialize your data structure here. \*/  
 public RandomizedSet() {  
 nums = new ArrayList<Integer>();  
 locs = new HashMap<Integer, Integer>();  
 }  
   
 /\*\* Inserts a value to the set. Returns true if the set did not already contain the specified element. \*/  
 public boolean insert(int val) {  
 boolean contain = locs.containsKey(val);  
 if ( contain ) return false;  
 locs.put( val, nums.size());  
 nums.add(val);  
 return true;  
 }  
   
 /\*\* Removes a value from the set. Returns true if the set contained the specified element. \*/  
 public boolean remove(int val) {  
 boolean contain = locs.containsKey(val);  
 if ( ! contain ) return false;  
 int loc = locs.get(val);  
 if (loc < nums.size() - 1 ) { // not the last one than swap the last one with this val  
 int lastone = nums.get(nums.size() - 1 );  
 nums.set( loc , lastone );  
 locs.put(lastone, loc);  
 }  
 locs.remove(val);  
 nums.remove(nums.size() - 1);  
 return true;  
 }  
   
 /\*\* Get a random element from the set. \*/  
 public int getRandom() {  
 return nums.get( rand.nextInt(nums.size()) );  
 }  
}  
  
  
*381,Insert Delete GetRandom O(1) - Duplicates allowed:***Python\_solution:**Frugal Python code   
import random  
  
class RandomizedCollection(object):  
  
 def \_\_init\_\_(self):  
 self.vals, self.idxs = [], collections.defaultdict(set)  
   
  
 def insert(self, val):  
 self.vals.append(val)  
 self.idxs[val].add(len(self.vals) - 1)  
 return len(self.idxs[val]) == 1  
   
  
 def remove(self, val):  
 if self.idxs[val]:  
 out, ins = self.idxs[val].pop(), self.vals[-1]  
 self.vals[out] = ins  
 if self.idxs[ins]:  
 self.idxs[ins].add(out)  
 self.idxs[ins].discard(len(self.vals) - 1)  
 self.vals.pop()  
 return True  
 return False   
  
 def getRandom(self):  
 return random.choice(self.vals)  
**Best\_solution:**C++ 128m Solution, Real O(1) Solution   
class RandomizedCollection {  
public:  
 /\*\* Initialize your data structure here. \*/  
 RandomizedCollection() {  
   
 }  
   
 /\*\* Inserts a value to the collection. Returns true if the collection did not already contain the specified element. \*/  
 bool insert(int val) {  
 auto result = m.find(val) == m.end();  
   
 m[val].push\_back(nums.size());  
 nums.push\_back(pair<int, int>(val, m[val].size() - 1));  
   
 return result;  
 }  
   
 /\*\* Removes a value from the collection. Returns true if the collection contained the specified element. \*/  
 bool remove(int val) {  
 auto result = m.find(val) != m.end();  
 if(result)  
 {  
 auto last = nums.back();  
 m[last.first][last.second] = m[val].back();  
 nums[m[val].back()] = last;  
 m[val].pop\_back();  
 if(m[val].empty()) m.erase(val);  
 nums.pop\_back();  
 }  
 return result;  
 }  
   
 /\*\* Get a random element from the collection. \*/  
 int getRandom() {  
 return nums[rand() % nums.size()].first;  
 }  
private:  
 vector<pair<int, int>> nums;  
 unordered\_map<int, vector<int>> m;  
};  
  
  
*382,Linked List Random Node:***Python\_solution:**Python reservoir sampling solution (when the length of linked list changes dynamically)   
class Solution(object):  
  
 def \_\_init\_\_(self, head):  
 self.head = head  
  
 def getRandom(self):  
 result, node, index = self.head, self.head.next, 1  
 while node:  
 if random.randint(0, index) is 0:  
 result = node  
 node = node.next  
 index += 1  
 return result.val  
  
**Best\_solution:**Brief explanation for Reservoir Sampling   
k  
  
*383,Ransom Note:***Python\_solution:**O(m+n) one-liner Python   
def canConstruct(self, ransomNote, magazine):  
 return not collections.Counter(ransomNote) - collections.Counter(magazine)  
**Best\_solution:**Java O(n) Solution---Easy to understand   
public class Solution {  
 public boolean canConstruct(String ransomNote, String magazine) {  
 int[] arr = new int[26];  
 for (int i = 0; i < magazine.length(); i++) {  
 arr[magazine.charAt(i) - 'a']++;  
 }  
 for (int i = 0; i < ransomNote.length(); i++) {  
 if(--arr[ransomNote.charAt(i)-'a'] < 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}  
  
  
*384,Shuffle an Array:***Python\_solution:**Python hack   
class Solution(object):  
 def \_\_init\_\_(self, nums):  
 self.reset = lambda: nums  
 self.shuffle = lambda: random.sample(nums, len(nums))  
  
**Best\_solution:**First Accepted Solution - Java   
import java.util.Random;  
  
public class Solution {  
 private int[] nums;  
 private Random random;  
  
 public Solution(int[] nums) {  
 this.nums = nums;  
 random = new Random();  
 }  
   
 /\*\* Resets the array to its original configuration and return it. \*/  
 public int[] reset() {  
 return nums;  
 }  
   
 /\*\* Returns a random shuffling of the array. \*/  
 public int[] shuffle() {  
 if(nums == null) return null;  
 int[] a = nums.clone();  
 for(int j = 1; j < a.length; j++) {  
 int i = random.nextInt(j + 1);  
 swap(a, i, j);  
 }  
 return a;  
 }  
   
 private void swap(int[] a, int i, int j) {  
 int t = a[i];  
 a[i] = a[j];  
 a[j] = t;  
 }  
}  
  
  
  
*385,Mini Parser:***Python\_solution:**Python & C++ solutions   
eval  
**Best\_solution:**An Java Iterative Solution   
public NestedInteger deserialize(String s) {  
 if (s.isEmpty())  
 return null;  
 if (s.charAt(0) != '[') // ERROR: special case  
 return new NestedInteger(Integer.valueOf(s));  
   
 Stack<NestedInteger> stack = new Stack<>();  
 NestedInteger curr = null;  
 int l = 0; // l shall point to the start of a number substring;   
 // r shall point to the end+1 of a number substring  
 for (int r = 0; r < s.length(); r++) {  
 char ch = s.charAt(r);  
 if (ch == '[') {  
 if (curr != null) {  
 stack.push(curr);  
 }  
 curr = new NestedInteger();  
 l = r+1;  
 } else if (ch == ']') {  
 String num = s.substring(l, r);  
 if (!num.isEmpty())  
 curr.add(new NestedInteger(Integer.valueOf(num)));  
 if (!stack.isEmpty()) {  
 NestedInteger pop = stack.pop();  
 pop.add(curr);  
 curr = pop;  
 }  
 l = r+1;  
 } else if (ch == ',') {  
 if (s.charAt(r-1) != ']') {  
 String num = s.substring(l, r);  
 curr.add(new NestedInteger(Integer.valueOf(num)));  
 }  
 l = r+1;  
 }  
 }  
   
 return curr;  
}  
  
*386,Lexicographical Numbers:***Python\_solution:**Python with Sorting   
sorted  
**Best\_solution:**Java O(n) time, O(1) space iterative solution 130ms   
public List<Integer> lexicalOrder(int n) {  
 List<Integer> list = new ArrayList<>(n);  
 int curr = 1;  
 for (int i = 1; i <= n; i++) {  
 list.add(curr);  
 if (curr \* 10 <= n) {  
 curr \*= 10;  
 } else if (curr % 10 != 9 && curr + 1 <= n) {  
 curr++;  
 } else {  
 while ((curr / 10) % 10 == 9) {  
 curr /= 10;  
 }  
 curr = curr / 10 + 1;  
 }  
 }  
 return list;  
 }  
  
  
*387,First Unique Character in a String:***Python\_solution:**1-liners in Python, 76ms   
class Solution(object):  
 def firstUniqChar(self, s):  
 return min([s.find(c) for c in string.ascii\_lowercase if s.count(c)==1] or [-1])  
  
**Best\_solution:**Java 7 lines solution 29ms   
public class Solution {  
 public int firstUniqChar(String s) {  
 int freq [] = new int[26];  
 for(int i = 0; i < s.length(); i ++)  
 freq [s.charAt(i) - 'a'] ++;  
 for(int i = 0; i < s.length(); i ++)  
 if(freq [s.charAt(i) - 'a'] == 1)  
 return i;  
 return -1;  
 }  
}  
  
  
*388,Longest Absolute File Path:***Python\_solution:**Simple Python solution   
depth  
**Best\_solution:**9 lines 4ms Java solution   
public int lengthLongestPath(String input) {  
 Deque<Integer> stack = new ArrayDeque<>();  
 stack.push(0); // "dummy" length  
 int maxLen = 0;  
 for(String s:input.split("\n")){  
 int lev = s.lastIndexOf("\t")+1; // number of "\t"  
 while(lev+1<stack.size()) stack.pop(); // find parent  
 int len = stack.peek()+s.length()-lev+1; // remove "/t", add"/"  
 stack.push(len);  
 // check if it is file  
 if(s.contains(".")) maxLen = Math.max(maxLen, len-1);   
 }  
 return maxLen;  
 }  
  
  
*389,Find the Difference:***Python\_solution:**1-liners and 2-liner in Python   
class Solution(object):  
 def findTheDifference(self, s, t):  
 return chr(reduce(operator.xor, map(ord, s + t)))  
  
**Best\_solution:**Java solution using bit manipulation   
public char findTheDifference(String s, String t) {  
 char c = 0;  
 for (int i = 0; i < s.length(); ++i) {  
 c ^= s.charAt(i);  
 }  
 for (int i = 0; i < t.length(); ++i) {  
 c ^= t.charAt(i);  
 }  
 return c;  
}  
  
  
*390,Elimination Game:***Python\_solution:**3 lines Iterative code in Python, O(log N), O(1) space   
class Solution(object):  
 def lastRemaining(self, n):  
 start, size, inv = 1, 1, 1  
 while n > 1: start, size, inv, n = start + inv \* size + 2 \* (n // 2 - 1) \* inv \* size,\  
 size \* 2, inv \* -1,\  
 n // 2  
 return start  
  
**Best\_solution:**JAVA: Easiest solution O(logN) with explanation   
 public int lastRemaining(int n) {  
 boolean left = true;  
 int remaining = n;  
 int step = 1;  
 int head = 1;  
 while (remaining > 1) {  
 if (left || remaining % 2 ==1) {  
 head = head + step;  
 }  
 remaining = remaining / 2;  
 step = step \* 2;  
 left = !left;  
 }  
 return head;  
 }  
  
  
*391,Perfect Rectangle:***Python\_solution:**Easy Understanding O(n) Python Solution   
class Solution(object):  
 def isRectangleCover(self, rectangles):  
 def recordCorner(point):  
 if point in corners:  
 corners[point] += 1  
 else:  
 corners[point] = 1  
  
 corners = {} # record all corners   
 L, B, R, T, area = float('inf'), float('inf'), -float('inf'), -float('inf'), 0  
  
 for sub in rectangles:  
 L, B, R, T = min(L, sub[0]), min(B, sub[1]), max(R, sub[2]), max(T, sub[3])  
 ax, ay, bx, by = sub[:]  
 area += (bx-ax)\*(by-ay) # sum up the area of each sub-rectangle  
 map(recordCorner, [(ax, ay), (bx, by), (ax, by), (bx, ay)])  
  
 if area != (T-B)\*(R-L): return False # check the area  
  
 big\_four = [(L,B),(R,T),(L,T),(R,B)]  
  
 for bf in big\_four: # check corners of big rectangle  
 if bf not in corners or corners[bf] != 1:  
 return False  
  
 for key in corners: # check existing "inner" points  
 if corners[key]%2 and key not in big\_four:  
 return False  
  
 return True  
  
**Best\_solution:**Really Easy Understanding Solution(O(n), Java)   
public boolean isRectangleCover(int[][] rectangles) {  
  
 if (rectangles.length == 0 || rectangles[0].length == 0) return false;  
  
 int x1 = Integer.MAX\_VALUE;  
 int x2 = Integer.MIN\_VALUE;  
 int y1 = Integer.MAX\_VALUE;  
 int y2 = Integer.MIN\_VALUE;  
   
 HashSet<String> set = new HashSet<String>();  
 int area = 0;  
   
 for (int[] rect : rectangles) {  
 x1 = Math.min(rect[0], x1);  
 y1 = Math.min(rect[1], y1);  
 x2 = Math.max(rect[2], x2);  
 y2 = Math.max(rect[3], y2);  
   
 area += (rect[2] - rect[0]) \* (rect[3] - rect[1]);  
   
 String s1 = rect[0] + " " + rect[1];  
 String s2 = rect[0] + " " + rect[3];  
 String s3 = rect[2] + " " + rect[3];  
 String s4 = rect[2] + " " + rect[1];  
   
 if (!set.add(s1)) set.remove(s1);  
 if (!set.add(s2)) set.remove(s2);  
 if (!set.add(s3)) set.remove(s3);  
 if (!set.add(s4)) set.remove(s4);  
 }  
   
 if (!set.contains(x1 + " " + y1) || !set.contains(x1 + " " + y2) || !set.contains(x2 + " " + y1) || !set.contains(x2 + " " + y2) || set.size() != 4) return false;  
   
 return area == (x2-x1) \* (y2-y1);  
 }  
  
  
*392,Is Subsequence:***Python\_solution:**2 lines Python   
def isSubsequence(self, s, t):  
 t = iter(t)  
 return all(c in t for c in s)  
  
**Best\_solution:**3 lines C   
bool isSubsequence(char\* s, char\* t) {  
 while (\*t)  
 s += \*s == \*t++;  
 return !\*s;  
}  
  
  
*393,UTF-8 Validation:***Python\_solution:**Short'n'Clean 12-lines Python solution   
def check(nums, start, size):  
 for i in range(start + 1, start + size + 1):  
 if i >= len(nums) or (nums[i] >> 6) != 0b10: return False  
 return True  
  
class Solution(object):  
 def validUtf8(self, nums, start=0):  
 while start < len(nums):  
 first = nums[start]  
 if (first >> 3) == 0b11110 and check(nums, start, 3): start += 4  
 elif (first >> 4) == 0b1110 and check(nums, start, 2): start += 3  
 elif (first >> 5) == 0b110 and check(nums, start, 1): start += 2  
 elif (first >> 7) == 0: start += 1  
 else: return False  
 return True  
  
# 45 / 45 test cases passed.  
# Status: Accepted  
# Runtime: 89 ms  
  
**Best\_solution:**Concise C++ implementation   
class Solution {  
public:  
 bool validUtf8(vector<int>& data) {  
 int count = 0;  
 for (auto c : data) {  
 if (count == 0) {  
 if ((c >> 5) == 0b110) count = 1;  
 else if ((c >> 4) == 0b1110) count = 2;  
 else if ((c >> 3) == 0b11110) count = 3;  
 else if ((c >> 7)) return false;  
 } else {  
 if ((c >> 6) != 0b10) return false;  
 count--;  
 }  
 }  
 return count == 0;  
 }  
};  
  
  
*394,Decode String:***Python\_solution:**Share my Python Stack Simple Solution (Easy to understand)   
class Solution(object):  
 def decodeString(self, s):  
 stack = []  
 stack.append(["", 1])  
 num = ""  
 for ch in s:  
 if ch.isdigit():  
 num += ch  
 elif ch == '[':  
 stack.append(["", int(num)])  
 num = ""  
 elif ch == ']':  
 st, k = stack.pop()  
 stack[-1][0] += st\*k  
 else:  
 stack[-1][0] += ch  
 return stack[0][0]  
  
**Best\_solution:**0ms simple C++ solution   
class Solution {  
public:  
 string decodeString(const string& s, int& i) {  
 string res;  
   
 while (i < s.length() && s[i] != ']') {  
 if (!isdigit(s[i]))  
 res += s[i++];  
 else {  
 int n = 0;  
 while (i < s.length() && isdigit(s[i]))  
 n = n \* 10 + s[i++] - '0';  
   
 i++; // '['  
 string t = decodeString(s, i);  
 i++; // ']'  
   
 while (n-- > 0)  
 res += t;  
 }  
 }  
   
 return res;  
 }  
  
 string decodeString(string s) {  
 int i = 0;  
 return decodeString(s, i);  
 }  
};  
  
  
*395,Longest Substring with At Least K Repeating Characters:***Python\_solution:**4 lines Python   
def longestSubstring(self, s, k):  
 for c in set(s):  
 if s.count(c) < k:  
 return max(self.longestSubstring(t, k) for t in s.split(c))  
 return len(s)  
  
**Best\_solution:**4 lines Python   
def longestSubstring(self, s, k):  
 for c in set(s):  
 if s.count(c) < k:  
 return max(self.longestSubstring(t, k) for t in s.split(c))  
 return len(s)  
  
  
*396,Rotate Function:***Python\_solution:**Python O(n), Math with explaination   
class Solution(object):  
 def maxRotateFunction(self, A):  
 sumA=sum(A)  
 temp=0  
 for i,c in enumerate(A):  
 temp+=i\*c  
 maxx=temp  
 for j in xrange(len(A)):  
 temp+=(len(A)\*A[j]-sumA)  
 maxx=max(temp,maxx)  
 return maxx  
  
**Best\_solution:**Java O(n) solution with explanation   
F(k) = 0 \* Bk[0] + 1 \* Bk[1] + ... + (n-1) \* Bk[n-1]  
F(k-1) = 0 \* Bk-1[0] + 1 \* Bk-1[1] + ... + (n-1) \* Bk-1[n-1]  
 = 0 \* Bk[1] + 1 \* Bk[2] + ... + (n-2) \* Bk[n-1] + (n-1) \* Bk[0]  
  
  
*397,Integer Replacement:***Python\_solution:**Python O(log n) time, O(1) space with explanation and proof   
class Solution(object):  
 def integerReplacement(self, n):  
 rtn = 0  
 while n > 1:  
 rtn += 1  
 if n % 2 == 0:  
 n //= 2  
 elif n % 4 == 1 or n == 3:  
 n -= 1  
 else:  
 n += 1  
 return rtn  
  
**Best\_solution:**A couple of Java solutions with explanations   
111011 -> 111010 -> 11101 -> 11100 -> 1110 -> 111 -> 1000 -> 100 -> 10 -> 1  
  
  
*398,Random Pick Index:***Python\_solution:**Simple Python solution   
class Solution(object):  
  
 def \_\_init\_\_(self, nums):  
 self.nums = nums  
   
  
 def pick(self, target):  
 return random.choice([k for k, v in enumerate(self.nums) if v == target])  
  
**Best\_solution:**Simple Reservoir Sampling solution   
public class Solution {  
  
 int[] nums;  
 Random rnd;  
  
 public Solution(int[] nums) {  
 this.nums = nums;  
 this.rnd = new Random();  
 }  
   
 public int pick(int target) {  
 int result = -1;  
 int count = 0;  
 for (int i = 0; i < nums.length; i++) {  
 if (nums[i] != target)  
 continue;  
 if (rnd.nextInt(++count) == 0)  
 result = i;  
 }  
   
 return result;  
 }  
}```  
  
*399,Evaluate Division:***Python\_solution:**9 lines "Floyd–Warshall" in Python   
A/B=k  
**Best\_solution:**Java AC Solution using graph   
 public double[] calcEquation(String[][] equations, double[] values, String[][] queries) {  
 HashMap<String, ArrayList<String>> pairs = new HashMap<String, ArrayList<String>>();  
 HashMap<String, ArrayList<Double>> valuesPair = new HashMap<String, ArrayList<Double>>();  
 for (int i = 0; i < equations.length; i++) {  
 String[] equation = equations[i];  
 if (!pairs.containsKey(equation[0])) {  
 pairs.put(equation[0], new ArrayList<String>());  
 valuesPair.put(equation[0], new ArrayList<Double>());  
 }  
 if (!pairs.containsKey(equation[1])) {  
 pairs.put(equation[1], new ArrayList<String>());  
 valuesPair.put(equation[1], new ArrayList<Double>());  
 }  
 pairs.get(equation[0]).add(equation[1]);  
 pairs.get(equation[1]).add(equation[0]);  
 valuesPair.get(equation[0]).add(values[i]);  
 valuesPair.get(equation[1]).add(1/values[i]);  
 }  
   
 double[] result = new double[queries.length];  
 for (int i = 0; i < queries.length; i++) {  
 String[] query = queries[i];  
 result[i] = dfs(query[0], query[1], pairs, valuesPair, new HashSet<String>(), 1.0);  
 if (result[i] == 0.0) result[i] = -1.0;  
 }  
 return result;  
 }  
   
 private double dfs(String start, String end, HashMap<String, ArrayList<String>> pairs, HashMap<String, ArrayList<Double>> values, HashSet<String> set, double value) {  
 if (set.contains(start)) return 0.0;  
 if (!pairs.containsKey(start)) return 0.0;  
 if (start.equals(end)) return value;  
 set.add(start);  
   
