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
```