15. 3Sum

Given numbers, return the combination whose sum is 0, the res should not contain duplicate triplets.

```
class Solution:
    def threeSum(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        nums.sort()
        res = []
        l = len(nums)
        for i in range(l-2):
            if i == 0 or nums[i] != nums[i-1]:
                p1 = i+1
                p2 = l-1
                while p1 < p2:
                    if p1 == i+1 or nums[p1] != nums[p1-1]:
                        if nums[i] + nums[p1] + nums[p2] > 0:
                            p2 = p2 -1
                        elif nums[i] + nums[p1] + nums[p2] < 0:
                            p1 = p1 +1
                        else:
                            res.append([nums[i] , nums[p1] , nums[p2]])
                            p1 = p1+1
                    else:
                        p1=p1+1
            else:
                i=i+1
        return res
```

16. 3Sum Closest

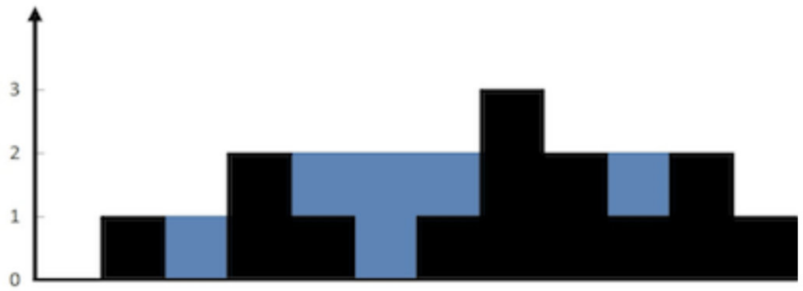
what is the absolute value in python?

answer: abs()

```
class Solution:
    def threeSumClosest(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        nums.sort()

        min_distance = abs(nums[0] + nums[1] + nums[2] - target)
        res = nums[0] + nums[1] + nums[2]

        for i in range(len(nums) - 2):

            l = i + 1
            r = len(nums) - 1

            while l < r:
                sum_ = nums[i] + nums[l] + nums[r]
                if abs(sum_ - target) < min_distance:
                    min_distance = abs(sum_ - target)
                    res = sum_

                if sum_ < target:
                    l = l+1
                elif sum_ > target:
                    r = r-1
                else:
                    return target

        return res
```

42. Trapping Rain Water

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



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

Example:

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

use two pointers, the time complexity is O(N), space is O(1)

---

class Solution: def trap(self, height): """ :type height: List[int] :rtype: int 0,1,0,2,1,0,1,3,2,1,2,1 i j left_max = 0 right_max = 1

```
    so we firstly move i forward,
    """

    # use two pointers still
    if len(height) == 0 or len(height) == 1:
        return  0

    i = 0
    j = len(height) - 1

    right_max = height[j]
    left_max = height[i]
    area = 0

    while i < j:
        if left_max<= right_max:
            i = i+1
            if height[i] < left_max :
                area = area + left_max - height[i]
            else:
                left_max = height[i]
        else:

            j = j-1
            if height[j] < right_max:
                area = area + right_max - height[j]
            else:
                right_max = height[j]

    return area
```

48. Rotate Image

```
class Solution:
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        matrix.reverse()
        for i in range(len(matrix)):
            for j in range(i):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
```

56. Merge Intervals

Given a collection of intervals, merge all overlapping intervals.

Example 1:

```
Input: [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].
```

Example 2:

```
Input: [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

1. how to sort by the first number in python?

sorted([('abc', 121),('abc', 231),('abc', 148), ('abc',221)], key=lambda x: x[1])

```
# Definition for an interval.
# class Interval:
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution:
    def merge(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: List[Interval]
        """
        if len(intervals) == 0 or len(intervals) == 1:
            return  intervals

        intervals = sorted(intervals, key = lambda x:x.start)
        start = intervals[0].start
        end = intervals[0].end
        res = []

        for i in range(1, len(intervals)):
            if intervals[i].start > end:
                res.append(Interval(start, end))
                start = intervals[i].start
                end = intervals[i].end
            else:
                end = max(end, intervals[i].end)

        res.append(Interval(start,end))
        return res
```

152. Maximum Product Subarray

```
class Solution:
    def maxProduct(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        [2,3,-2,4]


        """

        min_ = nums[0]
        max_ = nums[0]
        res = nums[0]

        for i in range(1,len(nums)):
            num = nums[i]
            if num < 0:
                temp = min_
                min_ = max_
                max_ = temp

            max_ = max(max_*num, num)
            min_ = min(min_*num, num)
            res = max(max_, res)
        return res
```

238. Product of Array Except Self

```
class Solution:
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        1 2 3 4
        1 1 2 6
        24  12  4  1
        """

        res = []
        cur = 1
        l = len(nums)
        for i in range(l-1):
            res.append(cur)
            cur = cur * nums[i]
        res.append(cur)

        cur = 1
        for i in range(l-1, 0, -1):
            res[i] = res[i] * cur
            cur = cur * nums[i]
        res[0] = res[0] * cur

        return res
```

31. Next Permutation

```
class Solution:
    def nextPermutation(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place instead.