 ArrayList<String> strList = pairs.get(start);  
 ArrayList<Double> valueList = values.get(start);  
 double tmp = 0.0;  
 for (int i = 0; i < strList.size(); i++) {  
 tmp = dfs(strList.get(i), end, pairs, values, set, value\*valueList.get(i));  
 if (tmp != 0.0) {  
 break;  
 }  
 }  
 set.remove(start);  
 return tmp;  
 }  
  
*400,Nth Digit:***Python\_solution:**Short Python+Java   
def findNthDigit(self, n):  
 n -= 1  
 for digits in range(1, 11):  
 first = 10\*\*(digits - 1)  
 if n < 9 \* first \* digits:  
 return int(str(first + n/digits)[n%digits])  
 n -= 9 \* first \* digits  
  
**Best\_solution:**Java solution   
 public int findNthDigit(int n) {  
 int len = 1;  
 long count = 9;  
 int start = 1;  
  
 while (n > len \* count) {  
 n -= len \* count;  
 len += 1;  
 count \*= 10;  
 start \*= 10;  
 }  
  
 start += (n - 1) / len;  
 String s = Integer.toString(start);  
 return Character.getNumericValue(s.charAt((n - 1) % len));  
 }  
  
  
*401,Binary Watch:***Python\_solution:**Simple Python+Java   
def readBinaryWatch(self, num):  
 return ['%d:%02d' % (h, m)  
 for h in range(12) for m in range(60)  
 if (bin(h) + bin(m)).count('1') == num]  
  
**Best\_solution:**Simple Python+Java   
def readBinaryWatch(self, num):  
 return ['%d:%02d' % (h, m)  
 for h in range(12) for m in range(60)  
 if (bin(h) + bin(m)).count('1') == num]  
  
  
*402,Remove K Digits:***Python\_solution:**Short Python, one O(n) and one RegEx   
prep  
**Best\_solution:**A greedy method using stack, O(n) time and O(n) space   
public class Solution {  
 public String removeKdigits(String num, int k) {  
 int digits = num.length() - k;  
 char[] stk = new char[num.length()];  
 int top = 0;  
 // k keeps track of how many characters we can remove  
 // if the previous character in stk is larger than the current one  
 // then removing it will get a smaller number  
 // but we can only do so when k is larger than 0  
 for (int i = 0; i < num.length(); ++i) {  
 char c = num.charAt(i);  
 while (top > 0 && stk[top-1] > c && k > 0) {  
 top -= 1;  
 k -= 1;  
 }  
 stk[top++] = c;  
 }  
 // find the index of first non-zero digit  
 int idx = 0;  
 while (idx < digits && stk[idx] == '0') idx++;  
 return idx == digits? "0": new String(stk, idx, digits - idx);  
 }  
}  
  
  
*403,Frog Jump:***Python\_solution:**Python DFS easy understanding using memo   
class Solution(object):  
 def canCross(self, stones):  
 self.memo = set()  
 target = stones[-1]  
 stones = set(stones)  
  
 res = self.bt(stones, 1, 1, target)  
 return res  
  
 def bt(self, stones, cur, speed, target):  
 # check memo  
 if (cur, speed) in self.memo:  
 return False  
  
 if cur==target:  
 return True  
   
 if cur>target or cur<0 or speed<=0 or cur not in stones:  
 return False  
 # dfs  
 candidate = [speed-1, speed, speed+1]  
 for c in candidate:  
 if (cur + c) in stones:  
 if self.bt(stones, cur+c, c, target):  
 return True  
  
 self.memo.add((cur,speed))  
 return False  
  
**Best\_solution:**Very easy to understand JAVA solution with explanations   
 public boolean canCross(int[] stones) {  
 if (stones.length == 0) {  
 return true;  
 }  
   
 HashMap<Integer, HashSet<Integer>> map = new HashMap<Integer, HashSet<Integer>>(stones.length);  
 map.put(0, new HashSet<Integer>());  
 map.get(0).add(1);  
 for (int i = 1; i < stones.length; i++) {  
 map.put(stones[i], new HashSet<Integer>() );  
 }  
   
 for (int i = 0; i < stones.length - 1; i++) {  
 int stone = stones[i];  
 for (int step : map.get(stone)) {  
 int reach = step + stone;  
 if (reach == stones[stones.length - 1]) {  
 return true;  
 }  
 HashSet<Integer> set = map.get(reach);  
 if (set != null) {  
 set.add(step);  
 if (step - 1 > 0) set.add(step - 1);  
 set.add(step + 1);  
 }  
 }  
 }  
   
 return false;  
 }   
  
  
*404,Sum of Left Leaves:***Python\_solution:**4 Lines Python Recursive AC Solution   
class Solution(object):  
 def sumOfLeftLeaves(self, root):  
 if not root: return 0  
 if root.left and not root.left.left and not root.left.right:  
 return root.left.val + self.sumOfLeftLeaves(root.right)  
 return self.sumOfLeftLeaves(root.left) + self.sumOfLeftLeaves(root.right) # isn't leave  
  
**Best\_solution:**Java iterative and recursive solutions   
public int sumOfLeftLeaves(TreeNode root) {  
 if(root == null) return 0;  
 int ans = 0;  
 if(root.left != null) {  
 if(root.left.left == null && root.left.right == null) ans += root.left.val;  
 else ans += sumOfLeftLeaves(root.left);  
 }  
 ans += sumOfLeftLeaves(root.right);  
   
 return ans;  
}  
  
  
*405,Convert a Number to Hexadecimal:***Python\_solution:**1-liner in Python   
class Solution(object):  
 def toHex(self, num):  
 return ''.join(  
 '0123456789abcdef'[(num >> 4 \* i) & 15]   
 for i in range(8)  
 )[::-1].lstrip('0') or '0'  
  
**Best\_solution:**Simple Java solution with comment   
/\*  
Basic idea: each time we take a look at the last four digits of  
 binary verion of the input, and maps that to a hex char  
 shift the input to the right by 4 bits, do it again  
 until input becomes 0.  
  
\*/  
  
public class Solution {  
   
 char[] map = {'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};  
   
 public String toHex(int num) {  
 if(num == 0) return "0";  
 String result = "";  
 while(num != 0){  
 result = map[(num & 15)] + result;   
 num = (num >>> 4);  
 }  
 return result;  
 }  
   
   
}````  
  
*406,Queue Reconstruction by Height:***Python\_solution:**Easy concept with Python/C++/Java Solution   
class Solution(object):  
 def reconstructQueue(self, people):  
 if not people: return []  
  
 # obtain everyone's info  
 # key=height, value=k-value, index in original array  
 peopledct, height, res = {}, [], []  
   
 for i in xrange(len(people)):  
 p = people[i]  
 if p[0] in peopledct:  
 peopledct[p[0]] += (p[1], i),  
 else:  
 peopledct[p[0]] = [(p[1], i)]  
 height += p[0],  
  
 height.sort() # here are different heights we have  
  
 # sort from the tallest group  
 for h in height[::-1]:  
 peopledct[h].sort()  
 for p in peopledct[h]:  
 res.insert(p[0], people[p[1]])  
  
 return res  
  
  
**Best\_solution:**Easy concept with Python/C++/Java Solution   
class Solution(object):  
 def reconstructQueue(self, people):  
 if not people: return []  
  
 # obtain everyone's info  
 # key=height, value=k-value, index in original array  
 peopledct, height, res = {}, [], []  
   
 for i in xrange(len(people)):  
 p = people[i]  
 if p[0] in peopledct:  
 peopledct[p[0]] += (p[1], i),  
 else:  
 peopledct[p[0]] = [(p[1], i)]  
 height += p[0],  
  
 height.sort() # here are different heights we have  
  
 # sort from the tallest group  
 for h in height[::-1]:  
 peopledct[h].sort()  
 for p in peopledct[h]:  
 res.insert(p[0], people[p[1]])  
  
 return res  
  
  
  
*407,Trapping Rain Water II:***Python\_solution:**python solution with heap   
class Solution(object):  
 def trapRainWater(self, heightMap):  
 if not heightMap or not heightMap[0]:  
 return 0  
   
 import heapq   
 m, n = len(heightMap), len(heightMap[0])  
 heap = []  
 visited = [[0]\*n for \_ in xrange(m)]  
  
 # Push all the block on the border into heap  
 for i in xrange(m):  
 for j in xrange(n):  
 if i == 0 or j == 0 or i == m-1 or j == n-1:  
 heapq.heappush(heap, (heightMap[i][j], i, j))  
 visited[i][j] = 1  
   
 result = 0  
 while heap:  
 height, i, j = heapq.heappop(heap)   
 for x, y in ((i+1, j), (i-1, j), (i, j+1), (i, j-1)):  
 if 0 <= x < m and 0 <= y < n and not visited[x][y]:  
 result += max(0, height-heightMap[x][y])  
 heapq.heappush(heap, (max(heightMap[x][y], height), x, y))  
 visited[x][y] = 1  
 return result  
  
**Best\_solution:**Java solution using PriorityQueue   
  
public class Solution {  
  
 public class Cell {  
 int row;  
 int col;  
 int height;  
 public Cell(int row, int col, int height) {  
 this.row = row;  
 this.col = col;  
 this.height = height;  
 }  
 }  
  
 public int trapRainWater(int[][] heights) {  
 if (heights == null || heights.length == 0 || heights[0].length == 0)  
 return 0;  
  
 PriorityQueue<Cell> queue = new PriorityQueue<>(1, new Comparator<Cell>(){  
 public int compare(Cell a, Cell b) {  
 return a.height - b.height;  
 }  
 });  
   
 int m = heights.length;  
 int n = heights[0].length;  
 boolean[][] visited = new boolean[m][n];  
  
 // Initially, add all the Cells which are on borders to the queue.  
 for (int i = 0; i < m; i++) {  
 visited[i][0] = true;  
 visited[i][n - 1] = true;  
 queue.offer(new Cell(i, 0, heights[i][0]));  
 queue.offer(new Cell(i, n - 1, heights[i][n - 1]));  
 }  
  
 for (int i = 0; i < n; i++) {  
 visited[0][i] = true;  
 visited[m - 1][i] = true;  
 queue.offer(new Cell(0, i, heights[0][i]));  
 queue.offer(new Cell(m - 1, i, heights[m - 1][i]));  
 }  
  
 // from the borders, pick the shortest cell visited and check its neighbors:  
 // if the neighbor is shorter, collect the water it can trap and update its height as its height plus the water trapped  
 // add all its neighbors to the queue.  
 int[][] dirs = new int[][]{{-1, 0}, {1, 0}, {0, -1}, {0, 1}};  
 int res = 0;  
 while (!queue.isEmpty()) {  
 Cell cell = queue.poll();  
 for (int[] dir : dirs) {  
 int row = cell.row + dir[0];  
 int col = cell.col + dir[1];  
 if (row >= 0 && row < m && col >= 0 && col < n && !visited[row][col]) {  
 visited[row][col] = true;  
 res += Math.max(0, cell.height - heights[row][col]);  
 queue.offer(new Cell(row, col, Math.max(heights[row][col], cell.height)));  
 }  
 }  
 }  
   
 return res;  
 }  
}  
  
  
*409,Longest Palindrome:***Python\_solution:**What are the odds? (Python & C++)   
def longestPalindrome(self, s):  
 odds = sum(v & 1 for v in collections.Counter(s).values())  
 return len(s) - odds + bool(odds)  
  
**Best\_solution:**Simple HashSet solution Java   
public int longestPalindrome(String s) {  
 if(s==null || s.length()==0) return 0;  
 HashSet<Character> hs = new HashSet<Character>();  
 int count = 0;  
 for(int i=0; i<s.length(); i++){  
 if(hs.contains(s.charAt(i))){  
 hs.remove(s.charAt(i));  
 count++;  
 }else{  
 hs.add(s.charAt(i));  
 }  
 }  
 if(!hs.isEmpty()) return count\*2+1;  
 return count\*2;  
}  
  
  
*410,Split Array Largest Sum:***Python\_solution:**Python solution dp and binary search   
import sys  
class Solution(object):  
 def splitArray(self, nums, m):  
 """  
 :type nums: List[int]  
 :type m: int  
 :rtype: int  
 """  
 dp = [[sys.maxint]\*(m) for \_ in range(len(nums)+1)]  
 acc = 0  
 dp[0][0] = 0  
 for i in range(1, len(nums)+1):  
 acc += nums[i - 1]  
 dp[i][0] = acc  
  
 for j in range(m):  
 dp[0][j] = 0  
  
 for i in range(1, len(nums)+1):  
 for i\_ in range(i):  
 for j in range(1, m):  
 dp[i][j] = min(dp[i][j], max(dp[i\_][j-1], dp[i][0]-dp[i\_][0]))  
 #print dp  
 return dp[len(nums)][m-1]  
  
**Best\_solution:**Clear Explanation: 8ms Binary Search Java   
l = max number of array; r = sum of all numbers in the array;  
  
*412,Fizz Buzz:***Python\_solution:**Python Golf   
def fizzBuzz(self, n):  
 return['FizzBuzz'[i%-3&-4:i%-5&8^12]or`i`for i in range(1,n+1)]  
  
**Best\_solution:**Java 4ms solution , Not using "%" operation   
public class Solution {  
 public List<String> fizzBuzz(int n) {  
 List<String> ret = new ArrayList<String>(n);  
 for(int i=1,fizz=0,buzz=0;i<=n ;i++){  
 fizz++;  
 buzz++;  
 if(fizz==3 && buzz==5){  
 ret.add("FizzBuzz");  
 fizz=0;  
 buzz=0;  
 }else if(fizz==3){  
 ret.add("Fizz");  
 fizz=0;  
 }else if(buzz==5){  
 ret.add("Buzz");  
 buzz=0;  
 }else{  
 ret.add(String.valueOf(i));  
 }  
 }   
 return ret;  
 }  
}  
  
  
*413,Arithmetic Slices:***Python\_solution:**Python DP solution   
def numberOfArithmeticSlices(self, A):  
 """  
 :type A: List[int]  
 :rtype: int  
 """  
 opt, i = [0,0], 1  
 for j in xrange(2,len(A)):  
 if A[j]-A[j-1] == A[j-1]-A[j-2]:  
 opt.append(opt[j-1]+i)  
 i += 1  
 else:  
 opt.append(opt[j-1])  
 i = 1  
 return opt[-1]  
**Best\_solution:**Simple Java solution 9 lines, 2ms   
public int numberOfArithmeticSlices(int[] A) {  
 int curr = 0, sum = 0;  
 for (int i=2; i<A.length; i++)  
 if (A[i]-A[i-1] == A[i-1]-A[i-2]) {  
 curr += 1;  
 sum += curr;  
 } else {  
 curr = 0;  
 }  
 return sum;  
}  
  
*414,Third Maximum Number:***Python\_solution:**Intuitive and Short Python solution   
class Solution(object):  
 def thirdMax(self, nums):  
 v = [float('-inf'), float('-inf'), float('-inf')]  
 for num in nums:  
 if num not in v:  
 if num > v[0]: v = [num, v[0], v[1]]  
 elif num > v[1]: v = [v[0], num, v[1]]  
 elif num > v[2]: v = [v[0], v[1], num]  
 return max(nums) if float('-inf') in v else v[2]  
  
**Best\_solution:**Java neat and easy understand solution, O(n) time, O(1) space   
 public int thirdMax(int[] nums) {  
 Integer max1 = null;  
 Integer max2 = null;  
 Integer max3 = null;  
 for (Integer n : nums) {  
 if (n.equals(max1) || n.equals(max2) || n.equals(max3)) continue;  
 if (max1 == null || n > max1) {  
 max3 = max2;  
 max2 = max1;  
 max1 = n;  
 } else if (max2 == null || n > max2) {  
 max3 = max2;  
 max2 = n;  
 } else if (max3 == null || n > max3) {  
 max3 = n;  
 }  
 }  
 return max3 == null ? max1 : max3;  
 }  
  
  
*415,Add Strings:***Python\_solution:**Python: 7-line & 52ms (+ 1-liner for fun)   
def addStrings(self, num1, num2):  
 z = itertools.izip\_longest(num1[::-1], num2[::-1], fillvalue='0')  
 res, carry, zero2 = [], 0, 2\*ord('0')  
 for i in z:  
 cur\_sum = ord(i[0]) + ord(i[1]) - zero2 + carry  
 res.append(str(cur\_sum % 10))  
 carry = cur\_sum // 10  
 return ('1' if carry else '') + ''.join(res[::-1])  
  
**Best\_solution:**Straightforward Java 8 main lines 25ms   
public class Solution {  
 public String addStrings(String num1, String num2) {  
 StringBuilder sb = new StringBuilder();  
 int carry = 0;  
 for(int i = num1.length() - 1, j = num2.length() - 1; i >= 0 || j >= 0 || carry == 1; i--, j--){  
 int x = i < 0 ? 0 : num1.charAt(i) - '0';  
 int y = j < 0 ? 0 : num2.charAt(j) - '0';  
 sb.append((x + y + carry) % 10);  
 carry = (x + y + carry) / 10;  
 }  
 return sb.reverse().toString();  
 }  
}  
  
  
*416,Partition Equal Subset Sum:***Python\_solution:**7 Lines 59ms Recursive Python Solution   
class Solution(object):  
 def canPartition(self, nums):  
 nums.sort(reverse=True)  
 def helper(start, target): # Here path is not needed  
 if target < 0: return  
 elif target == 0: return True  
 for i in xrange(start, len(nums)):  
 if helper(i+1, target-nums[i]): return True  
 return False  
  
 return False if sum(nums)%2 else helper(0, sum(nums)/2)  
  
**Best\_solution:**0/1 knapsack detailed explanation   
public boolean canPartition(int[] nums) {  
 int sum = 0;  
   
 for (int num : nums) {  
 sum += num;  
 }  
   
 if ((sum & 1) == 1) {  
 return false;  
 }  
 sum /= 2;  
  
 int n = nums.length;  
 boolean[][] dp = new boolean[n+1][sum+1];  
 for (int i = 0; i < dp.length; i++) {  
 Arrays.fill(dp[i], false);  
 }  
   
 dp[0][0] = true;  
   
 for (int i = 1; i < n+1; i++) {  
 dp[i][0] = true;  
 }  
 for (int j = 1; j < sum+1; j++) {  
 dp[0][j] = false;  
 }  
   
 for (int i = 1; i < n+1; i++) {  
 for (int j = 1; j < sum+1; j++) {  
 dp[i][j] = dp[i-1][j];  
 if (j >= nums[i-1]) {  
 dp[i][j] = (dp[i][j] || dp[i-1][j-nums[i-1]]);  
 }  
 }  
 }  
   
 return dp[n][sum];  
}  
  
  
*417,Pacific Atlantic Water Flow:***Python\_solution:**Python DFS bests 85%. Tips for all DFS in matrix question.   
self.directions = [(1,0),(-1,0),(0,1),(0,-1)]  
**Best\_solution:**Java BFS & DFS from Ocean   
public class Solution {  
 int[][]dir = new int[][]{{1,0},{-1,0},{0,1},{0,-1}};  
 public List<int[]> pacificAtlantic(int[][] matrix) {  
 List<int[]> res = new LinkedList<>();  
 if(matrix == null || matrix.length == 0 || matrix[0].length == 0){  
 return res;  
 }  
 int n = matrix.length, m = matrix[0].length;  
 //One visited map for each ocean  
 boolean[][] pacific = new boolean[n][m];  
 boolean[][] atlantic = new boolean[n][m];  
 Queue<int[]> pQueue = new LinkedList<>();  
 Queue<int[]> aQueue = new LinkedList<>();  
 for(int i=0; i<n; i++){ //Vertical border  
 pQueue.offer(new int[]{i, 0});  
 aQueue.offer(new int[]{i, m-1});  
 pacific[i][0] = true;  
 atlantic[i][m-1] = true;  
 }  
 for(int i=0; i<m; i++){ //Horizontal border  
 pQueue.offer(new int[]{0, i});  
 aQueue.offer(new int[]{n-1, i});  
 pacific[0][i] = true;  
 atlantic[n-1][i] = true;  
 }  
 bfs(matrix, pQueue, pacific);  
 bfs(matrix, aQueue, atlantic);  
 for(int i=0; i<n; i++){  
 for(int j=0; j<m; j++){  
 if(pacific[i][j] && atlantic[i][j])  
 res.add(new int[]{i,j});  
 }  
 }  
 return res;  
 }  
 public void bfs(int[][]matrix, Queue<int[]> queue, boolean[][]visited){  
 int n = matrix.length, m = matrix[0].length;  
 while(!queue.isEmpty()){  
 int[] cur = queue.poll();  
 for(int[] d:dir){  
 int x = cur[0]+d[0];  
 int y = cur[1]+d[1];  
 if(x<0 || x>=n || y<0 || y>=m || visited[x][y] || matrix[x][y] < matrix[cur[0]][cur[1]]){  
 continue;  
 }  
 visited[x][y] = true;  
 queue.offer(new int[]{x, y});  
 }   
 }  
 }  
}  
  