        1 5 2 4 3 5 6 7 8 - > 8 7
        1 3 5 4 3 2 -> 2 1 3 4 5
        """


        # find the first number that is larger than the last number backward

        ind = len(nums) - 1 # 5
        if ind <=0:
            return
        while ind > 0 and nums[ind] <= nums[ind-1]:
            ind = ind-1

        if ind == 0:
            nums=  nums.reverse()
            return

        # ind = 2 for the second example
        ind2 = len(nums)-1
        while ind2 > 0 and nums[ind2] <= nums[ind-1]:
            ind2 = ind2-1

        temp = nums[ind-1]
        nums[ind-1] = nums[ind2]
        nums[ind2] = temp

        l = ind
        r = len(nums) - 1

        while l < r:
            temp = nums[l]
            nums[l] = nums[r]
            nums[r] =temp
            l = l+1
            r= r-1
```

33. Search in Rotated Sorted Array

this problem is still very hard for me to solve bug-free.

```
class Solution:
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """

        if len(nums) == 0 :return -1
        l = 0
        r = len(nums) - 1




        if r == 0:
            return 0 if nums[0] == target else -1

        while l <= r:
            mid = (l + r) // 2

            if nums[mid] < nums[l] and target < nums[l]:
                num_mid = nums[mid]
            elif nums[mid] >= nums[l] and target >= nums[l]:
                num_mid = nums[mid]
            elif target < nums[l]:
                num_mid = -9999
            else:
                num_mid = 9999

            if num_mid == target:
                return mid

            if num_mid > target:
                r = mid
            else:
                l = mid + 1

        return -1
```

41. First Missing Positive

```
class Solution:
    def firstMissingPositive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l = len(nums)
        for i in range(l):

            if nums[i] < 1 or nums[i] > l:
                continue
            else:
                while nums[i] != i+1 and nums[i] >=1 and nums[i] <=l:
                    if nums[nums[i] - 1] == nums[i]:
                        break
                    temp = nums[i]
                    nums[i] = nums[nums[i] - 1]
                    nums[temp - 1] = temp

        for i in range(l):
            if nums[i] != i+1:
                return  i+1
        return l+1
```

54. Spiral Matrix

```
class Solution:
    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        res = []

        num_row = len(matrix)
        if num_row == 0 or len(matrix[0]) == 0: return res
        num_col = len(matrix[0])

        row_s = 0
        row_e = num_row - 1
        col_s = 0
        col_e = num_col - 1

        while row_s <= row_e and col_s <= col_e:
            for i in range(col_s, col_e + 1):
                res.append(matrix[row_s][i])

            row_s += 1
            if row_s > row_e:
                break

            for i in range(row_s, row_e + 1):
                res.append(matrix[i][col_e])
            col_e = col_e - 1
            if col_s > col_e:
                break

            for i in range(col_e, col_s - 1, -1):
                res.append(matrix[row_e][i])
            row_e = row_e - 1
            if row_s > row_e:
                break

            for i in range(row_e, row_s - 1, -1):
                res.append(matrix[i][col_s])
            col_s += 1
            if col_s > col_e:
                break

        return res
```

128. Longest Consecutive Sequence

need to redo

```
class Solution:
    def longestConsecutive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums) == 0 : return  0
        res = 1

        nums = set(nums)

        for n in nums:
            if n - 1 not in nums:
                y = n+1
                while y in nums:
                    y = y+1
                    res = max(res, y-n)
        return res
```

153. Find Minimum in Rotated Sorted Array

```
class Solution:
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        l = 0
        r = len(nums) - 1

        while l < r:
            if nums[l] < nums[r]:
                return nums[l]
            mid = (l + r) // 2
            if nums[mid] > nums[l]:
                l = mid + 1
            elif nums[mid] < nums[l]:
                r = mid
            else:
                return min(nums[l], nums[r])
        return nums[l]
```

229. Majority Element II

don't have any idea, need to redo this one

```
class Solution:
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]

        Boyer-Moore Majority Vote
        """