  
*419,Battleships in a Board:***Python\_solution:**Python solution   
class Solution(object):  
 def countBattleships(self, board):  
 if len(board) == 0: return 0  
 m, n = len(board), len(board[0])  
 count = 0  
 for i in range(m):  
 for j in range(n):  
 if board[i][j] == 'X' and (i == 0 or board[i-1][j] == '.') and (j == 0 or board[i][j-1] == '.'):  
 count += 1  
 return count  
  
**Best\_solution:**Simple Java Solution   
  
 public int countBattleships(char[][] board) {  
 int m = board.length;  
 if (m==0) return 0;  
 int n = board[0].length;  
   
 int count=0;  
   
 for (int i=0; i<m; i++) {  
 for (int j=0; j<n; j++) {  
 if (board[i][j] == '.') continue;  
 if (i > 0 && board[i-1][j] == 'X') continue;  
 if (j > 0 && board[i][j-1] == 'X') continue;  
 count++;  
 }  
 }  
   
 return count;  
 }  
  
  
*420,Strong Password Checker:***Python\_solution:**Simple Python solution   
class Solution(object):  
 def strongPasswordChecker(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 missing\_type = 3  
 if any('a' <= c <= 'z' for c in s): missing\_type -= 1  
 if any('A' <= c <= 'Z' for c in s): missing\_type -= 1  
 if any(c.isdigit() for c in s): missing\_type -= 1  
  
 change = 0  
 one = two = 0  
 p = 2  
 while p < len(s):  
 if s[p] == s[p-1] == s[p-2]:  
 length = 2  
 while p < len(s) and s[p] == s[p-1]:  
 length += 1  
 p += 1  
   
 change += length / 3  
 if length % 3 == 0: one += 1  
 elif length % 3 == 1: two += 1  
 else:  
 p += 1  
   
 if len(s) < 6:  
 return max(missing\_type, 6 - len(s))  
 elif len(s) <= 20:  
 return max(missing\_type, change)  
 else:  
 delete = len(s) - 20  
   
 change -= min(delete, one)  
 change -= min(max(delete - one, 0), two \* 2) / 2  
 change -= max(delete - one - 2 \* two, 0) / 3  
   
 return delete + max(missing\_type, change)  
  
**Best\_solution:**C++ 0ms O(n) 35 lines solution with detailed explanation   
s.length() < 6  
  
*421,Maximum XOR of Two Numbers in an Array:***Python\_solution:**Python 6 lines, bit by bit   
def findMaximumXOR(self, nums):  
 answer = 0  
 for i in range(32)[::-1]:  
 answer <<= 1  
 prefixes = {num >> i for num in nums}  
 answer += any(answer^1 ^ p in prefixes for p in prefixes)  
 return answer  
  
**Best\_solution:**Java O(n) solution using bit manipulation and HashMap   
public class Solution {  
 public int findMaximumXOR(int[] nums) {  
 int max = 0, mask = 0;  
 for(int i = 31; i >= 0; i--){  
 mask = mask | (1 << i);  
 Set<Integer> set = new HashSet<>();  
 for(int num : nums){  
 set.add(num & mask);  
 }  
 int tmp = max | (1 << i);  
 for(int prefix : set){  
 if(set.contains(tmp ^ prefix)) {  
 max = tmp;  
 break;  
 }  
 }  
 }  
 return max;  
 }  
}  
  
  
*423,Reconstruct Original Digits from English:***Python\_solution:**python: solve valid equation problem   
class Solution(object):  
 def originalDigits(self, s):  
 """  
 :type s: str  
 :rtype: str  
 """  
 dic = {}  
 for ch in s:  
 dic[ch] = dic.get(ch, 0) + 1  
 ret = []  
 ret.extend( ['0'] \* dic.get('z', 0) )  
 ret.extend( ['1'] \* (dic.get('o', 0)-dic.get('z', 0)-dic.get('w', 0)-dic.get('u', 0)) )  
 ret.extend( ['2'] \* dic.get('w', 0) )  
 ret.extend( ['3'] \* (dic.get('h', 0)-dic.get('g', 0)) )  
 ret.extend( ['4'] \* dic.get('u', 0) )  
 ret.extend( ['5'] \* (dic.get('f', 0)-dic.get('u', 0)) )  
 ret.extend( ['6'] \* dic.get('x', 0) )  
 ret.extend( ['7'] \* (dic.get('s', 0)-dic.get('x', 0)) )  
 ret.extend( ['8'] \* dic.get('g', 0) )  
 ret.extend( ['9'] \* (dic.get('i', 0)-dic.get('g', 0)-dic.get('x', 0)-dic.get('f', 0)+dic.get('u', 0) ) )  
 return ''.join( ret )  
  
  
**Best\_solution:**one pass O(n) JAVA Solution, Simple and Clear   
public String originalDigits(String s) {  
 int[] count = new int[10];  
 for (int i = 0; i < s.length(); i++){  
 char c = s.charAt(i);  
 if (c == 'z') count[0]++;  
 if (c == 'w') count[2]++;  
 if (c == 'x') count[6]++;  
 if (c == 's') count[7]++; //7-6  
 if (c == 'g') count[8]++;  
 if (c == 'u') count[4]++;   
 if (c == 'f') count[5]++; //5-4  
 if (c == 'h') count[3]++; //3-8  
 if (c == 'i') count[9]++; //9-8-5-6  
 if (c == 'o') count[1]++; //1-0-2-4  
 }  
 count[7] -= count[6];  
 count[5] -= count[4];  
 count[3] -= count[8];  
 count[9] = count[9] - count[8] - count[5] - count[6];  
 count[1] = count[1] - count[0] - count[2] - count[4];  
 StringBuilder sb = new StringBuilder();  
 for (int i = 0; i <= 9; i++){  
 for (int j = 0; j < count[i]; j++){  
 sb.append(i);  
 }  
 }  
 return sb.toString();  
}  
  
*424,Longest Repeating Character Replacement:***Python\_solution:**Consise Python sliding window   
def characterReplacement(self, s, k):  
 res = lo = hi = 0  
 counts = collections.Counter()  
 for hi in range(1, len(s)+1):  
 counts[s[hi-1]] += 1  
 max\_char\_n = counts.most\_common(1)[0][1]  
 if hi - lo - max\_char\_n > k:  
 counts[s[lo]] -= 1  
 lo += 1  
 return hi - lo  
  
**Best\_solution:**Java 12 lines O(n) sliding window solution with explanation   
 public int characterReplacement(String s, int k) {  
 int len = s.length();  
 int[] count = new int[26];  
 int start = 0, maxCount = 0, maxLength = 0;  
 for (int end = 0; end < len; end++) {  
 maxCount = Math.max(maxCount, ++count[s.charAt(end) - 'A']);  
 while (end - start + 1 - maxCount > k) {  
 count[s.charAt(start) - 'A']--;  
 start++;  
 }  
 maxLength = Math.max(maxLength, end - start + 1);  
 }  
 return maxLength;  
 }  
  
  
*426,All O`one Data Structure:***Python\_solution:**Accepted Java and Python solution   
public class AllOne {  
 Node head;  
 Node tail;  
   
 Map<String, Integer> keyCountMap;  
 Map<Integer, Node> countNodeMap;  
 Map<Integer, Set<String>> countKeyMap;  
   
 class Node {  
 int count;  
 Node prev;  
 Node next;  
   
 public Node(int cnt) {  
 count = cnt;  
 prev = null;  
 next = null;  
 }  
 }  
 /\*\* Initialize your data structure here. \*/  
 public AllOne() {  
 head = new Node(0);  
 tail = new Node(Integer.MAX\_VALUE);  
 head.next = tail;  
 tail.prev = head;  
   
 keyCountMap = new HashMap<>();  
 countNodeMap = new HashMap<>();  
 countKeyMap = new HashMap<>();  
   
 countNodeMap.put(0, head);  
 countNodeMap.put(Integer.MAX\_VALUE, tail);  
 }  
   
 /\*\* Inserts a new key <Key> with value 1. Or increments an existing key by 1. \*/  
 public void inc(String key) {  
 if (!keyCountMap.containsKey(key)) {  
 keyCountMap.put(key, 0);  
 }  
   
 int preCount = keyCountMap.get(key);  
 Node preNode = countNodeMap.get(preCount);  
   
 keyCountMap.put(key, preCount + 1);  
 int newCount = keyCountMap.get(key);  
   
 //insert  
 //new count is created  
 if (newCount != preNode.next.count) {  
 Node newNode = new Node(newCount);  
 insert(preNode, newNode);  
   
 countKeyMap.put(newCount, new HashSet<String>());  
 countNodeMap.put(newCount, newNode);  
 }  
 countKeyMap.get(newCount).add(key);  
   
 //delete old  
 if (preCount > 0) {  
 Set<String> oldSet = countKeyMap.get(preCount);  
 oldSet.remove(key);  
 if (oldSet.isEmpty()) {  
 delete(preNode);  
 countKeyMap.remove(preCount);  
 countNodeMap.remove(preCount);  
 }   
 }  
  
 }  
   
 /\*\* Decrements an existing key by 1. If Key's value is 1, remove it from the data structure. \*/  
 public void dec(String key) {  
 if (!keyCountMap.containsKey(key)) return;  
   
 int preCount = keyCountMap.get(key);  
 Node preNode = countNodeMap.get(preCount);  
   
 keyCountMap.put(key, preCount - 1);  
 int newCount = keyCountMap.get(key);  
   
 //insert  
 //new count occurs  
 if (newCount != 0) {  
 if (newCount != preNode.prev.count) {  
 Node newNode = new Node(newCount);  
 insert(preNode.prev, newNode);  
   
 countKeyMap.put(newCount, new HashSet<String>());  
 countNodeMap.put(newCount, newNode);  
 }  
 countKeyMap.get(newCount).add(key);   
 }  
 else keyCountMap.remove(key);  
  
 //delete  
 Set<String> oldSet = countKeyMap.get(preCount);  
 oldSet.remove(key);  
 if (oldSet.isEmpty()) {  
 delete(preNode);  
 countKeyMap.remove(preCount);  
 countNodeMap.remove(preCount);  
 }  
 }  
   
 /\*\* Returns one of the keys with maximal value. \*/  
 public String getMaxKey() {  
 if (head.next == tail) {  
 System.out.println("head == tail");  
 return "";  
 }  
 Set<String> set = countKeyMap.get(tail.prev.count);  
 return set.iterator().next();  
 }  
   
 /\*\* Returns one of the keys with Minimal value. \*/  
 public String getMinKey() {  
 if (head.next == tail) return "";  
 Set<String> set = countKeyMap.get(head.next.count);  
 return set.iterator().next();  
 }  
   
 public void insert(Node preNode, Node node) {  
 node.next = preNode.next;  
 node.prev = preNode;  
   
 node.next.prev = node;  
 node.prev.next = node;  
 }  
   
 public void delete(Node node) {  
 node.next.prev = node.prev;  
 node.prev.next = node.next;  
 }  
}  
  
**Best\_solution:**All in O(1), with detailed explantation   
"A": 4, "B": 4, "C": 2, "D": 1  
  
  
*427,Minimum Genetic Mutation:***Best\_solution:**Java Solution using BFS   
public class Solution {  
 public int minMutation(String start, String end, String[] bank) {  
 if(start.equals(end)) return 0;  
   
 Set<String> bankSet = new HashSet<>();  
 for(String b: bank) bankSet.add(b);  
   
 char[] charSet = new char[]{'A', 'C', 'G', 'T'};  
   
 int level = 0;  
 Set<String> visited = new HashSet<>();  
 Queue<String> queue = new LinkedList<>();  
 queue.offer(start);  
 visited.add(start);  
   
 while(!queue.isEmpty()) {  
 int size = queue.size();  
 while(size-- > 0) {  
 String curr = queue.poll();  
 if(curr.equals(end)) return level;  
   
 char[] currArray = curr.toCharArray();  
 for(int i = 0; i < currArray.length; i++) {  
 char old = currArray[i];  
 for(char c: charSet) {  
 currArray[i] = c;  
 String next = new String(currArray);  
 if(!visited.contains(next) && bankSet.contains(next)) {  
 visited.add(next);  
 queue.offer(next);  
 }  
 }  
 currArray[i] = old;  
 }  
 }  
 level++;  
 }  
 return -1;  
 }  
}  
  
  
*428,Number of Segments in a String:***Best\_solution:**Clean java solution O(n)   
public int countSegments(String s) {  
 int res=0;  
 for(int i=0; i<s.length(); i++)  
 if(s.charAt(i)!=' ' && (i==0 || s.charAt(i-1)==' '))  
 res++;   
 return res;  
}  
  
Time complexity: O(n)  
Space complexity: O(1)  
  
*429,Non-overlapping Intervals:***Python\_solution:**Short Ruby and Python   
end  
**Best\_solution:**Java: Least is Most   
 public int eraseOverlapIntervals(Interval[] intervals) {  
 if (intervals.length == 0) return 0;  
  
 Arrays.sort(intervals, new myComparator());  
 int end = intervals[0].end;  
 int count = 1;   
  
 for (int i = 1; i < intervals.length; i++) {  
 if (intervals[i].start >= end) {  
 end = intervals[i].end;  
 count++;  
 }  
 }  
 return intervals.length - count;  
 }  
   
 class myComparator implements Comparator<Interval> {  
 public int compare(Interval a, Interval b) {  
 return a.end - b.end;  
 }  
 }  
  
  
*430,Find Right Interval:***Python\_solution:**Python O(nlogn) short solution with explanation   
def findRightInterval(self, intervals):  
 l = sorted((e.start, i) for i, e in enumerate(intervals))  
 res = []  
 for e in intervals:  
 r = bisect.bisect\_left(l, (e.end,))  
 res.append(l[r][1] if r < len(l) else -1)  
 return res  
  
**Best\_solution:**Java clear O(n logn) solution based on TreeMap   
public class Solution {  
 public int[] findRightInterval(Interval[] intervals) {  
 int[] result = new int[intervals.length];  
 java.util.NavigableMap<Integer, Integer> intervalMap = new TreeMap<>();  
   
 for (int i = 0; i < intervals.length; ++i) {  
 intervalMap.put(intervals[i].start, i);   
 }  
   
 for (int i = 0; i < intervals.length; ++i) {  
 Map.Entry<Integer, Integer> entry = intervalMap.ceilingEntry(intervals[i].end);  
 result[i] = (entry != null) ? entry.getValue() : -1;  
 }  
   
 return result;  
 }  
}  
  
  
*431,Path Sum III:***Python\_solution:**Python solution with detailed explanation   
class SolutionBruteForce(object):  
 def find\_paths(self, root, target):  
 if root:  
 return int(root.val == target) + self.find\_paths(root.left, target-root.val) + self.find\_paths(root.right, target-root.val)  
 return 0  
  
 def pathSum(self, root, sum):  
 """  
 :type root: TreeNode  
 :type sum: int  
 :rtype: int  
 """  
 if root:  
 return self.find\_paths(root, sum) + self.pathSum(root.left, sum) + self.pathSum(root.right, sum)  
 return 0  
  
**Best\_solution:**17 ms O(n) java Prefix sum method   
 public int pathSum(TreeNode root, int sum) {  
 HashMap<Integer, Integer> preSum = new HashMap();  
 preSum.put(0,1);  
 helper(root, 0, sum, preSum);  
 return count;  
 }  
 int count = 0;  
 public void helper(TreeNode root, int currSum, int target, HashMap<Integer, Integer> preSum) {  
 if (root == null) {  
 return;  
 }  
   
 currSum += root.val;  
  
 if (preSum.containsKey(currSum - target)) {  
 count += preSum.get(currSum - target);  
 }  
   
 if (!preSum.containsKey(currSum)) {  
 preSum.put(currSum, 1);  
 } else {  
 preSum.put(currSum, preSum.get(currSum)+1);  
 }  
   
 helper(root.left, currSum, target, preSum);  
 helper(root.right, currSum, target, preSum);  
 preSum.put(currSum, preSum.get(sum) - 1);  
 }  
  
  
*432,Find All Anagrams in a String:***Python\_solution:**Python Sliding Window Solution using Counter   
 from collections import Counter  
  
 def findAnagrams(self, s, p):  
 """  
 :type s: str  
 :type p: str  
 :rtype: List[int]  
 """  
 res = []  
 pCounter = Counter(p)  
 sCounter = Counter(s[:len(p)-1])  
 for i in range(len(p)-1,len(s)):  
 sCounter[s[i]] += 1 # include a new char in the window  
 if sCounter == pCounter: # This step is O(1), since there are at most 26 English letters   
 res.append(i-len(p)+1) # append the starting index  
 sCounter[s[i-len(p)+1]] -= 1 # decrease the count of oldest char in the window  
 if sCounter[s[i-len(p)+1]] == 0:  
 del sCounter[s[i-len(p)+1]] # remove the count if it is 0  
 return res  
  
**Best\_solution:**Shortest/Concise JAVA O(n) Sliding Window Solution   
public List<Integer> findAnagrams(String s, String p) {  
 List<Integer> list = new ArrayList<>();  
 if (s == null || s.length() == 0 || p == null || p.length() == 0) return list;  
 int[] hash = new int[256]; //character hash  
 //record each character in p to hash  
 for (char c : p.toCharArray()) {  
 hash[c]++;  
 }  
 //two points, initialize count to p's length  
 int left = 0, right = 0, count = p.length();  
 while (right < s.length()) {  
 //move right everytime, if the character exists in p's hash, decrease the count  
 //current hash value >= 1 means the character is existing in p  
 if (hash[s.charAt(right++)]-- >= 1) count--;   
   
 //when the count is down to 0, means we found the right anagram  
 //then add window's left to result list  
 if (count == 0) list.add(left);  
   
 //if we find the window's size equals to p, then we have to move left (narrow the window) to find the new match window  
 //++ to reset the hash because we kicked out the left  
 //only increase the count if the character is in p  
 //the count >= 0 indicate it was original in the hash, cuz it won't go below 0  
 if (right - left == p.length() && hash[s.charAt(left++)]++ >= 0) count++;  
 }  
 return list;  
}  
  
*434,K-th Smallest in Lexicographical Order:***Python\_solution:**C++/Python 0ms O((log n)^2)-time O(1)-space super easy solution with detailed explanations   
result  
**Best\_solution:**Concise/Easy-to-understand Java 5ms solution with Explaination   
public int findKthNumber(int n, int k) {  
 int curr = 1;  
 k = k - 1;  
 while (k > 0) {  
 int steps = calSteps(n, curr, curr + 1);  
 if (steps <= k) {  
 curr += 1;  
 k -= steps;  
 } else {  
 curr \*= 10;  
 k -= 1;  
 }  
 }  
 return curr;  
}  
//use long in case of overflow  
public int calSteps(int n, long n1, long n2) {  
 int steps = 0;  
 while (n1 <= n) {  
 steps += Math.min(n + 1, n2) - n1;  
 n1 \*= 10;  
 n2 \*= 10;  
 }  
 return steps;  
}  
  
*435,Arranging Coins:***Best\_solution:**[JAVA] Clean Code with Explanations and Running Time [2 Solutions]   
public class Solution {  
 public int arrangeCoins(int n) {  
 int start = 0;  
 int end = n;  
 int mid = 0;  
 while (start <= end){  
 mid = (start + end) >>> 1;  
 if ((0.5 \* mid \* mid + 0.5 \* mid ) <= n){  
 start = mid + 1;  
 }else{  
 end = mid - 1;  
 }  
 }  
 return start - 1;  
 }  
}  
  