        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
]
```

287. Find the Duplicate Number

```
class Solution:
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        slow = nums[0]
        fast = nums[slow]

        while slow != fast:
            slow= nums[slow]
            fast = nums[nums[fast]]

        fast = 0
        while fast != slow:
            slow = nums[slow]
            fast = nums[fast]
        return fast
```

289. Game of Life

```
class Solution:
    def isAlive(self, board,i,j ):
        if i<0:
            return 0
        if j<0:
            return 0
        if i>= len(board):
            return 0
        if j>=len(board[0]):
            return 0

        if board[i][j] % 2 == 1:
            return 1
        else:
            return  0


    def count(self, board, i, j ):

        return self.isAlive(board, i-1,j-1) + self.isAlive(board, i-1,j) + self.isAl
ive(board, i-1,j+1) + self.isAlive(board, i,j-1) + self.isAlive(board, i,j+1) + sel
f.isAlive(board, i+1,j-1) + self.isAlive(board, i+1,j) + self.isAlive(board, i+1,j+
1)



    def gameOfLife(self, board):
        """
        :type board: List[List[int]]
        :rtype: void Do not return anything, modify board in-place instead.
        """
        n_row = len(board)
        n_col = len(board[0])

        for i in range(n_row):
            for j in range(n_col):
                n_live = self.count(board,i,j)
                if board[i][j] == 1:
                    if n_live == 2 or n_live==3:
                        board[i][j] = 3
                    else:
                        board[i][j] = 1
                else:
                    if n_live == 3:
                        board[i][j] = 2

        for i in range(n_row):
            for j in range(n_col):
                board[i][j]  =  board[i][j] //2
```

695. Max Area of Island

```
class Solution:
    def mark(self, grid,i,j):
        if i<0 or j<0 or i >= len(grid) or j>= len(grid[0]) or grid[i][j] == 0:
            return 0

        grid[i][j] = 0
        return self.mark(grid, i, j+1) + self.mark(grid, i, j-1) + self.mark(grid, i
-1, j) + self.mark(grid, i+1, j) +1

    def maxAreaOfIsland(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int

        """
        res = 0

        n_row = len(grid)
        n_col = len(grid[0])
        for i in range(n_row):
            for j in range(n_col):
                if grid[i][j] == 1:
                    res = max(res, self.mark(grid,i,j))
        return  res
```

39. Combination Sum

for example [2, 3, 6, 7] target = 7,

1. how to do list deep copy in python ?

don't have a very clear idea to solve this.

---

79. Word Search

so the problem here, given a matrix 2d. given a word, search it in the matrix, so each time we can replace the char with 0 for example, then repalce the 0 with the original char.

```
class Solution:
    def search( self, board, word, cur, i, j, n_row, n_col ):

        if i<0 or j<0 or i >= n_row or j >= n_col or word[cur] != board[i][j] :
            return  False

        if cur == len(word)-1:
            return True

        board[i][j] = '0'
        res = self.search(board, word, cur+1, i-1, j, n_row, n_col) or  self.search
(board, word, cur+1, i+1, j, n_row, n_col) or  self.search(board, word, cur+1, i, j-
1, n_row, n_col) or  self.search(board, word, cur+1, i, j+1, n_row, n_col)
        board[i][j] = word[cur]

        return res

    def exist(self, board, word):
        """
        :type board: List[List[str]]
        :type word: str
        :rtype: bool
        """
        n_row = len(board); n_col = len(board[0])

        for i in range(n_row):
            for j in range(n_col):
                if word[0] == board[i][j]:
                    if self.search(board, word, 0, i, j,n_row,n_col):
                        return True

        return False
```

4. Median of Two Sorted Arrays

数组下标控制，很烦，然后binary search，本质上

```
class Solution:
    def findMedianSortedArrays(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: float
        """

        # 1. be sure nums1 is shorter than nums2
        if len(nums1) > len(nums2):
            return self.findMedianSortedArrays(nums2, nums1)

        l1 = len(nums1)
        l2 = len(nums2)

        if l1 == 0:
            return nums2[l2 // 2] if l2 % 2 == 1 else (nums2[l2 // 2 - 1] + nums2[l2
// 2]) / 2.0

        half_length = (l1 + l2) // 2

        '''
        take [2] and [1,3] as an example, half_length = 1
        start to search in nums1
        '''

        l = 0
        r = l1

        while l <= r:
            # iter = 1, l = 0, r = 0
            ind_1 = (l + r) // 2  # 0
            ind_2 = half_length - ind_1  # 1