  
*436,Find All Duplicates in an Array:***Python\_solution:**Python O(n) time O(1) space   
class Solution(object):  
 def findDuplicates(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: List[int]  
 """  
 res = []  
 for x in nums:  
 if nums[abs(x)-1] < 0:  
 res.append(abs(x))  
 else:  
 nums[abs(x)-1] \*= -1  
 return res  
  
**Best\_solution:**Java Simple Solution   
public class Solution {  
 // when find a number i, flip the number at position i-1 to negative.   
 // if the number at position i-1 is already negative, i is the number that occurs twice.  
   
 public List<Integer> findDuplicates(int[] nums) {  
 List<Integer> res = new ArrayList<>();  
 for (int i = 0; i < nums.length; ++i) {  
 int index = Math.abs(nums[i])-1;  
 if (nums[index] < 0)  
 res.add(Math.abs(index+1));  
 nums[index] = -nums[index];  
 }  
 return res;  
 }  
}  
  
  
*438,Add Two Numbers II:***Python\_solution:**There is no maximum of INT in python, so.....   
def addTwoNumbers(self, l1, l2):  
  
 x1, x2 = 0, 0  
 while l1:  
 x1 = x1\*10+l1.val  
 l1 = l1.next  
 while l2:  
 x2 = x2\*10+l2.val  
 l2 = l2.next  
 x = x1 + x2  
   
 head = ListNode(0)  
 if x == 0: return head  
 while x:  
 v, x = x%10, x//10  
 head.next, head.next.next = ListNode(v), head.next  
   
 return head.next  
  
**Best\_solution:**Easy O(n) Java Solution using Stack   
public class Solution {  
 public ListNode addTwoNumbers(ListNode l1, ListNode l2) {  
 Stack<Integer> s1 = new Stack<Integer>();  
 Stack<Integer> s2 = new Stack<Integer>();  
   
 while(l1 != null) {  
 s1.push(l1.val);  
 l1 = l1.next;  
 };  
 while(l2 != null) {  
 s2.push(l2.val);  
 l2 = l2.next;  
 }  
   
 int sum = 0;  
 ListNode list = new ListNode(0);  
 while (!s1.empty() || !s2.empty()) {  
 if (!s1.empty()) sum += s1.pop();  
 if (!s2.empty()) sum += s2.pop();  
 list.val = sum % 10;  
 ListNode head = new ListNode(sum / 10);  
 head.next = list;  
 list = head;  
 sum /= 10;  
 }  
   
 return list.val == 0 ? list.next : list;  
 }  
}  
  
  
*439,Arithmetic Slices II - Subsequence:***Python\_solution:**Python Solution from the Author   
class Solution(object):  
 def numberOfArithmeticSlices(self, A):  
 """  
 :type A: List[int]  
 :rtype: int  
 """  
  
 lookup = {}  
  
 for i, a in enumerate(A):  
 if a in lookup:  
 lookup[a].append(i)  
 else:  
 lookup[a] = [i]  
  
 dp = []  
 for \_ in range(len(A)):  
 dp.append({})  
  
 for k, num in enumerate(A):  
 for i in range(0, k):  
 diff = A[k] - A[i]  
 X = A[i] - diff  
 if X in lookup:  
 for index in lookup[X]:  
 if index < i:  
 dp[k][diff] = dp[k].get(diff, 0) + 1  
  
 if diff in dp[i]:  
 dp[k][diff] = dp[k].get(diff, 0) + dp[i][diff]  
  
 res = 0  
 for x in dp:  
 for k in x:  
 res += x[k]  
  
 return res  
**Best\_solution:**Detailed explanation for Java O(n^2) solution   
T(i)  
  
*440,Number of Boomerangs:***Python\_solution:**Short Python O(n^2) hashmap solution   
 res = 0  
 for p in points:  
 cmap = {}  
 for q in points:  
 f = p[0]-q[0]  
 s = p[1]-q[1]  
 cmap[f\*f + s\*s] = 1 + cmap.get(f\*f + s\*s, 0)  
 for k in cmap:  
 res += cmap[k] \* (cmap[k] -1)  
 return res  
  
  
**Best\_solution:**Clean java solution: O(n^2) 166ms   
public int numberOfBoomerangs(int[][] points) {  
 int res = 0;  
  
 Map<Integer, Integer> map = new HashMap<>();  
 for(int i=0; i<points.length; i++) {  
 for(int j=0; j<points.length; j++) {  
 if(i == j)  
 continue;  
   
 int d = getDistance(points[i], points[j]);   
 map.put(d, map.getOrDefault(d, 0) + 1);  
 }  
   
 for(int val : map.values()) {  
 res += val \* (val-1);  
 }   
 map.clear();  
 }  
   
 return res;  
}  
  
private int getDistance(int[] a, int[] b) {  
 int dx = a[0] - b[0];  
 int dy = a[1] - b[1];  
   
 return dx\*dx + dy\*dy;  
}  
  
Time complexity: O(n^2)  
Space complexity: O(n)  
  
*441,Find All Numbers Disappeared in an Array:***Python\_solution:**Python 4 lines with short explanation   
class Solution(object):  
 def findDisappearedNumbers(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: List[int]  
 """  
 # For each number i in nums,  
 # we mark the number that i points as negative.  
 # Then we filter the list, get all the indexes  
 # who points to a positive number  
 for i in xrange(len(nums)):  
 index = abs(nums[i]) - 1  
 nums[index] = - abs(nums[index])  
  
 return [i + 1 for i in range(len(nums)) if nums[i] > 0]  
  
  
**Best\_solution:**Java accepted simple solution   
nums[nums[i] -1] = -nums[nums[i]-1]  
  
*442,Serialize and Deserialize BST:***Python\_solution:**Python O( N ) solution. easy to understand   
class Codec:  
  
 def serialize(self, root):  
 vals = []  
  
 def preOrder(node):  
 if node:  
 vals.append(node.val)  
 preOrder(node.left)  
 preOrder(node.right)  
  
 preOrder(root)  
  
 return ' '.join(map(str, vals))  
  
 # O( N ) since each val run build once  
 def deserialize(self, data):  
 vals = collections.deque(int(val) for val in data.split())  
  
 def build(minVal, maxVal):  
 if vals and minVal < vals[0] < maxVal:  
 val = vals.popleft()  
 node = TreeNode(val)  
 node.left = build(minVal, val)  
 node.right = build(val, maxVal)  
 return node  
  
 return build(float('-infinity'), float('infinity'))  
  
  
**Best\_solution:**Java PreOrder + Queue solution   
root left1 left2 leftX right1 rightX  
  
  
*443,Delete Node in a BST:***Python\_solution:**Bottom-up Recursive Python Solution. O(log(n)) Time.   
class Solution(object):  
 def deleteNode(self, root, key):  
 """  
 :type root: TreeNode  
 :type key: int  
 :rtype: TreeNode  
 """  
 if not root: return None  
   
 if root.val == key:  
 if root.left:  
 # Find the right most leaf of the left sub-tree  
 left\_right\_most = root.left  
 while left\_right\_most.right:  
 left\_right\_most = left\_right\_most.right  
 # Attach right child to the right of that leaf  
 left\_right\_most.right = root.right  
 # Return left child instead of root, a.k.a delete root  
 return root.left  
 else:  
 return root.right  
 # If left or right child got deleted, the returned root is the child of the deleted node.  
 elif root.val > key:  
 root.left = self.deleteNode(root.left, key)  
 else:  
 root.right = self.deleteNode(root.right, key)  
   
 return root  
  
**Best\_solution:**Recursive Easy to Understand Java Solution   
public TreeNode deleteNode(TreeNode root, int key) {  
 if(root == null){  
 return null;  
 }  
 if(key < root.val){  
 root.left = deleteNode(root.left, key);  
 }else if(key > root.val){  
 root.right = deleteNode(root.right, key);  
 }else{  
 if(root.left == null){  
 return root.right;  
 }else if(root.right == null){  
 return root.left;  
 }  
   
 TreeNode minNode = findMin(root.right);  
 root.val = minNode.val;  
 root.right = deleteNode(root.right, root.val);  
 }  
 return root;  
}  
  
private TreeNode findMin(TreeNode node){  
 while(node.left != null){  
 node = node.left;  
 }  
 return node;  
}  
  
  
*444,Sort Characters By Frequency:***Python\_solution:**1 line Python code.   
class Solution(object):  
 def frequencySort(self, str):  
 """  
 :type str: str  
 :rtype: str  
 """  
 return "".join([char \* times for char, times in collections.Counter(str).most\_common()])  
  
**Best\_solution:**C++ O(n) solution without sort()   
class Solution {  
public:  
 string frequencySort(string s) {  
 unordered\_map<char,int> freq;  
 vector<string> bucket(s.size()+1, "");  
 string res;  
   
 //count frequency of each character  
 for(char c:s) freq[c]++;  
 //put character into frequency bucket  
 for(auto& it:freq) {  
 int n = it.second;  
 char c = it.first;  
 bucket[n].append(n, c);  
 }  
 //form descending sorted string  
 for(int i=s.size(); i>0; i--) {  
 if(!bucket[i].empty())  
 res.append(bucket[i]);  
 }  
 return res;  
 }  
};  
  
  
*445,Minimum Number of Arrows to Burst Balloons:***Python\_solution:**Greedy, Python (132 ms)   
class Solution(object):  
 def findMinArrowShots(self, points):  
 """  
 :type points: List[List[int]]  
 :rtype: int  
 """  
 points = sorted(points, key = lambda x: x[1])  
 res, end = 0, -float('inf')  
 for interval in points:  
 if interval[0] > end:  
 res += 1  
 end = interval[1]  
 return res  
  
**Best\_solution:**Java Greedy Soution   
public int findMinArrowShots(int[][] points) {  
 if(points==null || points.length==0 || points[0].length==0) return 0;  
 Arrays.sort(points, new Comparator<int[]>() {  
 public int compare(int[] a, int[] b) {  
 if(a[0]==b[0]) return a[1]-b[1];  
 else return a[0]-b[0];  
 }  
 });  
   
 int minArrows = 1;  
 int arrowLimit = points[0][1];  
 for(int i=1;i<points.length;i++) {  
 int[] baloon = points[i];  
 if(baloon[0]<=arrowLimit) {  
 arrowLimit=Math.min(arrowLimit, baloon[1]);  
 } else {  
 minArrows++;  
 arrowLimit=baloon[1];  
 }  
 }  
 return minArrows;  
}  
  
  
*446,Minimum Moves to Equal Array Elements:***Best\_solution:**Java O(n) solution. Short.   
1  
  
*447,4Sum II:***Python\_solution:**Easy 2 lines O(N^2) Python   
def fourSumCount(self, A, B, C, D):  
 AB = collections.Counter(a+b for a in A for b in B)  
 return sum(AB[-c-d] for c in C for d in D)  
**Best\_solution:**Clean java solution O(n^2)   
public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {  
 Map<Integer, Integer> map = new HashMap<>();  
   
 for(int i=0; i<C.length; i++) {  
 for(int j=0; j<D.length; j++) {  
 int sum = C[i] + D[j];  
 map.put(sum, map.getOrDefault(sum, 0) + 1);  
 }  
 }  
   
 int res=0;  
 for(int i=0; i<A.length; i++) {  
 for(int j=0; j<B.length; j++) {  
 res += map.getOrDefault(-1 \* (A[i]+B[j]), 0);  
 }  
 }  
   
 return res;  
}  
  
Time complexity: O(n^2)  
Space complexity: O(n^2)  
  
*448,Assign Cookies:***Python\_solution:**Python concise & efficient solution   
def findContentChildren(self, g, s):  
 g.sort()  
 s.sort()  
 res = 0  
 i = 0  
 for e in s:  
 if i == len(g):  
 break  
 if e >= g[i]:  
 res += 1  
 i += 1  
 return res  
  
**Best\_solution:**Simple Greedy Java Solution   
Arrays.sort(g);  
Arrays.sort(s);  
int i = 0;  
for(int j=0;i<g.length && j<s.length;j++) {  
 if(g[i]<=s[j]) i++;  
}  
return i;  
  
  
*449,132 Pattern:***Python\_solution:**Python solution in O(nlogn)   
left  
**Best\_solution:**Single pass C++ O(n) space and time solution (8 lines) with detailed explanation.   
s1,s2,s3  
  
*450,Circular Array Loop:***Python\_solution:**Python O(n) solution with explaination   
def circularArrayLoop(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: bool  
 """  
 for i in range(len(nums)):  
 count = 0  
 pre = i  
 isloop = True  
 if nums[i] == 0:   
 break  
 is\_forward = nums[i]>0  
 while count<len(nums):  
 count += 1  
 cur = (pre+nums[pre])%len(nums)  
 if pre == cur or (nums[cur]>0) ^ is\_forward:# stop if running into a dead end or different sign element  
 isloop = False  
 break  
 else:  
 pre = cur  
 if isloop:  
 return True  
 else: # mark all the elements on the wrong path as visited   
 pre = i  
 while count > 0:  
 cur = (pre+nums[pre])%len(nums)  
 nums[pre] = 0  
 pre = cur  
 count -= 1  
 return False  
  
**Best\_solution:**I cannot understand why test case [-2, 1, -1, -2, -2] gives false?   
None  
  
*451,Poor Pigs:***Best\_solution:**Another explanation and solution   
 1 2 3 4 5  
 6 7 8 9 10  
11 12 13 14 15  
16 17 18 19 20  
21 22 23 24 25  
  
  
*452,Repeated Substring Pattern:***Python\_solution:**Easy python solution with explaination   
def repeatedSubstringPattern(self, str):  
  
 """  
 :type str: str  
 :rtype: bool  
 """  
 if not str:  
 return False  
   
 ss = (str + str)[1:-1]  
 return ss.find(str) != -1  
  
**Best\_solution:**Easy python solution with explaination   
def repeatedSubstringPattern(self, str):  
  
 """  
 :type str: str  
 :rtype: bool  
 """  
 if not str:  
 return False  
   
 ss = (str + str)[1:-1]  
 return ss.find(str) != -1  
  
  
*453,LFU Cache:***Python\_solution:**Python shitty O(1) solution with two dict and one linkedlist   
class ListNode(object):  
 def \_\_init\_\_(self, key, val):  
 self.prev = None  
 self.next = None  
 self.val = val  
 self.key = key  
  
 def connect(self, nextNode):  
 self.next = nextNode  
 nextNode.prev = self  
  
class LFUCache(object):  
  
 def \_\_init\_\_(self, capacity):  
 """  
   
 :type capacity: int  
 """  
 self.cap = capacity  
 self.head = ListNode(None, None)  
 self.tail = ListNode(None, None)  
 self.head.connect(self.tail)  
 #use to record the first ListNode of this count number  
 self.cnt = {0: self.tail}  
 # key: key , value:[ListNode, visit count]  
 self.kv = {None:[self.tail, 0]}  
  
 def moveforward(self, key):  
 node, cnt = self.kv[key]  
 self.add('tmp', node.val, cnt + 1)  
 self.remove(key)  
 self.kv[key] = self.kv['tmp']  
 self.kv[key][0].key = key  
 del self.kv['tmp']  
  
 def get(self, key):  
 """  
 :type key: int  
 :rtype: int  
 """  
 if key not in self.kv:  
 return -1  
 self.moveforward(key)  
 return self.kv[key][0].val  
  
 def set(self, key, value):  
 """  
 :type key: int  
 :type value: int  
 :rtype: void  
 """  
 if self.cap == 0:  
 return  
 if key in self.kv:  
 self.kv[key][0].val = value  
 self.moveforward(key)  
 return  
 if len(self.kv) > self.cap:  
 self.remove(self.tail.prev.key)  
 self.add(key, value, 0)  
  
  
 def remove(self, key):  
 node, cnt = self.kv[key]  
 if self.cnt[cnt] != node:  
 node.prev.connect(node.next)  
 elif self.kv[node.next.key][1] == cnt:  
 node.prev.connect(node.next)  
 self.cnt[cnt] = self.cnt[cnt].next  
 else:  
 node.prev.connect(node.next)  
 del self.cnt[cnt]  
 del self.kv[key]  
  
 def add(self, key, value, cnt):  
 if cnt in self.cnt:  
 loc = self.cnt[cnt]  
 else:  
 loc = self.cnt[cnt - 1]  
 node = ListNode(key, value)  
 loc.prev.connect(node)  
 node.connect(loc)  
 self.cnt[cnt] = node  
 self.kv[key] = [node, cnt]  
   
  
  
# Your LFUCache object will be instantiated and called as such:  
# obj = LFUCache(capacity)  
# param\_1 = obj.get(key)  
# obj.set(key,value)  
  
**Best\_solution:**Java O(1) Accept Solution Using HashMap, DoubleLinkedList and LinkedHashSet   
public class LFUCache {  
 private Node head = null;  
 private int cap = 0;  
 private HashMap<Integer, Integer> valueHash = null;  
 private HashMap<Integer, Node> nodeHash = null;  
   
 public LFUCache(int capacity) {  
 this.cap = capacity;  
 valueHash = new HashMap<Integer, Integer>();  
 nodeHash = new HashMap<Integer, Node>();  
 }  
   
 public int get(int key) {  
 if (valueHash.containsKey(key)) {  
 increaseCount(key);  
 return valueHash.get(key);  
 }  
 return -1;  
 }  
   
 public void set(int key, int value) {  
 if ( cap == 0 ) return;  
 if (valueHash.containsKey(key)) {  
 valueHash.put(key, value);  
 } else {  
 if (valueHash.size() < cap) {  
 valueHash.put(key, value);  
 } else {  
 removeOld();  
 valueHash.put(key, value);  
 }  
 addToHead(key);  
 }  
 increaseCount(key);  
 }  
   
 private void addToHead(int key) {  
 if (head == null) {  
 head = new Node(0);  
 head.keys.add(key);  
 } else if (head.count > 0) {  
 Node node = new Node(0);  
 node.keys.add(key);  
 node.next = head;  
 head.prev = node;  
 head = node;  
 } else {  
 head.keys.add(key);  
 }  
 nodeHash.put(key, head);   
 }  
   
 private void increaseCount(int key) {  
 Node node = nodeHash.get(key);  
 node.keys.remove(key);  
   
 if (node.next == null) {  
 node.next = new Node(node.count+1);  
 node.next.prev = node;  
 node.next.keys.add(key);  
 } else if (node.next.count == node.count+1) {  
 node.next.keys.add(key);  
 } else {  
 Node tmp = new Node(node.count+1);  
 tmp.keys.add(key);  
 tmp.prev = node;  
 tmp.next = node.next;  
 node.next.prev = tmp;  
 node.next = tmp;  
 }  
  
 nodeHash.put(key, node.next);  
 if (node.keys.size() == 0) remove(node);  
 }  
   
 private void removeOld() {  
 if (head == null) return;  
 int old = 0;  
 for (int n: head.keys) {  
 old = n;  
 break;  
 }  
 head.keys.remove(old);  
 if (head.keys.size() == 0) remove(head);  
 nodeHash.remove(old);  
 valueHash.remove(old);  
 }  
   
 private void remove(Node node) {  
 if (node.prev == null) {  
 head = node.next;  
 } else {  
 node.prev.next = node.next;  
 }   
 if (node.next != null) {  
 node.next.prev = node.prev;  
 }  
 }  
   
 class Node {  
 public int count = 0;  
 public LinkedHashSet<Integer> keys = null;  
 public Node prev = null, next = null;  
   
 public Node(int count) {  
 this.count = count;  
 keys = new LinkedHashSet<Integer>();  
 prev = next = null;  
 }  
 }  
}  
  
*454,Hamming Distance:***Python\_solution:**Python 1 line 49ms   
class Solution(object):  
 def hammingDistance(self, x, y):  
 z = x ^ y  
 count = 0  
 while z > 0:  
 count += z & 1  
 z >>= 1  
   
 return count  
  
**Best\_solution:**Java 1 Line Solution :D   
"corresponding bits are different"  
  
*455,Minimum Moves to Equal Array Elements II:***Python\_solution:**2 lines Python, 2 ways   
def minMoves2(self, nums):  
 median = sorted(nums)[len(nums) / 2]  
 return sum(abs(num - median) for num in nums)  
  
def minMoves2(self, nums):  
 nums.sort()  
 return sum(nums[~i] - nums[i] for i in range(len(nums) / 2))  
**Best\_solution:**Java(just like meeting point problem)   
public class Solution {  
 public int minMoves2(int[] nums) {  
 Arrays.sort(nums);  
 int i = 0, j = nums.length-1;  
 int count = 0;  
 while(i < j){  
 count += nums[j]-nums[i];  
 i++;  
 j--;  
 }  
 return count;  
 }  
}  
  
*456,Island Perimeter:***Python\_solution:**Short Python   
def islandPerimeter(self, grid):  
 return sum(sum(map(operator.ne, [0] + row, row + [0]))  
 for row in grid + map(list, zip(\*grid)))  
**Best\_solution:**clear and easy java solution   
public class Solution {  
 public int islandPerimeter(int[][] grid) {  
 int islands = 0, neighbours = 0;  
  
 for (int i = 0; i < grid.length; i++) {  
 for (int j = 0; j < grid[i].length; j++) {  
 if (grid[i][j] == 1) {  
 islands++; // count islands  
 if (i < grid.length - 1 && grid[i + 1][j] == 1) neighbours++; // count down neighbours  
 if (j < grid[i].length - 1 && grid[i][j + 1] == 1) neighbours++; // count right neighbours  
 }  
 }  
 }  
  
 return islands \* 4 - neighbours \* 2;  
 }  
}  
  
  
*457,Can I Win:***Python\_solution:**Python solution, easy to understand   
 def canIWin(self, maxChoosableInteger, desiredTotal):  
 """  
 :type maxChoosableInteger: int  
 :type desiredTotal: int  
 :rtype: bool  
 """  
 if (1 + maxChoosableInteger) \* maxChoosableInteger/2 < desiredTotal:  
 return False  
 self.memo = {}  
 return self.helper(range(1, maxChoosableInteger + 1), desiredTotal)  
  
   
 def helper(self, nums, desiredTotal):  
   
 hash = str(nums)  
 if hash in self.memo:  
 return self.memo[hash]  
   
 if nums[-1] >= desiredTotal:  
 return True  
   
 for i in range(len(nums)):  
 if not self.helper(nums[:i] + nums[i+1:], desiredTotal - nums[i]):  
 self.memo[hash]= True  
 return True  
 self.memo[hash] = False  
 return False  
**Best\_solution:**Java solution using HashMap with detailed explanation   
O(2^n)  
  
*459,Count The Repetitions:***Best\_solution:**Ugly Java brute force solution, but accepted. 1088ms.   
public class Solution {  
 public int getMaxRepetitions(String s1, int n1, String s2, int n2) {  
 char[] array1 = s1.toCharArray(), array2 = s2.toCharArray();  
 int count1 = 0, count2 = 0, i = 0, j = 0;  
   
 while (count1 < n1) {  
 if (array1[i] == array2[j]) {  
 j++;  
 if (j == array2.length) {  
 j = 0;  
 count2++;  
 }  
 }  
 i++;  
 if (i == array1.length) {  
 i = 0;  
 count1++;  
 }  
 }  
   
 return count2 / n2;  
 }  
}  
  
  
*460,Unique Substrings in Wraparound String:***Python\_solution:**Concise O(n) 6-liner in Python   
'abcdefghijklmnopqrstuvwxyza’  
**Best\_solution:**Concise Java solution using DP   
p  
  
*461,Validate IP Address:***Python\_solution:**Python Solution   
class Solution(object):  
 def validIPAddress(self, IP):  
 def is\_hex(s):  
 hex\_digits = set("0123456789abcdefABCDEF")  
 for char in s:  
 if not (char in hex\_digits):  
 return False  
 return True  
 ary = IP.split('.')  
 if len(ary) == 4:  
 for i in xrange(len(ary)):  
 if not ary[i].isdigit() or not 0 <= int(ary[i]) < 256 or (ary[i][0] == '0' and len(ary[i]) > 1):  
 return "Neither"  
 return "IPv4"  
 ary = IP.split(':')  
 if len(ary) == 8:  
 for i in xrange(len(ary)):  
 tmp = ary[i]  
 if len(tmp) == 0 or not len(tmp) <= 4 or not is\_hex(tmp):   
 return "Neither"  
 return "IPv6"  
 return "Neither"  
  
**Best\_solution:**Java Simple Solution   
public String validIPAddress(String IP) {  
 if(isValidIPv4(IP)) return "IPv4";  
 else if(isValidIPv6(IP)) return "IPv6";  
 else return "Neither";  
}  
  
public boolean isValidIPv4(String ip) {  
 if(ip.length()<7) return false;  
 if(ip.charAt(0)=='.') return false;  
 if(ip.charAt(ip.length()-1)=='.') return false;  
 String[] tokens = ip.split("\\.");  
 if(tokens.length!=4) return false;  
 for(String token:tokens) {  
 if(!isValidIPv4Token(token)) return false;  
 }  
 return true;  
}  
public boolean isValidIPv4Token(String token) {  
 if(token.startsWith("0") && token.length()>1) return false;  
 try {  
 int parsedInt = Integer.parseInt(token);  
 if(parsedInt<0 || parsedInt>255) return false;  
 if(parsedInt==0 && token.charAt(0)!='0') return false;  
 } catch(NumberFormatException nfe) {  
 return false;  
 }  
 return true;  
}  
   
public boolean isValidIPv6(String ip) {  
 if(ip.length()<15) return false;  
 if(ip.charAt(0)==':') return false;  
 if(ip.charAt(ip.length()-1)==':') return false;  
 String[] tokens = ip.split(":");  
 if(tokens.length!=8) return false;  
 for(String token: tokens) {  
 if(!isValidIPv6Token(token)) return false;  
 }  
 return true;  
}  
public boolean isValidIPv6Token(String token) {  
 if(token.length()>4 || token.length()==0) return false;  
 char[] chars = token.toCharArray();  
 for(char c:chars) {  
 boolean isDigit = c>=48 && c<=57;  
 boolean isUppercaseAF = c>=65 && c<=70;  
 boolean isLowerCaseAF = c>=97 && c<=102;  
 if(!(isDigit || isUppercaseAF || isLowerCaseAF))   
 return false;  
 }  
 return true;  
}  
  
  
*464,Concatenated Words:***Python\_solution:**Python Explanation   
S = set(A)  
ans = []  
for word in A:  
 if not word: continue  
 stack = [0]  
 seen = {0}  
 M = len(word)  
 while stack:  
 node = stack.pop()  
 if node == M:  
 ans.append(word)  
 break  
 for j in xrange(M - node + 1):  
 if (word[node:node+j] in S and   
 node + j not in seen and  
 (node > 0 or node + j != M)):  
 stack.append(node + j)  
 seen.add(node + j)  
  
return ans  
  
**Best\_solution:**Java DP Solution   
DP  
  
*465,Matchsticks to Square:***Python\_solution:**Python Explanation   
if len(A) < 4 or sum(A) % 4 or max(A) > sum(A) / 4:  
 return False  
  
T = sum(A) / 4  
N = len(A)  
A.sort()  
  
memo = {}  
def dp(mask, cur = T):  
 if (mask, cur) in memo: return memo[mask, cur]  
 if mask == 0: return cur == 0  
 if cur == 0: return dp(mask, T)  
  
 ans = False  
 for bit in xrange(N):  
 if mask & (1 << bit):  
 if A[bit] > cur:  
 break  
 if dp(mask ^ (1 << bit), cur - A[bit]):  
 ans = True  
 break  
 memo[mask, cur] = ans  
 return ans  
  
return dp(2\*\*N - 1)  
  
**Best\_solution:**Java DFS Solution with Explanation   
S  
  
*466,Ones and Zeroes:***Python\_solution:**0-1 knapsack in python   
dp(k, x, y) = max(dp(k-1, x-z, y-o) + 1, dp(k-1, x, y)) (z is zeroes in strs[k], o is ones in strs[k])  
  
**Best\_solution:**c++ DP solution with comments   
int findMaxForm(vector<string>& strs, int m, int n) {  
 vector<vector<int>> memo(m+1, vector<int>(n+1, 0));  
 int numZeroes, numOnes;  
  
 for (auto &s : strs) {  
 numZeroes = numOnes = 0;  
 // count number of zeroes and ones in current string  
 for (auto c : s) {  
 if (c == '0')  
 numZeroes++;  
 else if (c == '1')  
 numOnes++;  
 }  
  
 // memo[i][j] = the max number of strings that can be formed with i 0's and j 1's  
 // from the first few strings up to the current string s  
 // Catch: have to go from bottom right to top left  
 // Why? If a cell in the memo is updated(because s is selected),  
 // we should be adding 1 to memo[i][j] from the previous iteration (when we were not considering s)  
 // If we go from top left to bottom right, we would be using results from this iteration => overcounting  
 for (int i = m; i >= numZeroes; i--) {  
 for (int j = n; j >= numOnes; j--) {  
 memo[i][j] = max(memo[i][j], memo[i - numZeroes][j - numOnes] + 1);  
 }  
 }  
 }  
 return memo[m][n];  
}  
  
  
*467,Heaters:***Python\_solution:**Short Python   
i  
**Best\_solution:**Short and Clean Java Binary Search Solution   
Arrays.binarySearch()  
  
*468,Number Complement:***Python\_solution:**Simple Python   
class Solution(object):  
 def findComplement(self, num):  
 i = 1  
 while i <= num:  
 i = i << 1  
 return (i - 1) ^ num  
  
**Best\_solution:**3 line C++   
class Solution {  
public:  
 int findComplement(int num) {  
 unsigned mask = ~0;  
 while (num & mask) mask <<= 1;  
 return ~mask & ~num;  
 }  
};  
  
  
*469,Total Hamming Distance:***Python\_solution:**Python via Strings   
def totalHammingDistance(self, nums):  
 return sum(b.count('0') \* b.count('1') for b in zip(\*map('{:032b}'.format, nums)))  
**Best\_solution:**Java O(n) time O(1) Space   
public int totalHammingDistance(int[] nums) {  
 int total = 0, n = nums.length;  
 for (int j=0;j<32;j++) {  
 int bitCount = 0;  
 for (int i=0;i<n;i++)   
 bitCount += (nums[i] >> j) & 1;  
 total += bitCount\*(n - bitCount);  
 }  
 return total;  
}  
  
  
*470,Largest Palindrome Product:***Python\_solution:**Time limit exceeded in Python   
class Solution(object):  
 def largestPalindrome(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 if n==1: return 9  
   
 upper = 10\*\*n-1 # Largest n-digit number  
 firstHalf = int(upper\*upper/10\*\*n) # First n digits of largest palindrome  
   
 # Loop over palindromes  
 found = False  
 while not found:  
 secondHalf = int(str(firstHalf)[::-1])  
 tryThis = firstHalf\*10\*\*n + secondHalf  
   
 # Loop over a. If a is an integer multiple of tryThis, you win.  
 for a in xrange(upper, 0, -1):  
   
 # Test for b more than n digits or a^2 greater than the palindrome.  
 # (The second check is valid because we are searching in decreasing order of a).  
 if tryThis/a > upper or a\*a < tryThis:   
 break  
 if tryThis % a == 0:   
 found = True  
 break  
 firstHalf -= 1  
 return tryThis % 1337  
  
**Best\_solution:**Java Solution using assumed max palindrom   
 public int largestPalindrome(int n) {  
 // if input is 1 then max is 9   
 if(n == 1){  
 return 9;  
 }  
   
 // if n = 3 then upperBound = 999 and lowerBound = 99  
 int upperBound = (int) Math.pow(10, n) - 1, lowerBound = upperBound / 10;  
 long maxNumber = (long) upperBound \* (long) upperBound;  
   
 // represents the first half of the maximum assumed palindrom.  
 // e.g. if n = 3 then maxNumber = 999 x 999 = 998001 so firstHalf = 998  
 int firstHalf = (int)(maxNumber / (long) Math.pow(10, n));  
   
 boolean palindromFound = false;  
 long palindrom = 0;  
   
 while (!palindromFound) {  
 // creates maximum assumed palindrom  
 // e.g. if n = 3 first time the maximum assumed palindrom will be 998 899  
 palindrom = createPalindrom(firstHalf);  
   
 // here i and palindrom/i forms the two factor of assumed palindrom  
 for (long i = upperBound; upperBound > lowerBound; i--) {  
 // if n= 3 none of the factor of palindrom can be more than 999 or less than square root of assumed palindrom   
 if (palindrom / i > maxNumber || i \* i < palindrom) {  
 break;  
 }  
   
 // if two factors found, where both of them are n-digits,  
 if (palindrom % i == 0) {  
 palindromFound = true;  
 break;  
 }  
 }  
  
 firstHalf--;  
 }  
  
 return (int) (palindrom % 1337);  
 }  
  
 private long createPalindrom(long num) {  
 String str = num + new StringBuilder().append(num).reverse().toString();  
 return Long.parseLong(str);  
 }  
  
  
*471,Sliding Window Median:***Python\_solution:**Easy Python O(nk)   
def medianSlidingWindow(self, nums, k):  
 window = sorted(nums[:k])  
 medians = []  
 for a, b in zip(nums, nums[k:] + [0]):  
 medians.append((window[k/2] + window[~(k/2)]) / 2.)  
 window.remove(a)  
 bisect.insort(window, b)  
 return medians  
**Best\_solution:**O(n log k) C++ using multiset and updating middle-iterator   
vector<double> medianSlidingWindow(vector<int>& nums, int k) {  
 multiset<int> window(nums.begin(), nums.begin() + k);  
 auto mid = next(window.begin(), k / 2);  
 vector<double> medians;  
 for (int i=k; ; i++) {  
  
 // Push the current median.  
 medians.push\_back((double(\*mid) + \*prev(mid, 1 - k%2)) / 2);  
  
 // If all done, return.  
 if (i == nums.size())  
 return medians;  
   
 // Insert nums[i].  
 window.insert(nums[i]);  
 if (nums[i] < \*mid)  
 mid--;  
  
 // Erase nums[i-k].  
 if (nums[i-k] <= \*mid)  
 mid++;  
 window.erase(window.lower\_bound(nums[i-k]));  
 }  
}  
  
*472,Magical String:***Python\_solution:**Short Python using queue   
class Solution(object):  
 def magicalString(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 S = [1,2,2]  
 idx = 2  
 while len(S) < n:  
 S += S[idx] \* [(3 - S[-1])]  
 idx += 1  
 return S[:n].count(1)  
  
**Best\_solution:**Simple Java solution using one array and two pointers   
int  
  
*473,License Key Formatting:***Python\_solution:**Python solution   
class Solution(object):  
 def licenseKeyFormatting(self, S, K):  
 """  
 :type S: str  
 :type K: int  
 :rtype: str  
 """  
 S = S.upper().replace('-','')  
 size = len(S)  
 s1 = K if size%K==0 else size%K  
 res = S[:s1]  
 while s1<size:  
 res += '-'+S[s1:s1+K]  
 s1 += K  
 return res  
  
**Best\_solution:**Java 5 lines clean solution   
 public String licenseKeyFormatting(String s, int k) {  
 StringBuilder sb = new StringBuilder();  
 for (int i = s.length() - 1; i >= 0; i--)  
 if (s.charAt(i) != '-')  
 sb.append(sb.length() % (k + 1) == k ? '-' : "").append(s.charAt(i));  
 return sb.reverse().toString().toUpperCase();  
 }   
  
  
*474,Smallest Good Base:***Python\_solution:**Python solution with detailed mathematical explanation and derivation   
import math  
class Solution(object):  
 def smallestGoodBase(self, n):  
 """  
 :type n: str  
 :rtype: str  
 """  
 n = int(n)  
 max\_m = int(math.log(n,2)) # Refer [7]  
 for m in range(max\_m,1,-1):  
 k = int(n\*\*m\*\*-1) # Refer [6]  
 if (k\*\*(m+1)-1)//(k-1) == n:  
 # Refer [3]  
 return str(k)  
   
 return str(n-1)   
  
**Best\_solution:**3ms, AC, C++, long long int + binary search   
class Solution {  
public:  
 string smallestGoodBase(string n) {  
 unsigned long long tn=(unsigned long long)stoll(n);  
 unsigned long long x=1;  
 for (int i=62;i>=1;i--) {  
 if ((x<<i)<tn) {  
 unsigned long long cur=mysolve(tn,i);  
 if (cur!=0) return to\_string(cur);  
 }  
 }  
 return to\_string(tn-1);  
 }  
   
unsigned long long mysolve(unsigned long long n,int d) {  
 double tn=(double) n;  
 unsigned long long right=(unsigned long long)(pow(tn,1.0/d)+1);  
 unsigned long long left=1;  
 while (left<=right){  
 unsigned long long mid=left+(right-left)/2;  
 unsigned long long sum=1,cur=1;  
 for (int i=1;i<=d;i++) {  
 cur\*=mid;  
 sum+=cur;  
 }  
 if (sum==n) return mid;  
 if (sum>n) right=mid-1;  
 else left=mid+1;  
 }  
 return 0;  
}  
  
};  
  
  
*476,Max Consecutive Ones:***Python\_solution:**Simple Python   
class Solution(object):  
 def findMaxConsecutiveOnes(self, nums):  
 cnt = 0  
 ans = 0  
 for num in nums:  
 if num == 1:  
 cnt += 1  
 ans = max(ans, cnt)  
 else:  
 cnt = 0  
 return ans  
  
**Best\_solution:**Java 4 lines concise solution with explanation   
 public int findMaxConsecutiveOnes(int[] nums) {  
 int maxHere = 0, max = 0;  
 for (int n : nums)  
 max = Math.max(max, maxHere = n == 0 ? 0 : maxHere + 1);  
 return max;   
 }   
  
  
*477,Predict the Winner:***Python\_solution:**Python with memorization [48 ms]   
class Solution(object):  
 def PredictTheWinner(self, nums):  
 def check(left, right, memo):  
 if left > right:  
 return 0  
 if left == right:  
 return nums[left]  
 if not (left, right) in memo:  
 ss = sum(nums[left: right + 1])  
 l, r = ss - check(left + 1, right, memo) + nums[left], ss - check(left, right - 1, memo) + nums[right]  
 memo[(left, right)] = max(l, r)  
 return memo[(left, right)]  
  
 s = sum(nums)  
 c1 = check(0, len(nums) - 1, {})  
 return c1 >= s - c1  
  
**Best\_solution:**Java 1 Line Recursion Solution   
public class Solution {  
 public boolean PredictTheWinner(int[] nums) {  
 return helper(nums, 0, nums.length-1)>=0;  
 }  
 private int helper(int[] nums, int s, int e){   
 return s==e ? nums[e] : Math.max(nums[e] - helper(nums, s, e-1), nums[s] - helper(nums, s+1, e));  
 }  
}  
  
  
*479,Zuma Game:***Python\_solution:**DP O(N^3) Solution in Python with explanation   
getBalls()  
**Best\_solution:**Standard test program is wrong?   
"RRWWRRBBRR", "WB"  
  
*481,Increasing Subsequences:***Python\_solution:**Simple Python   
def findSubsequences(self, nums):  
 subs = {()}  
 for num in nums:  
 subs |= {sub + (num,)  
 for sub in subs  
 if not sub or sub[-1] <= num}  
 return [sub for sub in subs if len(sub) >= 2]  
**Best\_solution:**Java 20 lines backtracking solution using set, beats 100%.   
public class Solution {  
  
 public List<List<Integer>> findSubsequences(int[] nums) {  
 Set<List<Integer>> res= new HashSet<List<Integer>>();  
 List<Integer> holder = new ArrayList<Integer>();  
 findSequence(res, holder, 0, nums);  
 List result = new ArrayList(res);  
 return result;  
 }  
  
 public void findSequence(Set<List<Integer>> res, List<Integer> holder, int index, int[] nums) {  
 if (holder.size() >= 2) {  
 res.add(new ArrayList(holder));  
 }  
 for (int i = index; i < nums.length; i++) {  
 if(holder.size() == 0 || holder.get(holder.size() - 1) <= nums[i]) {  
 holder.add(nums[i]);  
 findSequence(res, holder, i + 1, nums);  
 holder.remove(holder.size() - 1);  
 }  
 }  
 }  
}  
  
*482,Construct the Rectangle:***Python\_solution:**Simple Python   
class Solution(object):  
 def constructRectangle(self, area):  
 mid = int(math.sqrt(area))  
 while mid > 0:  
 if area % mid == 0:  
 return [int(area / mid), int(mid)]  
 mid -= 1  
  
**Best\_solution:**3 line Clean and easy understand solution   
public int[] constructRectangle(int area) {  
 int w = (int)Math.sqrt(area);  
 while (area%w!=0) w--;  
 return new int[]{area/w, w};  
}  
  