            if ind_1 > 0 and nums1[ind_1 - 1] > nums2[ind_2]:
                r = ind_1 - 1
            elif ind_1 < l1 and nums1[ind_1] < nums2[ind_2 - 1]:
                l = ind_1 + 1
            else:
                if ind_1 == l1:  # all number in nums1 is belong to left
                    right_smallest = nums2[ind_2]
                elif ind_2 == l2:  # all number in nums2 is belong to left
                    right_smallest = nums1[ind_1]
                else:
                    right_smallest = min(nums1[ind_1], nums2[ind_2])
                if (l1 + l2) % 2 == 1:
                    return right_smallest

                if ind_1 == 0:  # all number in nums1 is belong to right
                    left_biggest = nums2[ind_2 - 1]
                elif ind_2 == 0:  # all number in nums2 is belong to right
                    left_biggest = nums1[ind_1 - 1]
                else:
```

```
                left_biggest = max(nums1[ind_1 - 1], nums2[ind_2 - 1])
            return (right_smallest + left_biggest) / 2.0
```

45. Jump Game II

my first idea about this: use a bds to iterate them layer by layer

[2, 3, 1, 1, 4]

2 can reach 3 and 1,

2 3 1 1 4

```
class Solution:
    def jump(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        if len(nums) <= 1:
            return 0

        # 2 3 1 1 4
        level = 0
        start = 0
        end = 1
        l = len(nums)  # l =5

        while end < l:  # end = 1 < 5
            new_end = end  # = 1
            for ind in range(start, end):  # form [0 to 1)
                if ind + nums[ind] >= l - 1:
                    return level + 1

                new_end = max(ind + nums[ind], new_end)
            start = end
            end = new_end+1
            level += 1
        return l
```

57. Insert Interval

search the insert position firstly. What is ~ in python?

```
class Solution:


    def insert(self, intervals, newInterval):
        s, e = newInterval.start, newInterval.end
        left = [i for i in intervals if i.end < s]
        right = [i for i in intervals if i.start > e]
        if left + right != intervals:
            s = min(s, intervals[len(left)].start)
            e = max(e, intervals[~len(right)].end)
        return left + [Interval(s, e)] + right
```

127. Word Ladder

```
class Solution:
    def ladderLength(self, beginWord, endWord, wordList):
        """
        :type beginWord: str
        :type endWord: str
        :type wordList: List[str]
        :rtype: int
        """

        beginSet = set()
        beginSet.add(beginWord)

        visited = {}
        for word in wordList:
            visited[word] = False

        if endWord not in visited:
            return 0

        import string

        level = 0
        while len(beginSet) != 0:

            level += 1
            newBeginSet = set()
            for word in beginSet:
                for ind in range(len(word)):

                    originalChar = word[ind]
                    for repl in string.ascii_lowercase:
                        if repl == originalChar:
                            continue
                        else:
                            newWord = word[:ind] + repl + word[ind + 1:]
                        if newWord in visited and not visited[newWord]:
                            if newWord == endWord:
                                return level+1
                            newBeginSet.add(newWord)
                            visited[newWord] = True
            print(newBeginSet)

            beginSet = newBeginSet
        return 0
```

560. Subarray Sum Equals K

Given an array of integers and an integer k, you need to find the total number of continuous subarrays whose sum equals to k.

Example 1: Input:nums = [1,1,1], k = 2 Output: 2

---

use a hashmap, for example

[1,2,0,3,4,1] target = 5

map 1->0 map 1->0, 3->1 map 1->0,

for each number, curSum = the sum from 0 to the current number, is curSum-target is in the map , then add the len(map[curSum - target]) to the res

return res

with this idea in mind, the code is like this :

```
class Solution:
    def subarraySum(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """

        curSum = 0
        res = 0
        m_ = {}
        m_[0] = [-1]

        for i in range(len(nums)):
            curSum += nums[i]

            if (curSum - k) in m_:
                res += len(m_[curSum-k])

            if curSum not in m_:
                m_[curSum] = []
            m_[curSum].append(i)
        return res
```

the time complexity is O(N), the sapce, here we used a map , so basically it's still O(N). but it's quite slow. So let's see others' code.

one improvement of my code, is that they used a counter rather than a list to store each one's index.

```
class Solution:
    def subarraySum(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """

        curSum = 0
        res = 0
        m_ = {}
        m_[0] = 1

        for i in range(len(nums)):
            curSum += nums[i]

            if (curSum - k) in m_:
                res += m_[curSum-k]

            if curSum not in m_:
                m_[curSum] = 0
            m_[curSum] +=1

        return res
```