  
*483,Reverse Pairs:***Python\_solution:**Python divide & conquer and DP   
class Solution(object):  
 def reversePairs(self, nums):  
 return self.helper(nums, 0, len(nums))  
   
 def helper(self, nums, l, r):  
 mid = l + r >> 1  
 if mid == l: return 0  
 total = self.helper(nums, l, mid) + self.helper(nums, mid, r)  
 prev\_total = 0  
 for i in range(l, mid):  
 target = nums[i] - 1 >> 1  
 idx = bisect.bisect\_right(nums, target, mid, r)  
 prev\_total += idx - mid  
 mid = idx  
 total += prev\_total  
 nums[l: r] = sorted(nums[l: r])  
 return total  
  
**Best\_solution:**General principles behind problems similar to "Reverse Pairs"   
BST  
  
*484,Target Sum:***Python\_solution:**Python DP   
class Solution(object):  
 def findTargetSumWays(self, nums, S):  
 if not nums:  
 return 0  
 dic = {nums[0]: 1, -nums[0]: 1} if nums[0] != 0 else {0: 2}  
 for i in range(1, len(nums)):  
 tdic = {}  
 for d in dic:  
 tdic[d + nums[i]] = tdic.get(d + nums[i], 0) + dic.get(d, 0)  
 tdic[d - nums[i]] = tdic.get(d - nums[i], 0) + dic.get(d, 0)  
 dic = tdic  
 return dic.get(S, 0)  
  
**Best\_solution:**Java (15 ms) C++ (3 ms) O(ns) iterative DP solution using subset sum with explanation   
nums  
  
*485,Teemo Attacking:***Python\_solution:**Python Solution for Teemo   
class Solution(object):  
 def findPoisonedDuration(self, timeSeries, duration):  
 ans = duration \* len(timeSeries)  
 for i in range(1,len(timeSeries)):  
 ans -= max(0, duration - (timeSeries[i] - timeSeries[i-1]))  
 return ans  
  
**Best\_solution:**Python Solution for Teemo   
class Solution(object):  
 def findPoisonedDuration(self, timeSeries, duration):  
 ans = duration \* len(timeSeries)  
 for i in range(1,len(timeSeries)):  
 ans -= max(0, duration - (timeSeries[i] - timeSeries[i-1]))  
 return ans  
  
  
*486,Next Greater Element I:***Python\_solution:**Python Solution with O(n)   
 d = {}  
 st = []  
 ans = []  
   
 for x in nums:  
 while len(st) and st[-1] < x:  
 d[st.pop()] = x  
 st.append(x)  
  
 for x in findNums:  
 ans.append(d.get(x, -1))  
   
 return ans  
  
**Best\_solution:**Java 10 lines linear time complexity O(n) with explanation   
[5, 4, 3, 2, 1, 6]  
  
*487,Diagonal Traverse:***Python\_solution:**sorting and normal Python   
def findDiagonalOrder(self, matrix):  
 entries = [(i+j, (j, i)[(i^j)&1], val)  
 for i, row in enumerate(matrix)  
 for j, val in enumerate(row)]  
 return [e[2] for e in sorted(entries)]  
  
**Best\_solution:**Concise Java Solution   
bottom border  
  
*489,Keyboard Row:***Python\_solution:**one-liner Ruby + Python   
def find\_words(words)  
 words.select { |w| w =~ /^([qwertyuiop]\*|[asdfghjkl]\*|[zxcvbnm]\*)$/i }  
end  
  
**Best\_solution:**Java 1-Line Solution via Regex and Stream   
public String[] findWords(String[] words) {  
 return Stream.of(words).filter(s -> s.toLowerCase().matches("[qwertyuiop]\*|[asdfghjkl]\*|[zxcvbnm]\*")).toArray(String[]::new);  
}  
  
  
*490,Find Mode in Binary Search Tree:***Python\_solution:**Simple Python Explanation   
count = collections.Counter()  
  
def dfs(node):  
 if node:  
 count[node.val] += 1  
 dfs(node.left)  
 dfs(node.right)  
   
dfs(root)  
max\_ct = max(count.itervalues())  
return [k for k, v in count.iteritems() if v == max\_ct]  
  
**Best\_solution:**Proper O(1) space   
public class Solution {  
   
 public int[] findMode(TreeNode root) {  
 inorder(root);  
 modes = new int[modeCount];  
 modeCount = 0;  
 currCount = 0;  
 inorder(root);  
 return modes;  
 }  
  
 private int currVal;  
 private int currCount = 0;  
 private int maxCount = 0;  
 private int modeCount = 0;  
   
 private int[] modes;  
  
 private void handleValue(int val) {  
 if (val != currVal) {  
 currVal = val;  
 currCount = 0;  
 }  
 currCount++;  
 if (currCount > maxCount) {  
 maxCount = currCount;  
 modeCount = 1;  
 } else if (currCount == maxCount) {  
 if (modes != null)  
 modes[modeCount] = currVal;  
 modeCount++;  
 }  
 }  
   
 private void inorder(TreeNode root) {  
 if (root == null) return;  
 inorder(root.left);  
 handleValue(root.val);  
 inorder(root.right);  
 }  
}  
  
  
*491,IPO:***Python\_solution:**Python solution   
def findMaximizedCapital(self, k, W, Profits, Capital):  
 current = []  
 future = sorted(zip(Capital, Profits))[::-1]  
 for \_ in range(k):  
 while future and future[-1][0] <= W:  
 heapq.heappush(current, -future.pop()[1])  
 if current:  
 W -= heapq.heappop(current)  
 return W  
**Best\_solution:**Very Simple (Greedy) Java Solution using two PriorityQueues   
max  
  
*492,Next Greater Element II:***Python\_solution:**Python 6 lines solution using stack   
def nextGreaterElements(self, nums):  
 stack, res = [], [-1] \* len(nums)  
 for i in range(len(nums)) \* 2:  
 while stack and (nums[stack[-1]] < nums[i]):  
 res[stack.pop()] = nums[i]  
 stack.append(i)  
 return res  
**Best\_solution:**Java 10 lines and C++ 12 lines linear time complexity O(n) with explanation   
stack  
  
*493,Base 7:***Python\_solution:**Python easy understand solution   
def convertTo7(self, num):  
 if num < 0: return '-' + self.convertTo7(-num)  
 if num < 7: return str(num)  
 return self.convertTo7(num // 7) + str(num % 7)  
  
  
**Best\_solution:**Simple Java, oneliner Ruby   
public String convertTo7(int num) {  
 if (num < 0)  
 return '-' + convertTo7(-num);  
 if (num < 7)  
 return num + "";  
 return convertTo7(num / 7) + num % 7;  
}  
  
  
*495,Relative Ranks:***Python\_solution:**Python solution   
def findRelativeRanks(self, nums):  
 sort = sorted(nums)[::-1]  
 rank = ["Gold Medal", "Silver Medal", "Bronze Medal"] + map(str, range(4, len(nums) + 1))  
 return map(dict(zip(sort, rank)).get, nums)  
**Best\_solution:**Easy Java Solution, Sorting.   
score  
  
*496,Perfect Number:***Python\_solution:**Python, Straightforward with Explanation   
def prime\_factorization(N):  
 d = 2  
 while d \* d <= n:  
 expo = 0  
 while N % d == 0:  
 expo += 1  
 N /= d  
 if expo:  
 yield (d, expo)  
 d += 1  
 if N > 1:  
 yield (N, 1)  
  
ans = 1  
for prime, expo in prime\_factorization(abs(N)):  
 ans \*= sum(prime \*\* k for k in xrange(expo + 1))  
return ans == 2\*N  
  
**Best\_solution:**Simple Java Solution   
public class Solution {  
 public boolean checkPerfectNumber(int num) {  
 if (num == 1) return false;  
   
 int sum = 0;  
 for (int i = 2; i <= Math.sqrt(num); i++) {  
 if (num % i == 0) {  
 sum += i;  
 if (i != num / i) sum += num / i;  
 }  
 }  
 sum++;  
   
 return sum == num;  
 }  
}  
  
  
*497,Most Frequent Subtree Sum:***Python\_solution:**Python easy understand solution   
ctr  
**Best\_solution:**Verbose Java solution, postOrder traverse, HashMap (18ms)   
post-order  
  
*498,Find Bottom Left Tree Value:***Python\_solution:**Right-to-Left BFS (Python + Java)   
def findLeftMostNode(self, root):  
 queue = [root]  
 for node in queue:  
 queue += filter(None, (node.right, node.left))  
 return node.val  
  
**Best\_solution:**Right-to-Left BFS (Python + Java)   
def findLeftMostNode(self, root):  
 queue = [root]  
 for node in queue:  
 queue += filter(None, (node.right, node.left))  
 return node.val  
  
  
*499,Freedom Trail:***Python\_solution:**Python Solution (222 ms)   
dist(i, j) = min(|i - j|, n - |i - j|)  
  
**Best\_solution:**Concise Java DP Solution   
public class Solution {  
 public int findRotateSteps(String ring, String key) {  
 int n = ring.length();  
 int m = key.length();  
 int[][] dp = new int[m + 1][n];  
   
 for (int i = m - 1; i >= 0; i--) {  
 for (int j = 0; j < n; j++) {  
 dp[i][j] = Integer.MAX\_VALUE;  
 for (int k = 0; k < n; k++) {  
 if (ring.charAt(k) == key.charAt(i)) {  
 int diff = Math.abs(j - k);  
 int step = Math.min(diff, n - diff);  
 dp[i][j] = Math.min(dp[i][j], step + dp[i + 1][k]);  
 }  
 }  
 }  
 }  
   
 return dp[0][0] + m;  
 }  
}  
  
  
*500,Find Largest Value in Each Tree Row:***Python\_solution:**Python BFS   
def findValueMostElement(self, root):  
 maxes = []  
 row = [root]  
 while any(row):  
 maxes.append(max(node.val for node in row))  
 row = [kid for node in row for kid in (node.left, node.right) if kid]  
 return maxes  
**Best\_solution:**9ms JAVA DFS solution   
public class Solution {  
 public List<Integer> largestValues(TreeNode root) {  
 List<Integer> res = new ArrayList<Integer>();  
 helper(root, res, 0);  
 return res;  
 }  
 private void helper(TreeNode root, List<Integer> res, int d){  
 if(root == null){  
 return;  
 }  
 //expand list size  
 if(d == res.size()){  
 res.add(root.val);  
 }  
 else{  
 //or set value  
 res.set(d, Math.max(res.get(d), root.val));  
 }  
 helper(root.left, res, d+1);  
 helper(root.right, res, d+1);  
 }  
}  
  
  
*501,Longest Palindromic Subsequence:***Python\_solution:**Python DP O(n) space O(n^2) time   
class Solution(object):  
 def longestPalindromeSubseq(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 n = len(s)  
 dp = [[1] \* 2 for \_ in range(n)]  
 for j in xrange(1, len(s)):  
 for i in reversed(xrange(0, j)):  
 if s[i] == s[j]:  
 dp[i][j%2] = 2 + dp[i + 1][(j - 1)%2] if i + 1 <= j - 1 else 2  
 else:  
 dp[i][j%2] = max(dp[i + 1][j%2], dp[i][(j - 1)%2])  
 return dp[0][(n-1)%2]  
  
**Best\_solution:**Straight forward Java DP solution   
dp[i][j]  
  
*502,Super Washing Machines:***Python\_solution:**Python solution   
class Solution(object):  
 def findMinMoves(self, machines):  
 """  
 :type machines: List[int]  
 :rtype: int  
 """  
 if not machines:  
 return 0  
 n = len(machines)  
 s = sum(machines)  
 if s % n:  
 return -1  
 avg = s / n  
 ans = 0  
 left\_sum = 0  
 for x in machines:  
 delta = x - avg  
 ans = max(ans, -left\_sum, delta + max(0, left\_sum))  
 left\_sum += delta  
 return ans  
  
**Best\_solution:**Super Short & Easy Java O(n) Solution   
public class Solution {  
 public int findMinMoves(int[] machines) {  
 int total = 0;   
 for(int i: machines) total+=i;  
 if(total%machines.length!=0) return -1;  
 int avg = total/machines.length, cnt = 0, max = 0;  
 for(int load: machines){  
 cnt += load-avg; //load-avg is "gain/lose"  
 max = Math.max(Math.max(max, Math.abs(cnt)), load-avg);  
 }  
 return max;  
 }  
}  
  
  
*503,Coin Change 2:***Python\_solution:**python O(n) space dp solution   
 def change(self, amount, coins):  
 """  
 :type amount: int  
 :type coins: List[int]  
 :rtype: int  
 """  
 dp = [0] \* (amount + 1)  
 dp[0] = 1  
 for i in coins:  
 for j in range(1, amount + 1):  
 if j >= i:  
 dp[j] += dp[j - i]  
 return dp[amount]  
  
**Best\_solution:**Knapsack problem - Java solution with thinking process O(nm) Time and O(m) Space   
dp[i][j]  
  
*504,Detect Capital:***Python\_solution:**In Python, these are called...   
def detectCapitalUse(self, word):  
 return word.isupper() or word.islower() or word.istitle()  
**Best\_solution:**3 Lines   
public class Solution {  
 public boolean detectCapitalUse(String word) {  
 int cnt = 0;  
 for(char c: word.toCharArray()) if('Z' - c >= 0) cnt++;  
 return ((cnt==0 || cnt==word.length()) || (cnt==1 && 'Z' - word.charAt(0)>=0));  
 }  
}  
  
  
*505,Longest Uncommon Subsequence I:***Python\_solution:**Python, Simple Explanation   
def findLUSlength(self, A, B):  
 if A == B:  
 return -1  
 return max(len(A), len(B))  
  
**Best\_solution:**I feel this problem is just perfect for April Fools' day   
Both strings' lengths will not exceed 100.  
  
*506,Longest Uncommon Subsequence II:***Python\_solution:**Python, Simple Explanation   
def subseq(w1, w2):  
 #True iff word1 is a subsequence of word2.  
 i = 0  
 for c in w2:  
 if i < len(w1) and w1[i] == c:  
 i += 1  
 return i == len(w1)  
   
A.sort(key = len, reverse = True)  
for i, word1 in enumerate(A):  
 if all(not subseq(word1, word2)   
 for j, word2 in enumerate(A) if i != j):  
 return len(word1)  
return -1  
  
**Best\_solution:**Python, Simple Explanation   
def subseq(w1, w2):  
 #True iff word1 is a subsequence of word2.  
 i = 0  
 for c in w2:  
 if i < len(w1) and w1[i] == c:  
 i += 1  
 return i == len(w1)  
   
A.sort(key = len, reverse = True)  
for i, word1 in enumerate(A):  
 if all(not subseq(word1, word2)   
 for j, word2 in enumerate(A) if i != j):  
 return len(word1)  
return -1  
  
  
*507,Continuous Subarray Sum:***Python\_solution:**Python with explanation. 62ms Time O(min(n, k)) mostly   
if k == 0  
**Best\_solution:**Java O(n) time O(k) space   
public boolean checkSubarraySum(int[] nums, int k) {  
 Map<Integer, Integer> map = new HashMap<Integer, Integer>(){{put(0,-1);}};;  
 int runningSum = 0;  
 for (int i=0;i<nums.length;i++) {  
 runningSum += nums[i];  
 if (k != 0) runningSum %= k;   
 Integer prev = map.get(runningSum);  
 if (prev != null) {  
 if (i - prev > 1) return true;  
 }  
 else map.put(runningSum, i);  
 }  
 return false;  
}  
  
  
*508,Longest Word in Dictionary through Deleting:***Python\_solution:**Short Python solutions   
def findLongestWord(self, s, d):  
 def isSubsequence(x):  
 it = iter(s)  
 return all(c in it for c in x)  
 return max(sorted(filter(isSubsequence, d)) + [''], key=len)  
  
**Best\_solution:**Short Java Solutions - Sorting Dictionary and Without Sorting   
public String findLongestWord(String s, List<String> d) {  
 Collections.sort(d, (a,b) -> a.length() != b.length() ? -Integer.compare(a.length(), b.length()) : a.compareTo(b));  
 for (String dictWord : d) {  
 int i = 0;  
 for (char c : s.toCharArray())   
 if (i < dictWord.length() && c == dictWord.charAt(i)) i++;  
 if (i == dictWord.length()) return dictWord;  
 }  
 return "";  
}  
  
  
*509,Contiguous Array:***Python\_solution:**Python O(n) Solution with Visual Explanation   
count  
**Best\_solution:**Easy Java O(n) Solution, PreSum + HashMap   
0  
  
*510,Beautiful Arrangement:***Python\_solution:**Python recursion + DP 66ms   
cache = {}  
class Solution(object):  
 def countArrangement(self, N):  
 def helper(i, X):  
 if i == 1:  
 return 1  
 key = (i, X)  
 if key in cache:  
 return cache[key]  
 total = 0  
 for j in xrange(len(X)):  
 if X[j] % i == 0 or i % X[j] == 0:  
 total += helper(i - 1, X[:j] + X[j + 1:])  
 cache[key] = total  
 return total  
 return helper(N, tuple(range(1, N + 1)))  
  
**Best\_solution:**Java Solution, Backtracking   
public class Solution {  
 int count = 0;  
   
 public int countArrangement(int N) {  
 if (N == 0) return 0;  
 helper(N, 1, new int[N + 1]);  
 return count;  
 }  
   
 private void helper(int N, int pos, int[] used) {  
 if (pos > N) {  
 count++;  
 return;  
 }  
   
 for (int i = 1; i <= N; i++) {  
 if (used[i] == 0 && (i % pos == 0 || pos % i == 0)) {  
 used[i] = 1;  
 helper(N, pos + 1, used);  
 used[i] = 0;  
 }  
 }  
 }  
}  
  
  
*512,Minesweeper:***Python\_solution:**Simple Python (DFS)   
def updateBoard(self, A, click):  
 click = tuple(click)  
 R, C = len(A), len(A[0])  
   
 def neighbors(r, c):  
 for dr in xrange(-1, 2):  
 for dc in xrange(-1, 2):  
 if (dr or dc) and 0 <= r + dr < R and 0 <= c + dc < C:  
 yield r + dr, c + dc  
   
 stack = [click]  
 seen = {click}  
 while stack:  
 r, c = stack.pop()  
 if A[r][c] == 'M':  
 A[r][c] = 'X'  
 else:  
 mines\_adj = sum( A[nr][nc] in 'MX' for nr, nc in neighbors(r, c) )  
 if mines\_adj:  
 A[r][c] = str(mines\_adj)  
 else:  
 A[r][c] = 'B'  
 for nei in neighbors(r, c):  
 if A[nei[0]][nei[1]] in 'ME' and nei not in seen:  
 stack.append(nei)  
 seen.add(nei)  
 return A  
  
**Best\_solution:**Java Solution, DFS + BFS   
Search  
  
*513,Minimum Absolute Difference in BST:***Python\_solution:**Python 7 lines AC solution   
def getMinimumDifference(self, root):  
 def dfs(node, l=[]):  
 if node.left: dfs(node.left, l)  
 l.append(node.val)  
 if node.right: dfs(node.right, l)  
 return l  
 l = dfs(root)  
 return min([abs(a-b) for a,b in zip(l, l[1:])])  
  
**Best\_solution:**Two Solutions, in-order traversal and a more general way using TreeSet   
inOrder  
  
*515,K-diff Pairs in an Array:***Python\_solution:**1-liner in Python, O(n) time   
 def findPairs(self, nums, k):  
 return len(set(nums)&{n+k for n in nums}) if k>0 else sum(v>1 for v in collections.Counter(nums).values()) if k==0 else 0  
  
**Best\_solution:**Java O(n) solution - one Hashmap, easy to understand   
public class Solution {  
 public int findPairs(int[] nums, int k) {  
 if (nums == null || nums.length == 0 || k < 0) return 0;  
   
 Map<Integer, Integer> map = new HashMap<>();  
 int count = 0;  
 for (int i : nums) {  
 map.put(i, map.getOrDefault(i, 0) + 1);  
 }  
   
 for (Map.Entry<Integer, Integer> entry : map.entrySet()) {  
 if (k == 0) {  
 //count how many elements in the array that appear more than twice.  
 if (entry.getValue() >= 2) {  
 count++;  
 }   
 } else {  
 if (map.containsKey(entry.getKey() + k)) {  
 count++;  
 }  
 }  
 }  
   
 return count;  
 }  
}  
  
  
*517,Design TinyURL:***Best\_solution:**Suggestion on extra questions   
last\_updated\_timestamp  
  
*518,Encode and Decode TinyURL:***Python\_solution:**Python Solution   
class Codec:  
 def \_\_init\_\_(self):  
 self.d = {}  
 self.r = {}  
  
 def encode(self, longUrl):  
 self.d[longUrl] = longUrl.\_\_hash\_\_()  
 self.r[longUrl.\_\_hash\_\_()] = longUrl  
 return longUrl.\_\_hash\_\_()  
  
 def decode(self, shortUrl):  
 return self.r[shortUrl]  
  
**Best\_solution:**Two solutions and thoughts   
http://tinyurl.com/0  
  
*520,Complex Number Multiplication:***Best\_solution:**Java 3-liner   
public String complexNumberMultiply(String a, String b) {  
 int[] coefs1 = Stream.of(a.split("\\+|i")).mapToInt(Integer::parseInt).toArray(),   
 coefs2 = Stream.of(b.split("\\+|i")).mapToInt(Integer::parseInt).toArray();  
 return (coefs1[0]\*coefs2[0] - coefs1[1]\*coefs2[1]) + "+" + (coefs1[0]\*coefs2[1] + coefs1[1]\*coefs2[0]) + "i";  
}  
  
  
*521,Convert BST to Greater Tree:***Python\_solution:**Python, Simple with Explanation   
def convertBST(self, root):  
 def visit1(root):  
 if root:  
 visit1(root.left)  
 vals.append(root.val)  
 visit1(root.right)  
 vals = []  
 visit1(root)  
   
 self.s = 0  
 def visit2(root):  
 if root:  
 visit2(root.right)  
 self.s += vals.pop()  
 root.val = self.s  
 visit2(root.left)  
 visit2(root)  
  
 return root  
  
**Best\_solution:**Java Recursive O(n) time   
public class Solution {  
  
 int sum = 0;  
   
 public TreeNode convertBST(TreeNode root) {  
 convert(root);  
 return root;  
 }  
   
 public void convert(TreeNode cur) {  
 if (cur == null) return;  
 convert(cur.right);  
 cur.val += sum;  
 sum = cur.val;  
 convert(cur.left);  
 }  
   
}  
  
  
*522,Minimum Time Difference:***Python\_solution:**Python, Straightforward with Explanation   
def findMinDifference(self, A):  
 def convert(time):  
 return int(time[:2]) \* 60 + int(time[3:])  
 minutes = map(convert, A)  
 minutes.sort()  
   
 return min( (y - x) % (24 \* 60)   
 for x, y in zip(minutes, minutes[1:] + minutes[:1]) )  
  
**Best\_solution:**Verbose Java Solution, Bucket   
public class Solution {  
 public int findMinDifference(List<String> timePoints) {  
 boolean[] mark = new boolean[24 \* 60];  
 for (String time : timePoints) {  
 String[] t = time.split(":");  
 int h = Integer.parseInt(t[0]);  
 int m = Integer.parseInt(t[1]);  
 if (mark[h \* 60 + m]) return 0;  
 mark[h \* 60 + m] = true;  
 }  
   
 int prev = 0, min = Integer.MAX\_VALUE;  
 int first = Integer.MAX\_VALUE, last = Integer.MIN\_VALUE;  
 for (int i = 0; i < 24 \* 60; i++) {  
 if (mark[i]) {  
 if (first != Integer.MAX\_VALUE) {  
 min = Math.min(min, i - prev);  
 }  
 first = Math.min(first, i);  
 last = Math.max(last, i);  
 prev = i;  
 }  
 }  
   
 min = Math.min(min, (24 \* 60 - last + first));  
   
 return min;  
 }  
}  
  
  
*523,Single Element in a Sorted Array:***Python\_solution:**Python in 3 lines.   
class Solution(object):  
 def singleNonDuplicate(self, nums):  
 odd\_set =set(nums[0::2])  
 even\_set =set(nums[1::2])  
 return next(iter(odd\_set-even\_set))  
  
**Best\_solution:**Java Binary Search O(log(n)) Shorter Than Others   
public class Solution {  
 public int singleNonDuplicate(int[] nums) {  
 // binary search  
 int n=nums.length, lo=0, hi=n/2;  
 while (lo < hi) {  
 int m = (lo + hi) / 2;  
 if (nums[2\*m]!=nums[2\*m+1]) hi = m;  
 else lo = m+1;  
 }  
 return nums[2\*lo];  
 }  
}  
  
  
*524,Reverse String II:***Python\_solution:**Python, Straightforward with Explanation   
def reverseStr(self, s, k):  
 s = list(s)  
 for i in xrange(0, len(s), 2\*k):  
 s[i:i+k] = reversed(s[i:i+k])  
 return "".join(s)  
  
**Best\_solution:**Java Concise Solution   
public class Solution {  
 public String reverseStr(String s, int k) {  
 char[] arr = s.toCharArray();  
 int n = arr.length;  
 int i = 0;  
 while(i < n) {  
 int j = Math.min(i + k - 1, n - 1);  
 swap(arr, i, j);  
 i += 2 \* k;  
 }  
 return String.valueOf(arr);  
 }  
 private void swap(char[] arr, int l, int r) {  
 while (l < r) {  
 char temp = arr[l];  
 arr[l++] = arr[r];  
 arr[r--] = temp;  
 }  
 }  
}  
  
  
*525,01 Matrix:***Python\_solution:**Python, Simple with Explanation   
def updateMatrix(self, A):  
 R, C = len(A), len(A[0])  
 def neighbors(r, c):  
 for cr, cc in ((r-1,c),(r+1,c),(r,c-1),(r,c+1)):  
 if 0 <= cr < R and 0 <= cc < C:  
 yield cr, cc  
   
 q = collections.deque([((r, c), 0)   
 for r in xrange(R)   
 for c in xrange(C)   
 if A[r][c] == 0])  
 seen = {x for x,\_ in q}  
 ans = [[0]\*C for \_ in A]  
 while q:  
 (r, c), depth = q.popleft()  
 ans[r][c] = depth  
 for nei in neighbors(r, c):  
 if nei not in seen:  
 seen.add(nei)  
 q.append((nei, depth + 1))  
   
 return ans  
  
**Best\_solution:**Java Solution, BFS   
BFS  
  
*526,Diameter of Binary Tree:***Python\_solution:**Python, Simple with Explanation   
def diameterOfBinaryTree(self, root):  
 self.best = 1  
 def depth(root):  
 if not root: return 0  
 ansL = depth(root.left)  
 ansR = depth(root.right)  
 self.best = max(self.best, ansL + ansR + 1)  
 return 1 + max(ansL, ansR)  
   
 depth(root)  
 return self.best - 1  
  
**Best\_solution:**Java Solution, MaxDepth   
every  
  
*529,Remove Boxes:***Python\_solution:**Python, Straightforward [but slow] with Explanation   
def removeBoxes(self, A):  
 def outside\_ranges(ranges, i, j):  
 prev = i  
 for r1, r2 in ranges:  
 yield prev, r1 - 1  
 prev = r2 + 1  
 yield prev, j  
   
 memo = {}  
 def dp(i, j):  
 if i >= j: return +(i==j)  
 if (i,j) not in memo:  
 good = []  
 for k, v in itertools.groupby(range(i, j+1),  
 key = lambda x: A[x] == A[i]):  
 if k:  
 w = list(v)  
 good.append((w[0], w[-1]))  
   
 ans = 0  
 for size in xrange(1, len(good) + 1):  
 for subset in itertools.combinations(good, size):  
 cand = sum( g[-1] - g[0] + 1 for g in subset ) \*\* 2  
 cand += sum( dp(L, R) for L, R in outside\_ranges(subset, i, j) )  
 ans = max(ans, cand)  
   
 memo[i, j] = ans  
 return memo[i, j]  
 return dp(0, len(A)-1)  
  
**Best\_solution:**Java top-down and bottom-up DP solutions   
boxes  
  
*530,Friend Circles:***Python\_solution:**Python, Simple Explanation   
def findCircleNum(self, A):  
 N = len(A)  
 seen = set()  
 def dfs(node):  
 for nei, adj in enumerate(A[node]):  
 if adj and nei not in seen:  
 seen.add(nei)  
 dfs(nei)  
   
 ans = 0  
 for i in xrange(N):  
 if i not in seen:  
 dfs(i)  
 ans += 1  
 return ans  
  
**Best\_solution:**Neat DFS java solution   
public class Solution {  
 public void dfs(int[][] M, int[] visited, int i) {  
 for (int j = 0; j < M.length; j++) {  
 if (M[i][j] == 1 && visited[j] == 0) {  
 visited[j] = 1;  
 dfs(M, visited, j);  
 }  
 }  
 }  
 public int findCircleNum(int[][] M) {  
 int[] visited = new int[M.length];  
 int count = 0;  
 for (int i = 0; i < M.length; i++) {  
 if (visited[i] == 0) {  
 dfs(M, visited, i);  
 count++;  
 }  
 }  
 return count;  
 }  
}  
  
*533,Student Attendance Record I:***Python\_solution:**Tiny Ruby, Short Python/Java/C++   
def check\_record(s)  
 !s[/A.\*A|LLL/]  
end  
  
**Best\_solution:**Java 1-liner   
public boolean checkRecord(String s) {  
 return !s.matches(".\*LLL.\*|.\*A.\*A.\*");  
}  
  
  
*534,Student Attendance Record II:***Python\_solution:**Python DP with explanation   
dp[i]  
**Best\_solution:**Improving the runtime from O(n) to O(log n)   
f[i][j][k]  
  
*535,Optimal Division:***Python\_solution:**Python, Straightforward with Explanation (Insightful Approach)   
def optimalDivision(self, A):  
 A = map(str, A)  
 if len(A) <= 2: return '/'.join(A)  
 return '{}/({})'.format(A[0], '/'.join(A[1:]))  
  
**Best\_solution:**Easy to understand simple O(n) solution with explanation   
class Solution {  
public:  
 string optimalDivision(vector<int>& nums) {  
 string ans;  
 if(!nums.size()) return ans;  
 ans = to\_string(nums[0]);  
 if(nums.size()==1) return ans;  
 if(nums.size()==2) return ans + "/" + to\_string(nums[1]);  
 ans += "/(" + to\_string(nums[1]);  
 for(int i = 2; i < nums.size();++i)  
 ans += "/" + to\_string(nums[i]);  
 ans += ")";  
 return ans;  
};  
  
  
*536,Brick Wall:***Python\_solution:**Python, with simple explanation   
class Solution(object):  
 def leastBricks(self, wall):  
 """  
 :type wall: List[List[int]]  
 :rtype: int  
 """  
 if len(wall)==0: return -1  
 Counter,cumSum=collections.defaultdict(int),[0]\*len(wall)  
 for i in range(0,len(wall)):  
 for y in wall[i]:  
 cumSum[i]+=y  
 Counter[cumSum[i]]+=1  
 Counter[cumSum[0]]=0  
 return len(wall)-max(Counter.values())  
  
**Best\_solution:**I DON'T THINK THERE IS A BETTER PERSON THAN ME TO ANSWER THIS QUESTION   
public class Solution {  
 public int leastBricks(List<List<Integer>> wall) {  
 if(wall.size() == 0) return 0;  
 int count = 0;  
 Map<Integer, Integer> map = new HashMap<Integer, Integer>();  
 for(List<Integer> list : wall){  
 int length = 0;  
 for(int i = 0; i < list.size() - 1; i++){  
 length += list.get(i);  
 map.put(length, map.getOrDefault(length, 0) + 1);  
 count = Math.max(count, map.get(length));  
 }  
 }  
 return wall.size() - count;  
 }  
}  
  
*538,Next Greater Element III:***Python\_solution:**Clear Python Solution   
class Solution(object):  
 def nextGreaterElement(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 num = str(n)  
 for i in range(len(num)-2, -1, -1):  
 if num[i] < num[i+1]:  
 t = list(num[i:])  
 for j in range(len(t)-1, 0, -1):  
 if t[j]>t[0]:  
 first = t.pop(j)  
 rest = sorted(t)  
 res = int(num[:i] + first + ''.join(rest))   
 return res if res <= (2\*\*31-1) else -1   
 #print t  
 #raise ValueError('Error: cannot find bigger value!')  
 return -1  
  
**Best\_solution:**Simple Java solution (4ms) with explanation.   
public class Solution {  
 public int nextGreaterElement(int n) {  
 char[] number = (n + "").toCharArray();  
   
 int i, j;  
 // I) Start from the right most digit and   
 // find the first digit that is  
 // smaller than the digit next to it.  
 for (i = number.length-1; i > 0; i--)  
 if (number[i-1] < number[i])  
 break;  
  
 // If no such digit is found, its the edge case 1.  
 if (i == 0)  
 return -1;  
   
 // II) Find the smallest digit on right side of (i-1)'th   
 // digit that is greater than number[i-1]  
 int x = number[i-1], smallest = i;  
 for (j = i+1; j < number.length; j++)  
 if (number[j] > x && number[j] <= number[smallest])  
 smallest = j;  
   
 // III) Swap the above found smallest digit with   
 // number[i-1]  
 char temp = number[i-1];  
 number[i-1] = number[smallest];  
 number[smallest] = temp;  
   
 // IV) Sort the digits after (i-1) in ascending order  
 Arrays.sort(number, i, number.length);  
   
 long val = Long.parseLong(new String(number));  
 return (val <= Integer.MAX\_VALUE) ? (int) val : -1;  
 }  
}  
  
  
*539,Reverse Words in a String III:***Python\_solution:**1 line Ruby / Python   
def reverse\_words(s)  
 s.split.map(&:reverse).join(" ")  
end  
  
**Best\_solution:**[C++] [Java] Clean Code   
class Solution {  
public:  
 string reverseWords(string s) {  
 for (int i = 0; i < s.length(); i++) {  
 if (s[i] != ' ') { // when i is a non-space  
 int j = i;  
 for (; j < s.length() && s[j] != ' '; j++) { } // move j to the next space  
 reverse(s.begin() + i, s.begin() + j);  
 i = j - 1;  
 }  
 }  
   
 return s;  
 }  
};  
  
  
*540,Subarray Sum Equals K:***Python\_solution:**Python, Simple with Explanation   
A[0] + A[1] + ... + A[t-1] = W  
**Best\_solution:**Java Solution, PreSum + HashMap   
SUM[i, j]  
  
*541,Array Partition I:***Python\_solution:**Python 1 line (sorting is accepted)   
class Solution(object):  
  
 def arrayPairSum(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 return sum(sorted(nums)[::2])  
**Best\_solution:**Java Solution, Sorting. And rough proof of algorithm.   
public class Solution {  
 public int arrayPairSum(int[] nums) {  
 Arrays.sort(nums);  
 int result = 0;  
 for (int i = 0; i < nums.length; i += 2) {  
 result += nums[i];  
 }  
 return result;  
 }  
}  
  
  
*543,Binary Tree Tilt:***Python\_solution:**Python, Simple with Explanation   
for each node: ans += abs(node.left.subtreesum - node.right.subtreesum)  
**Best\_solution:**Java Solution, post-order traversal   
public class Solution {  
 int result = 0;  
   
 public int findTilt(TreeNode root) {  
 postOrder(root);  
 return result;  
 }  
   
 private int postOrder(TreeNode root) {  
 if (root == null) return 0;  
   
 int left = postOrder(root.left);  
 int right = postOrder(root.right);  
   
 result += Math.abs(left - right);  
   
 return left + right + root.val;  
 }  
}  
  
  
*544,Find the Closest Palindrome:***Python\_solution:**Python, Simple with Explanation   
S  
**Best\_solution:**Python, Simple with Explanation   
S  
  
*545,Array Nesting:***Python\_solution:**Short Python   
i  
**Best\_solution:**[C++] [Java] Clean Code - O(N)   
circle  
  
*546,Reshape the Matrix:***Python\_solution:**Python Solutions   
NumPy  
**Best\_solution:**Java Concise O(nm) time   
public int[][] matrixReshape(int[][] nums, int r, int c) {  
 int n = nums.length, m = nums[0].length;  
 if (r\*c != n\*m) return nums;  
 int[][] res = new int[r][c];  
 for (int i=0;i<r\*c;i++)   
 res[i/c][i%c] = nums[i/m][i%m];  
 return res;  
}  
  
  
*547,Permutation in String:***Python\_solution:**Python, Simple with Explanation   
window  
**Best\_solution:**Java Solution, Sliding Window   
p  
  
*552,Subtree of Another Tree:***Python\_solution:**Python, Straightforward with Explanation (O(ST) and O(S+T) approaches)   
s  
**Best\_solution:**Java Solution, tree traversal   
s  
  
*555,Distribute Candies:***Python\_solution:**Python, Straightforward with Explanation   
len(set(candies))  
**Best\_solution:**Java Solution, 3 lines, HashSet   
public class Solution {  
 public int distributeCandies(int[] candies) {  
 Set<Integer> kinds = new HashSet<>();  
 for (int candy : candies) kinds.add(candy);  
 return kinds.size() >= candies.length / 2 ? candies.length / 2 : kinds.size();  
 }  
}  
  
  
*556,Out of Boundary Paths:***Python\_solution:**Python, Straightforward with Explanation   
cur[r][c]  
**Best\_solution:**C++ 6 lines DP O(N \* m \* n), 6 ms   
int findPaths(int m, int n, int N, int i, int j) {  
 uint dp[51][50][50] = {};  
 for (auto Ni = 1; Ni <= N; ++Ni)  
 for (auto mi = 0; mi < m; ++mi)  
 for (auto ni = 0; ni < n; ++ni)  
 dp[Ni][mi][ni] = ((mi == 0 ? 1 : dp[Ni - 1][mi - 1][ni]) + (mi == m - 1? 1 : dp[Ni - 1][mi + 1][ni])  
 + (ni == 0 ? 1 : dp[Ni - 1][mi][ni - 1]) + (ni == n - 1 ? 1 : dp[Ni - 1][mi][ni + 1])) % 1000000007;  
 return dp[N][i][j];  
}  
  
  
*561,Shortest Unsorted Continuous Subarray:***Best\_solution:**Java O(n) Time O(1) Space   
beg  
  
*563,Delete Operation for Two Strings:***Best\_solution:**Java DP Solution (Longest Common Subsequence)   
public int minDistance(String word1, String word2) {  
 int dp[][] = new int[word1.length()+1][word2.length()+1];  
 for(int i = 0; i <= word1.length(); i++) {  
 for(int j = 0; j <= word2.length(); j++) {  
 if(i == 0 || j == 0) dp[i][j] = 0;  
 else dp[i][j] = (word1.charAt(i-1) == word2.charAt(j-1)) ? dp[i-1][j-1] + 1  
 : Math.max(dp[i-1][j], dp[i][j-1]);  
 }  
 }  
 int val = dp[word1.length()][word2.length()];  
 return word1.length() - val + word2.length() - val;  
}  
  
*567,Erect the Fence:***Python\_solution:**Python, AM Chain with Explanation   
drive  
**Best\_solution:**Java Solution, Convex Hull Algorithm - Gift wrapping aka Jarvis march   
Gift wrapping aka Jarvis march  
  
*569,Tag Validator:***Python\_solution:**Short Python, accepted but not sure if correct   
c  
**Best\_solution:**Java Solution: Use startsWith and indexOf   
public class Solution {  
 public boolean isValid(String code) {  
 Stack<String> stack = new Stack<>();  
 for(int i = 0; i < code.length();){  
 if(i>0 && stack.isEmpty()) return false;  
 if(code.startsWith("<![CDATA[", i)){  
 int j = i+9;  
 i = code.indexOf("]]>", j);  
 if(i < 0) return false;  
 i += 3;  
 }else if(code.startsWith("</", i)){  
 int j = i + 2;  
 i = code.indexOf('>', j);  
 if(i < 0 || i == j || i - j > 9) return false;  
 for(int k = j; k < i; k++){  
 if(!Character.isUpperCase(code.charAt(k))) return false;  
 }  
 String s = code.substring(j, i++);  
 if(stack.isEmpty() || !stack.pop().equals(s)) return false;  
 }else if(code.startsWith("<", i)){  
 int j = i + 1;  
 i = code.indexOf('>', j);  
 if(i < 0 || i == j || i - j > 9) return false;  
 for(int k = j; k < i; k++){  
 if(!Character.isUpperCase(code.charAt(k))) return false;  
 }  
 String s = code.substring(j, i++);  
 stack.push(s);  
 }else{  
 i++;  
 }  
 }  
 return stack.isEmpty();  
 }  
}  
  
  
*570,Fraction Addition and Subtraction:***Python\_solution:**Small simple C++/Java/Python   
A / B  
**Best\_solution:**Concise Java Solution   
public String fractionAddition(String expression) {  
 String[] fracs = expression.split("(?=[-,+])"); // splits input string into individual fractions  
 String res = "0/1";  
 for (String frac : fracs) res = add(res, frac); // add all fractions together  
 return res;  
}  
  
public String add(String frac1, String frac2) {  
 int[] f1 = Stream.of(frac1.split("/")).mapToInt(Integer::parseInt).toArray(),   
 f2 = Stream.of(frac2.split("/")).mapToInt(Integer::parseInt).toArray();  
 int numer = f1[0]\*f2[1] + f1[1]\*f2[0], denom = f1[1]\*f2[1];  
 String sign = "";  
 if (numer < 0) {sign = "-"; numer \*= -1;}  
 return sign + numer/gcd(numer, denom) + "/" + denom/gcd(numer, denom); // construct reduced fraction  
}  
  
// Computes gcd using Euclidean algorithm  
public int gcd(int x, int y) { return x == 0 || y == 0 ? x + y : gcd(y, x % y); }  
  
  
*571,Valid Square:***Python\_solution:**Share my simple Python solution   
class Solution(object):  
 def validSquare(self, p1, p2, p3, p4):  
 points = [p1, p2, p3, p4]  
   
 dists = collections.Counter()  
 for i in range(len(points)):  
 for j in range(i+1, len(points)):  
 dists[self.getDistance(points[i], points[j])] += 1  
   
 return len(dists.values())==2 and 4 in dists.values() and 2 in dists.values()  
   
 def getDistance(self, p1, p2):  
 return (p1[0] - p2[0])\*\*2 + (p1[1] - p2[1])\*\*2  
  
**Best\_solution:**C++ 3 lines (unordered\_set)   
int d(vector<int>& p1, vector<int>& p2) {  
 return (p1[0] - p2[0]) \* (p1[0] - p2[0]) + (p1[1] - p2[1]) \* (p1[1] - p2[1]);  
}  
bool validSquare(vector<int>& p1, vector<int>& p2, vector<int>& p3, vector<int>& p4) {  
 unordered\_set<int> s({ d(p1, p2), d(p1, p3), d(p1, p4), d(p2, p3), d(p2, p4), d(p3, p4) });  
 return !s.count(0) && s.size() == 2;  
}  
  
  
*572,Longest Harmonious Subsequence:***Python\_solution:**Python, Straightforward with Explanation   
count[x]  
**Best\_solution:**Simple Java HashMap Solution   
public int findLHS(int[] nums) {  
 Map<Long, Integer> map = new HashMap<>();  
 for (long num : nums) {  
 map.put(num, map.getOrDefault(num, 0) + 1);  
 }  
 int result = 0;  
 for (long key : map.keySet()) {  
 if (map.containsKey(key + 1)) {  
 result = Math.max(result, map.get(key + 1) + map.get(key));  
 }  
 }  
 return result;  
}  
  
*573,Big Countries:***Best\_solution:**TLE- Union distinct.   
select name, population, area   
from World  
where area > 3000000  
Union distinct  
select name, population, area  
from World  
where population > 25000000;  
  
*574,Classes More Than 5 Students:***Best\_solution:**"More than" or "no less than"??   
None  
  
*576,Range Addition II:***Python\_solution:**Python solution , beat 100%   
 def maxCount(self, m, n, ops):  
 """  
 :type m: int  
 :type n: int  
 :type ops: List[List[int]]  
 :rtype: int  
 """  
 if not ops:  
 return m\*n  
 return min(op[0] for op in ops)\*min(op[1] for op in ops)  
**Best\_solution:**Java Solution, find Min   
public class Solution {  
 public int maxCount(int m, int n, int[][] ops) {  
 if (ops == null || ops.length == 0) {  
 return m \* n;  
 }  
   
 int row = Integer.MAX\_VALUE, col = Integer.MAX\_VALUE;  
 for(int[] op : ops) {  
 row = Math.min(row, op[0]);  
 col = Math.min(col, op[1]);  
 }  
   
 return row \* col;  
 }  
}  
  
  
*577,Minimum Index Sum of Two Lists:***Best\_solution:**Java O(n+m) Time O(n) Space   
public String[] findRestaurant(String[] list1, String[] list2) {  
 Map<String, Integer> map = new HashMap<>();  
 List<String> res = new LinkedList<>();  
 int minSum = Integer.MAX\_VALUE;  
 for (int i=0;i<list1.length;i++) map.put(list1[i], i);  
 for (int i=0;i<list2.length;i++) {  
 Integer j = map.get(list2[i]);  
 if (j != null && i + j <= minSum) {  
 if (i + j < minSum) { res = new LinkedList<>(); minSum = i+j; }  
 res.add(list2[i]);  
 }  
 }  
 return res.toArray(new String[res.size()]);  
}  
  
  
*578,Non-negative Integers without Consecutive Ones:***Python\_solution:**python dp solution easily understood   
class Solution(object):  
 def findIntegers(self, num):  
 """  
 :type num: int  
 :rtype: int  
 """  
 # A[0] is the lowest bit, A[-1] is the highest bit  
 A=bin(num)[2:][::-1]  
 # dp[i][0] is the number of integers with (i+1)bits, highest bit is 0 and without consecutive ones  
 # dp[i][1] is the number of integers with (i+1)bits, highest bit is 1 and without consecutive ones  
 dp=[[1,1] for \_ in range(len(A))]  
 # res is the number of integers less than A[:i] without consecutive ones.  
 res=1 if A[0]=='0' else 2  
 for i in range(1, len(A)):  
 dp[i][0]=dp[i-1][0]+dp[i-1][1]  
 dp[i][1]=dp[i-1][0]  
 # try to get the number of integers less than A[:i+1]  
 if A[i-1:i+1]=='01':  
 # if A[i-1:i+1]=='01', we can append '1' after integers less than A[:i] without consecutive ones,   
 # also any integer with (i+1) bits, highest bit is '0', without consecutive ones  
 # is less than A[:i+1]  
 res+=dp[i][0]  
 elif A[i-1:i+1]=='11':  
 # if A[i-1:i+1]=='11', then any integer with i+1 bits and without consecutive ones   
 # is less than A[:i+1]  
 res=dp[i][0]+dp[i][1]  
 # if A[i]=='0', the number of integers with i+1 bits, less than A[:i+1] and without   
 # consecutive ones is the same as A[:i]  
 return res  
  
**Best\_solution:**Java Solution, DP   
public class Solution {  
 public int findIntegers(int num) {  
 StringBuilder sb = new StringBuilder(Integer.toBinaryString(num)).reverse();  
 int n = sb.length();  
   
 int a[] = new int[n];  
 int b[] = new int[n];  
 a[0] = b[0] = 1;  
 for (int i = 1; i < n; i++) {  
 a[i] = a[i - 1] + b[i - 1];  
 b[i] = a[i - 1];  
 }  
   
 int result = a[n - 1] + b[n - 1];  
 for (int i = n - 2; i >= 0; i--) {  
 if (sb.charAt(i) == '1' && sb.charAt(i + 1) == '1') break;  
 if (sb.charAt(i) == '0' && sb.charAt(i + 1) == '0') result -= b[i];  
 }  
   
 return result;  
 }  
}  
  
  
*579,Human Traffic of Stadium:***Best\_solution:**What's wrong with this answer?   
SELECT DISTINCT s1.id, s1.date, s1.people  
FROM stadium s1, stadium s2, stadium s3  
WHERE s1.people >= 100  
AND s2.people >= 100  
AND s3.people >= 100  
AND ((DATEDIFF(s2.date, s1.date) = 1 AND DATEDIFF(s3.date, s2.date) = 1)  
OR (DATEDIFF(s2.date, s1.date) = -1 AND DATEDIFF(s3.date, s1.date) = 1)  
OR (DATEDIFF(s2.date, s1.date) = -1 AND DATEDIFF(s3.date, s2.date) = -1)  
)  
ORDER BY s1.date;  
  
  
*583,Can Place Flowers:***Python\_solution:**Python, Straightforward with Explanation   
def canPlaceFlowers(self, A, N):  
 for i, x in enumerate(A):  
 if (not x and (i == 0 or A[i-1] == 0)   
 and (i == len(A)-1 or A[i+1] == 0)):  
 N -= 1  
 A[i] = 1  
 return N <= 0  
  
**Best\_solution:**Java - Greedy solution - O(flowerbed) - beats 100%   
public class Solution {  
 public boolean canPlaceFlowers(int[] flowerbed, int n) {  
 int count = 0;  
 for(int i = 0; i < flowerbed.length && count < n; i++) {  
 if(flowerbed[i] == 0) {  
 //get next and prev flower bed slot values. If i lies at the ends the next and prev are considered as 0.   
 int next = (i == flowerbed.length - 1) ? 0 : flowerbed[i + 1];   
 int prev = (i == 0) ? 0 : flowerbed[i - 1];  
 if(next == 0 && prev == 0) {  
 flowerbed[i] = 1;  
 count++;  
 }  
 }  
 }  
   
 return count == n;  
 }  
}  
  
  
*584,Construct String from Binary Tree:***Python\_solution:**Python recursion   
 def tree2str(self, t):  
 """  
 :type t: TreeNode  
 :rtype: str  
 """  
 def preorder(root):  
 if root is None:  
 return ""  
 s=str(root.val)  
 l=preorder(root.left)  
 r=preorder(root.right)  
 if r=="" and l=="":  
 return s   
 elif l=="":  
 s+="()"+"("+r+")"  
 elif r=="":   
 s+="("+l+")"  
 else :   
 s+="("+l+")"+"("+r+")"  
 return s  
 return preorder(t)  
  
**Best\_solution:**Java Solution, Tree Traversal   
public class Solution {  
 public String tree2str(TreeNode t) {  
 if (t == null) return "";  
   
 String result = t.val + "";  
   
 String left = tree2str(t.left);  
 String right = tree2str(t.right);  
   
 if (left == "" && right == "") return result;  
 if (left == "") return result + "()" + "(" + right + ")";  
 if (right == "") return result + "(" + left + ")";  
 return result + "(" + left + ")" + "(" + right + ")";  
 }  
}  
  
  
*587,Find Duplicate File in System:***Best\_solution:**C++ clean solution, answers to follow up   
vector<vector<string>> findDuplicate(vector<string>& paths) {  
 unordered\_map<string, vector<string>> files;  
 vector<vector<string>> result;  
  
 for (auto path : paths) {  
 stringstream ss(path);  
 string root;  
 string s;  
 getline(ss, root, ' ');  
 while (getline(ss, s, ' ')) {  
 string fileName = root + '/' + s.substr(0, s.find('('));  
 string fileContent = s.substr(s.find('(') + 1, s.find(')') - s.find('(') - 1);  
 files[fileContent].push\_back(fileName);  
 }  
 }  
  
 for (auto file : files) {  
 if (file.second.size() > 1)  
 result.push\_back(file.second);  
 }  
  
 return result;  
}  
  
  
*589,Valid Triangle Number:***Python\_solution:**Can this problem possibly be solved by python?   
def triangleNumber(self,nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 final = 0  
 nums = sorted(nums)  
 for i in range(2,len(nums))[::-1]:  
 l = 0  
 r = i-1  
 while (r>l):  
 if nums[l]+nums[r] > nums[i]:  
 final += r-l  
 r-=1  
 else:  
 l+=1  
 return final  
  
**Best\_solution:**Java O(n^2) Time O(1) Space   
public static int triangleNumber(int[] A) {  
 Arrays.sort(A);  
 int count = 0, n = A.length;  
 for (int i=n-1;i>=2;i--) {  
 int l = 0, r = i-1;  
 while (l < r) {  
 if (A[l] + A[r] > A[i]) {  
 count += r-l;  
 r--;  
 }  
 else l++;  
 }  
 }  
 return count;  
}  
  
  
*595,Merge Two Binary Trees:***Python\_solution:**Short Recursive Solution w/ Python & C++   
class Solution(object):  
 def mergeTrees(self, t1, t2):  
 if t1 and t2:  
 root = TreeNode(t1.val + t2.val)  
 root.left = self.mergeTrees(t1.left, t2.left)  
 root.right = self.mergeTrees(t1.right, t2.right)  
 return root  
 else:  
 return t1 or t2  
  
**Best\_solution:**Java Solution, 6 lines, Tree Traversal   
public class Solution {  
 public TreeNode mergeTrees(TreeNode t1, TreeNode t2) {  
 if (t1 == null && t2 == null) return null;  
   
 int val = (t1 == null ? 0 : t1.val) + (t2 == null ? 0 : t2.val);  
 TreeNode newNode = new TreeNode(val);  
   
 newNode.left = mergeTrees(t1 == null ? null : t1.left, t2 == null ? null : t2.left);  
 newNode.right = mergeTrees(t1 == null ? null : t1.right, t2 == null ? null : t2.right);  
   
 return newNode;  
 }  
}  
  
  
*598,Not Boring Movies:***Best\_solution:**The problem description could be worded better   
select \*  
from cinema  
where mod(id, 2) = 1 and description != 'boring'  
order by rating DESC  
;  
  
  
*599,Task Scheduler:***Python\_solution:**6 lines O(N) solutions w/ C++ & Python   
class Solution {  
public:  
 int leastInterval(vector<char>& tasks, int n) {  
 vector<int> counter(256);  
 for ( char t : tasks )  
 ++counter[t];  
 int m = \*max\_element(counter.begin(), counter.end());  
 int l = count(counter.begin(), counter.end(), m);  
 return max(int(tasks.size()), (m - 1) \* (n + 1) + l);  
 }  
};  
  
// 64 / 64 test cases passed.  
// Status: Accepted  
// Runtime: 63 ms  
  
**Best\_solution:**concise Java Solution O(N) time O(26) space   
// (c[25] - 1) \* (n + 1) + 25 - i is frame size  
// when inserting chars, the frame might be "burst", then tasks.length takes precedence  
// when 25 - i > n, the frame is already full at construction, the following is still valid.  
public class Solution {  
 public int leastInterval(char[] tasks, int n) {  
  
 int[] c = new int[26];  
 for(char t : tasks){  
 c[t - 'A']++;  
 }  
 Arrays.sort(c);  
 int i = 25;  
 while(i >= 0 && c[i] == c[25]) i--;  
  
 return Math.max(tasks.length, (c[25] - 1) \* (n + 1) + 25 - i);  
 }  
}  
  
  
*600,Add One Row to Tree:***Python\_solution:**Short Python BFS   
def addOneRow(self, root, v, d):  
 dummy, dummy.left = TreeNode(None), root  
 row = [dummy]  
 for \_ in range(d - 1):  
 row = [kid for node in row for kid in (node.left, node.right) if kid]  
 for node in row:  
 node.left, node.left.left = TreeNode(v), node.left  
 node.right, node.right.right = TreeNode(v), node.right  
 return dummy.left  
**Best\_solution:**[C++] [Java] 10 line Solution - no helper   
1  
  
*601,Maximum Distance in Arrays:***Best\_solution:**Java Solution, Min and Max   
public class Solution {  
 public int maxDistance(int[][] arrays) {  
 int result = Integer.MIN\_VALUE;  
 int max = arrays[0][arrays[0].length - 1];  
 int min = arrays[0][0];  
   
 for (int i = 1; i < arrays.length; i++) {  
 result = Math.max(result, Math.abs(arrays[i][0] - max));  
 result = Math.max(result, Math.abs(arrays[i][arrays[i].length - 1] - min));  
 max = Math.max(max, arrays[i][arrays[i].length - 1]);  
 min = Math.min(min, arrays[i][0]);  
 }  
   
 return result;  
 }  
}  
  
  
*602,Minimum Factorization:***Python\_solution:**Python DP solution   
a < 10  
**Best\_solution:**Java Solution, result array   
public class Solution {  
 public int smallestFactorization(int n) {  
 // Case 1: If number is smaller than 10  
 if (n < 10) return n;  
   
 // Case 2: Start with 9 and try every possible digit  
 List<Integer> res = new ArrayList<>();  
 for (int i = 9; i > 1; i--) {  
 // If current digit divides n, then store all  
 // occurrences of current digit in res  
 while (n % i == 0) {  
 n = n / i;  
 res.add(i);  
 }  
 }  
  
 // If n could not be broken in form of digits  
 if (n != 1) return 0;  
  
 // Get the result from the array in reverse order  
 long result = 0;  
 for (int i = res.size() - 1; i >= 0; i--) {  
 result = result \* 10 + res.get(i);  
 if (result > Integer.MAX\_VALUE) return 0;  
 }  
   
 return (int)result;  
 }  
}  
  
  
*604,Swap Salary:***Best\_solution:**Accept solution with xor   
update salary set sex = CHAR(ASCII('f') ^ ASCII('m') ^ ASCII(sex));  
  
  
*605,Maximum Product of Three Numbers:***Python\_solution:**Python, Straightforward with Explanation   
def maximumProduct(self, A):  
 A.sort()  
 if len(A) > 6:  
 A = A[:3] + A[-3:]  
   
 return max(A[i] \* A[j] \* A[k]  
 for i in xrange(len(A))  
 for j in xrange(i+1, len(A))  
 for k in xrange(j+1, len(A)))  
  
**Best\_solution:**Java Easy AC...   
 public int maximumProduct(int[] nums) {  
   
 Arrays.sort(nums);  
 //One of the Three Numbers is the maximum value in the array.  
  
 int a = nums[nums.length - 1] \* nums[nums.length - 2] \* nums[nums.length - 3];  
 int b = nums[0] \* nums[1] \* nums[nums.length - 1];  
 return a > b ? a : b;  
 }  
  
  
*606,K Inverse Pairs Array:***Python\_solution:**Python, Straightforward with Explanation   
dp[n][k]  
**Best\_solution:**Python, Straightforward with Explanation   
dp[n][k]  
  
*607,Course Schedule III:***Python\_solution:**Python, Straightforward with Explanation   
end  
**Best\_solution:**Python, Straightforward with Explanation   
end  
  
*608,Design Excel Sum Formula:***Python\_solution:**Python `Run Code` showed error in expected answer   
None  
**Best\_solution:**C++ straight forward   
class Excel {  
private:  
 int \*\*dict;  
 int offset, H, W;  
 unordered\_map<int, vector<string>> mp;  
public:  
 Excel(int H, char W) {  
 offset = 'A';  
 this->H = H;  
 this->W = W - offset + 1;  
 mp.clear();  
 dict = new int\*[H];  
 for (int i = 0; i < H; i++) {  
 dict[i] = new int[this->W];  
 memset(dict[i], 0, sizeof(int)\*this->W);  
 }  
 }  
   
 void set (int r, char c, int v) {  
 int k = (r << 10) + c;  
 dict[r - 1][c - offset] = v;  
 mp.erase(k);  
 }  
   
 int get (int r, char c) {  
 int k = (r << 10) + c;  
 if (mp.find(k) == mp.end())  
 return dict[r - 1][c - offset];  
 return get\_cells(mp[k]);  
 }  
   
 int sum (int r, char c, vector<string> strs) {  
 int k = (r << 10) + c;  
 dict[r - 1][c - offset] = get\_cells(strs);  
 mp[k] = strs;  
 return dict[r - 1][c - offset];  
 }  
   
 int get\_cells(vector<string> &strs) {  
 int res = 0;  
 for (auto s : strs) {  
 if (s.find(':') == -1)  
 res += get\_cell(s);  
 else  
 res += get\_cell\_range(s);  
 }  
 return res;  
 }  
   
 int get\_cell(string &cell) {  
 int r = 0, idx = 0;  
 char c = cell[idx++];  
 while (idx < cell.length())  
 r = 10 \* r + cell[idx++] - '0';  
 return get(r, c);  
 }  
   
 int get\_cell\_range(string &cell\_range) {  
 int rs = 0, re = 0, idx = 0, res = 0;  
 char cs, ce;  
   
 int seg = cell\_range.find(':');  
 cs = cell\_range[idx++];  
 while (idx < seg)  
 rs = 10 \* rs + cell\_range[idx++] - '0';  
   
 idx++;  
 ce = cell\_range[idx++];  
 while (idx < cell\_range.length())  
 re = 10 \* re + cell\_range[idx++] - '0';  
   
 for (int r = rs; r <= re; r++) {  
 for (char c = cs; c <= ce; c++) {  
 res += get(r, c);  
 }  
 }  
 return res;  
 }  
};  
  
/\*\*  
 \* Your Excel object will be instantiated and called as such:  
 \* Excel obj = new Excel(H, W);  
 \* obj.set(r,c,v);  
 \* int param\_2 = obj.get(r,c);  
 \* int param\_3 = obj.sum(r,c,strs);  
 \*/